Theory of Computation

Date : / /

1. Introduction
2. Regular Language & Finite Automata 50%
3. Context Free Language & Push Down Automata
5. ~~Turing~~ REC & RE languages & Turing Machine

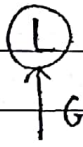
REC: Recursive
RE: Recursive Enumerable

Context Sensitive Language (belongs to turing Machine)

Introduction- | Languages
| Grammar
| Machine

Languages- bunch of string

Grammar- Way to represent language, rules so that valid string can be generated.

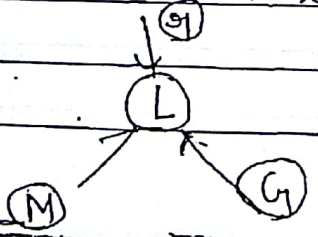


L(G)
↳ language which follow G
L(M) - Machine corresponding to language L, it is unique.

Machine- set of command which accept only valid string

L(M) = accept string if {string ∈ L}

Regular Expression(r.e.) - formal description or way to represent language.

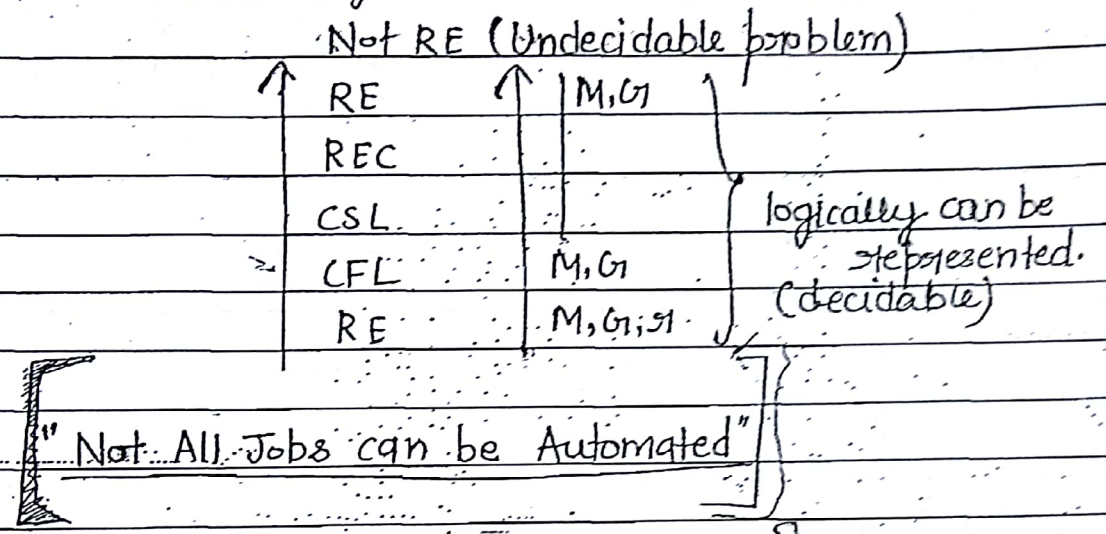


EXCELLENT

Date: / /

- R.e can not be generated for every language.
- It can only generate regular language (simplest one)
- R.e. is basically a formulae representation of all rules.

- There are some language which can not be Represented by Machine & Grammars; these are "not RE" which are those ie No logix can be written



"Not All Jobs can be Automated"

Positive Aspect of TOC - can be done
(-)ve Aspect of TOC - cannot be done.

If A problem cannot be solved if they can not be represented by a particular language.

Recognizing a language solving the problem

$L = \{s_1, s_2, s_3, s_4\}$

if $L = s_2$ (as input) it is recognizable

$L = s_5$ (not a recognizable)

(True but cannot be proved.

EXCELLENT

Language - 1. Alphabet - (Σ)
2. string
3. Concatenation ($L_1 . L_2$)
4. Reversal L^R
5. Null String
6. Length of String
7. prefix
8. Suffix
9. Substring
10. Power of string (w^n)
11. Σ^*, Σ^+ w-string
12. Language, $L \subseteq \Sigma^*$

Σ^* - Universal Language on Atbat Alphabet Σ.

13. REpresentation of languages - (Informal or formal)
- You can not write formal language for "NOT RE"

If two language represent the same logic, known as equivalent language.

Desirable problem - if any algo. exist to solve that particular problem.

- two formal language are interconvertable.
$M_1 = M_2$ if $L(M_1) = L(M_2)$

14. Operations on Languages
- Union . Intersection . Set Difference . Complement
- Symmetric difference.

15. L^R (Reversal ——————)
16. $L_1 . L_2$ = Concatenation of two Language
17. $L^2, \ldots L^n$ $(n \geq 0)$ - power of language
(Here n can never be negative as there can never inverse of language possible)
18. L^*, L^+ $\left(\frac{1}{L}\right)^{\alpha}$

EXCELLENT

@ ε (ε doesnot belong to Alphabet)

1. ALPHABET- non empty finite set of symbols
which can't be divided ↑ you can not break them down

$\Sigma = \{a, b\}$

$\Sigma = \{0, 1\}$ – Binary alphabet

$\Sigma = \{0\}$ – Unary Alphabet

$\Sigma = \{\}$ ✗ (not an alphabet)

$\Sigma = \{a, b, c, 0, 1, \ldots\}$ ✗ (it is not finite)

technically allowed ?

$\Sigma = \{123, 345, \ldots\}$ you can not use them

$\Sigma = \{01, 0, 1\}$ → Not valid as you are dividing the symbol

$\Sigma = \{0, 1, 2\}$ – ternary alphabet

(Σ^*)

2. String- ~~combinat~~ sequence of "0 or more" symbol (finite no of) taken from alphabet set (Σ)

@ ε- Epsilon - reserved for null string ie a valid string

$\Sigma = \{\varepsilon\}$ → not a valid alphabet as it is a valid string not a symbol

(ε- can replaced by 1)

$\Sigma = \{a, b\}$

$L = \{a, b, ab, ba\}$ Valid

3. Null String- length = ε

4. length of string- no of symbols in a-string

$|aba| = 3$

$|\varepsilon| = 0$

$|a| = 1$

EXCELLENT

- Every symbol is a string of length 1 = True but not every string is not a symbol.

- Order of alphabet does not matter but in string, order matter

$\{a, b\} = \{b, a\}$

$ab \neq ba$

Concatenation - if two strings u, v then concatenation is u.v or uv

$u = aab$ $\quad uv = aabba$

$v = ba$

length of given $|uv| = |u| + |v|$

1. $uv \neq vu$ (not commutative)
2. $|uv| = |u| + |v|$
3. $u(vw) = (uv)w$ - associative
4. $u \cdot \varepsilon = u$
5. [ε - identity element for concatenation as after concatenation no change occurs.]

Reversal - string - u^R , It is also known as Transpose operation.

If $u = aab$

$u^R = baa$

[$\neq$ means may or may not]

1. $u \neq u^R$
2. $|u| = |u^R|$
3. $(uv)^R = v^R u^R$ (Reversal law)
4. $(u^R)^R = u$
5. $u = u^R$ iff u is palindrome

Ex- u = abba

EXCELLENT

Page No.
Date : / /

5. Every string on unary ~~fa~~ alphabet are palindrome.

Every palindrome must be in form ww^R (Even palindrome)
wxw^R (odd palindrome)

language
- L_O - odd palindrome
- L_E - even palindrome
- L_P - Palindrome

$(ww^R)^R = (w^R)^R.(w^R)$
$= ww^R$

$(wxw^R)^R = (wx)^R w$
$= (w^R)^R (wx)^R$
$= wx^R w^R$
$= wxw^R$

as x is a sigmbol
$\therefore x \in \Sigma$
$w \in \Sigma^*$

Σ^* is collection of every possible string on Σ.

- String cannot be infinite but Σ^* always infinite.

~~7 Prefix~~ Theorem- Min. cardinality/size of alphabet = 1

Ques - On Alphabet Σ, how many string of length n is possible?

$\{|\Sigma|^n\}$ $N = \{|\Sigma|^n\}$

as for ~~A~~
$\Sigma = \{a, b\}$
$n = 3$

aab, aba, aaa, abb, baa, bba, bba, bab
8 (2^3)

Ques On Alphabet Σ, how many string ~~of length~~ upto or atmost length n?

$= \sum_{i=0}^{n} (\Sigma^i)$ $= \left\{\frac{|\Sigma|^{n+1} - 1}{|\Sigma| - 1}\right\}$

In $\Sigma = 2$ $= 2^0, 2^1, 2^2, 2^3$
$n = 3$

EXCELLENT

Ques - on a Alphabet. Σ, how many even palindrome of length n, odd palindrome?

$\Sigma = \{a, b\}$ $n=2$ even- ab ba = #2 (aa, bb)
aa bb

n = ③ ~~if n odd~~ ~~are odd~~

n even abba abab aaaa bbbb ~~abab~~ aabb bbaa, baba
baab

w abba
$n=3 = w w^R$ $|\Sigma|^1 = 2$

fixed last aab|baa
$\overbrace{n/2}$ $|\Sigma|^{n/2}$
combination

for even palindrome - $|\Sigma|^{n/2}$ for even = $|\Sigma|^{n/2}$
for odd $\uparrow|\Sigma|$ for odd = $|\Sigma|^{n/2} \times |\Sigma|$
aa|b|a|b|a a = n ↙ for w ↘ for x
$\frac{n}{2} \times \Sigma$ $= \{|\Sigma|^{n/2} \times |\Sigma|\}$ for x
↑ odd palindrome

odd palindrome - $|\Sigma|^{n+1/2}$

Ex ~~n=4~~ length, n = 11; Alphabet = binary; No of ~~even~~ odd palindr
$2^6 = 64$

for even palindrome upto length ~~ter~~ n, if n is even
$|\Sigma|^0 + |\Sigma|^{2/2} + |\Sigma|^{4/2} + \cdots + |\Sigma|^{n/2}$
$= \frac{|\Sigma|^{n/2+1} - 1}{|\Sigma| - 1}$

if n is odd = $[|\Sigma|^0 + |\Sigma|^1 + |\Sigma|^2 + \cdots + |\Sigma|^{n-1/2}] * |\Sigma|$

[finding no of even palindrome upto n]

EXCELLENT

$\left[\frac{|\Sigma|^{n+1}-1}{|\Sigma|-1}\right]$

- You are initially finding the no of even palindrome & then again finding odd no of palindrome by multiplying by Σ

$\omega x \omega^R$ → no of even palindromes

↑

$|\Sigma|$

for even palindrome of length atmost n

for $n=1 = |\Sigma|$ for $n=8$ $\Sigma=2$

for 0 $|\Sigma|^{0/2}$

2 $|\Sigma|^{2/2}$

4 $|\Sigma|^{4/2}$

8 $|\Sigma|^{8/2}$

$|\Sigma|^{0/2} + |\Sigma|^{2/2} + |\Sigma|^{4/2} + |\Sigma|^{8/2}$

$\therefore$ After Generalization

$\left\{ |\Sigma|^{0/2} + |\Sigma|^{2/2} + |\Sigma|^{4/2} + \dots\dots |\Sigma|^{n/2} \right\}$

$= \left\{ \frac{|\Sigma|^{n/2+1} - 1}{|\Sigma| - 1} \right\}$

for odd Let $n=9$

$n=0$ $|\Sigma|^{0/2} \times |\Sigma|$

$n=2$ $|\Sigma|^{2/2} \times |\Sigma|$

$n=4$ $|\Sigma|^{4/2} \times |\Sigma|$

$n=8$ $|\Sigma|^{8/2} \times |\Sigma|$

$= \left[\frac{|\Sigma|^{n+1/2} - 1}{|\Sigma| - 1}\right] \times |\Sigma|$

Date : / /

PREFIX - it is set of front part of string

Prefix of $\omega = \{ u \mid \omega = uv \}$

if $\omega = 00$, prefix of $\omega = \{ \epsilon, 0, 00 \}$

Null is a prefix of string always & string itself a prefix.

Suffix Back part-

Suffix of $\omega = \{ v \mid \omega = uv \}$

$\omega = 001$, suffix $= \{ \epsilon, 1, 01, 001 \}$

Suffix is can be null & the string itself.

Ques - Prefix(ω) $\cap$ Suffix(ω) $= \{ \epsilon, \omega \}$

~~False~~ False

$\omega = 000$

prefix(ω) $= \{0, \epsilon, 00, 000\}$

suffix(ω) $= \{0, \epsilon, 00, 000\} \neq \{\epsilon, \omega\}$ as

~~yes it is~~

$\{ \epsilon, \omega \} \subseteq$ prefix(ω) $\cap$ suffix(ω)

Ques If a string of length n, prefix(ω) = n+1

{ no of elmt in prefix(ω) = n+1 }

Ques - { string of length n, suffix(ω) = n+1 }

cardinality

EXCELLENT

Prefix & Suffixes are always unique.

Proper Prefix- if $\omega = 00$

$\{\text{prefix}(\omega) = \{0, \varepsilon\} \Rightarrow$ proper prefix

i.e ω is excluded.

every prefix other than string itself

proper suffix - same

$\{$Max length $= n\}$

↓ cardinality.

Substring- A substring s of ω,

$s = \{z \mid uzv\}$

- set of 0 or more consecutive symbol of given ω.
- Prefix & suffix both are substring.
- Null is always a substring.

$\omega = abcd$, $s = \{\varepsilon, a, b, c, d, ab, bc, bcd, abcd\}$

but $s \neq \{abd\}$ as not consecutive

- substring may or may not unique.

$\{\text{prefix}(\omega) \cup \text{suffix}(\omega) = \text{subset of substring}\}$

Ques- a string of length n with no repeated symbol.

No of ~~repeated sym substring of length n~~

$n = 0 = 1 \longrightarrow 1$

$n = 1 = \varepsilon, a \longrightarrow 2$

ab $n = 2 = \varepsilon, a, b, ab, ba, aa, bb \longrightarrow 5$

$\{$No of substring of length $n = 5\}$

EXCELLENT

No of sub string possible $= \frac{n(n+1)}{2} + 1$

No of subwords without $\varepsilon = \frac{n(n+1)}{2}$

Ques

No of substring of string of length n, with repeation of symbol in it?

~~AXIOMATIABLE~~

AXIOMATIZABLE $n = 13$

A - 3 times

I = 2 time

no of diff word = 13 - 2 - 1 = 10

∴ 1 length → 10

2 length ⇒ check for repeatition

AX AT AB ....

MA ZA

Io IZ → only different element

IX ~~ZT~~ TI

⇒ (no repitition around)

No repeated words

∴ Now for R bcoz

12 possible combination for 2 length

$\Rightarrow \frac{12 \times (12+1)}{2}$

$= 78$

$78 + 10 = 88$

If AX

AX then we will subtract 1 $12 - 1 = 11$

We do not check for Repitition we are not checking for 3 length for 3 length bcoz when no repitition for 2 as no repition there is no chance for repitition of in 2 as well string of length 3.

EXCELLENT

as $w \cdot w$ → nothing such can happen

$w^0 = \varepsilon$ (it means it is independent of what actually string is)

$w^1 = w$

$w^2 = w \cdot w$

$w^3 = w \cdot w^2$

$= w \cdot w \cdot w$

$w^m \cdot w^n = w^{m+n}$

$(w^m)^n = w^{mn}$

$w = 01 \quad w^2 = 0101$

w^{-1} is not possible as for $w^0 = w \cdot w^{-1}$

$\varepsilon = w \cdot w^{-1}$

nothing is possible which after concatenation result into ε (null string)

$((01)^2(100)^3)^R$

$= (0101\,100\,100\,100)^R$

$= 001001001010)$

$= ((001)^3(10)^2)$

$((u)^m(v)^m)^R = (v^R)^m(u^R)^m$

EXCELLENT

11. Σ^*, Σ^+

Σ^* - Every possible string on given alphabet.
- also known as star closure / Kleen's Closure / iteration.

$\Sigma = \{0\}$

$\Sigma^* = \{0, \epsilon, 00, 000, 0^9 \ldots\}$

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

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000 \ldots\}$

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

- Σ^* is always infinite. It can never be finite.

Σ^+ - positive closure

$\Sigma^+ = \{\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\}$

$\Sigma = \{0\}$

$\Sigma^+ = \{0, 00, 000, 0^9\}$

$\boxed{\Sigma^+ = \Sigma^* - \epsilon}$ it does not contain null string

$(\Sigma^+ \cup \epsilon = \Sigma^*)$

Ques - $(\Sigma^+, \cdot)$ → it has identity element (no) as it does not contain ϵ.

$(\Sigma^*, \cdot)$ - it contain identity elmt. (Yes)

~~Closed - Groupoid~~

~~CL + Asso~~

Closed + Associative - semigroup
Closed + Associative + Identity → Sem Monoid
Closed + Associative + Identity + inverse → Group
Closed + Associative + Identity + Inverse + commutative - Abelian Group

Σ^+ → semigroup
Σ^* → Monoid

∴ Inverse of a string does not exist.

{(Z, +) - Abelian Group}

13. Representation of Languages -

12. $L \subseteq \Sigma^*$ = bunch of strings on Σ of specific meaning for Grammar to them.

Ex - $\Sigma = \{0\}$
$\Sigma^* = \{\varepsilon, 0, 00, 000, \ldots\}$ (language should not accept anything)
$L_1 = \{\}$ → Null Language — length = 0 (cardinality)
$L_2 = \{\varepsilon\}$ → Language having null string (language is not null but it accept null string)
↓ No of elem = 1

$\lvert L_1 \rvert = 0$	$L_3 = \{0, 00\}$	→ Finite
$\lvert L_2 \rvert = 1$	$L_4 = \{00, 0000, 0^5\}$	

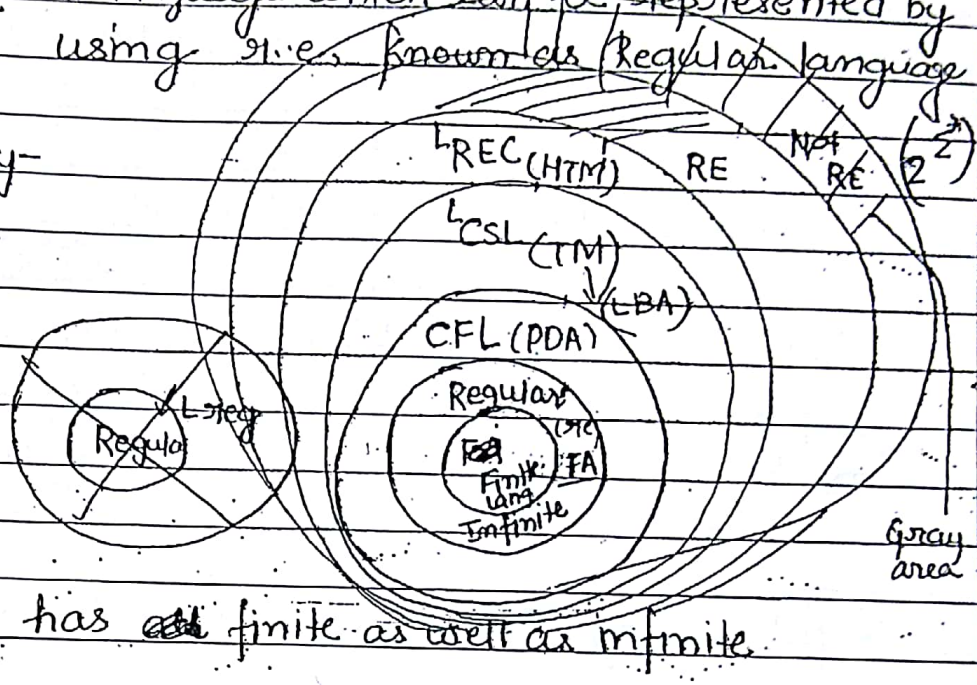
A Language can be finite or infinite
$L = \{0^{2n} \mid n \geq 0\}$
↓ infinite

EXCELLENT

When some restrictions are putted on ~~a language~~ Σ^*, then it forms a language.

Regular Language- A language which can be represented by using R.E. is known as Regular language

Chomsky Hierarchy-



- Regular language has ~~all~~ finite as well as infinite automata.

Regular Expression $*, +, (\ ), 0, 1$

(~~It donot allow intersection & compliment~~)

- Regular Expression is one which for which FA can be defined.
- FA has only finite memory i.e limited Memory.
- ~~Each R.L is a finite automat~~
- Each finite automata is RL but each RL is not FA.
- Each CFL is not RL but each RL is CFL.

$L = \{ 0^n 1^n \mid n \geq 0 \}$ CFL but not RL as it require memory for remembering n no of zero.

EXCELLENT

Page No.
Date : / /

CFL - context free Language - (Finite Automata with one stack)

- May be FA or infinite Automata.

Every Regular language is CFL. True.

CSL - Context Sensitive language

- Requires ~~PDA~~ Turing Machine

$0^n1^n2^n$ ~~$0^n1^n2^n$~~

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

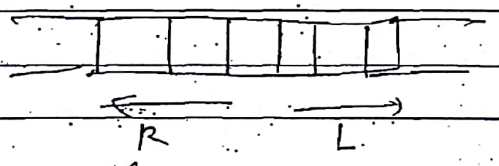
) cannot be solved by PDA

- TM has tape where it can read/write from both ends

TM = (FA + R/W) + R↔L

- In tape using FA & PDA you cannot move backward

- In TM, you can move in both direction right as well as left



000 111 222

Move for first match.

come back find for 1 untick value

0 0 0 1 1 1 2 2 2

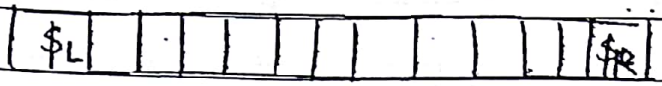
and so on

if no symbol left then its is accepted

EXCELLENT

used by CSL then we can use LBA (Lower bound Automata)

- LBA is a special TM where length of tape is bounded with ~~length~~ Left end Marker & Right end Marker



i.e, here working space of the tape is limited ie limited M/M.

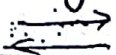
- You can also move right ~~to~~ or left acc to required.
- Here M/M of (LBA) automata is depending upon input if input expands, M/M also expands. Due to which it is able to solve infinite Automata as well. So, it is diff from FA because FA will have M/M which is independent of input.
- In FA, M/M size depends upon states in FA.

- LBA will Not be able to solve the problem which are recursive because req of M/M is not clear through input

For Recursive problem L_{REC} we use Halting Turing Machine (FA + R/W + R/L)

- the limitation with HTM is it ~~can~~ must halt ie, it must never get stuck in infinite loop.
- ~~It never halt as~~

A Machine does not halt if it get stuck in Right-Left Move, for



never ~~halting~~ hang of machine, what we did, for each move we enter into new state.

EXCELLENT

Date : / /

RE - a language in which for some input it may halt or may not halt.

After REC, system comes into state where it may halt or may not halt.

Ex- Software

- It uses RE (FA + R/W + R/E)
  - ↳ it may or may not halt

- Turing Machine has power equivalent to logic. no machine powerful than TM exist.

Church turing thesis, "Turing Machine has same power to logic"

- if logic exists for a problem, TM will work/exist.

Machine has two uses -
1. Compiler (to recognize special language) / Acceptor
2. Transducer (to perform computation)

The problems which a turing machine cannot handle is Not RE
↓ the problem which can not be represented by logic.

the whole classification is 2^{Σ^*} as it covers all the aspect of given string set
↓ power set of Σ^*
↓ possible on Σ.

EXCELLENT

- FA is not "not RE" & RE is not FA

13 Representation of Languages

Informal Method of Representation (Suitable for Human but not for system)
1. Listing Method
2. Statement Method
3. Set-Builder Method

for not RE

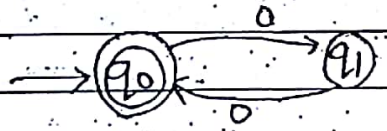
Formal (suitable for CSL, CFL, RL (all, every))
4. Machine
5. Grammar
6. Regular expression

↳ Least power as only can represent RL

Ex:- even no. of zero-

3. Set-Builder Method- $L = \{0^{2n} \mid n \geq 0\}$
2. Stmt - { even no of zero }
1. Listing = $\{\epsilon, 00, 0000, 000000, \ldots\}$

(4) Machine



(0) - final state
→ start/initial state
(practice)

(5) Grammar - $G_1 = (V, T, P, S)$ (S: Start Symbol)

$S \to 00S \mid \epsilon$

$V = \{S\}$

$T = \{0\}$

$P: S \to 00S \mid \epsilon$

6 Regular Expression i.e. $= (00)^*$

EXCELLENT

Operation on Languages- $\cup, \cap, -, L^c, L_1 \oplus L_2$

$-$ → Set Difference

L^c → Complement

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

$L_1 = \{10, 0, 11\}$

$L_2 \{11, 01\}$

$L_1 \cup L_2 = \{11, 0, 11, 01\}$

$L_1 \cap L_2 = \{11\}$

$L_1 - L_2 = \{10, 0\}$

$L_1^c = \overline{L_1} = \Sigma^* - L_1$

~~$L_1 \oplus L_2$~~ $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

$\overline{L_1} = \{\epsilon, 1, 00, 01, 000, \ldots\}$

$\overline{L_2} = \Sigma^* - L_2$

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

Ques $\Sigma = \{0\}$

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

$\overline{L} = \Sigma^* - L$

$= \{0^{(2n+1)} \mid n \geq 0\}$

$L \cup \overline{L} = \Sigma^*$

$L \cap \overline{L} = \phi$

The complement of a finite language is alway infinite = True

The complement of a infinite language is alway finite = False (may or may not be finite)

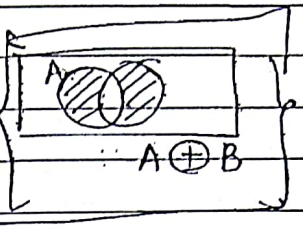
if the $\overline{L}$ is infinite then L may or may not be finite.

if the $\overline{L}$ is finite then L must be infinite

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

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

$L_2 - L_1 = \{10\}$

$A \oplus B = (A-B) \cup (B-A)$ → 

$L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$

$= \{10, 0, 01\}$ » [belong to L_1 only & L_2 only but not Both]

$L_1 \oplus L_2 = (L_1 \cup L_2) - (L_1 \cap L_2)$

$L_1 \oplus L_2 \neq (L_1 \cup L_2) - (L_1 \cap L_2)$

15. Concatenation & Reversal-

Reversal of a Language- $L^R = \{w^R \mid w \in L\}$

$L_1 = \{01, 100, 00\}$

$L^R = \{10, 001, 00\}$

1. $|L^R| = |L|$

2. $w_1 = w_2$ iff $\{w_1^R \equiv w_2^R\}$

To Reverse an infinite language,

$L = (01)^* 11$

$L^R = 11(10)^*$

$L = (01)^* 10$

$L^R = 01(10)^*$

$L = (10+00)^* 100$

$L^R = 001(01+00)^*$ ✓

$L^R = 001(00+01)^*$ ✓

$L^R \neq 001(10+00)^*$ ✗

as + is commutative

EXCELLENT

$(a+b)^* = \{a, ab, abb, baa\}$

$(ab+ba)^* \neq abba$ ~~extra term~~ extra term

$(a+b)^*$

Ques - { $L^R = L$ iff L is palindrome } → maybe true or maynot be.

as $L = \{10, 01\}$

$L^R = \{01, 10\} \Rightarrow$ true

but if

$L = \{101, 010\}$

$L^R = \{101, 010\} \Rightarrow$ true

} true for 'if'

{ but iff is used for definition so 1st condition get violated as 10 and 01 are not palindrome but $L^R = L$ }

16. Concatenation- $L_1 \cdot L_2 = \{uv \mid u \in L_1 \& v \in L_2\}$

$L_1 = \{01, 10, 11\}$ } it must cover every combination

$L_2 = \{1, 0\}$ }

$L_1 \cdot L_2 = \{011\ 101\ 111\ 010\ 100\ 110\}$

⇓ must not contain duplicate

Ie.

$|L_1 \cdot L_2| \neq |L_1| \times |L_2|$

{ $|L_1 \cdot L_2| \leq |L_1||L_2|$ }

↓ because ~~there may be some duplicat~~ combination

~~$L_1 = \{01, 10, 110\}$~~

~~$L_2 = \{1,$~~

EXCELLENT

$L_2L_1 = \{101, 110, 111, 001, 010, 011\}$

Also $|L_2L_1| \neq |L_1L_2|$

1. $L_2L_1 \neq L_1L_2$ (Not commutative)
2. $L_1(L_2L_3) = (L_1L_2)L_3$ (but associative)

Ex- $L_1 = \{0^n \mid n \geq 0\}$
$L_2 = \{1^n \mid n \geq 0\}$

$L_1 \cdot L_2 = \{$ (a) $\{0^n 1^n \mid n \geq 0\}$
(b) $\{0^m 1^n \mid m, n \geq 0\}$
(c) $\{0^m 1^m \mid m \geq 0\}$

as $L_1 = \{000\}$
$L_2 = \{11\}$ > as both n are local variable

$L_1 \cdot L_2 = \{0^m 1^n \mid m, n \geq 0\}$

$L_1 = \{0^n \mid n \geq 0\}$
$L_2 = \{1^n 2^n \mid n > 0\}$

$L_1 \cdot L_2 = \{0^m 1^n 2^n \mid m, n \geq 0\}$
∴ as their both must be equal

Power of a Language (L^n)

$L^0 = \{\epsilon\}$ (as contributing to identity element)

$L^1 = L$

$L^2 = L \cdot L$

$L^3 = L \cdot L^2 = L^2 \cdot L = L \cdot L \cdot L$

$L \cdot \phi = \phi$ {attaching nothing to L will result to ϕ}

$L = \{10, 01\}$

$L^0 = \{\epsilon\}$

$L^1 = \{10, 01\}$

$L^2 = \{10, 01\} \cdot \{10, 01\}$

$= \{1010, 1001, 0110, 0101\}$

To given string belong to L^2

- break into two parts
  - 10 10
- if successful break i.e the atom elmt is in L
  else doesnot belong.

$x \in L^n$ iff $x = \underbrace{-\cdot-\cdot-\cdot-}_{n \text{ parts}}$

ϵ_L

* if $L = \{01, 10, \epsilon\}$
  then
  $L^2 = 01$ as {$01 \in L^2$ as $01.\epsilon$}

EXCELLENT

if ε is there in L to check $x \in L^n$ divide it in to ~~n~~ less than n parts. ~~if it works~~

Ques
$L = \{0^n \mid n \geq 0\}$
(or) $L^2 = \{0^n \mid n \geq 0\}$ ✓ as both n are independent (??) local variable
(b) $L^2 = \{0^{2n} \mid n \geq 0\}$ false
bas if 0^{2n}

16. L^* & L^+ -
- closure of L, $L^* = L^0 \cup L^1 \cup L^2 \ldots\ldots$
(+)ve closure on L, $L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots\ldots$

Ex $L = \{01, 10\}$
$L^0 = \varepsilon$
$L^1 = \{01, 10\}$
$L^2 = \{0110, 0101, 1001, 1010\}$
$L^* = \{\varepsilon, 01, 10, 0110, 0101, 1001, 1010, \ldots\}$

- to check given string belong to L^*
  - break into any no of pieces (acc. to L).
  as 01 10 1001 011) Do not belong.

- if $L = \{01, 101, 110\}$
  01 101 00) Not possible.

- in L, if ε is there, it doesnot make any difference.

(false)

$\emptyset$

Ques - a. belonging of string to L^2 doesnot mean belonging to L^3 as if there is ϵ in L, it also belong to L^3.

Ques - $L^+ = L^* - \epsilon$ (false) (because L^+ can contain ϵ)

* [L^+ can contain ϵ as L contain ϵ but it will not contain L^0.]

Ques : $L^+ = L^* - \epsilon$ iff $\epsilon \notin L$

i.e; if a language is a null free language.

Ques $1^+ = L^*$ if L contains ϵ

EXCELLENT

Grammar is a 4-tuple

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

- V — Set of Variable (non empty & finite)
- T — set of Terminals
- P — Production
- S — start symbol $\in V$

by default S is start symbol, if not mentioned

- In every grammar, atleast one variable is required.
- Variable - capitals
- Terminal - small - finite. it can be empty.
  Empty - $S \to \epsilon$
- Production is also non empty.

  $|P| \geq 1$

  as for $\emptyset$, we need a dummy production.

  $S \to A$

all 3 terminal variable & production must be finite.

- for a single grammar, many single machine is possible but a minimal DFA, i.e no of states in a unique machine.

Let $P = \{S \to 00S, S \to \epsilon\}$ ..... if $P = \{S \to 00S \mid \epsilon\}$ $|P| = 2$

{cardinality of P = No of production = 2}

- for a single grammar, a single language is possible only.

Derivation — < > — to represent variable

→ Backus Nour Representation

—— without bracket, terminal

EXCELLENT

<Adj> → a | the

<Noun phrase> → boy | dog

<verb> → walks | jump | run }

S → <sentence>

T = { a, the, boy, dog, walks, jump, run }

V = { <S>, <N>, <A> <V> }

Cardinality, |P| = 9

Derivation is useful to match the I/P string to the pattern i.e for checking for compilation.

<sen> → <Noun><verb>

→ <Art><Noun><verb> When left variable

→ a <Noun><verb> getting substituted

→ a boy walks

- at one step of derivation, do substitute one variable only.
- A production can be used many no of times.
- A Grammar to generate infinite set, must contain recursion.

S → 0 S → Right Recursion.

S → S 0 → Left Recursion.

- Ambiguity has no mean with left or Right recursion.
- Recursion will always not lead to infinite language

EXCELLENT

Theorem

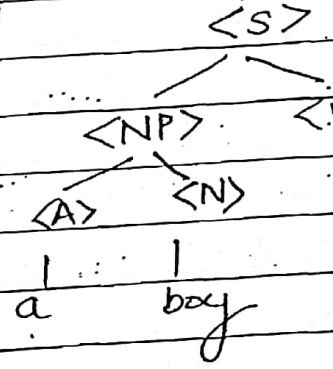
A string w belong to L iff
$\{ w \in L(G)$ iff $\exists$ derivation of w using P $\}$

[iff & only if]

- Derivation of a string cannot contain any variable.
  i.e $\{ w \in T^* \}$

- Derivation may or may not be unique for a given string. it can be atleast 1.

- Ambiguity donot come due to multiple derivation but due to multiple derivation tree



(for derivation tree, no ambiguity due to diff substitution bcoz you can do any side (Right or left) anytime

- Ambiguity is related to semantic & derivation tree is relate to semantic but derivation is related to syntax.

{ Derivation - Syntax
Derivation Tree - Semantics }

EXCELLENT

Page No.
Date : / /

$S \to 00S \mid \varepsilon$

$S \to 00S \to 0000S$

$\to 000000$

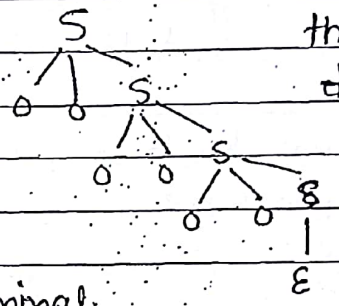
Sentential form - $(V \cup T)^*$

$\{00S, 0000S\}$

000000 — Sentence $[(T)^*]$

[A sentence is always a ~~sential~~ sentential form]

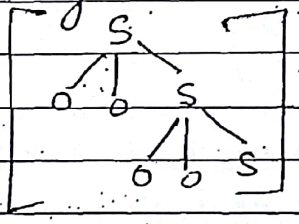
yield = 000000ε



the yield of a given ~~direct~~ derivation tree is the sentence

- Every leaf node is terminal

Partial derivation tree - • the yield can be sentential form
• it may have variable at leaf



Derivation-Tree - • Root must be start symbol
• Right derivation i.e correct
• ~~yield~~ yield must be sentence.

Partial DT - • Root may or may not be start symbol.
• yield may or may not be sentence.
• It may or may not be having ~~no~~ variable at leaf.

EXCELLENT

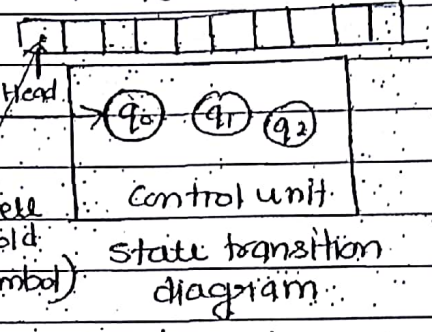
↳ Undecidable problem

[A Derivation Tree is always a Partial DT but PDT is not always DT.]

Machine

- The equivalence of two grammar is decidable iff they are regular (both)
- checking Ambiguity of a grammar is undecidable.

Finite Automata-



In both FA & PDA, tape is read only

- At end of tape, ϵ is placed.
- After Reading an I/P, it moves toward Right
- Read only head can move forward only

→ (O) — final state (accepting state)
→ At end of tape, machine must halt,
if halt at non final state - string rejected
if halt at final state - string accepted.

- PDA is provided with stack of infinite size.

for transducer - FA and TM
(Mealy Machine, Moore Machine)

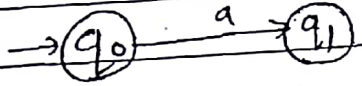
for acceptance — DFA & NFA

EXCELLENT

Deterministic Finite state Acceptance

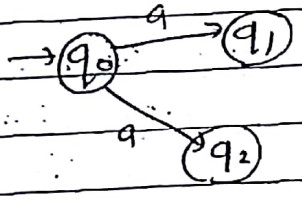
- single path i.e for an I/P, only one O/P exist.

i.e If you execute program multiple ~~out~~ time for one I/P, it must produce same result.



Non-Deterministic FA -

- for a single ~~of~~ Input, it may produce diff: O/P.



Machine work by divide itself & traces both path to move.

- It alway produce a result, if exist.

- It help to perform Backtracking to given problem by moving into many branch.
- It is easier to program.

- both DFA & NFA has same power of work.

DFA ≡ NFA

- if a program is Require backtracking & searching, NFA is used to simulate that program.

EXCELLENT

Acceptor (HTM, LBA)

↓ it can give o/p yes or no whether string accepted or rejected

Date : / /

~~Mealy &~~

- FA can transduce simple f^n but TM can transduce every logically computable f^n.
- A logically computable f^n is also partially recursive function.

Mealy & Moore - both have same power

in Mealy machine - O/P depending on current state as well as input symbol

in Moore Machine- O/P is only f^n of the current state. it doesnot depend on input symbol.

- for a specific state, O/P is specified, no matter whatever the input symbol is

- Mealy can be converted to moore & vice versa

- Mealy is more superior to moore as Mealy is easier to implement.

EXCELLENT

(Remove trap from DFA) Date : / /

Chapter-2

Regular language & Finite Automata

1. FA Theory
2. FA Design [L→M , M→L]
3. Regular Expression — language corresponding to a ~~grammar~~ expression
4. Regular Grammar:
   - L→G
   - G→L
   - L→π
   - π→L

5. Algorithms in FA (these algo runs on machine.)

- if you have n state in NFA, almost 2^n will be there in DFA

- ~~NFA to DFA~~ NFA to equivalent DFA (Subset Const^n algorithm)
- DFA to min. dfa (minimization of Automata)
- ε-closure

(NFA with ε Moves equivalent to NFA without Null Moves)

- The problem of NFA minimization exist, known as decidable problem. the Algo name is NFA minimization
- NP problem, slow down the process (exponential).

Tractable Problem – polynomial time problem.

Undecidable – no logic exist.

- if Algo exist to solve a problem, then problem is decidable.

6. Regular ? CFL ? CSL

7. Closure properties of regular language

8. Decidability properties of regular language

EXCELLENT

9. Application & Variations of FA
- Pattern Matching can be done by FA
- String Matching cannot be done by FA

10. TRANSDUCER (FA used as transducer)
- Mealy & Moore

Theorem -
- Mealy to Moore
- Moore to Mealy

FA Theory-

1. DFA & NFA specification
2. Diff b/w DFA & NFA
3. δ: transition f^n (δ)
4. δ^*: extended (δ^*) transition f^n
5. L(M). (Language representation of machine)
6. Theorem in FA

δ is used for mapping.

Ways to write transition f^n
- state diagram
- State table
- State ~~Diagram~~ transition function

(High User friendly) ↑

δ^* - is not the program where δ is the program.
↓ it is output

6. Theorem in FA -
- L($\bar{M}$) - language accepted by complimenting machine.
- finite language is always Regular.
- pumping lemma (to show a lang. is not regular)
- MYHILL NERODE THEOREM
- KLEEN'S Theorem

EXCELLENT

(bidirectional) ~~make a~~

- if A FA is not possible of given RL, then it is not RL.

DFA & NFA specification-

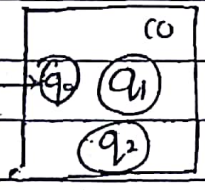
Automata-

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

- Q: finite non-empty set of internal states
- Σ: Input Alphabet finite nonempty
- q_0: initial state (represented by '→')
- F: final state

- It will halt at last of I/P
- IN each machine, state must be finite & non empty
- FA has finite M/M bcoz it remembered only through moving of states

(Ex) a b a a b



$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

Limitation - • finite state
• finite M/M

- Whatever is in Σ, can only be placed in I/P tape.

EXCELLENT

Page No. 13
Date : / /

δ- state transition function.
- Always finite
- No of command is finite

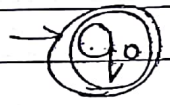
- In DFA, δ can ~~be~~ not be empty but in NFA, δ can be empty.

Ex- for $L = \phi$

NFA → q_0

DFA → q_0 0,1 (bcoz in DFA, each state must have 1 o/p for ~~DFA~~ each input symbol.)

NFA accepting $L = \epsilon$



- IN DFA, ε moves are not allowed

Final state - $F \subseteq Q$
- accepting state
- represented by ◎
- finite state
- can be empty

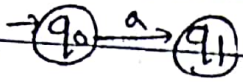
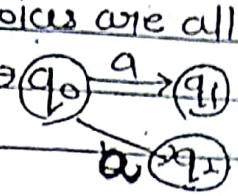
Diff. b/w NFA & DFA-

δ for DFA- $\delta_{Q \times \Sigma \to Q}$

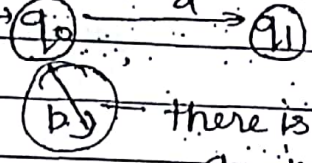
combination for current state with current I/P symbol will produce a resulting state

δ for NFA- $\delta_{Q \times (\Sigma \cup \epsilon) \to 2^Q}$

EXCELLENT

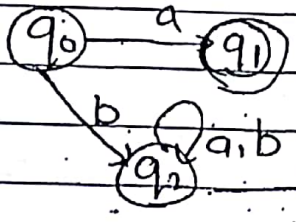
1.	No choices allowed 	1. Choices are allowed 
2.	No Null Moves Allowed	2. Null Moves allowed, known as ϵ-NFA
3.	No Dead configuration	Dead configuration Allowed

Dead Configuration- not defining transition for a input symbol so that M/C can not decide what to do.

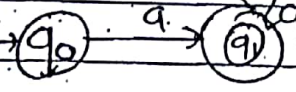
In DFA,  $\Sigma = \{a, b\}$

there is ~~not~~ no defining for action of q_0 in B.

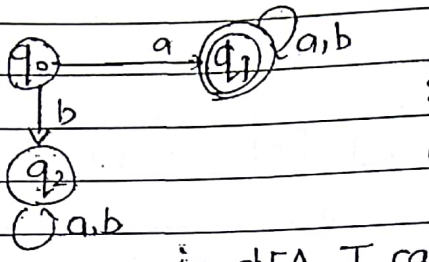
to do this, instead of Defining DC, use non final state



q_2 → trap state (when you enter to a particular state & can never go anywhere)

but in NFA, 

if "ba" comes
it get rejected in NFA

Ex - 

$\Sigma = \{a, b\}$

$Q = \{q_0, q_1, q_2\}$

in dFA I can only give

$\delta(q_0, a) = \{q_1\}$

but in NFA $\delta(q_0, a) = \{q_1, q_3\} \in$ power set of Q

$\Downarrow$

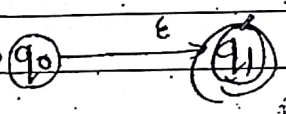
2^Q

i.e., we use 2^Q there

as 2^Q = power set of Q

$= \{\phi, \{q_0\}, \{q_1\}, \{q_2\} \{q_3\}, \{q_1, q_3\} \ldots\}$

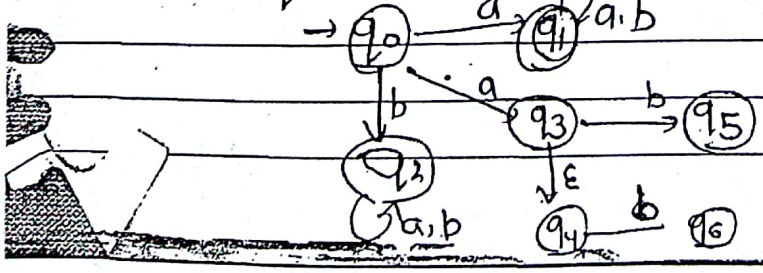
- Every DFA is a NFA but not vice versa.

Null Moves. 

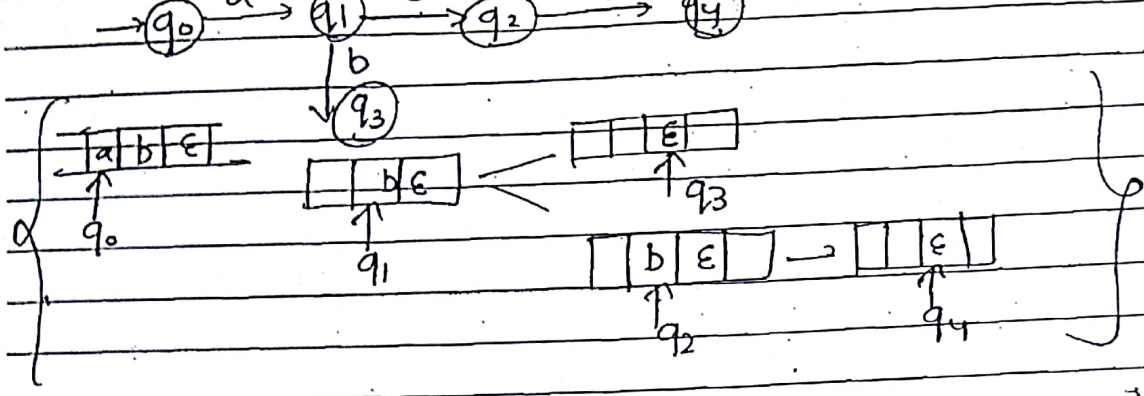
↓ it means a spontaneous jump from one state to another without reading an input.

- Null cannot be the part of I/P tape (it can be only at last of tape for showing end of tape)

- $\delta^*(q_0, ab)$ → means a question: (movement of machine on string)



machine will split itself at q_0, also at q_3 when ε move came, machine splits.



- NFA is helpful to apply search & backtrack.

NFA allow DC as mapping is to 2^Q if $\delta(q_0, b)$ given it will goto ϕ it can go to ϕ as $\phi \in 2^Q$

$\delta(q_0, b) = \phi$ → It can reject

δ^* ... $\delta^* Q \times \Sigma^* \to Q$ } for DFA

↑ as strings can be provided.

$\delta^* Q \times \Sigma^* \to 2^Q$ } for NFA

- if string is not accepted, it will ~~represent~~ return ϕ else return ~~final state~~ ~~set of visited states~~ final state

$\delta^*(q_1, \varepsilon) = \{q_1, q_2, q_3\}$ → q_1 —ε→ q_2 —ε→ q_3

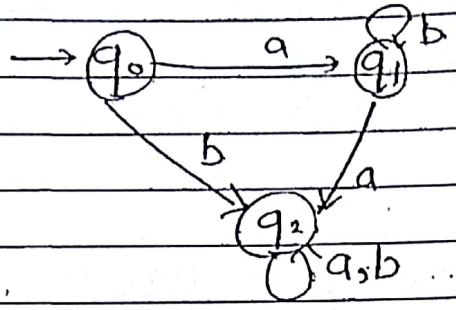
if ε is not specified as a move

$\delta(q_1, \varepsilon) = q_1$ as no move has done

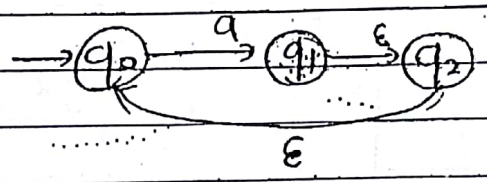
EXCELLENT

To check whether a M/C is NFA or DFA-

- firstly check for Null Move, if there, its NFA. otherwise
- check for alphabet
- check for exactly one arrow for all alphabet at each state. if there its DFA otherwise not. (there must be no choice & no DC)



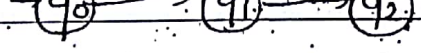
Ques whether it is NFA or DFA. $(q_2, \epsilon) = q$



$\delta^*(q_0, a) = \{q_1, q_2, q_0\}$
$\delta^*(q_0, \epsilon) = \{q_0\}$
$\delta^*(q_0, aaa) = \{q_0, q_1, q_2\}$
$\delta^*(q_1, a) = \{q_0, q_1, q_2\}$
$\delta^*(q_1, \epsilon) = \{q_1, q_0, q_2\}$
$\delta^*(q_1, aaa) = \{q_0, q_1, q_2\}$

or

q_2
$q_0 \to q_1 \to$

if 

$\delta^*(q_1, a) = \phi$ (bcoz it must have to be atleast on q_1)
$\delta^*(q_1, \epsilon) = \{q_1, q_2\}$

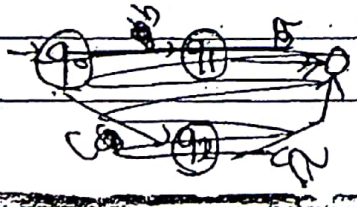
it tracks the final state that are visited after following the path of given string. since a is never traced, it returns ϕ.

EXCELLENT

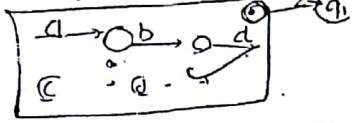
but in NFA

$q \in \delta^*(q, \varepsilon)$

DFA	NFA
1. $\delta^*(q, \varepsilon) = q$	1. $q \in \delta^*(q, \varepsilon)$
2. $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$ as only one path possible	if 2. $\delta^*(q, u) = \{q_j\}$ $j = 1 \cdots n$ moving to i state on u
3. $\delta^*(q, uv) \neq \delta^*(\delta^*(q, v), u)$	for v $\delta^*(q, uv) = \bigcup_{i=1}^{n} \delta^*(q_i, v)$ $= (q_1, v) \cup (q_2, v), \ldots$
4. if $\delta^*(q, u) = \delta^*(q, v)$ then, $\delta^*(q, uz) \Rightarrow \delta^*(q, vz)$ i.e, DFA is Right invariant but it is not left invarient i.e $\delta^*(q, zu) \neq \delta^*(q, zv)$ if implies $\{u \equiv v \Rightarrow uz \equiv vz\}$ (it must start from initial state)	
if $\delta^*(q, uz) = \delta^*(q, vz)$ then it is not always $\delta^*(q, u) \neq \delta^*(q, v)$	



EXCELLENT



Qur -

if given - $\delta^*(q,u) = \delta^*(q,v)$

then

$\delta^*(q,uz) \Rightarrow \delta^*(q,vz)$

↓

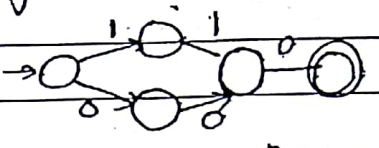
if it is starting from q_0

$\delta^*(q_0,uz)$

then u & v are equivalent

i.e $u \equiv v$ & $uz \equiv vz$

Not



11, 00

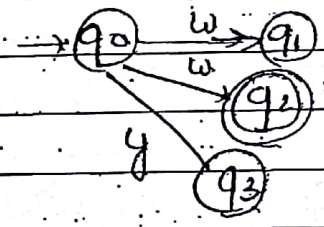
L(M) - Language accepted by a machine-

For DFA, All string which push machine to final state

$$L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \in F \}$$

for NFA, $L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0,w) \cap F \neq \phi \}$

↓ since a list of final state may exist



$\Rightarrow \{q_1, q_2\}$

$F = \{q_2, q_3\}$

$F \neq \{q_1, q_2\}$

so we take intersection

for $\overline{L(M)}$ - Rejected string by L(M)

DFA, $L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0,w) \notin F \}$

NFA $L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0,w) \cap F = \phi \}$

IN DFA, Rejection happen when no final state.

IN NFA- Rejection happen when no final state as well as stuck in DC

↓

it is handled as

$\delta^*(q_0, w) = \phi$ (if DC occurs)

$\phi \neq \phi$ (Rejected)

Theorems-

1. $\overline{L(M)} = L(\overline{M})$ (it is applicable only for DFA)

~~the language accep~~

if M accepts L(M) then $\overline{M}$ will accept $\overline{L(M)}$

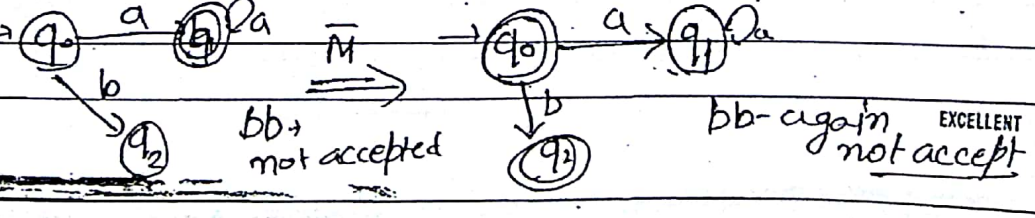
It is useful for machine designing for complement of a condition.

Ex - not starting 001

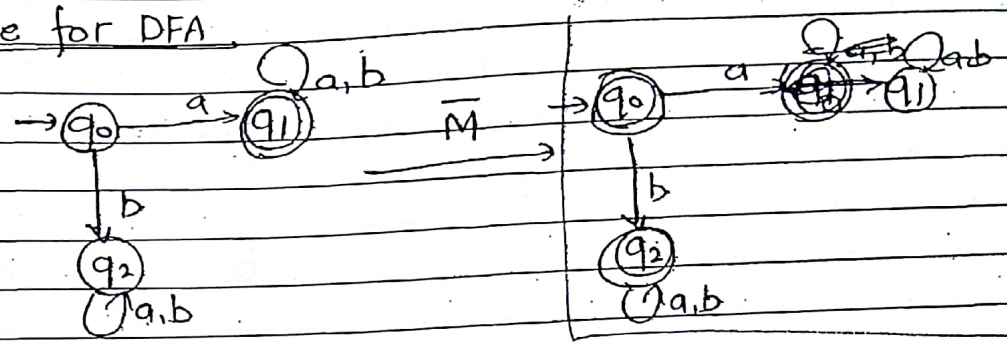
for complementing a Machine - Convert all final state to Non final state & non final state to final state

It is possible ~~bcz~~ in DFA because Acceptance & Rejection is one way through final state.

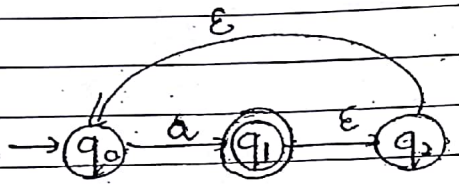
but in NFA, those dead state which were dead freely will also remain dead after even complement

as 

Example for DFA



Ques -



What is complement of language accepted by machine

[• Not possible directly as it is NFA, First convert in to DFA] as complement is possible for DFA by using that theorem.

You can find the language & then complement it.

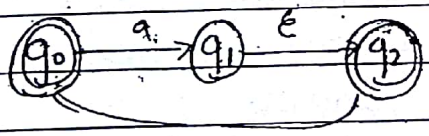
$L(M) = a^* = \{ a, aa, aaa \cdots \} = a^+$

[ε is not accepted as not input ε is given. as q_0 is not final state]

$\overline{L(M)} = a^* - a^+$

$= a(\varepsilon)$

Ques -



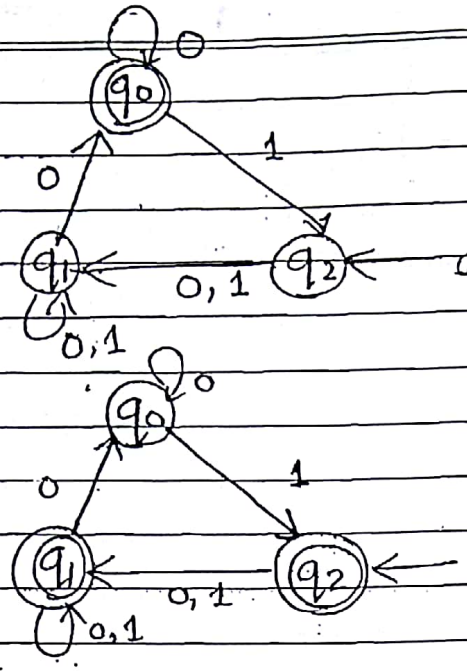
$L(M) = a^*$ (as ε is accepted

$\overline{L(M)} = \phi$

EXCELLENT

Page No.
Date : / /

Ques -



Construct the FA by convert the final state to Non final state & the language accepted by that Machine. : $L(\bar{M})$

$\epsilon.(0+1)^*$

$(0+1)^*$

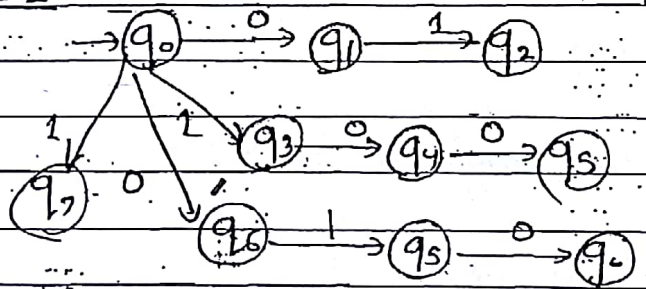
Do not apply that theorem as it is NFA.

Theorem

(1) A finite language is always regular i.e ~~you can design its FA~~

01

$L = \{01, 100, 010, 1\}$

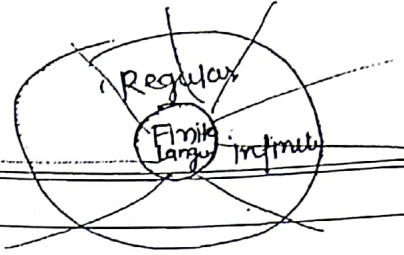


{FINITE ⇒ REGULAR}

- A Regular language need not to be finite always.
- A not regular language is infinite.

1. $p \Rightarrow q$ Direct
2. $q \Rightarrow p$ Converse
3. $\sim q \Rightarrow \sim p$ contrapositive
4. $\sim p \Rightarrow \sim q$ Inverse

EXCELLENT



- Every infinite language is ~~not~~ always not regular.
  ↳ False

Theorem 2 - for every regular language FA is possible.

REG $\Leftrightarrow$ ∃FA

- for every non re, fA is not possible
- for every re, fA is possible

Theorem 3 Pumping Lemma if L is Regular surely L will satisfy the pumping lemma.

- it is one way theorem.
- for Regular langu
- if L satisfy pumping lemma of RL, it may or may not be regular.
  i.e there are some non RL which satisfy pumping lemma
- Regular must satisfy pumping lemma.

Theorem 4 L is Regular if and only if No of myhill-Nerode equivalence classes is finite

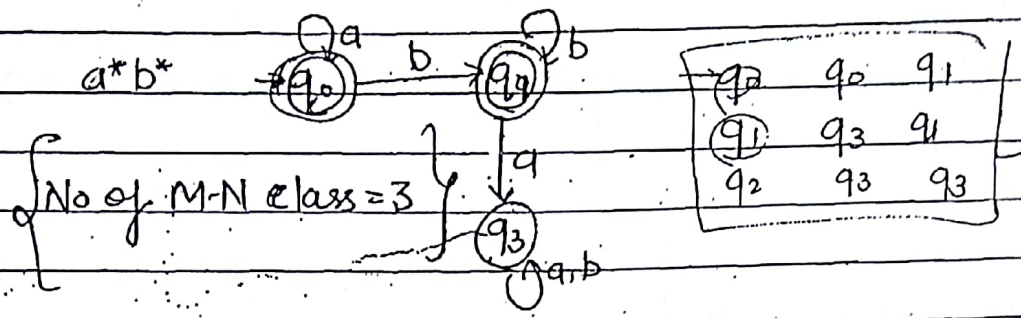
$\Leftrightarrow$

- two way theorem
- Each state in minimal DFA corresponding to Myhill-Nerode equivalence class
- So since no of state is finite L is regular bcoz for ever RL, DFA is possible.

EXCELLENT

• If M-N classes are infinite then L is not regular

Ques $L = \{a^n b^n \mid n \geq 0\}$ How many M-N classes are there
↓ infinite (Ans)
as it is not regular



[Myhill Nerode class cannot be connected with NFA.]

Myhill-Nerode class accept string if they are represented by string - (the classes are the strings which reaches that particular state)

a^*, a^*b^*, a^*b^*

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

• order of classes does not matter.

5. KLEENE's Theorem - A Language L is Regular if and only if there exist FA which accept L.

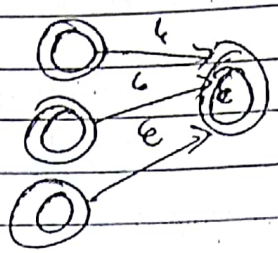
∴ L is regular ⇔ ∃ FA which accept L.
⇔ ∃ dfA which accept L.

• Every Machine that accept ~~FA~~ regular is FA.
↳ False
(as PDA, TM can accept FA)

L is regular if exist dFA - true
as NFA can be converted to DFA.

EXCELLENT

- L is Regular ⟺ ∃ nfa with single final state
  ↳ Yes
  (bcoz each Nfa with many final state can be converted to NFA with single final state by using ε-move)



- ⊙ L is Regular ⟺ ∃ dfa with single final state
  ↳ False

- ⊙ L is Regular ⟺ ∃ NFA without Null Moves
  ↳ Yes (from given NFA, null move can be removed).

- ⊙ L is Regular ⟺ ∃ NFA with DC
  ↳ Yes (it is equivalent NFA by increasing the state)

- L is regular ⟺ ∃ NFA without choice
  ↳ Yes

choice, Null, DC are the luxury to NFA.
•DFA, NFA both have same power

- L is regular ⟺ ∃ r.e. for L → (Yes)

- nfa ≡ dfa ≡ r.e. ≡ r.l. (have same power)

- L is regular ⟺ r.e for L without * (False)

- L is regular $\Leftrightarrow$ $\exists$ RG for L (True)
  (Regular Grammar)

- L is regular $\Leftrightarrow$ $\exists$ Right linear regular Grammar L
  (True)

~~Cont...~~

Grammar

1. Type 0 $\leftrightarrow$ TM $\leftrightarrow$ RE (Turing Machine)
2. Type 1 $\leftrightarrow$ CSL $\leftrightarrow$ LBA (context sensitive Grammar)
3. Type 2 $\leftrightarrow$ PDA $\leftrightarrow$ CFL (context free Grammar)
4. Type 3 connected to FA & RL FA $\Leftrightarrow$ RL $\Leftrightarrow$ Type 3
   also known as regular Grammar

Formal Grammar

for Type 2 grammar you can also write Type 0 & Type 1 Grammar

- There is no specific grammar specified for HTM and REC.

Not R.E - Not Machine no grammar. (Informal language)

RE ~~Gra~~ Languages are formal language & grammar known as formal Grammar

Type 0 : • Every Grammar is type 0 grammar
- Unrestricted Grammar (UG)
- Phrase Structured Grammar (PSG)
- Recursively enumerable Grammar.

EXCELLENT

for type 0, a single restriction is

for $u \to v$ $\quad u, v \in (V \cup T)^*$

{u must contain atleast one variable}

bcoz only terminals cannot be expanded.

$aa \to X$ ✗ $\quad aA \to bB$ ✓

- Type 1, 2, 3 are comes under type 0.

Type 1 grammar – at each step, there must be an expension or it may be same but can not contract.

i.e for $|u| \leq |v|$

as $BaA \to aB$, $|u| = 3$, $|v| = 2$, so Not a type 3

* contraction is not allowed in type 1, but it allow $S \to \varepsilon$.

for type 1- $u \to v$

$u, v = (V \cup T)^*$

but $v = \varepsilon$ if u = single terminal (start symbol)

ie $aS \to \varepsilon$ (not allowed)

$S \to \varepsilon$ is allowed only if S not appear in RHS of production

Ex- $AB \to aS$, $S \to \varepsilon$ — false

EXCELLENT

~~$v \in (VUT=$~~

$v \in (VUT)^*$

$S \to aAB \mid \epsilon$
$A \to ab$
$B \to c$

$+$ (Type 1) $+$ (Type 0)

* it must also be non contracting.

Type 3 Grammar:- Type 2 + Type 1 + Type 0

+

for $u \to v$

$v \in T^*V + T^*$ — Right linear Regular Grammar

$v \in VT^* + T^*$ — Left linear Regular Grammar

RLRG $A \to aaB \mid aa$

LLRG $A \to Baa \mid aa$ ↳ right hand side can contain only one variable.

A Type 3 Grammar can be RLRG or LLRG but not both-

$S \to aaB \mid aa$
$B \to ccD$
$D \to Aa$

> Not regular grammar as both LLRG & RLRG

Q. $A \to B$ → allowed in Type 3

EXCELLENT

Date : / /

- Every regular grammar is right linear or left linear

Cont....

- L is regular $\Leftrightarrow$ $\exists$ ~~RL~~ right ~~rel~~ linear regular grammar on L

Ng - Yes bcoz each Right linear can be converted to Left linear.

{ it implies Right Linear RG & Left Linear RG both have same power.

- L is regular $\Leftrightarrow$ $\exists$ r.g without recursion.

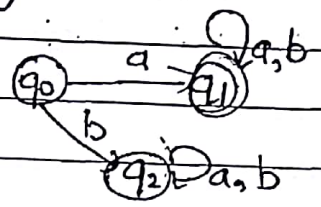
↳ False because recursion is *. (it is neccessity)

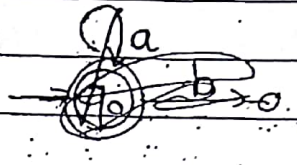
EXCELLENT

The no. of states is unique in DFA as well as NFA.

$\Sigma = \{a, b\}$

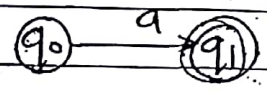
Ques1- starting with a.





To design a Minimal Machine,

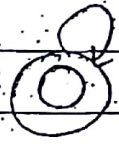
1. Theorem - if |Wmin| has n states then no of state in (min. fa) $\geq n+1$ (in case of infinite language) as for start with a

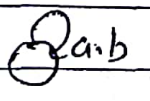


2 states are required atleast.

2. reuse the state if possible
3. filling up of blanks i.e for each state map for all I/P alphabet

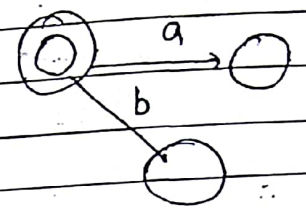
Types of state in FA:-

1. Permanent Accept -  (It is used when string already satisfy the cond") one you enter, always accept'

2. Permanent ~~Accept~~ Reject/Trap-  once enter just reject

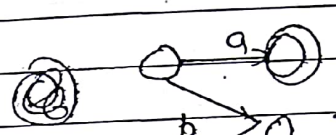
(if string in that state it will never satisfy the cond")

3. Temporary Accept / ~~Reject~~ -



(it is used when condⁿ satisfy at a ~~moment but may~~ fail in future)

4. Temporary Reject -

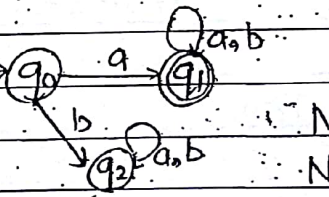


Can go anywhere. (when condⁿ is not satisfied now, but may be satisfied in future)

- In Minimal DFA design, every state does an unique work.

Moves - staticscope
Regressed ← (hope)
forward →
trap

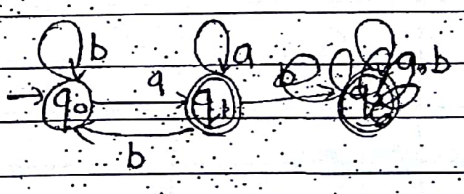
1) starting with a-



a(a+b)*

No. of state = 3 (DFA)
No of state in min NFA = 2

3) End with 'a'



→ e = (a+b)*a
No of state in min DFA = 2
No of state in min NFA = 2

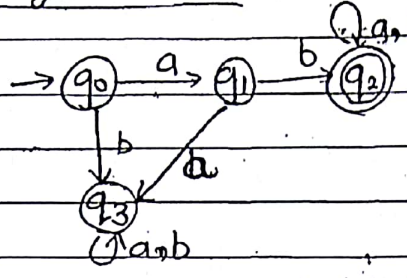
EXCELLENT

→ Use these tricks

- substring machine will always having PA state.
- starting with "a" will always have a trap & ending with "a" has regression

Page
Date

3. starting with "ab"



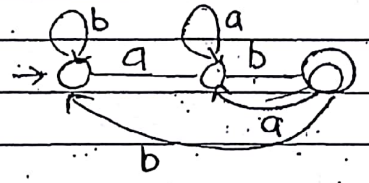
No of state (DFA min) = 4
No of state in (NFA min) = 3

re ab(a+b)*

n size starting- require atleast (n+1) state + 1 for trap
(n+1+1) → for dfa
(n+1) for NFA

5 size starting-
7 state + 1 for trap
8 for DFA minimal
7 for minimal DFA.

5. ending with ~~AB~~ ab



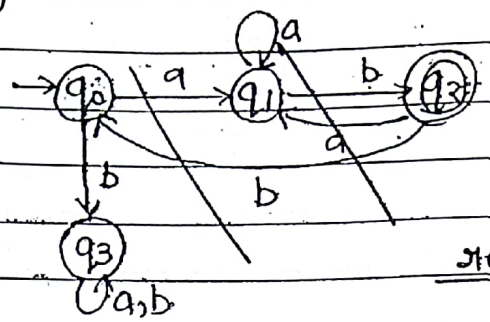
No of states = 3 (Minimal)

re- (a+b)*ab

n size end string- require n+1 state (atleast)
as for n=2
it requires 3 states

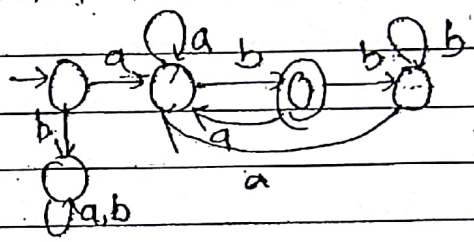
~~Std~~	TRAP	PA
starting with	✓	✓
Ending with	X	X
substring	X	✓

6. Starting with a & end with b



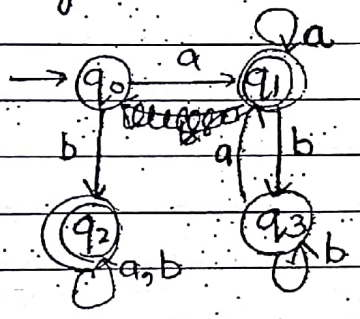
Min state (DFA) = 5
(NFA) = 4

re- a(a+b)*b



7. Starting with aa end with ab

(7) Starting with a end with a

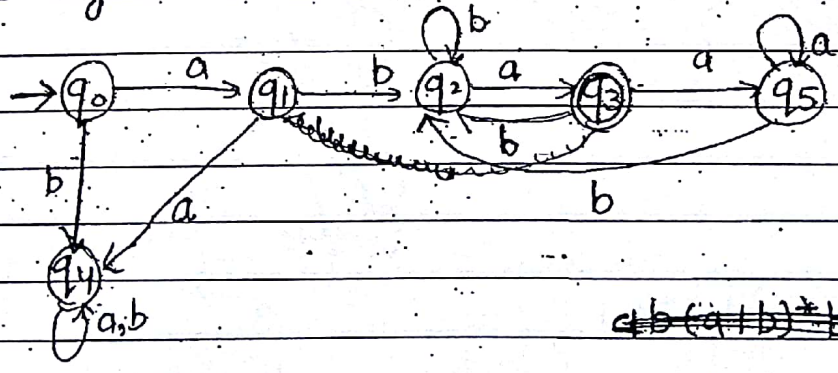


Min state (DFA) = 4
(NFA) = 3

a(a+b)*a

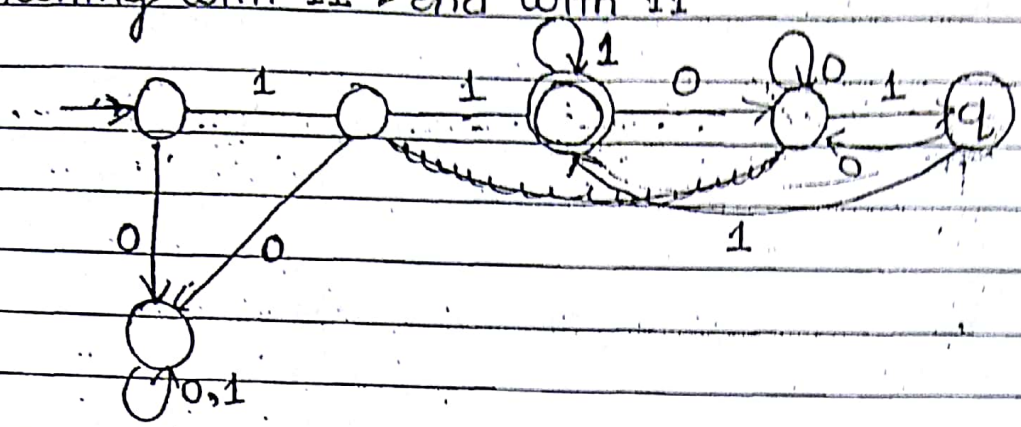
a(a+b)*a +a

(8) starting with 'ab' and end with 'ba'



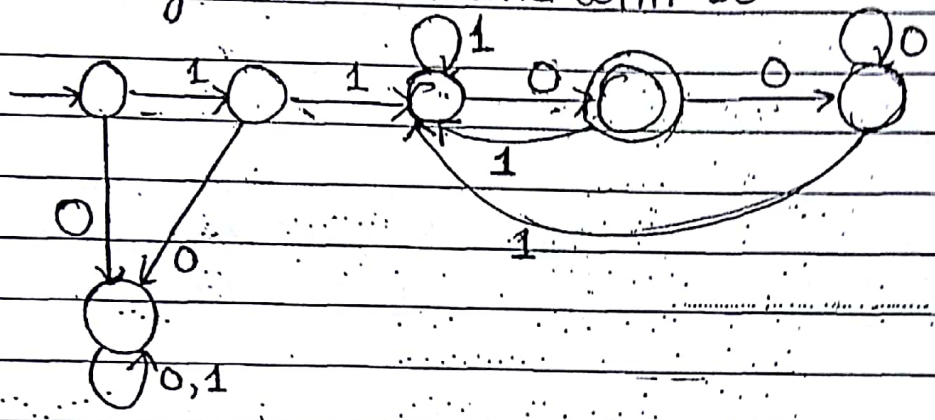
ab(a+b)*ba

EXCELLENT

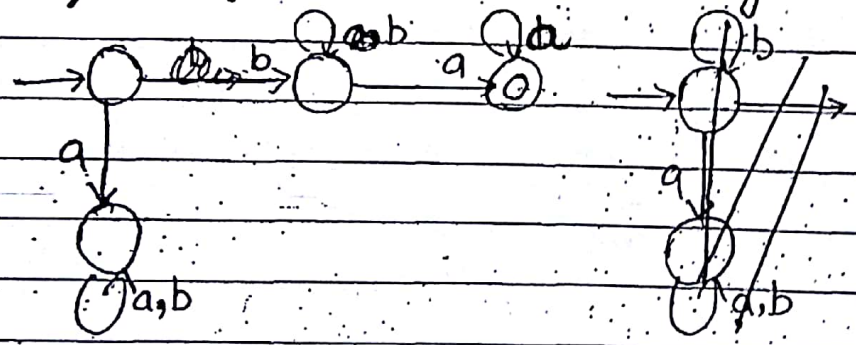


1111010

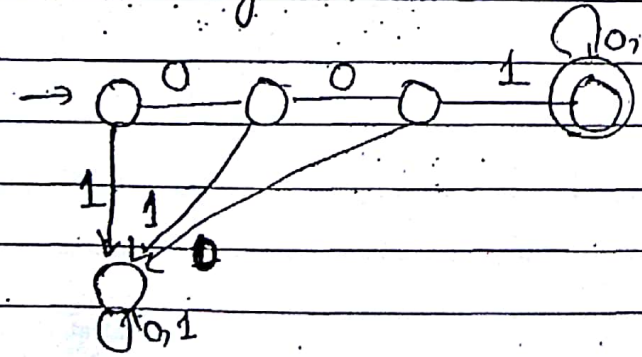
10. starting with 11 & End with 10



11. Not starting with a or not ending with b

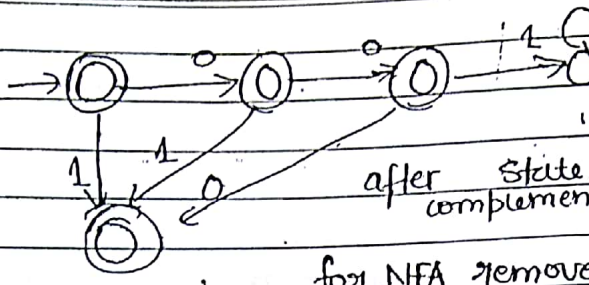


K. non starting with 001



Complement is

EXCELLENT



after complement State in Min DFA = state in original DFA

for NFA remove traps = state (min. NFA) = 4

* if a DFA is given, its complement will have same states i.e no. of states
* If a DFA has n state, state in its complement will be n
* if a NFA has n state, state in its complement will be nor less than n as it will remove all the trap states. (n-1)

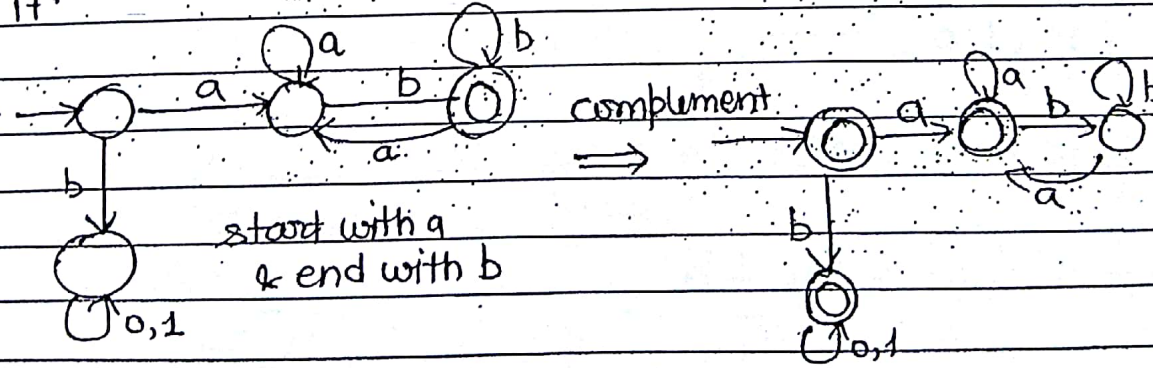
---

12) Not start 'a' or not end with b

start A → A ∪ B → start B

$(\overline{A \cup B}) = A \cap B$

∴ design start with a & end with B & then complement it.



EXCELLENT

Not A or Not B

↓

[just complement the machine & answer two & matches them]

* To complement NFA of state n

1. Convert into equivalent DFA by adding trap
   may be n or n+1
   (when by trap no need to add trap)
2. Complement then no of state is same.
3. Convert again to NFA by removing trap
   if trap added previously → no of state = n or n+1 as no removal of trap
   if trap not added → no of state = n or n-1

(13) {01, 100}

14) containing atleast 1 'a'

15) ——— exactly 1 'a'

16 . atmost 1 'a'

17) containing substring '0'

EXCELLENT

In case of finite language, if $|w_{max}| = n$ then

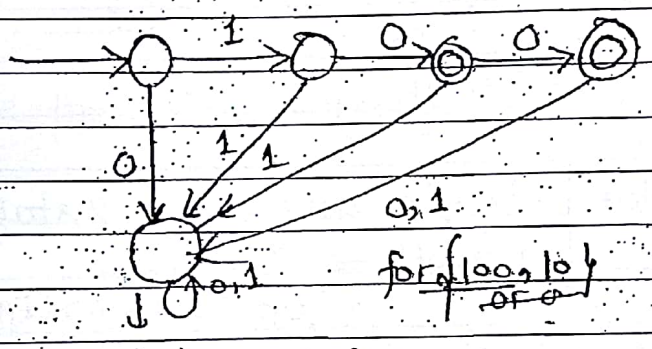
{no of states in min. ~~DFA~~ nfa = n+1} — trap state mayor may not required

{no of states in min dfa = n+2}

because n+1 + 1 → trap state again

↓ for max string

1. First accomodate biggest string:



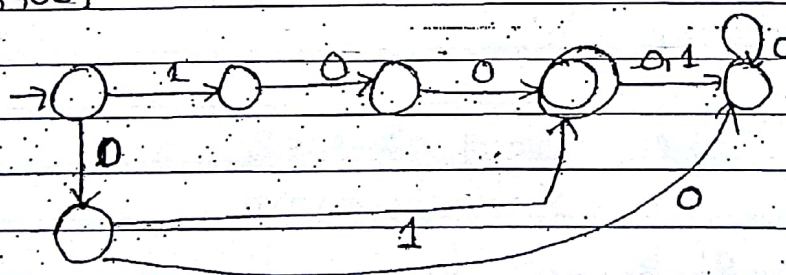
for {100, 10} or 0

if finite language & DFA is generated trap is required always at final state

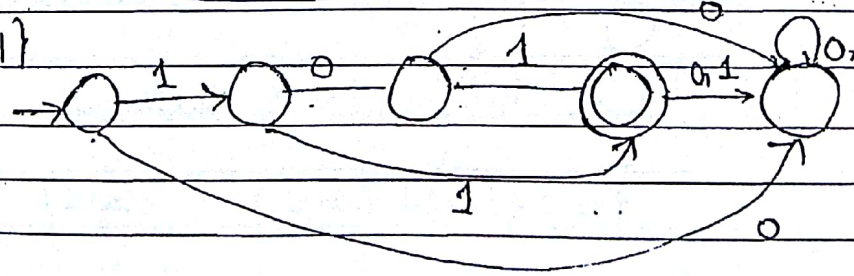
~~{10, 100}~~ {10, 100} No of state = 4 (NFA)
= 5 (DFA)

2. try to accomodate other in it if possible.

{01, 100}



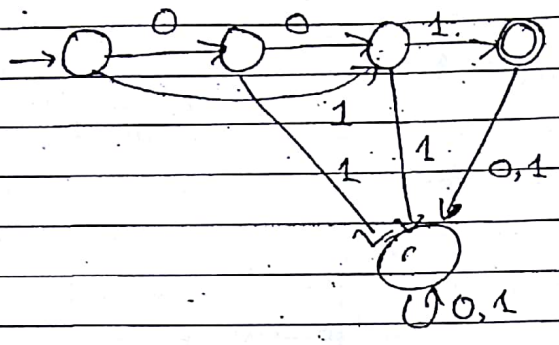
{101, 11}



{A → min NFA}

Page No.
Date : / /

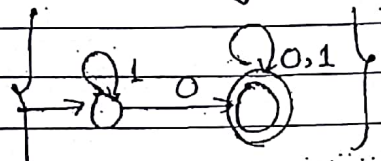
{001, 11}



{try to use each start or ending}

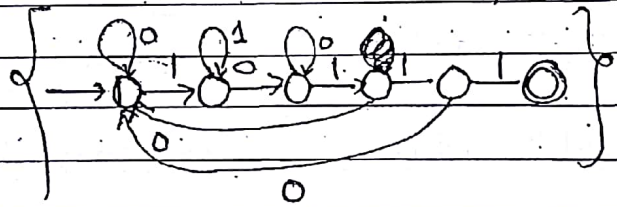
14 - containing the substring:- Never had a trap

substring-0

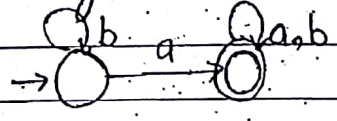


for [1 length substring = 2 states
n length → n+1 states
(for both NFA & DFA)]

{1011}

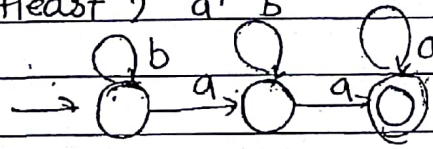


15) containing atleast 1 'a'



No of states = 2

atleast 2 'a' b

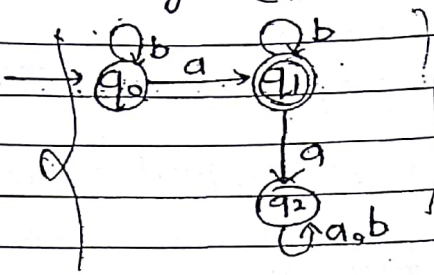


No of states = 3

{n a's - will have n+1 state}

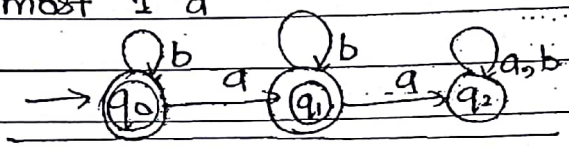
EXCELLI

16.) Exactly 1 'a' (trap exists here)



No of states = 3 (DFA)
= 2 (NFA)

17.) Atmost 1 'a'

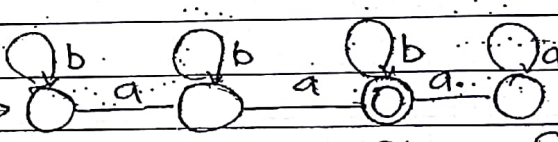


No of final state = 2
(trap also exists here)

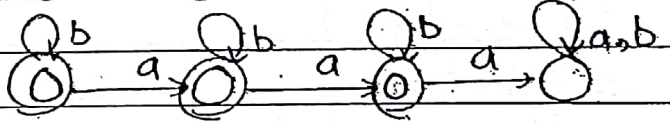
- If one final state, exactly come into existence

for 2a

exactly 2a's



atmost 2a's



for n (exactly) states = n+2 (DFA)
n (atmost) = n+2 (DFA). but no of final state = n+1

for NFA, n (exactly), states = n+1
n (atmost), state n+1

EXCELLENT

Page No.
Date : / /

for a language, there exist more than one regular Grammar

elegant expression- expression with no redundancy which is not required.

- It is possible to have more than one elegant expression

- Regular expression is defined asfollows-

1. ϕ, ϵ are regular expression.
   if $r = \phi$ $L(r) = \{\}$
   $r = \epsilon$ $L(r) = \{\epsilon\}$
2. $a \in \Sigma$ is regular expression.
   $r = a$ $L(r) = \{a\}$
3. r, s are r.e., then
   $\Rightarrow r + s$ is r.e.,
   $\downarrow$ + must operate b/w two r.e.
   $\{ L(r+s) = L(r) \cup L(s) \}$
   $r = a+b$
   $\therefore L(r) = \{a, b\}$
   - $r = ab + ba$
     $L(r) = \{ab, ba\}$
   - if $r = a^* + b^*$
     $L(r) = \{$ string of any no of a ; string of any no of b $\}$
4. if r is r.e., & s is r.e.
   then $r.s$ is also r.e
   $\{ L(r.s) = L(r).L(s) \}$

Ex- $r.s = a^*b^*$

$L(r.s) = \{$ any no of a with any no of b but a followed by b but not b not followed by a $\}$

EXCELLENT

+, . binary operation.

7. $a^*.+$ – not a regular expression

8. $a^*\phi$ – is a regular expression

$a^*.\phi = \phi$

9. $a^*.\epsilon = a^*$

10. if r is r.e., then r^* is also a r.e.,

11. $()^*$ is not a regular expression.

$(\phi)^*$ is regular expression.

12. if R is a regular expression

(R) is also a regular exp.

( ) is used to violate the precedence

$(() \geq * \geq . \geq +)$

- $r = (ab)^*$ $L(r) = \{\phi, \epsilon, ab, (ab)^2, (ab)^3 \ldots\}$

~~$r = (a^*+$~~

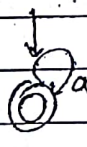
- ~~$r = a^*$~~

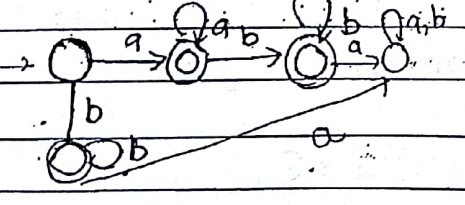
a^*b^* = any string without ba

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

$r = (a+b)^*$ = [any combination of a & b]

~~as~~ a can follow b as well b can follow a as + is commutative.





	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_2	0
q_3		

EXCELLENT

• Power of Finite Automata is pattern matching

1. for starting with a - $a(a+b)^*$
2. end with a - $(a+b)^*a$
3. for a at 3rd pos^n - $(a+b)(a+b)a(a+b)^*$
4. for 3 bit string & a at 3rd pos^n -
   $(a+b)(a+b)a$
5. 3 bit is a & 2nd last is b
   $\hookrightarrow (a+b)(a+b)a(a+b)^*b(a+b)$
6. .....

7. Starting with 11 & end with 11

$$11(0+1)^*11+111+11$$

adding out the missed bits

8. start - 10 end - 01

$$10(0+1)^*01+101$$

9. start with 00 & end with 11

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

10. not starting with 01

~~$(1+0)^*-01(1+0)^*$~~

$= 00(1+0)^*+11(1+0)^*+01(1+0)^*+0+1+\varepsilon$

↓ possible comb^n

can have lesser length strings

5. not starting with a or not ending with b

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

bcoz of 'OR'

as b will be in not ~~ending~~ starting with a

EXCELLENT

but OR will have

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

{Mode Machines are always regular}

12. $L = \{001, 11\}$
$re = 001 + 11$

- $L = \{11, 001, 10, 110\}$
$re = 11 + 001 + 10, 110$
$= 11(\epsilon + 0) + 001 + 10$
- $L = \{011, 111\}$
$re = (0+1)11$
- $L = \{110110 \quad 11001010\}$
$re = 11(01+00)10$

13. containing the substring 'a'
$re = (a+b)^*a(a+b)^*$

14. Contains atleast 1 a
$(a+b)^*a(a+b)^*$
exactly 1a b^*ab^*
atmost 1a = ~~atmost~~ exactly 1 a + exactly 0 a
$= b^* + b^*ab^*$
$= b^*(\epsilon + ab^*)$
atleast 2 a = $(a+b)^*a(a+b)^*a(a+b)^*$
exactly 2a = $b^*ab^*ab^*$
atmost 2a = $b^* + b^*ab^* + b^*ab^*ab^*$

atmost 3a = $b^* + b^*ab^* + b^*ab^*ab^* + b^*ab^*ab^*$
$= b^*(\epsilon + ab^*(\epsilon + ab^*)) = b^*(\epsilon + ab^*(\epsilon + ab^*))$

atleast 1a
↳ $(a+b)^*a(a+b)^* = b^*a(a+b)^* \neq (a+b)^*ab^*$

for atleast 2 a's

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

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

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

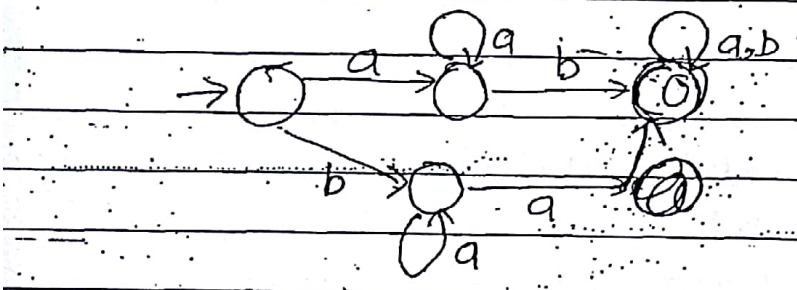
$\cong (b^*a(a+b)^*ab^*$

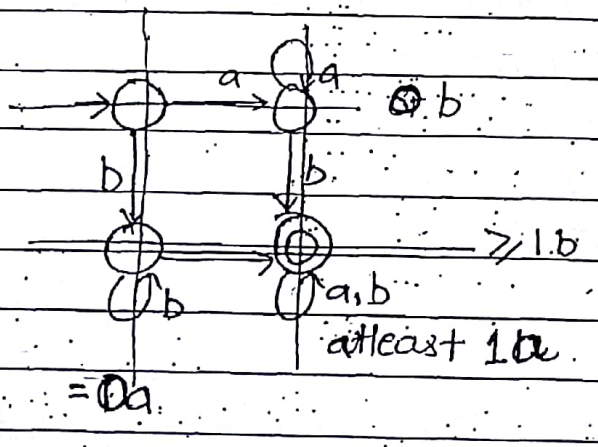
all these are equivalent to each other.

16-Apr

~~18 atleast 1 "a" &~~

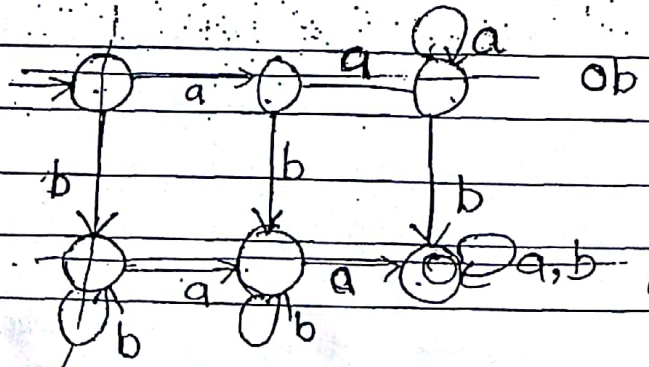
11. atleast 1 "a" & at least 1 "b"





final state is obtained by intersection of line meeting the result.

atleast 2a & atleast 1b

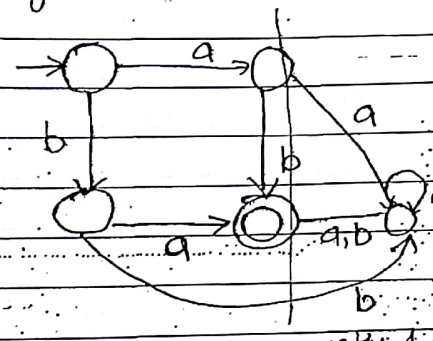


atleast 2a & atleast 1 b

for atleast m "a" & atleast n "b"s

Min. DFA	(m+1)(n+1)
Min. NFA	(m+1)(n+1) as there is no trap state.

12. Exactly ~~a~~ 1 "a" & exactly 1 "b" (finite language)



exactly 1a & 1b

if there is a trap condition lead to exactly)

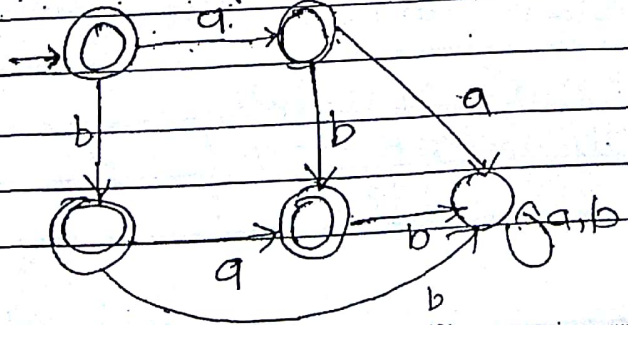
R.e = ab + ba

~~exactly 2'a' & exactly 1'b~~

Min DFA	(m+1)(n+1)+1 (as trap only introduced)
Min NFA	(m+1)(n+1)

13- atmost 1a & atmost 1 b

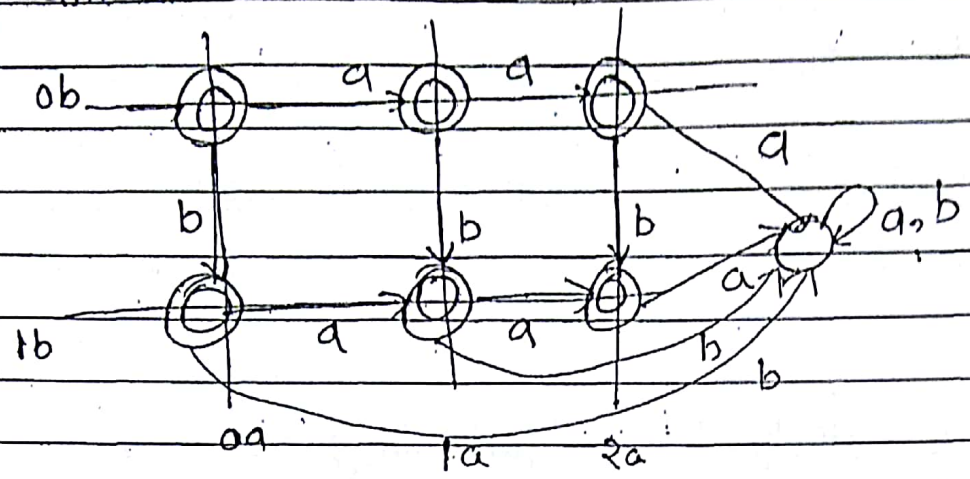
~~at~~ =0a or 1a (ab or b)



in any DFA, if all states are final state then language is Σ^* this property is only satisfied by ~~DFA~~

EXCELLENT

* atmost 2a's & ^1 b

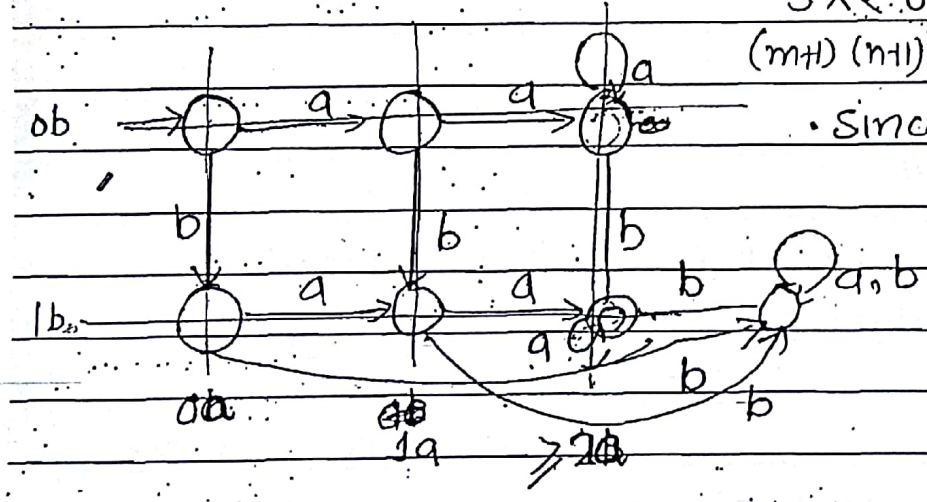


min. DFA = (m+1)(n+1)+1
min. NFA = (m+1)(n+1)

*

Atleast 2a & exactly 1b

[grid is required of size 3×2 or 2×3 (m+1)(n+1)]



• Since exactly is there, trap is required)

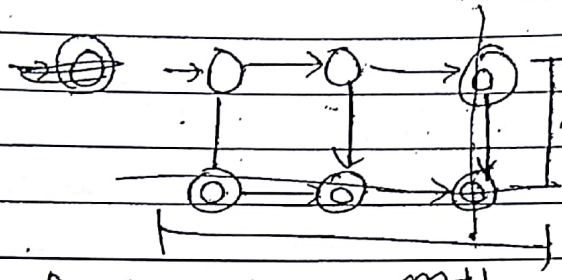
Min. DFA = (m+1)(n+1) + 1
(for atleast) (for trap (as exactly is there))

for shortcut — atleast m a → m+1 state
exactly n b → n+1 state
1 trap

EXCELLENT

OR

atleast 2a & atleast `1b'



no trap & L shaped

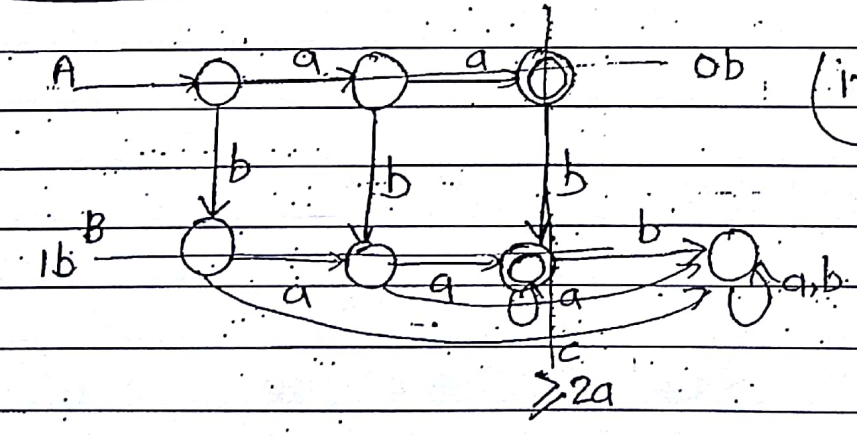
No of final states -

$m+1+n+1-1$

↓ common states

$= m+n+1$

* atleast 2'a' & atmost 1'b'' OR Not possible



= final state =
(intersection of A, B, C)

EXCELLENT

AND machine has less finite state in comparison to OR

Page No.
Date: / /

Product automata - AND operation can be done.

~~If then~~

Mode Machine - $\Sigma = \{a, b\}$

14. Even no "a" $\quad b^*(ab^*a)^*b^*$

$$r.e = (b^*ab^*ab^*)^* + b^*$$

15. odd no "a" $\quad r.e = [(b^*ab^*ab^*)^* + b^*]ab^*$

attachment of 1 a with to even no of a

Even no "a" $\Rightarrow L = \{\omega \mid n_a(\omega) \bmod 2 == 0\}$

odd no "a" $\Rightarrow L = \{\omega \mid n_a(\omega) \bmod 2 = 1\}$

even no "a" $\Rightarrow L = \{\omega \mid n_a(\omega) \bmod 2 = 2x,\ x \in Z^+\}$

for no of a's to be multiple of 3

$$L = \{\omega \mid n_a(\omega) \bmod 3 == 0\}$$

$$r.e. = (b^*ab^*ab^*ab^*)^* + b^*$$

for "not multiple of 3"

$3x+1$

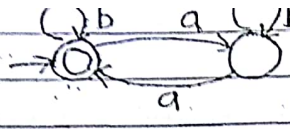
$$r.e = [(b^*ab^*ab^*ab^*)^*a + b^*]ab^*$$

for $3x+2$

$$r.e. = [(b^*ab^*ab^*ab^*)^* + b^*]ab^*ab^*$$

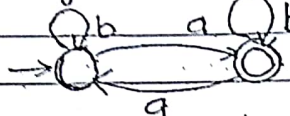
$$= ba^*((b^*ab^*ab^*ab^*)^* + b^*))ab^*$$

EXCELLENT



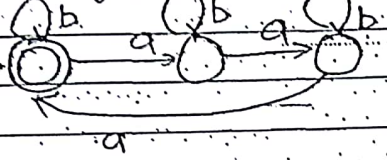
No of state = 2

15. odd no of 'a'



No of state = 2

Multible of 3



No of state = 3

{No of state in mod m = m+1}

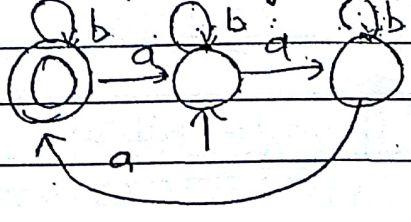
for 3x+1

i.e. {$n_a \bmod 3 = 1$}

No of state = 3

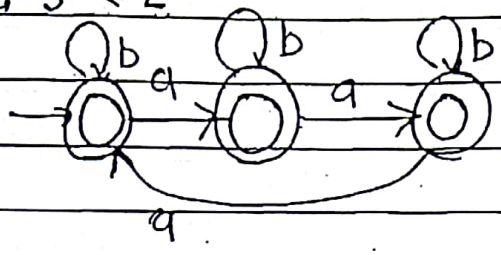
{No of states in residue of Mod m = m}

Ex

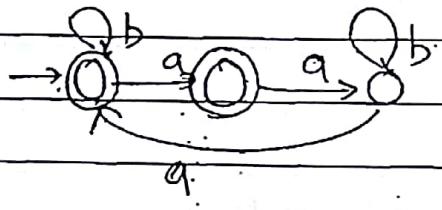


$\Rightarrow$ {$n_a(w) \bmod 3 = 2$}

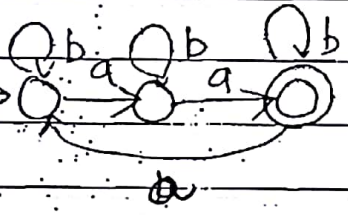
* $n_a \bmod 3 \leq 2$



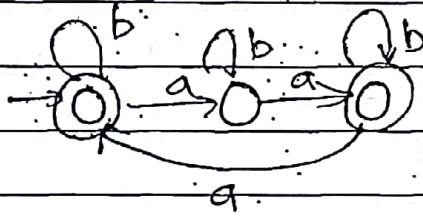
- $n_a \bmod 3 \geq 1$



- $n_a \bmod 3 > 1$



- $n_a \bmod 3 \neq 1$



- $\{n_a \bmod 3 < 1 = n_a \bmod 3 = 0\}$

- R.e. for $n_a \bmod 4 \leq 1$

$$re = [(b^*ab^*ab^*ab^*ab^*)^* + b^*] + [(b^*ab^*ab^*ab^*ab^*) + b^*]ab^*$$
$$= [(b^*ab^*ab^*ab^*ab^*) + b^*)][\epsilon + ab^*]$$

↳ 2 residue

if $\epsilon + ab^* + ab^*ab^*$

↳ 3 residue

$n_a \bmod 4 \geq 2$

i.e $= [(b^*ab^*ab^*ab^*ab^*)^*+b^*)] | [ab^*ab^*+ab^*ab^*ab^*]$

ba^*b

- [No of final states is controlled by no of residue in the required o/p]

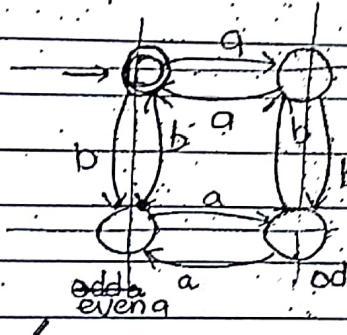
Mod Grid Machine-

16. Even no of a's & even no of b

Here if you have mod m & mod n on a & b

No of states, Grid size = m x n

$L = \{ w \mid n_a \bmod 2 == 0 \;\&\&\; n_b(w) \bmod 2 == 0 \}$



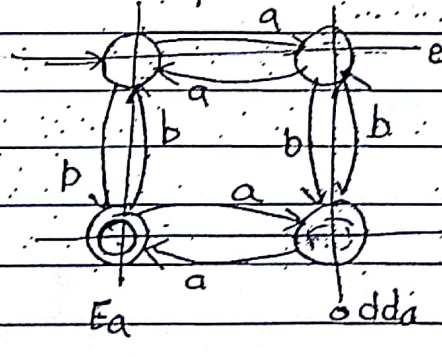
i.e

{ If No final state in a given DFA its language is empty }

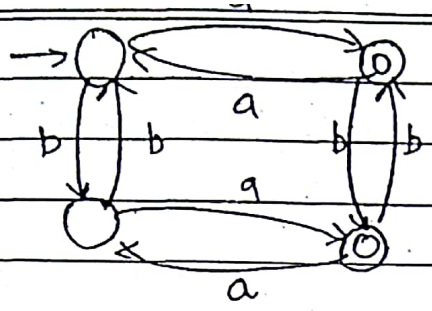
odd b-

17 Even a & odd b

$L = \{ w \mid n_a \bmod 2 == 0 \;\&\&\; n_b \bmod 2 == 1 \}$

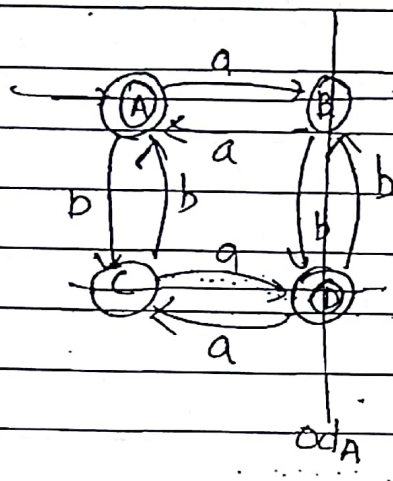


EXCELLENT



→ odd a acceptance
but it wont be minimal

Ex-



$\Rightarrow$ (even a & even b) or (odd a & odd B)

or odd a or even b but not both

$= O_a \oplus E_b$

$= E_b + O_a - O_a E_b$

or even a or odd b but not both

$(A + D - C)$

$$\text{or} = L = \{\omega \mid \underset{O}{n_a(\omega)} + \underset{O}{n_b(\omega)} = \underset{E}{\text{even}}\}$$

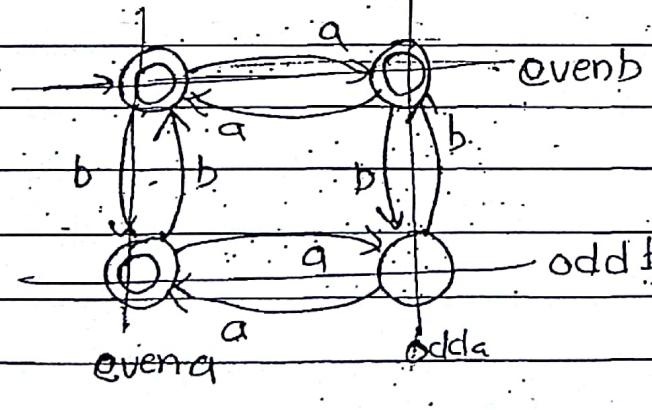
if B & C are final state, then

$$L = \{\omega \mid n_a(\omega) + n_b(\omega) = \text{odd}\}$$

odd Even

Even odd

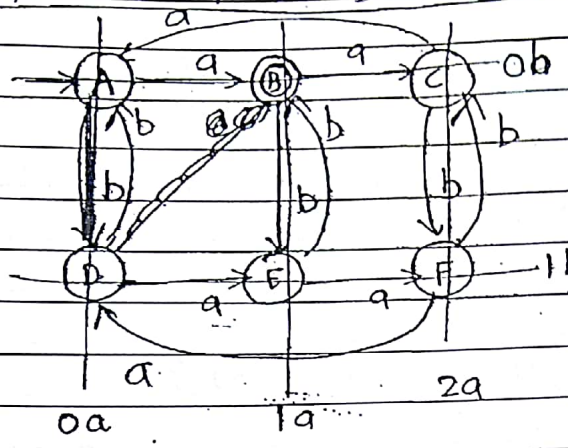
18- Even a's or Even b's



• Do n or U acc to final state pos

Page No.
Date: / /

19. $n_a \bmod 3 = 1$ & $n_b \bmod 2 = 0$



- Make a grid
- Choose for final state (by intersection of plane)

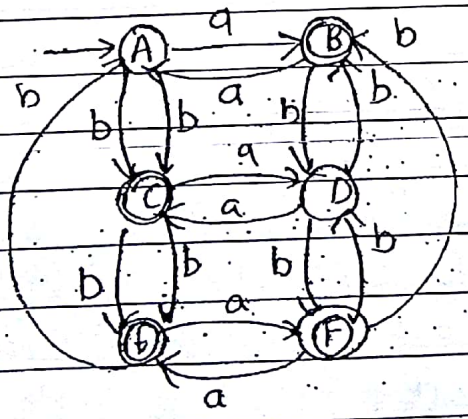
{No of states = m×n}

If B E DF are final state

~~$n_b \bmod 2 = 1$~~ or $n_a \bmod 3 = 1$

20. $\{w \mid n_a \bmod 2 \neq n_b \bmod 3\}$

- Do Grid view
- find the residue for each state



a b (residue combination)

~~Every~~ A - 0 0

B = 1 0

C = 0, 1

D = 1, 0

E = 0, 2

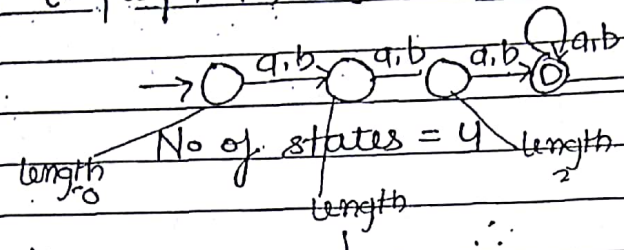
F = 1, 2

satisfies

for $\leq$ = A, C, D, E, F will be the final cond^n

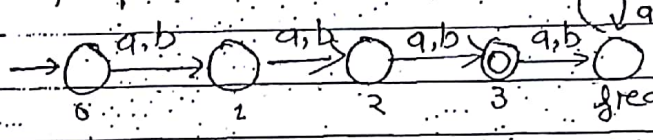
EXCELLENT

21. $L = \{w \mid |w| \geq 3\}$ Re - (a+b)(a+b)



{ for n length machine, No of states = n+1 }

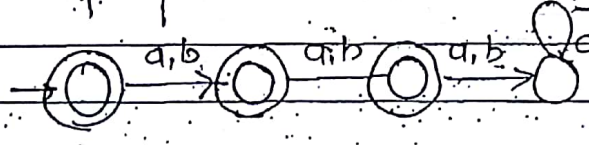
22. $L = \{w \mid |w| = 3\}$ RE - (a+b)(a+b)(a+b)



No of states = 5

{ No of states in DFA = n+2
No of states in NFA = n+1 }

23. $L = \{w \mid |w| \leq 2\}$ RE - $\epsilon + (a+b) + (a+b)(a+b)$



No of state in NFA = n+1
DFA = n+2

for $|w| \leq 2$

$RF = \epsilon + (a+b) + (a+b)^2 + (a+b)^3$

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

$= \cancel{(\epsilon + (a+b)(\epsilon + (a+b))^2)}$

$= \cancel{(\epsilon + a + b)}$

$= (\epsilon + a + b)^3$

(atmost 3 bit)

EXCELLENT

If $(\varepsilon+a+b)^{100}$ – atmost 100 bits

as $\underbrace{\quad\quad\quad\quad\quad\quad}_{\varepsilon ab\ \varepsilon a,b}$ ... $\underbrace{\quad\quad\quad\quad\quad\quad}_{100}$

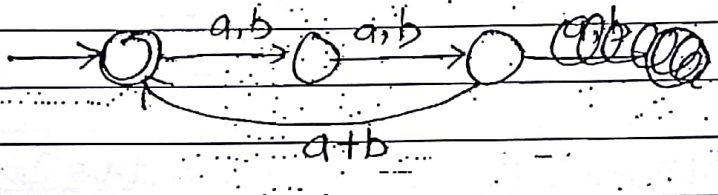
If $(\varepsilon+a+b)^{100}(a+b)^*$ – atleast 100 bits

If ~~$(\varepsilon+$~~ $(a+b)^{100}$ – exactly 100 bits

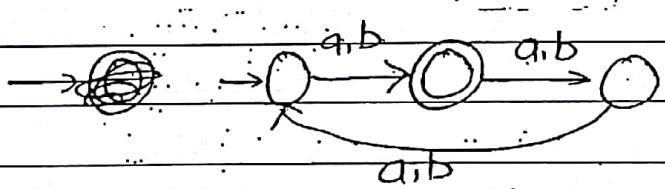
Q. $L=\{w \mid |w| \bmod 3 = 0\}$

$RE = \underbrace{(a+b)(a+b)(a+b)}^{*}$ ie $3x$

↳ as it must have 3 length



for 1 as residue $3x+1$



RE:- $\underbrace{(a+b)(a+b)(a+b)}^{*}(a+b)$

for $3x+2$ $(a+b)(\underbrace{(a+b)(a+b)(a+b)}^{*}(a+b)$

Ex:- $(aa+bb)^*$

I- Every string created is of even length. ✓

II- Every string is even length string ✓

~~$(a+$~~

III- It will generate all the string of even length

$(aa+ab+ba+bb)^*$ ↳ false

it will not create

EXCELLENT

Page No.
Date : / /

Ex - $((0+1)^{100})^* (\varepsilon+0+1)^{23}$

$L = \{w \mid w \bmod 100 \leq 23\}$

If $\left[((0+1)^{100})^*(0+1)^{23}\right] \Rightarrow L = \{w \mid w \bmod 100 = 23\}$

as for $L = \{w \mid w \bmod 100 \geq 23\}$

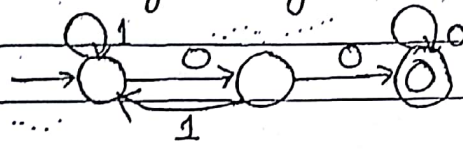
$(0+100)\left((0+1)^{100}\right)^* \left((0+1)^{23} + (0+1)^{24} + \cdots + (0+1)^{99}\right)$

$((0+1)^{100})^* (0+1)^{23} (\varepsilon + (0+1) + (0+1)^2 + \cdots (0+1)^{76})$

$((0+1)^{100})^* (0+1)^{23} (\varepsilon+0+1)^{76}$

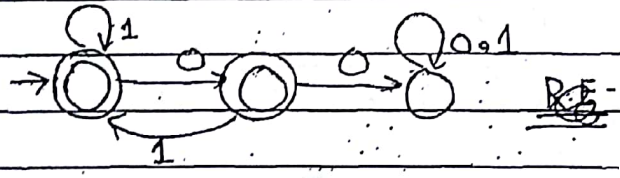
$\Sigma = \{0,1\}$

25) All the binary string with no two consecutive zeros.



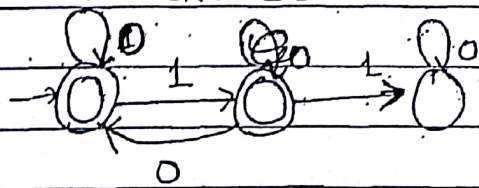
(for with consecutive zeros)

Now complement it



RE -

26) No two consecutive 1's



EXCELLENT

* No two consecutive zero-

1. take 0 blocked with 1..

~~(01+1)*~~ (01+1)*

~~(10+01+1~~ and then put 1 zero

RE = (01+1)* (ε+0) = ~~(10+1)* (ε+0)~~

= (ε+0)(10+1)*

(ε+0) is placed so that string can end with zero as well

* No two consecutive one.

1. block 1 with zero

RE = (0+10)*(ε+1)

It can also be written ~~(0+01)~~ as

(ε+1)(0+01)* // string can start with 1 as well

(01+1)* - Not two consecutive zero but string ending with

(0+01)* (0+01)* → No two consecutive zero and string with & with no null

~~(0+101)~~ = ~~Every 1 0 is~~

(0+10)* = every 1 is followed by atleast 1 0.

Q6. No three consecutive '000's

RE= (1+01)*(ε+0) (for no two consecutive zero)

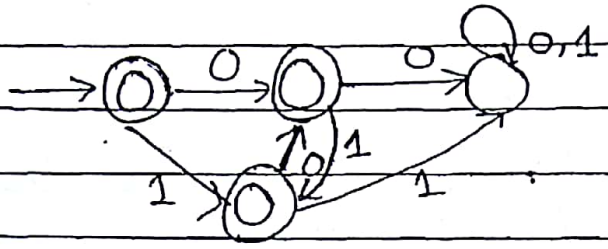
~~(1+00~~

RE = (1 + 0(1+01))*(ε+0+00)

or

RE = (ε+0+00)(1+(1+01)0)* (~~ε~~

$(\epsilon+0)(10+101)^*(\epsilon+0)$



RE $(\epsilon+1)(010+0$

RE $(\epsilon+0)(101)^*(\epsilon+0)$

RE $= (\epsilon+0)(101)^*(\epsilon+0)$

RE $= (\epsilon+1)(010)^*(\epsilon+1)$

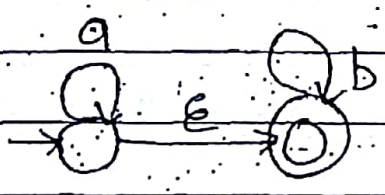
$\epsilon+0)$ is added as $(101)^*$ will only generate the string starting & as well as ending with 1.

Design for Regular expression-

Eg - If a Regular expression is given directly design NFA it is easy to ~~convert~~ design NFA by given RE.

Eg. a^*b^*

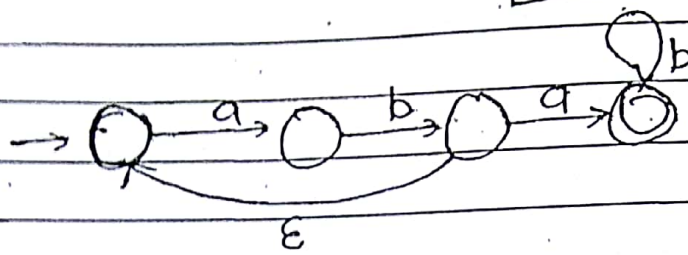
NFA is:



Minimum state = 2

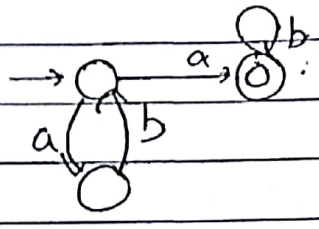
3o) $(ab)^*ab^*$

(•) when it came go on serial ie change of state



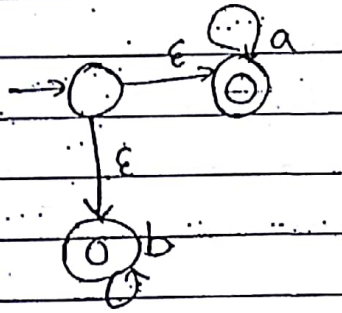
+ - parallel branch
• concatenated

its minimal NFA is—



3) $a^* + b^*$

it (+) just do parallel branching use ε move to go on...



for Regular to DFA

1. convert into NFA
2. ~~convert~~ if already DFA, occurs

else if it is not DFA, 3 reason

1. choice are multiple
2. DC
3. Null Move.

2. if DC only, but that move into trap directly no need of algorithm.

EXCELLENT

• If Null Move exist, algorithm is applied but it take time so use shortcut

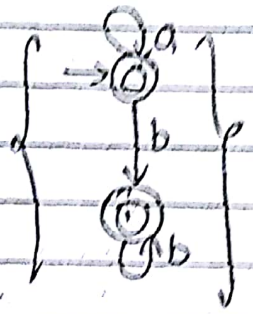
as ex-

If a^*b^*

$\Downarrow$

$b^* = (\varepsilon + bb^*)$

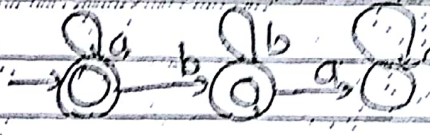
$\therefore a^*(\varepsilon + bb^*) = a^* + a^*bb^*$



Theorem :

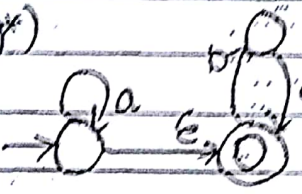
1. $x^* = (\varepsilon + xx^*)$

Now do DC Moves



Ex 2, $(a^*(ba)^*)$

NFA -

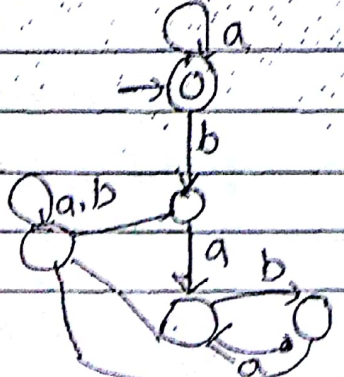


use like $x^* = \varepsilon + xx^*$

$(ba)^* = \varepsilon + (ba)(ba)^*$

$\therefore (a^*(ba)^*) = a^*(\varepsilon + (ba)(ba)^*)$

$= a^* + a^*(ba)(ba)^*$



EXCELLENT

• If choice comes, you

[It basically arise when setting is done in right side so if possible, avoid them.]

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

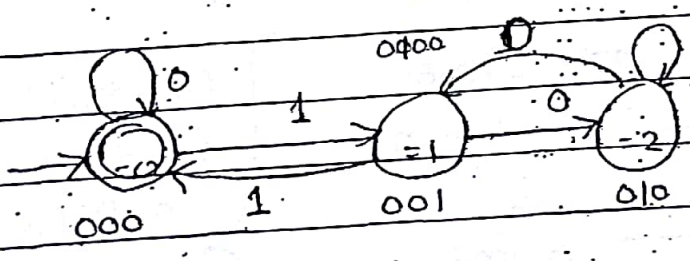
↳ You cannot remove directly the choice problem.

33 - $\{w \mid d(w) \bmod 3 = 0\}$ ie binary string of a decimal value must be multiple of ~~there~~ three.

• Mod is always regular.

• Since Mod 3, just you will have 3 states.

these three state has 0, 1, 2 as O/P (i.e residue)

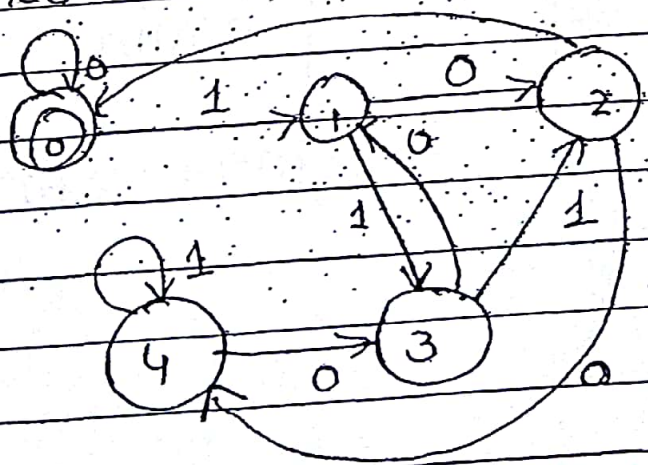


• if only 0000 comes must give 0 only

• when 1st 1 come it go to 1 o/p stat

• when 0 comes afte 1 it became 2 and so on.

for mod 5



EXCELLENT

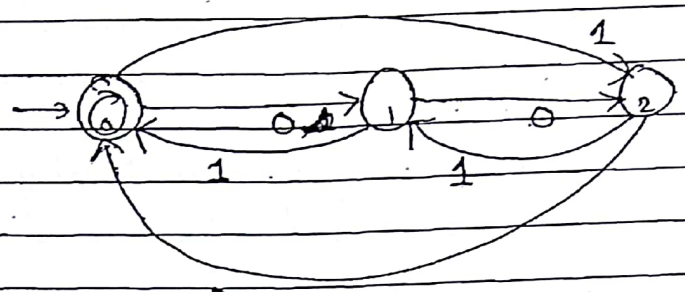
-1 mod 3

Page No.
Date : / /

a If you want to do $n_0(\omega) - n_1(\omega) = 2$ not a regular language

but

33 $L = \{ \omega \mid n_0(\omega) - n_1(\omega) \bmod 3 = 0 \}$



if only 0 comes
= 1
if only 1 comes
-1 mod 3
= 2
as 7 mod 3
7-6 = 1
-1 mod 3
-1 -(-3) = 2

just took some bits & proceed.

for mod n machine of any type,
only n state will exist.

Language with more than one minimal string

34.	First two symbols are same	$(00+11)(0+1)^*$
35	first ——— different	$(10+01)(0+1)^*$
36.	last ——— different	$(0+1)^*(10+01)$
37.	last ——— same	$(0+1)^*(11+00)$
38.	first ~~two~~ symbol are same & last ~~two~~ are also same.	$0(0+1)^*0 + 1(0+1)^*1 + 1 + 0$
39.	first & last symbol are different	$0(0+1)^*1 + 1(0+1)^*0$

Min state in NFA = 4
——— DFA = 5

EXCELLENT

• if choice comes, you can not directly remove them.

[it basically arise when setting is done in right side so if possible, avoid them.]

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

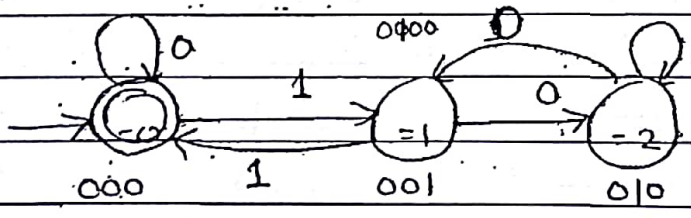
↳ You can not remove directly the choice problem.

33 - $\{w \mid d(w) \bmod 3 = 0\}$ ie binary string of a decimal value must be multiple of ~~there~~ three.

• Mod is always regular.

• since Mod 3, just you will have 3 states.

these three state has 0, 1, 2 as O/P (i.e residue)

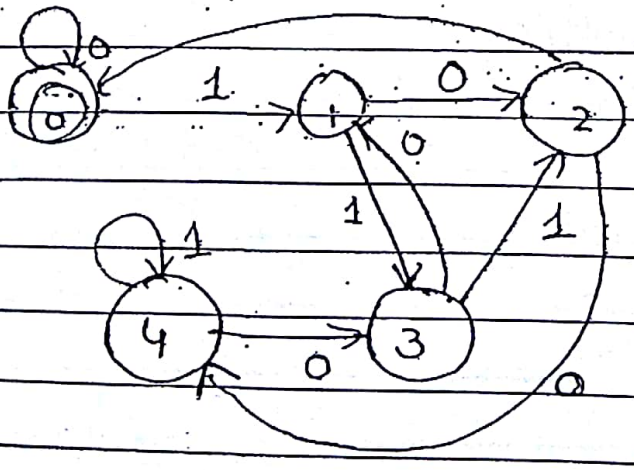


• if only 0000 comes must give 0 only

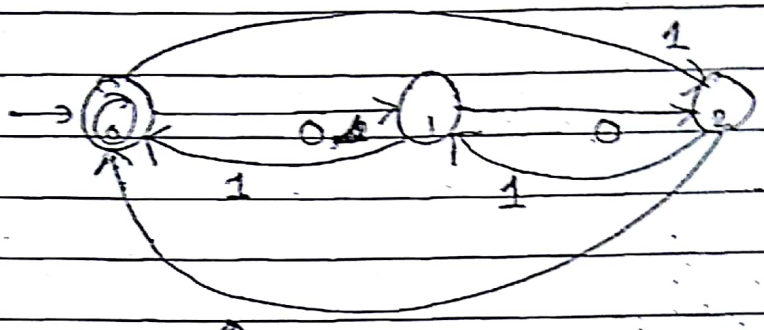
• when 1st 1 come it go to 1 o/p stat

• when 0 comes afte 1 it became 2 and so on.

for mod 5



33 $L = \{ w \mid n_0(w) - n_1(w) \bmod 3 = 0 \}$



just took some bits & proceed.

if only 0 comes
= 1
if only 1 comes
-1 mod 3
= 2
as 7 mod 3
7-6 = 1
-1 mod 3
-1-(-3) = 2

for mod n machine of any type,
only n state will exist.

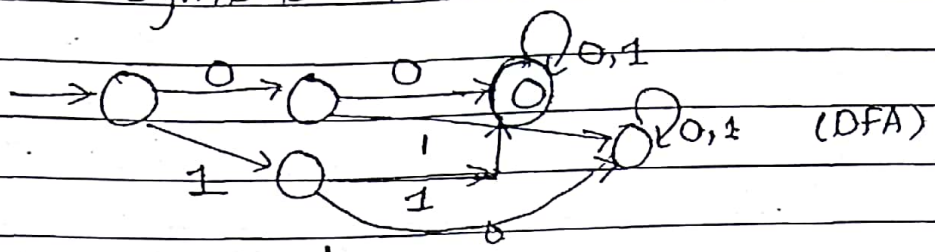
language with more than one minimal string:

34. First two symbols are same — $(00+11)(0+1)^*$
35. first — different — $(10+01)(0+1)^*$
36. last — different — $(0+1)^*(10+01)$
37. last — same — $(0+1)^*(11+00)$
38. first ~~two~~ symbol are same & last ~~two~~ are also same — $0(0+1)^*0 + 1(0+1)^*1 + 1 + 0$
39. first & last symbol are different — $0(0+1)^*1 + 1(0+1)^*0$

Min state in NFA = 4
DFA = 5

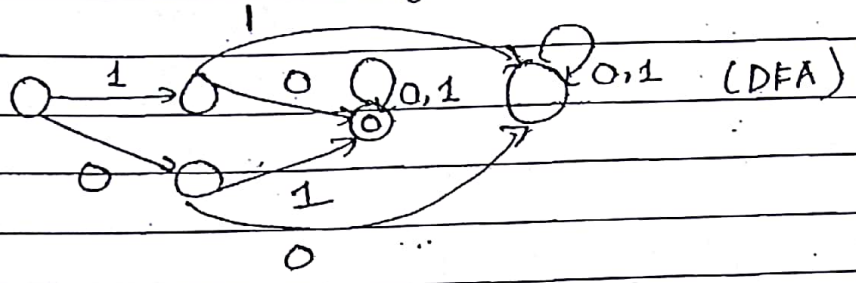
EXCELLENT

34. first two symbols are same



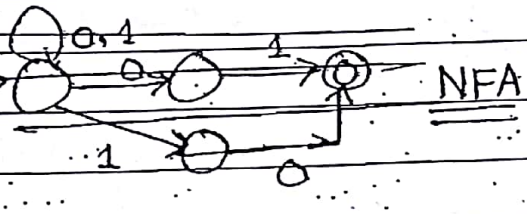
(DFA)

35 -



(DFA)

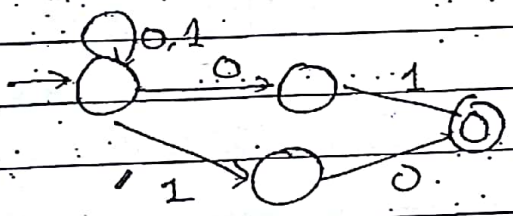
36.



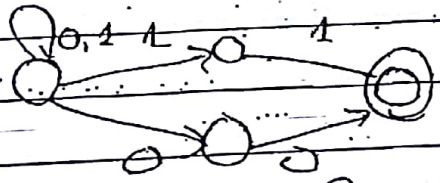
NFA

~~When~~

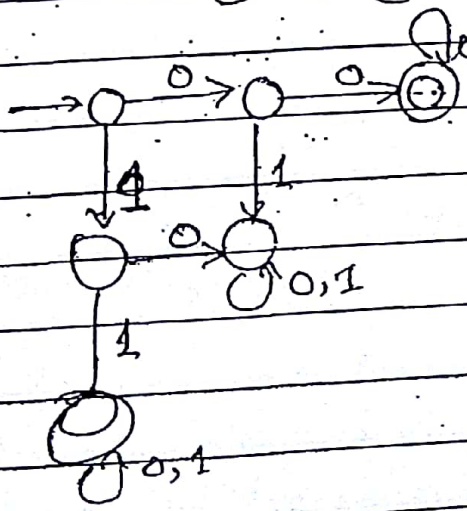
36.



37.



for 34-



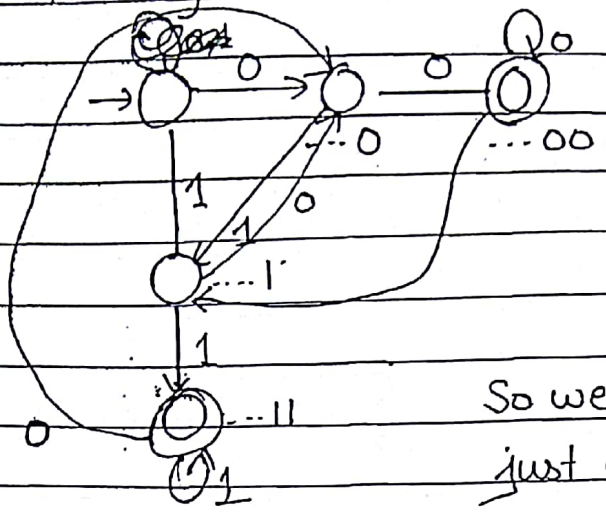
⇒ then Merge bcoz both have same behave.

(Whenever you are doing for two minimal string, in starting itself donot merge the final state, If those are permanent accept, then merge
- otherwise leave them i.e it is done so that no mistake can ever happen.

procedure)

EXCELLENT

3) last two symbol are same



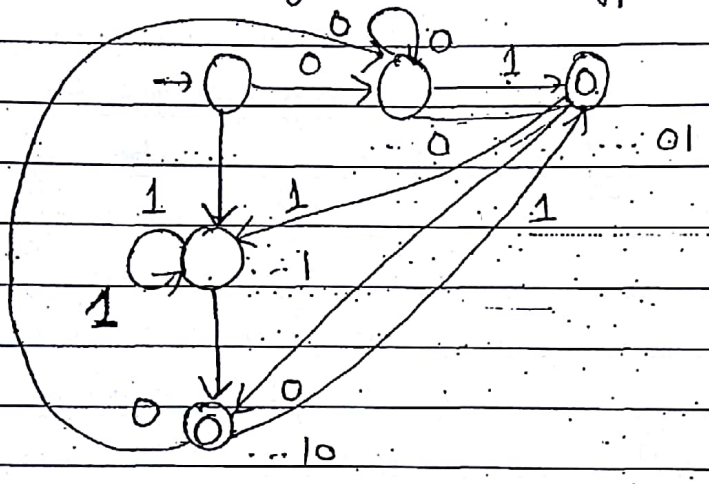
(00+11)(0

(0+1)*(00+11)

(0+1)*- ε +(0+1)(0+1)*

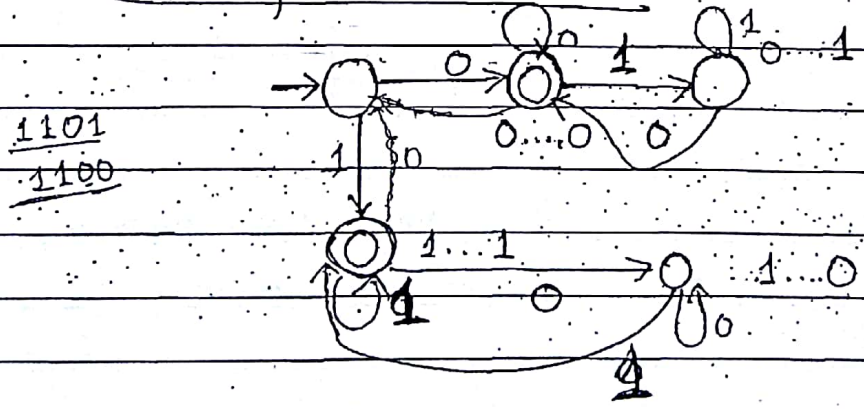
So we use the arrow filling ie just go with arrows.

38) Last two symbol are diff



- Initially Draw for minimal states - note down underneath it
- then fill up the arrows
- if you note down under neath no chances of mistake

38 - last & first are same-



EXCELLENT

Regular Expressions-

- Arideen's Theorem - for conversion of machine to RE.

1. Conversion of Machine to RE.
2. Equivalance of two RE & properties of RE

Ex - $L_1 = \{\epsilon\}$ $L_2 = \{a\}$, $L_3 = \phi$

$L_1 \cup L_2 \cdot L_3^*$

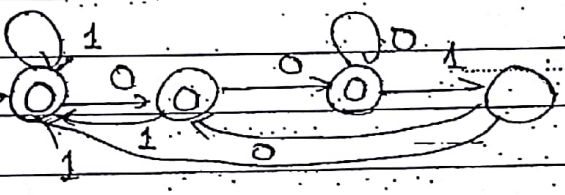
$\epsilon + a \cdot \phi^*$ $\quad \{\phi^* = \epsilon\}$

$\epsilon + a \cdot \epsilon$

$= \{(\epsilon + a)\}$

3. Arideen's Theorem

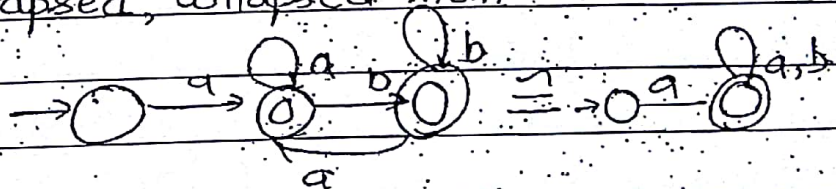
Ques ....



not end with 001

1. Machine to RE -

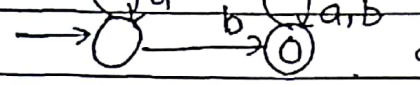
1. You should not bother whether it is Nfa or Dfa.
2. Before you convert it, simplify the machine.
   1. Remove any trap state
   2. Remove any non-reachable state
   3. In case of multiple final state, if they can be collapsed, collapsed them.



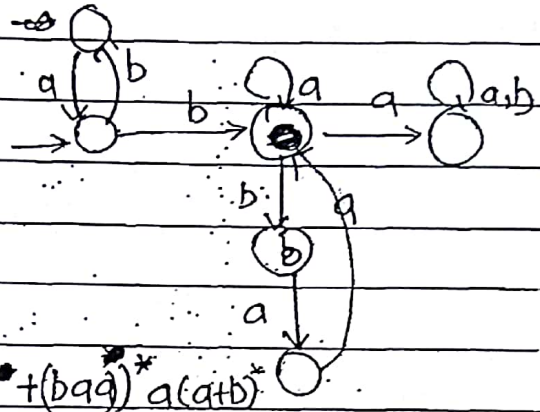
4. Remove all states which do not serves anything for final state.

3. If you have final state which can not be collapsed, write re for each final state & add them

EXCELLENT

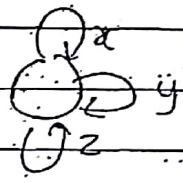


$a^*b(a+b)^*$

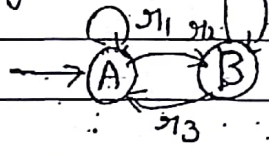


$(ab)^* b(a^* + (baa)^* a(a+b)^*$

$= (ab)^* b(a + bba)^* a(a+b)^*$

7. When you have  ij $\Rightarrow (x+y+z)^*$

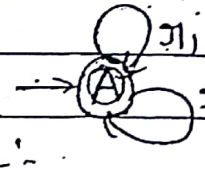
8. Try to reduce the machine to 2 state machine & can directly solve it.



r_1, r_2, r_3, r_4 = any expression

Let A be the final state.

$r.e. = r_A$ reduce to a single state



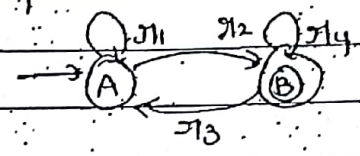
$$r_e = (r_1 + r_2 r_4^* r_3)^*$$

EXCELLENT

To simplify the machine-

1. In beginning, try to remove not initial & final statement or those where min actions are done.

if B is the final state



It will have two soln has it has loop. You can generate answer for left arrow & for right arrow known as left & right ~~resolution~~.

~~B~~ ~~$r_B = r_4^* r_1^* r_2 r_4^* r_3 (r_1^* r_2 r_4^*)$~~

~~kno~~

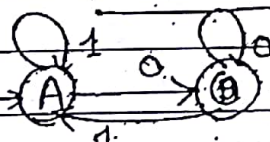
LR ; $r_B = (r_1 + r_2 r_4^* r_3)^* r_2 r_4^*$

RR ; $r_B = r_1^* r_2 (r_3 r_1^* r_2 + r_4^*)$

from A to B & B to A : A to B : A to B : B to A

in left, initially solve the left part by A when can have r_1 or $r_2 r_4^* r_3$

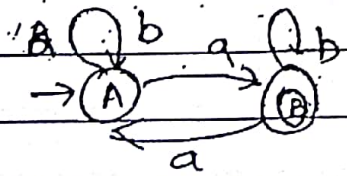
for



LR = $(1^* + 00^*1)^* 00^*$

RR = ~~$(00^*1)^*$~~ $1^*0(0 + 11^*0)^*$

for

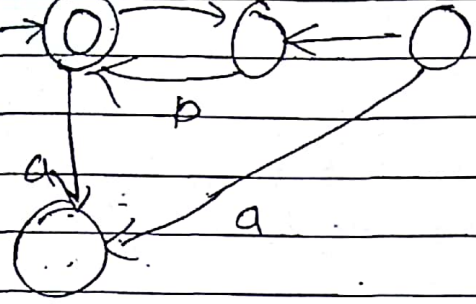


~~A~~ LR = $(b + baab^*a)ab^*$

RR = $b^*a(b + ab^*a)$

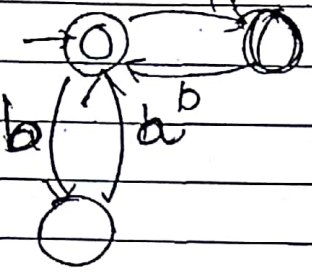
EXCELLENT

Ques1



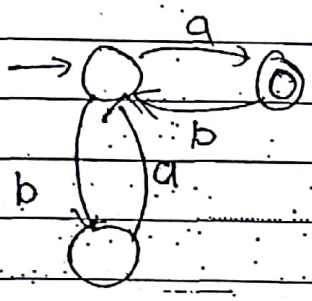
R.E = $(ab)^*$

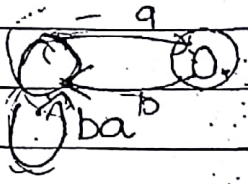
Ques2



R.E = $(abb^* + ba)^*$

Ques3

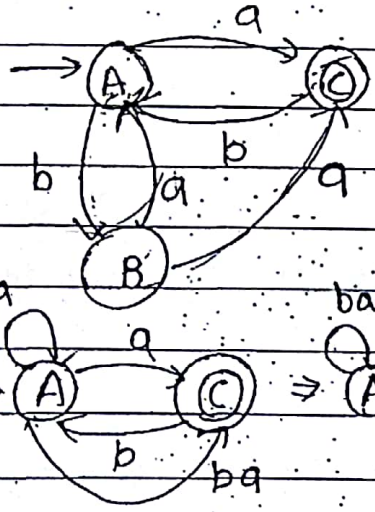


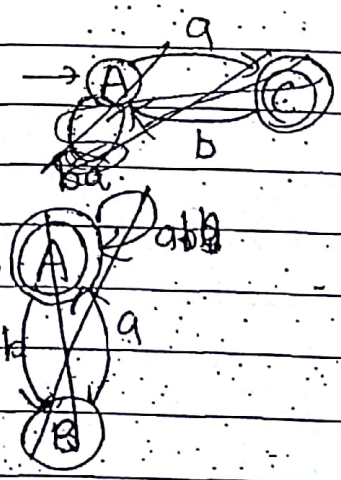


LR = $(ba + (ab)^*a)^*$

RR $(ba)^*a(b(ba)^*a)^*$

Ques4





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

RR = $(ba)^*(a+ba)(b(ba)^*(a+ba))^*$

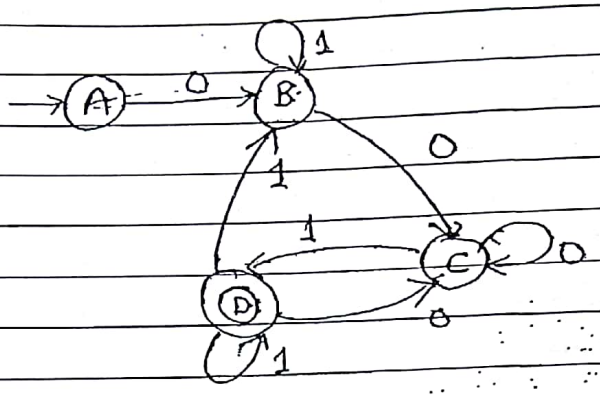
EXCELLENT

RR = While followed by B

Page No. 48
Date : / /

LR means B in terms of A

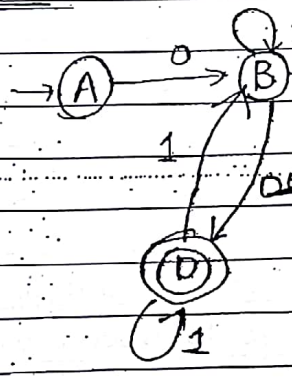
Ques -



~~(1+0~~

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

Delete C



~~$0^*(10)$~~ $0(0+11^*0)^*1 = \varepsilon$ (let)

D

$LR - 0[(1+\varepsilon 1^*1)^* \varepsilon 1^*]$

$RR - 0[\varepsilon 1^* \varepsilon (1+11^*\varepsilon)^*]$

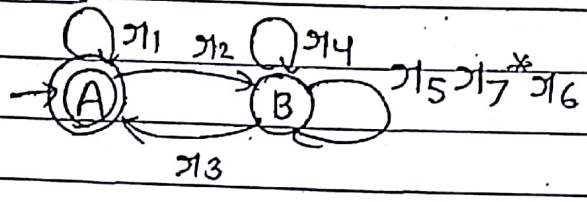
$LR = 0[(1+(0(0+11^*0)^*1^*1))^* + (0(0+11^*0)^*11^*]$

So many soln are possible

EXCELLENT

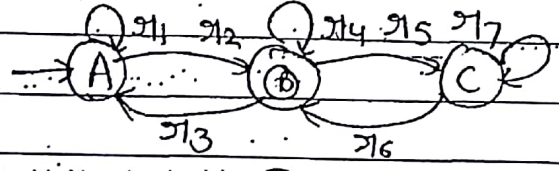
Let A is the final state

1. Delete C

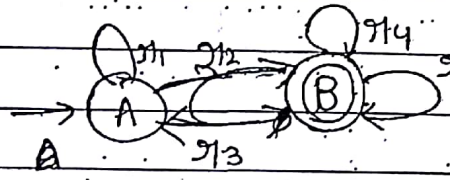


$$r_A = \left(r_1 + r_2 (r_4 + r_5 r_7^* r_6)^* r_3 \right)^*$$

Let B is the final state.



For best answer resolve the loop at B only



$x = r_4 + r_5 r_7^* r_6$

A

~~LR = $(r_1 + r_2 (r_4 + r$~~

$LR = (r_1 + r_2 x^* r_3)^* r_2 x^*$ (A get resolved.

$RR = r_1^* r_2 (x + r_3 r_1^* r_2)^*$ (when resolve is done at B only)

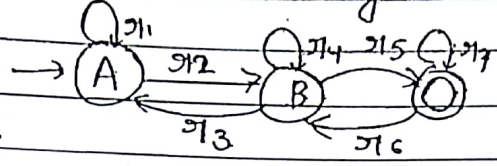
(Here RR is better)

∴ final answer = $r_1^* r_2 ((r_4 + r_5 r_7^* r_6) + r_3 r_1^* r_2)^*$

(You can also get it by directly taking loop only)

EXCELLENT

so use best answer of B & then followed by C.



$r_C = (RR \text{ of } B)\, r_5\, r_7^*$

Properties of Regular Expression-

If r & s are two regular expression on given alphabet Σ,

1. Commutative - $r + s = r + s = s + r$

$r \cdot s \neq s \cdot r$

Ex $a + b = b + a$

$a \cdot b \neq b \cdot a$

2. Associativity - $r + (s + t) = (r + s) + t$

$r \cdot (s \cdot t) = (r \cdot s) \cdot t$

Ex - I. $a + (b^*c^* + d^*)$

II. $d^* + (a + c^*b^*)$ I, III are equivalent.

III. $d^*(a + b^*c^*)$

3. Distributive - $r \cdot (s + t) = r \cdot s + r \cdot t$

$r + (st) \neq r + s \cdot r + t$ $(r + s) \cdot (r + t)$

1. $a + \varepsilon = \{a, \varepsilon\}$
2. $a + \phi = a$

Page No.
Date : / /

$(s + t)r = sr + tr$

Ex - I $(ab^*) + c^*$
II $(a + b^*)(b^* + c^*)$ → Not equivalent

4. Identity - 1. $r + \phi = \phi + r = r$ 3. $r \cdot \phi = \phi$
2. $r \cdot \varepsilon = \varepsilon \cdot r \neq \varepsilon$

Ex - $a^* = a^* \cdot \varepsilon + \phi = a^* + \phi$

5. properties of $*$, - 1) $\varepsilon^* = \varepsilon$
2) $\varepsilon^+ = \varepsilon$
3) $\phi^* = \varepsilon$
4) $\phi^+ = \phi$

1. $\varepsilon^* = \varepsilon^+$
2. $\phi^* \neq \phi^+$

5. $(r^*)^* = r^*$

6) $r^* \cdot r^* = r^*$
7) $r^* + r^* = r^*$
8) $r^* r^+ = r^+ = r r^*$
9) $(r^*)^+ = r^*$
10) $(r^+)^* = r^*$

(6) Union & $r + r = r$
Concatenation - $r \cdot r = r^2$

7. $(r + s)^* = (r^* + s^*)^*$

EXCELLENT

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

↓ as

$\{\epsilon + b + b^2 \ldots\}$

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

$= (a+b)$

Go

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

$(b, ab, \cancel{bab}, \ldots + b\epsilon + bb)^*$

↓ No we not getting $(a+b)^*$

$(r+s)^* = (r^* + s^*)^* = (r^*s^*)^*$

↓ because we are getting r & s

to check $(\_\_)^*$ is $(a+b)^*$ check whether you are getting r & s & a & b separately or not.

1) $(a^*b^*)^* = (b^*a^*)^*$

2) $\epsilon + r(r)^* = r^*$

3) if r, s, t are r.e.,

12) $p(q^*p)^* = (pq^*)^*p$

~~$p(q^*p)^*$~~ 14) $p(qp)^* = (pq)^*p$

Proof → (A) —p→ (B), (B) —q→ (A)

L.R = $(pq)^*p$ } equal

R.R $p(qp)^*$

Manual: ~~$p(pq)^n = p(pq)^n$~~

$p(qp)^n = (pq)^n p$

$pqpqpqp = pqpqpqp$

$n=3$

$(pq)^3 p = p(qp)^3$

EXCELLENT

✓

I) $(ab)^*(c^*(ab)^*)^*$
II $(ab)c^*)^*(ab)^*$ } not equal

15. $(r+s)^*r^* = (r+s)^*$
$(r+s)^*s^* = (r+s)^*$

$(r+s)^*(\epsilon + r + rr + \dots) = (r+s)^*$ Hence proved

Ex - $(a+b)^*b^*a = (a+b)^*a$

$(a+b)^*b^*a^*(ab)^*ab^* = (a+b)^*ab^*$ ....

Arcdeen theorem - if $r = q + rp$ $r = r$ is dependent on r if there is cycle or loop.
then $r = qp^*$ } p cannot contain ϵ

if $X = Xz + XTC^*$
$X = YZ(TC^*)^*$

EXCELLENT

Date : / /

if eqn is given as $r = q + rp$ → Left Recursion

$\{r = qp^*\}$

but if equn is $r = q + pr$

$\{r = p^*q\}$

as if $r = q + rp$

$= q + qp^*p = q(\varepsilon + p^*p) = qp^*$

If

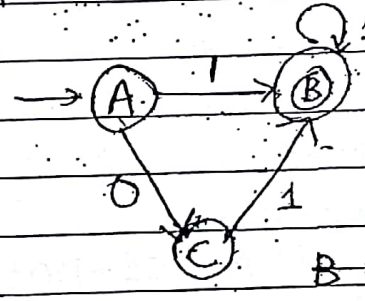
$r = q + pr$

$= q + pp^*q$

$= (\varepsilon + pp^*)q = p^*q$

~~but if r=~~

Ques -



~~01*~~ $(01+1)1^*$

~~B = 1+01 1+01+~~

for applying Arden theorem

- each state will store incoming array with state followed by input

$A = \varepsilon$

$B = A1 + B1 + C1$

$C = A0$

$B = \varepsilon \cdot 1 + B1 + C1$

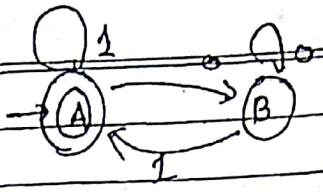
$C = 0$

$B = 1 + 01 + B1$

$B = (1+01)1^*$

EXCELLENT

Ques-



$(1+00^*1)^*$

$A = B1 + A1 + \epsilon$

$B = A0 + B0$

$B = A00^*$

$A = A00^*1 + A1 + \epsilon$

$A = \epsilon + (00^*1 + 1)$

$A = \epsilon(00^*1 + 1)^*$

.....

$B = (00^*1 + 1)^*00^*$

Ques - if the final state is D, L is L(D) if E is final state L is L(E)

(a) $L(D) = L(E)$

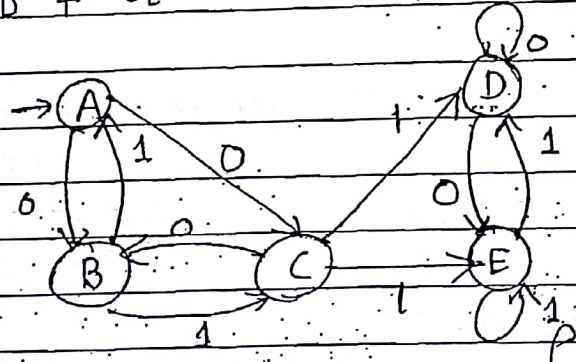
$L(D) \subseteq L(E)$

$L_E \subseteq L_D$

$L_D \neq L_E$

$D = C1 + D0 + E1$

$E = E1 + C\phi + D0$



$L_D = L_E$

You can find out the relation directly

EXCELLENT

Regular Grammar

G → L conversion of a Grammar to Language

for equivalent Grammar:

G_1: $S \to aS \mid \varepsilon$
G_2: $S \to aS \mid aaS \mid \varepsilon$ } equal

if $S \Rightarrow X \mid Y$
$S \to X \mid Y \mid Z$ } these are equal if Z is obtained from X, Y }

Solving a Grammar

To solve a Regular Grammar - two ways

substitution — conversion of M/C & solving it

to solve a CFG - substitution Method.

(You can solve it by M/C Method. but there, M/C are difficult to design) (by using PDA

Substitution is difficult if grammar has so much dependency

Ques - 1. for ϕ - $G = (V, T, P, S)$

$T = \{\ \}$

$V = \{S, A\}$

$P = \{S \to A\}$

2. for ε $S \to \varepsilon$

3. for a $S \to a$

4 for a+b $S \to a \mid b$

EXCELLENT

5 for aba + ba = $S \to aba | ba$

6. for a^* — three possible grammar

1. $S \to aS | \varepsilon$ ✓
2. $S \to Sa | \varepsilon$ ✓
3. $S \to a | SS | \varepsilon$

[only constant term in RHS of grammar - stopper]

7. for $(ab)^*$

$S \to ab | SS | \varepsilon$ ✓
$S \to abS | \varepsilon$
$S \to Sab | \varepsilon$

for a grammar to L^*

$S \to SS | \varepsilon | L$

* if you see $S \to ~~ab~~ | SS | \varepsilon$ just solve the left part & put * [related to *]

$S \to aa | bb | SS | \varepsilon \Rightarrow (aa+bb)^*$

for a^+ - ~~$S \to aS | \varepsilon$~~ $S \to aS | a$ $S \to SS | a$
~~$S \to SS | a$~~ $S \to Sa | a$

SS will create ambiguity in Grammar.

Grammar ambiguity is different from language ambiguity

if L(G) is ambigous - every grammar of L is ambiguous & it cannot be removed.

if L(G) is unambigous - a grammar of L exist which is unambigous.

EXCELLENT

D-CFL = D-PDA

A regular or DCFL language can never be ambigous.
∴ ambiguity in language is very harmful.

A RL is always unambigous bcoz for each RL, DFA exist & DFA cannot accept same string in two ways as no choice is allowed so multiple derivation tree cannot be generated.

A D-CFL is D PDA and here also, no choices allowed

CFG corresponds to NPDA

- for x is a string

x^* $S \to xS \mid x$
$S \to Sx \mid x$
$S \to x \mid SS$

$\{S \to aS \mid b = a^*b\}$

for $S \to xS \mid y$ $\{xe = x^*y\}$

for $(01)^*100$
$S \to 01S \mid 100$ (Right linear)

- for $100(01)^*$
$S \to S01 \mid 100$ Left linear

* $S \to Sx \mid y$ and $S \to xS \mid y$ are not equivalent.

$S \to Sx \mid y$ ⇓ yx^*

$S \to xS \mid y$ ⇓ x^*y

EXCELLENT

Page No.
Date : / /

- Every left linear Grammar has equivalent right linear grammar.

- if x^*y use LLG
  xy^* use RRG

- a^*+b^* → $S \to aS \mid \varepsilon$
  $S \to bS \mid \varepsilon$
  for +

$S_1 \to aS_1 \mid \varepsilon$
$S_2 \to bS_2 \mid \varepsilon$
start symbol → $S \to S_1 \mid S_2$
} substitution Method

- if $S_1 \Rightarrow aS \mid \varepsilon$
  $S_2 \Rightarrow aaS \mid \varepsilon$
  R.E $= a^* + (aa)^*$
  $= a^*$
  } If you can simplify the given expression just simplify it.

- a^*b^* ⋯
  $S \to S_1 S_2$
  $S_1 \to aS_1 \mid \varepsilon$
  $S_2 \to bS_2 \mid \varepsilon$

- When you do like $S \to S_1 S_2$ Grammar is not regular Grammar.
  (start)

Ques - RG_1 with S_1 produce L_1
RG_2 with S_2 ⟶ L_2

∴ New Grammar has
all production of G_1
⟶ G_2
$S \to S_1$
$S \to S_2$

Resulting Grammar is CFG as if $G_1 = LLG$ with $L_1 \cup L_2$ $G_2 = RRG$

EXCELLENT

if G_1 is Left linear & G_2 is right linear if they get mixed it will not be a regular Grammar.

if we have $G_1 = RRG$, $G_2 = RRG$, $S \to S_1 S_2$ } - CFL with RL and $L_1 \cdot L_2$

for concatenation- if Grammar is to be regular.

- for a^*b^* $S \to aS | b^* | \varepsilon$

  Here you cannot put the language ~~$S \to aS$~~ | so use variable of that language

  $S \to aS | bS | \varepsilon$ } then S_1 is known as Variable stopper
  $S_1 \to bS_1 | \varepsilon$

- $a^*b^*c^*$

  $S \to aS | A$
  $A \to bA | C$
  $C \to cC | \varepsilon$

- $(a+b)^*$ — $\{S \to aS | bS | \varepsilon\}$

  $S \to Sa | Sb | \varepsilon$
  $S \to a | b | SS | \varepsilon$

EXCELLENT

for end with a $S \to aS|bS|a$

for end with b $S \to aS|bS|b$

for start with a $S \to Sa|Sb|a$

$(x+y)^*$ $S \to xS|yS|\epsilon$

$S \to Sx|Sy|\epsilon$

$(x+y)^+$ ~~$S \to xy|\epsilon$~~ $S \to xS|yS|x|y$

$(x+y)^*z \Rightarrow$ $S \to xS|yS|z|\epsilon$

If $(x+y)^*(10)^*$ → $S \to xS|yS|A$

$A \to 10A|\epsilon$

If $S \to xS|Sy|\epsilon$ ⇒ ~~$(a+b)$~~ (a^*b^*)

$S \to 0S|S11|\epsilon$ ~~00~~ $((01)^*(11)^*)$

$S \to 0S|S11|011|\epsilon$ $((01)^*(11)^*)011$

$(01)^*011(11)^*$

Linearity of a Grammar-

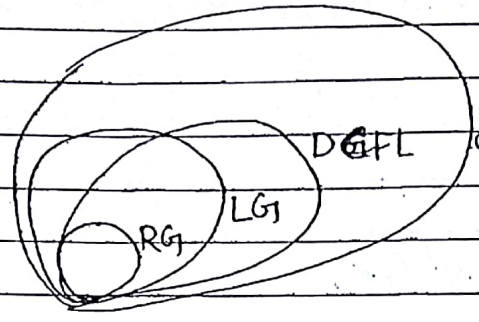
Left side must have single variable but allow ~~atmost~~ single variable is allow RHS at any pos^n

$V \to VT^* + T^*V + T^*VT^* + T^*$

EXCELLENT

Ex $S \to Aaa \mid aaB \mid aBa \mid aa \mid \varepsilon$ (Linear Grammar)

$S \to aBaA$ // Not a linear Grammar



17-NOV

Algorithm in FA-

1. NFA to equivalent DFA → Subset construction Algo // do not work for null Move
2. DFA to equivalent min. DFA
3. NFA with Null Moves to NFA without ε-moves.

} decidable

} NFA Minimization is decidable but it is NP problem }

ε-closure form transition system (so many initial state)

- NFA never has more than one initial state.

Subset Cons. Algo- (I/P - NFA without null Moves or any transition system)

NFA with Null Move → Transition system w/o ε-move
→ NFA w/o ε-move.

- DFA to NFA is also decidable as DFA is also a NFA.

EXCELLENT

1. Subset Construction Algo-

A. if you have n-state NFA, then its equivalent DFA has atmost 2^n states by using subset construction algo as

$Q = \{q_0, q_1, q_2\}$ $\quad 2^Q = \{\phi, \{q_1\}, \ldots \{q_1, q_0, q_1\}\}$

B. Let I/P NFA $= M_N(Q, \Sigma, \delta, q_0, F)$

&

equivalent o/P DFA $= M_D(Q', \Sigma', \delta', q_0', F)$

Here

1. $Q' \subseteq 2^Q$
2. $\Sigma' = \Sigma$
3. $q_0' = \{q_0\}$
   - as Now $Q' \in 2^Q$ as states are in bracket
4. $F' \subseteq Q' \subseteq 2^Q$
5. $F' \notin 2^F$ as it will not allow some new elmt of Q.
   $F' \neq F$

Example

	0	1
$\rightarrow q_0$	q_1, q_2	—
q_1	q_2	q_1
q_2	q_0	q_1

Transition table on NFA

If you have NFA with DC only, just put ~~DE~~trap.

Equivalent DFA is

M'	0	1
→ $\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\{q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_1\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_1\}$
ϕ	ϕ	ϕ
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_1\}$
$\{q_1, q_3\}$	$\{q_2\}$	$\{q_1\}$
$\{q_2\}$	$\{q_0\}$	$\{q_1\}$

- Start procedure from initial state
- go on with each state entering
- if ~~all~~ no new state came, just stop

$\{q_1, q_2\}$ - have behaviour of q_1, q_2.

ϕ is a valid state used as trap.

$\delta(\phi, a) = \phi$

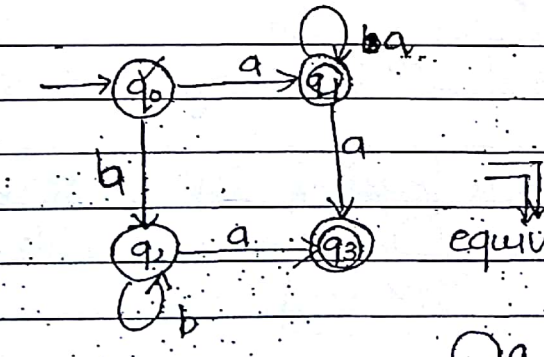
$\delta(\phi, 1) = \phi$

- Any state containing F then become final state

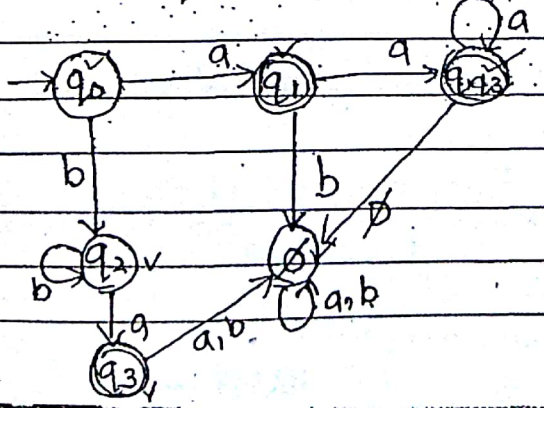
No of States in equivalent DFA = 7

No of final state ——— = 4

Ques



equivalent DFA is (directly convert it)



No of state = 6

F state = 3

EXCELLENT

{F = can be ∅}

Page No.
Date : / /

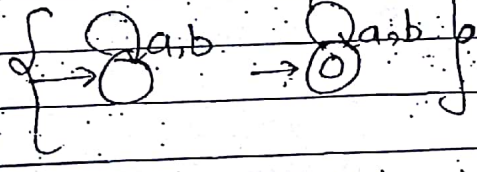
- If Input NFA has an DC without choice, it will be having trap state.
- If I/P NFA has DC state with choice, it may or may not have trap state

		a	b
Ex	→q_0	q_1, q_2	q_2
	q_1	q_2	-
	q_2	q_0	q_2

- but if I/P NFA has DC state with choice at initial state, it will have trap state.
- If a DFA has n state its equivalent NFA will have atmost n state.

Minimization of DFA -

- A DFA with n state, its min. DFA will have states as $1 \leq$ no of state $\leq n$
- O/P Min. DFA will n states if I/P DFA itself is minimal.
- for $\Sigma = \{a, b\}$ two DFA is possible with 1 state



- Working - it minimizes the machine by identifying equivalent states

ie, ∀ state $A \equiv B$

if $\forall w \in \Sigma^*$

$\delta^*(A, w)$ & $\delta^*(B, w)$ both same behaviour

will both accept w or reject w.

Here $\delta^*(A,w)$ & $\delta^*(B,w)$ one may or may not be equal

EXCELLENT

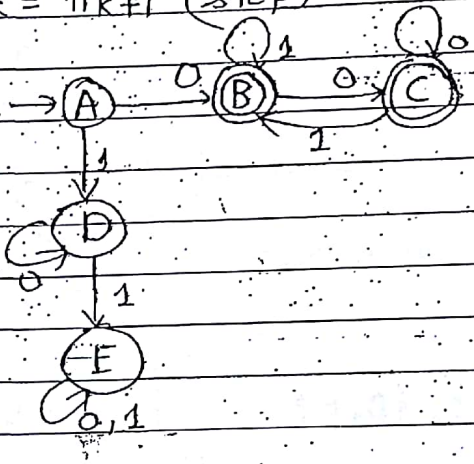
$\delta^*(A,\omega) \neq \delta^*(B,\omega)$ ⇒ it is not always required ~~for~~ both must reduce ~~same~~ produce same state.

We can not go with checking equivalence as there may be ~~an~~ endino a case where no end exist.

To check equivalence, we use equivalence class Π

Here if $\Pi_k = \Pi_{k+1}$ (stop)

Ques -



Π_0 → all states which are equivalent
i.e which shows same behaviour at ϵ strings

behaviour of state is Accept or Reject.

so Here B, C Accept Null (as both are final state)
A, D, E reject Null

$\Pi_0 = \{\{B,C\}\{A,D,E\}\}$

$\delta^*(B,\epsilon) = B$
$\delta^*(B,\epsilon) = C$
$\delta^*(A,\epsilon) = \phi$
$\delta^*(D,\epsilon) = \phi$
$\delta^*(E,\epsilon) = \phi$

B, C shows same behaviour and A, D, E show same behaviour
{B, C} are putted together & {A, D, E} are putted together.

If itself a min. DFA is given then it will partition into different state {A}, {B}, {C}, {D}, {E}

- this algo will take worst case time when ~~a~~ I/P itself is minimal DFA.

Page No.
Date: 15/ /

$\Pi_0 = \{\{B, C\}\ \{A, D, E\}\}$

if two states are not k equivalence, then they can never ~~have~~ be k+1 equal equivalence

B, A can never be equivalent

for

Π_1 = check for equivalence for B & C

check for 1-length sent string

if o/p of B & C belong to same block in Π_0 then they are Π_1 equivalent.

$B \xrightarrow{0} C$
$C \xrightarrow{0} C$ } same ~~block~~ block

$B \xrightarrow{1} B$
$C \xrightarrow{1} B$ } same block

$\Pi_1 = \{\{B, C\}, \{A\}, \{D, E\}\}$

$A \xrightarrow{0} B$
$D \xrightarrow{0} D$ } not in same block

separate A & D

$A \xrightarrow{0} B$
$E \xrightarrow{0} E$ } not in ~~same~~ block

~~so sp~~ separate A & E

$D \xrightarrow{0} D$
$E \xrightarrow{0} E$ } same block

$D \xrightarrow{1} E$
$E \xrightarrow{1} E$ } same block but we can not put bar

$B, C \xrightarrow{0} B$ and $B, C \xrightarrow{1} C$ } always be in same block

if something is like permanent

If $B \xrightarrow{0} D$
$C \xrightarrow{0} B$ } you can not put bar as it may get divided.

EXCELLENT

Max equivalence class - no of states

$\Pi_2 = \{\{B,C\}, \{A\}, \{D,E\}\}$

$\Pi_1 = \Pi_2$

i.e K-equivalent
1-equivalent

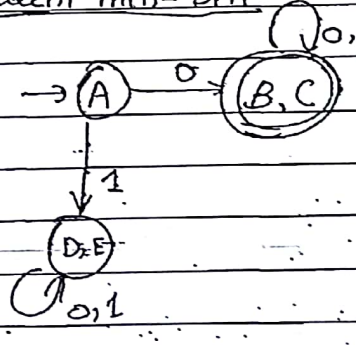
Steps -

for Π_0 - put final states & Non final state separated as ϵ-string is only accepted by final string.

for $\Pi_1 \cdots \Pi_k$

1. check for 1 length string for each state in same block
2. if $\Pi_k = \Pi_{k+1}$ stop
   else again

∴ equivalent min-DFA -



- Limitation -
  - This algo is incapable of removing non reachable states
  - so when a DFA contain Non-reachable state, firstly remove them.

EXCELLENT

Page No.
Date : / /

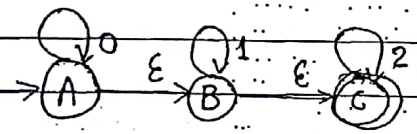
E-NFA - NFA having atleast one ε-move.

ε-closure

if a ε-nfa of n states with k ε-move & b size alphabet then its transition system with no ε-move will also have n states as no new states get added or no ~~per~~ previous getting deleted.

~~O/P-~~
~~if a ε-nfa of n states &~~
I/P - ε-nfa of n state
O/P - nfa with no of state
Ag - @ atmost 2^n

Ex $0^*1^*2^*$

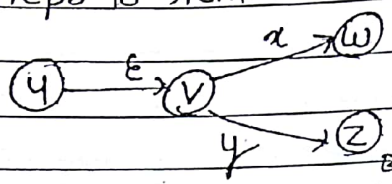
I/P 

transition state with ε- move-

ε- closure (A) = $\delta^*(A, \varepsilon)$
ε-closure (A) = {A, B, C}
ε-closure (B) = {B, C}
ε-closure (C) = {C}

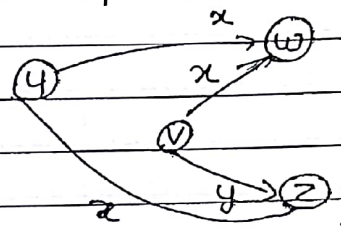
EXCELLENT

State steps to remove ε move

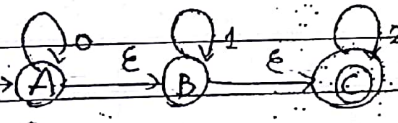


1. duplicate all the arrow on v are as if originating from u. (duplicate all the arrow originating for v as it is originating from u)
2. if u is a initial state then make v as initial state.
3. if v is a final state the make u as final state.

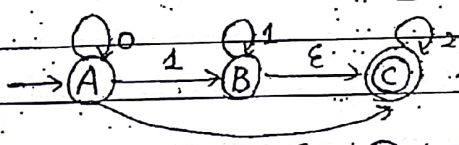
step 1



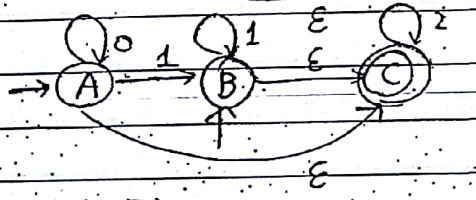
Ex -



A→B

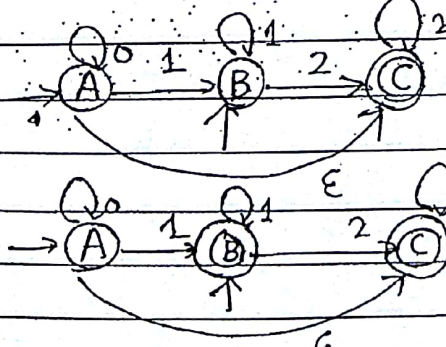


- duplication of arrow



- initial state

Remove Null B/w B & C



- final state travel backward
- Initial state travel forward

Conversion of ε to NFA

Guidelines-

1. A Finite Language is always regular

Ex $\{a^n b^n \mid n \geq 0\}$ - Not regular

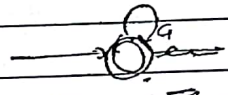
$\{a^n b^n \mid n \leq 10\}$ - Regular

$\{a^n b^n \mid n \geq 10\}$ - Not a Regular language

$\{a^n \mid n \geq 0\}$ - ~~Linear~~ Regular as FA can be drawn.

$\{a^{3n+1} \mid n \geq 0\}$ - Regular

↳ if powers are linear then grammar is regular



Non Linear powers strings are ~~CFL~~ or CSL

$\{a^{n^2} \mid n \geq 0\}$ - Not Regular

$\{a^{n^2} \mid n \leq 5\}$ - Regular

A ~~finr~~ A infinite language with non-linear power is always CSL.

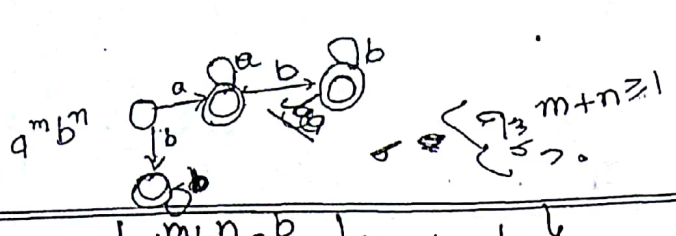
A infinite language with linear power is always regular when it does not require any string ~~machine~~ matching or any storage

$\{2^n \mid n \geq 0\}$ - Not a regular as multiplication is required

$\{a^m b^n \mid m+n = 10\}$ Regular

$\{a^m b^n \mid m+n = p\}$ - Not regular

(FA & PDA can't do multiplication & division)



Page No.
Date : / /

$\{a^m b^n c^p \mid m-n=1\}$

↓ Not Regular
as it is infinite & it requires comparison.

- if any lang. is infinite if
  - Non linear power then it is not regular
  - infinite sub-set (+, L, *, ?) then it is not regular
  - infinite comparison
  - infinite string machine

FA - can never do string Matching
PDA can do one string matching, but in reverse
ie PDA can do 1 comparison ie ww^R
CSL - can do more than one string matching.

$\{ww^R \mid w \in \{0,1\}^*\}$ - CFL

$\{ww^R \mid w \in \{0,1\}\}$ - Regular

$\{ww^Rw \mid w \in \{0,1\}^*\}$ - Not CFL & not regular

$\{a^{2n} b^n c^n \mid n \geq 0\}$ not CFL as require two comparison

$\{a^m b^n \mid m+n \geq 1\}$ regular

- $\{a^n b^m c^m d^n \mid m,n \geq 0\}$ - CFL as n, m has 1-1 comparison.

You can do one comp. on one variable in PDA but you can do 2 comp at two diff variable.

EXCELLENT

$\{a^n b^m c^n d^m\}$ → 2 comparison n
2 comparison m
↳ Not a PDA

$\{a^m b^n c^n d^m\}$ → PDA or CFL

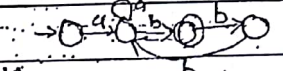
$\{a^n b^m c^n\}$ - PDA as [when b came, donot push it to stack.]

$\{a^{m+n} b^m c^n \mid m,n \geq 0\}$ - accepted by PDA

Language not accepted by CSL = which are recursive

$\{(ab)^n (cd)^n \mid n \geq 0\}$ ~~Regular~~ PDA

$\{a^m b^n \mid m,n \geq 0\}$ - Regular



~~$\{a^{m+1} b^{3m+1} \mid m,n \geq 0\}$ - CFL~~ $\{a^{m+1} b^{3n+1}\}$ - Regular

- $\{ww^R \mid w \in \{0,1\}^*\}$ - CFL
$\{ww^R \mid w \in \{0,1\}\}$ - Regular
$\{ww^R \mid w \in \{0,1\}^*, |w| \leq 10\}$ Regular
$\{w_1 w_2^R \mid w_1, w_2 \in (0,1)^*\}$ → Regular
~~$(0+1)^* (0+1)^*$~~

$L = \{wxw^R \mid w \in (0+1)^*, x \in (0+1)^*\}$ - Regular $\{w(0+1)^* w^R = (0+1)^*$
as
$\{x \mid x \in (0+1)^*\}$ is subset of L
if $w = \varepsilon$
~~$\varepsilon x \varepsilon$~~ $\varepsilon x \varepsilon = x = (0+1)^*$

EXCELLENT

$\{ wxw^R \mid w \in (0,1)^* \; x \in (0,1) \}$ - CFL

* All palindrome on binary alphabet are CFL
All palindrome on unary alphabet are regular as all string will be palindrom.

$\{ wxw^R \mid w \in (0)^* \; x \in \{0\} \}$
↳ Regular

*Regular is not closure over subset, superset, infinite intersection, infinite union, infinite difference.

If Regular is closure over subset (having) then $(0+1)^*$ is regular & then any language can become regular as $(0^n 1^n 0^n)$ also.

Superset :- $L = \emptyset$ is closet R.L.
then superset of L is covering all language.
∴ All language cannot be regular

Infinite intersection & Union -

$\{a^n b^n \mid n \geq 0\}$ → Illo regular
if U is regular then $\{\} \cup \{ab\} \cup \{aabb\} \ldots$ will make it regular

Infinite Intersection - $A \cap B = \overline{\overline{A} \cup \overline{B}}$
↓ Not regular as U is also not regular

Infinite difference $A - B = A \cap \overline{B}$ → Not regular

EXCELLENT

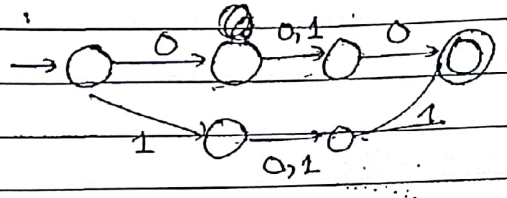
$x(0+1)^+$

Page No. 62
Date : / /

$\{ wxw^R \mid w \in (0,1)^*, x \in \{0,1\}^* \}$ – Regular

$\{ wxw^R \mid w \in (0,1)\ \ x \in \{0,1\} \}$

$= 0(0+1)0 + 1(0+1)1$



$\{ wxw^R \mid w \in \{0,1\}^*, x \in \{0,1\}^+ \}$ – Regular as $x\ (0+1)^+$ cover all cases

$(0+1)^+ \in (0+1)^+$

$\{ wxw^R \mid w \in (0,1)^+ \ x \in (0,1)^+ \}$ → Regular

$\{ wxw^R \mid w \in (0,1)^+ \ x \in (0,1)^* \}$ – Regular

~~$\{wxw$~~

$\{ ww \mid w \in (0+1)^* \}$ – CSL

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

$(0+1)^*$

$\{ ww \mid w \in (0,1) \}$ – Regular

$\{ wxw \mid w \in (0,1)^* \ x \in (0,1)^* \}$ – Regular as you can put every thing as x

$\{ wxw \mid w \in (0,1)^*, x \in (0,1)^+ \}$ – Regular

$\{ wxw \mid w \in (0,1)^+ \ x \in (0,1)^* \}$ – CSL

$0x0 + 1x1$

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

as but $\{ w = 0 \}$ not covered not covered by either $0(1+0)^*0$ $1(1+0)^*1$

EXCELLENT

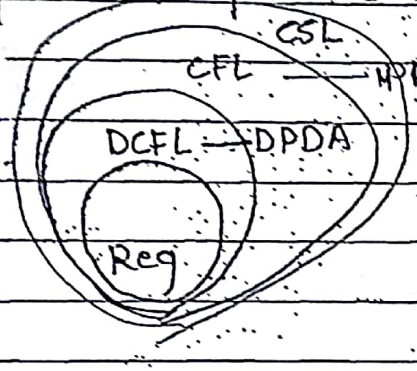
$0(0+1)^+0 + 1(0+1)^+\phi$ → Also CSL

$\{xww^R \mid w, x \in (0,1)^+\}$ — CFL

$x00+x11$ not covering. $w=01$

$ww^Rx \mid w, x \in (0,1)^+$ — ~~CFL~~ Regular CFL

$\{w(w^R)^* \mid w \in (0,1)^*\}$ - Regular



DPDA is having less power than NPDA.

CLOSURE PROPERTIES
- Union
- Intersection
- ~~Identity~~ Complement
- Concatenation
- Iteration / Closure
- Reversal of L / Transpose of L
- ~~Homo~~ Homomorphism of L
- Inverse Homomorphism.

EXCELLENT

L^R	✓	×	✓				No information available
$h(L)$	✓	×	✓				
$h^{-1}(L)$	✓	✓	✓				
$L \cup R$	✓	✓	✓	✓	✓		
$L \cap R$	✓	✓	✓	✓	✓		
$L - R$	✓	✓	✓	✓	✓		

- Regular language is closed for every op^n except $\subseteq$, $\supseteq$, in$\cap$ in$\cup$, in$-$.
- No language is closed for $\subseteq$, $\supseteq$, in$\cap$, in$\cup$, in$-$.
  R → Regular language

Except

Every language is closed for $\cup, \cap, -$ with regular expression

- $L_1 \cup L_2$ = × (may or may not be DCFL)
  DCFL DCFL
  $L_1 \uparrow \cup L_2 \uparrow$ → push up
  CFL CFL = ✓ (may
  ∴ L_1, L_2 Union give CFL.

  If you wanna correct answer push to known the boundary.

as $\overline{L_1}$ = × (may) comp of CFL is CSL
DCFL

Page No.
Date : / /

Ques - $\{a^n b^n c^{2n}\} \cap (a^* b^*)$ is a CFL?

$\{a^n b^n c^{2n}\}$ ↑ CFL ∩ regular = Yes it is a CFL.

Never do merge the language, try to go for closure.

Ques - DCFL↑ ∪ CFL ⇒ so push smaller one

CFL ∪ CFL = CFL

Ques DCFL.. reg ↑

DCFL . DCFL = X

↓

DCFL↑ . DCFL↑ = CFL . CFL = ✓

Try to give the closest Answer

Ques $(a^n b^n c^{2n}) \cap (a^* b (a+b)^* b a^*)$ = CFL ∩ reg = CFL

↓ DCFL

(a) CFL

(b) DCFL

(c) regular

(d) CSL

Ques - $L = \{a^n b^n\}$ $R = \Sigma^*$

$L \cup R = ?$ $L \cup R \Rightarrow$ CFL

$L \rightarrow$ CFL CFL reg

but $\{a^n b^n\} \cup \Sigma^*$ is Σ^*

↓ regular

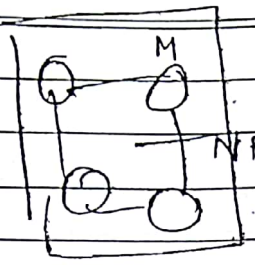
$L \cup R$ L:

∴ regular & non-regular

EXCELLENT

{- Difference is not commutative.}

Page No. 64
Date : / /

Ques 

$L(M)$ is regular or not?

Yes, it is regular.

Ques G_1 | Some Rule

G_2 | Some Rule

Ques $\overline{RE}$ = ?

May not may not be RE

Ques - L_1 - DCFL L_2 - CFL

$L_1 - L_2 = L_1 \cap \overline{L_2}$

$= DCFL \cap \overline{CFL}$

$= DCFL \cap CSL$

$= CSL$

$L_2 - L_1$

$= L_2 \cap \overline{L_1}$

$= CFL \cap \overline{DCFL}$

$= CFL \cap CFL$

$= CSL$

Ques $L_1 - L_2 = L_1 \cap \overline{L_2}$

XOR $L_1 \oplus L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$

NAND $L_1 \uparrow L_2 = \overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$

XNOR $L_1 \downarrow L_2 = \overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$

$L_1 \Rightarrow L_2$

XNOR $L_1 \Leftrightarrow L_2 = (L_1 \cap L_2) \cup (\overline{L_1} \cap \overline{L_2})$

Ques - L - Reg = L

Reg - L = may no be L

$L = CFL$

$Reg - CFL = Reg \cap \overline{CFL}$

$= Reg \cap CSL$

$= CSL$

EXCELLENT

Ques L_1 = DCFL

L_2 = reg

$L_1 \oplus L_2 = (L_1 \cap \bar{L_2}) \cup (\bar{L_2} \cap \bar{L_1})$

$= (DCFL \cap reg) \cup (Reg \cap DCFL)$

CFL $\cup$ CFL

= CFL

precedence – ( ) $\geq$ 'complement' $\geq \cap \geq \cup \geq \Rightarrow \geq \Leftrightarrow$

Ques :- L_1 - DCFL

$L_2 \rightarrow$ DCFL

$(L_1 \cap L_2)$ = first $\cap$ then comp. will generate weak answer so use demorgan law

$\overline{(L_1 \cap L_2)} = \bar{L_1} \cup \bar{L_2}$

DCFL $\cup$ CSL

= CSL

- For a language to be closed under ; $\oplus$, -, $\uparrow$, $\downarrow$, $\Rightarrow$, $\Leftrightarrow$, it has to be closed under $\cup$, $\cap$, — complement.
  $\cap$ is can also be converted to $\cup$ by demorgan law

∴ So

for a language to be closed under $\oplus$, -, $\uparrow$, $\downarrow$, $\Rightarrow$ $\Leftrightarrow$ (secondary op^n), it must be closed under $\cup$, complement.

- Reg, CSL, Rec are closed on secondary op^n.

~~$L_2 = q\,q$~~

~~$L_1 = \{ab^n c^n$~~

A CFL is DCFL if

1. the push to pop has to be cleared. i.e no confusion. i.e (when to push when to pop) or how much to push it is also to be clear
2. How much the push has to be clear. pop
3. $\{a^m b^n c^p \mid m = n \text{ or } m = p\}$ - for DCFL, language must not have it.

Ex - $\{ww^R \mid w \in \{a,b\}^*\}$ - It is a CFL not a DCFL as push to pop is not cleared.

as 1011101

how system will know when to push & when to pop but NDPA will do it by splitting of machine, storing each & every state

Union - Not closed

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

$L_2 = \{b^n c^{2n} \mid n \geq 0\}$ DPDA.

$L_1 \cup L_2 = b^n c^n$ or $b^n c^{2n}$: Not a DPDA or DCFL as it is confused how much to push or much to pop

bbcc

bbcccc

∴ double comp on a single variable is allowed with 'OR' cond^n in CFL or PDA

$\{a^m b^n c^p \mid m = n \text{ or } m = p\}$

↓CFL

but not DCFL

EXCELLENT

II $\{wxw^R \mid w \in (0,1)^* \quad x \in (0,1)^*\}$

↓

which one is DCFL

Ans- II as each RL is DCFL

Intersection- not closed

$L_1 = \{a^n b^n c^m\}$ — ~~CFL~~ ~~DCFL~~ DCFL

$L_2 = \{a^n b^m c^m\}$

$L_1 \cap L_2 = \{a^n b^n c^n\}$ - not a DCFL

Q ww $\{ab^n c^n\} \cup \{db^n c^{2n}\} \rightarrow$ DCFL

as no confusion

when 'd' came. push 1 & b

When 'd' came push 2 (d)

$\{c^n b^n a\} \cup \{c^{2n} b^n d\}$ = Not a DCFL

CFL - Intersection - not closed

$L_1 = \{a^n b^n c^m\}$

$L_2 = \{a^n b^m c^m\}$

$L_1 \cap L_2 = \{a^n b^n c^n\}$ - CSL

EXCELLENT

for Regular language-

Union - $L_1 \rightarrow$ regular $\rightarrow \exists$ re$_1$

$L_2 \rightarrow$ regular $\rightarrow \exists$ re$_2$

$\therefore L_1 \cup L_2 =$ re$_1$ + re$_2$ = re $\therefore$

$L_1 \cup L_2 =$ re $\therefore$ it is regular

- Kleen theorem for Generating Machine & Grammar guarantee Right linear regular grammar.

$\therefore L_1 \rightarrow$ re$_1$

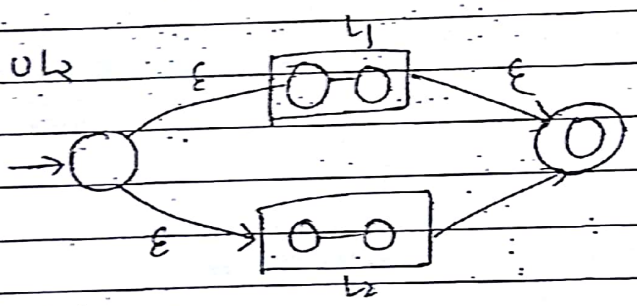
$L_2 \rightarrow$ re$_2$

$L_1 \cup L_2 \rightarrow$ draw Machine using Theorem

$\therefore$ it is also generate Grammar which is RLRG.

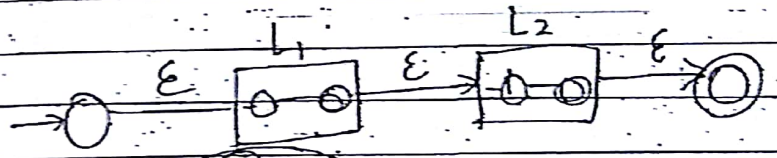
$\therefore$ it is regular.

for $L_1 \cup L_2$

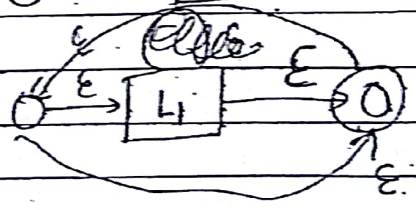


$L_1 \cap L_2$

$L_1 \cdot L_2$



L^*



∴ it is closed.

for ∩ Intersection- if L_1 is regular & L_2 is also regular
∴ ∃e M_1, M_2 corresponds to L_1, L_2
(Machine)

then $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$
↑ product Automata

$M_1 \times M_2$ can be obtained from DFA's only on same alphabet set

Product Automata- if M_1 = m state
M_2 = n state
$M_1 \times M_2$ = m×n state

$L(M_1 \times M_2) = L(M_2 \times M_1)$ // it is commutative
they will accept same language but M/C will be different.
∴ $L(M_1 \times M_2) = L(M_2 \times M_1)$
but $M_1 \times M_2$ configuration ≠ $M_2 \times M_1$ confi.

Ex-

M_1	0	1
→ A	B	A
(B)	A	B

M_2	0	1
→ (C)	D	D
(D)	C	D

$M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$
$M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$

Ans $M_1 \times M_2 = (Q, \Sigma, \delta, q_0, F)$
$Q = Q_1 \times Q_2$
$\Sigma = \Sigma_1 = \Sigma_2$
$q_0 = \{q_{01}, q_{02}\}$

$F_1 \times F_2$ will allow the o/p which is common to both & this is the property of intersection.

$\therefore Q = \{\{A,C\}, \{A,D\}, \{B,C\}, \{B,D\}\}$

- $\delta((X,Y), a) = (\delta_1(X,a), \delta_2(Y,a))$ } $L_1 \cap L_2$

where

$(X,Y) \in Q$ $\quad X \in Q_1$

$a \in \Sigma$ $\quad Y \in Q_2$

~~X X Y~~ $M_1 \times M_2$	0	1
$\rightarrow$ A C	BD	AD
A D	BC	AD
(B C)	AD	BD
BD	AC	BD

for its conversion for requirement of answer

If given

$\rightarrow$ A	A	B
(B)	B	C
C	C	D
D	D	A

initially take initial state & final state & check whether it matches $A \rightarrow AC$

$B \rightarrow BC$

you can check it intutively.

A	A
–	B
B	
–	A

F for $L_1 - L_2$ $= \{BD\}$

for $L_1 \cup L_2$ $F = \{AC, BC, BD\}$

for $L_1 \oplus L_2 = \{AC, BD\}$

- $L_1 \times L_2$ can be used to create any secondary op^n machine.

1. if $L_1 \cup L_2$ is required – construct $L_1 \times L_2$ and $F = ((X,Y)$ where $X \in F_1$ or $Y \in F_2)$

2. if $L_1 - L_2$ is required construct $L_1 - L_2$

$\Rightarrow F = \{F_1 \times$ (non final state of L_2)

$F = \{F_1 \times \overline{F_2}\}$

3. $L_1 \oplus L_2$ – $F = \{$ ~~Q~~ $X \in F_1$ or $Y \in F_2$ but $X, Y \notin F_1, F_2\}$

4. ~~$L_1 \cdot L_2$~~

EXCELLENT

If is possible ... regular but language gene... regular

Concatenation - $L_1 \to$ regular.e

$L_2 \to .$ regular e

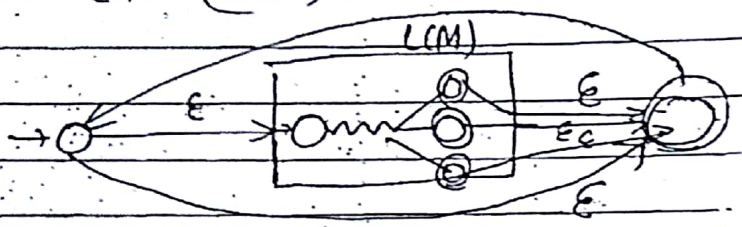
$L_1 L_2 \Rightarrow$ regular e

closure - $L - RL$

$L \to$ regular.e

$L^* = (\text{regular}.e)^*$ is also regular

To covert $L(M)$ to $(L(M))^*$ ϵ



Reversal - $\left\{ \begin{array}{l} S \to AB \\ A \to aA \mid \epsilon \\ B \to bB \mid \epsilon \end{array} \right\}$ non regular grammar.

generate RL

$\therefore L = a^* b^*$ bcoz CFL can also generate R.E

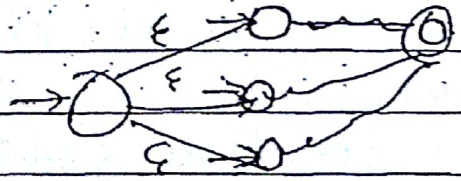
ξ

- ~~Type 3 cat~~
- but RG can not be written for Non-Regular language.

Reversal - 1. Exchange the final state & ~~new~~ initial state

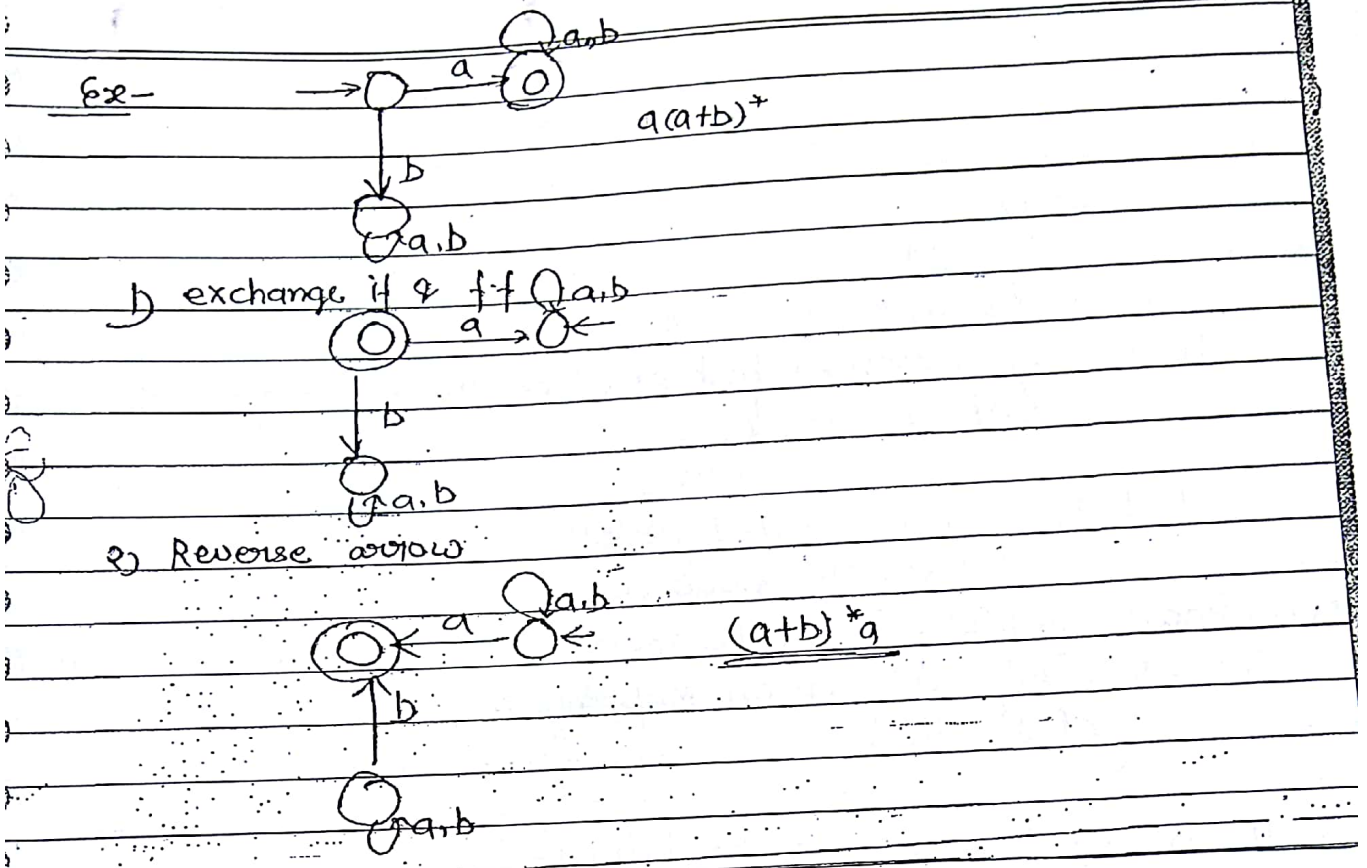
2. Reverse all the arrows

3. In case you have ~~initi~~ more than initial state convert into NFa by new initial state



- this algo may or may not give minimized Machine

EXCELLENT



Homomorphism of Language- If L is language on Σ then h(L) will be L' on Σ'

$\Sigma' = \Gamma^*$

as $L = \{01, 100\}$ on $\Sigma = \{0,1\}$

$\Sigma \to \Gamma^*$

$\{\Sigma \to \Gamma^*\}$

Let $h(0) = aa$

$h(1) = b$

$h(L) = \{aab, baaaa\}$

It is closed as L is RE ∴ and h(L) is also RE

- h(L) is of finite L is alway L.
- h(L) of Infinite L may be finite or infinite h(L) is many to one mapping.

Ex $L = (0|+100)^*00$ $h(0) = \epsilon$

$h(1) = \epsilon$

$h(L) = \epsilon$

EXCELLENT

Date : / /

$\Sigma = \{a, b, c\}$

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

$h(a) = 1$

$h(b) = 0$

$h(c) = 01$

$h(L) = \{1001\}$

$\therefore h^{-1}(L)$ = each string which can form this language on Σ

$\therefore \{abba, abc\}$

..... if $h(L) = \{1010\}$ $h(a) = 1$, $h(b) = 0$ $h(c) = 10$

$= \{abab, abc, cab, cc\}$

- cardinality of $h(L)$ ~~will~~ can be more than or less than or equal to L as it only depends on Mapping $h(a)$

$\therefore (h(L) \leq |L|)$

Converse of ~~two~~ closure properties-

Union-

- If $L_1 \cup L_2$ is regular then it may ~~or may not~~ be possible that L_1, L_2 are not regular.
- It is only 1 way regular theorem

L_1 is reg & L_2 is reg $\Rightarrow L_1 \cup L_2$ is reg.

as if

$L_1 = a^n b^n \rightarrow$ Non Reg.

$L_2 = (a+b)^* \rightarrow$ Reg

$L_1 \cup L_2 = a^*(a+b)^* = \Sigma^*$

and $L_1 = a^n b^n$

$L_2 = \overline{a^n b^n}$

$L_1 \cup L_2 = a^n b^n \cup \overline{a^n b^n}$

$= (a+b)^*$

EXCELLER

• Complement & Reversal of an Regular language is regular
• It is two way theorem
∴ If L is regular
then $\bar{L}$ is also regular
& if L is Non Regular then $\bar{L}$ is also Non regular.

Intersection - if $L_1 \cap L_2$ is regular, then L_1, L_2 may or may not be regular

Ex-

$\phi \cap a^n b^n = \phi$

L_1 L_2 $L_1 \cap L_2$

T ↑

~~regular~~ ~~Non reg~~

$a^n b^n \cap a^n b^n = \phi$

L_1 L_2 $L_1 \cap L_2$

↓ NR $\bar{T}_R$

Ques- if $L_1 \cup L_2$ is Non regular

- both are NR
- 1 is NR
- atleast 1 is NR

Ques- If $L_1 \cup L_2$ is Non regular & L_2 is reg.

$L_1 = ?$

= Must be non regular.

EXCELLENT

Page No.
Date : / /

for Complement- L is reg $\iff$ $\bar{L}$ is regular
i.e if L is reg then $\bar{L}$ is regular
if $\bar{L}$ is reg then L is regular
if L is ~~re~~ NR then $\bar{L}$ is Also non Regular
if $\bar{L}$ is NR then also $\bar{L}$ is regular

if L $\bar{L}$ are two language which is not possible:

(a) both are DCFL's
(b) both are regular
(c) both are RE → because if ~~not RE~~ if it is REC as REC is also RE
(d) ~~bo~~ one is REC, other is "Not RE"

L = rec (also an re)
$\bar{L}$ = rec
↳ possible

for concatenation- • L_1 is regular & L_2 is reg. ⇒ $L_1 \cdot L_2$ is regular
• if $L_1 L_2$ is regular then atleast one has to be regular.

$\phi \cdot a^n b^n = \phi$ $(L_1 \cdot L_2)$
L_1 L_2 ~~L=RL~~
↑ non regular

• if $L_1 L_2$ is not regular atleast one has to non Regular.

Qus $L_1 \cdot L_2$ = regular L_1 = regular
L_1 = may or may not be regular

EXCELLENT

for * L is reg. $\Rightarrow L^*$ is regular

- if L^* is regular, L may or may not be regular

if $L^* = \{a^n b^n + a + b\}^* = (a+b)^*$

$\{L = \{a^n b^n + a + b\}\}$
↓ non regular

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

L^* is $\{\varepsilon + a + aa + a^4 + a^9 + \ldots\}^*$

$= (a^*)$

= regular

It is possible that closure of non regular language is regular.

$L = \{a^n \mid n \text{ is prime}\}$

$= \{a^2 + a^5 + a^7 + a^{11} + a^{13} + a^{15} + \ldots\}$

$= a^2 a^* \quad = aaa^* \quad = a^* - \{a\}$

= regular

for Reversal L is reverse regular $\Leftrightarrow L^R$ is also regular.

L is CFL $\Leftrightarrow L^R$ is also CFL.

for h(L) if h(L) is regular $\Rightarrow$ h(L) is also regular

if h(L) is regular $\Rightarrow$ L may or may not be regular

Page No.
Date : / /

23-Apr

DECIDABILITY - for a problem to be decidable, an algorithm exist which solve problem in finite amount of time & it must halt.

Undecidable - No algorithm exist
if algorithm exist, it will give answer but for some input, it may go to hang.

A problem with answer 'Yes' or 'No' are decision problem

13 Problem - Decidable
27 —— - Undecidable

A RLRG can be converted to LLRG or vice versa

Trivial Decidable problem - O/P = Input
i.e, problem of converting DFA to NFA as each DFA itself a NFA.

Decidable
- Proper Decidable (like NFA to DFA) (13/40)
- Trivial Decidable (doing Nothing (or always the same)) (like DFA to NFA)

Undecidable
- Proper UD (No logic exist for this type of problem) (27/40)
- Trivial Undecidable (like NDPA → DPDA)

as these are UD because of power of NDPA is more than DPDA we cant convert it.
∴ there are problem which are TUD i.e which are UD because of power.

EXCELLENT

1. NPDA has more power than DPDA
2. LBA's & TM are going on to research.

is always decidable or vice-versa

	(FA Type 3) RE	(PDA Type 2)	(Type 1 LBA)	(HTM)	(Type 0 TM)
	RGG	CFL	CSL	REC	RE
1. MEMBERSHIP ($w \in L$?)	✓	✓	✓	✓	×
2. EMPTINESS ($L = \phi$?)	✓	✓	×	×	×
3. FINITENESS (L = Finite?)	✓	✓	×	×	×
4. EQUIVALENCE ($L_1 = L_2$?)	✓	×	×	×	×
5. REGULARITY (L = Regular?) (of Language)	✓	×	×	×	×
6. AMBIGUITY (L = Ambiguous?)	✓	×	×	×	×
7. UNIVERSALITY ($L = \Sigma^*$?)	✓	×	×	×	×
8. DISJOINTNESS ($L_1 \cap L_2 = \phi$)	✓	×	×	×	×

reduced by GUD

undecidable because they are not REC has HTM.

A problem is Decidable if and only if L(P) is REC.
So if Ques asked a problem is REC or not, check whether it is decidable or not

- all decidable problem are two ways
- if a problem is Undecidable then it is Not REC ie RE or Not RE
- ie if CSL finiteness is undecidable then infiniteness is also undecidable.

EXCELLENT

Page No.
Date : / /

Membership- • $w \in L(G)$ iff there exist an ~~algo~~ derivation of w using production

or

$w \in L(M)$ iff M accept w

or

$w \in L(r)$, iff r generate w.

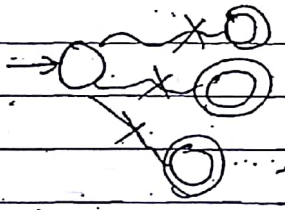
• PDA, HTM, FA, LBA Halts on every input.

• So Machine can tell in finite time where w is member of L or not.

• for RE, it may or may not halt so membership is undecidable as if you give some input, it go to loop so it is undecidable.

Emptiness - ~~L(M) =~~

$L(M) = \emptyset$ iff there is no directed path to reach final state



for Reg ~~lang~~ is decidable because if A is q_0 & F={B,C,D} I will use graph theory to check whether there any path b/w A,B A,C A,D it is decidable.

if A,B, AC, A.D is no path b/w them ∴ it is empty

For CFL - CFL can make list of useful & useless elements or variables.

If the list of useless variable has start symbol then it accept empty language.

EXCELLENT

for CFL

$L(G_1) = \phi$ if S e start symbol, belongs to useless variable

Emptiness is mainly discussed by grammar

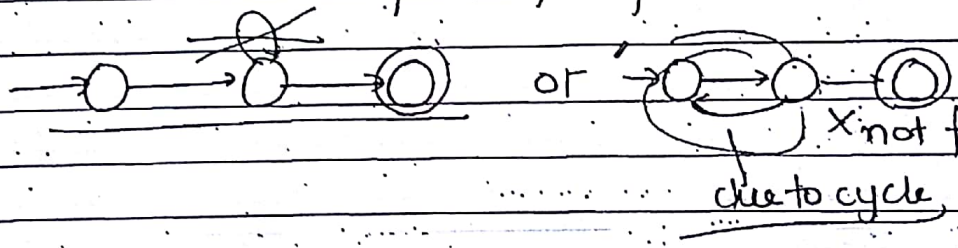
CYK Algorithm - to check whether $w \in L(G_1)$ for G_1 of type 2

complexity $= O(n^3)$

- n is the length of word given as input to the algorithm. ie (length of w)
- Most efficient algorithm.

Finite Ness - for Finite Automata -

for M, L(M) is finite if and only if there is no cycle ~~b/w directed between path from~~ in directed path b/w final & initial state



Ques - FA has loop

1) it is finite
2. it is infinite
3) May or may not be finite

FA has loop in Path b/w q_0 & F
i) it is surely infinite.

FA has loop but not in path of q_0 & F
i) it is surely finite

EXCELLENT

Page No.
Date : / /

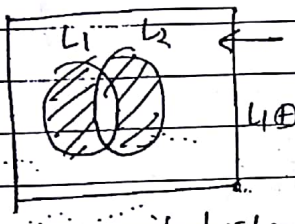
it is decidable bcause graph theory can tell about cycle.

for CFGs - bcoz there exist an algorithm to check about recursion & fruitful recursion.

EQUIVALENCE - Only two r.e are equivalent
one r.e & one pda are not equivalent
Type 0, 1, 2 are not decidable for equivalent

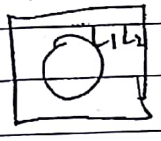
L_1, L_2 are equal iff $L_1 \oplus L_2 = \phi$

$L_1\bar{L}_2 + L_2\bar{L}_1$ ... as $L_2 = \bar{L}_1$
$\therefore L_2 = \bar{L}_1$
$L_1\bar{L}_1$



$(L_1 - L_2) \cup (L_2 - L_1)$
$\downarrow \phi \quad \downarrow \phi$

if $L_1 = L_2$



- develop M_1, M_2
- $M_1 \oplus M_2$ by product automata
- if $M_1 \oplus M_2$ produces ϕ, then (it is decidable as it is having an algorithm to solve)

for REC, Regular, CSL are closed under U & $\bar{L}$ so you can decide for $M_1 \oplus M_2$
but Regular is ~~not~~ decidable under emptiness only
but CSL, REC not so it is undecidable.

EXCELLENT

Halting problem of TM is undecidable & all undecidable & 73
fundamental problem can be reduced by if Post therefore, other
is are also undecidable.

Date: / /

- REGULARITY if L is Regular ⇔ a finite automata for L.
  ↓ it is a trivial decidability problem for regular language.

if a problem P_1 is reducable to another problem P_2
$\therefore P_1 \leq P_2$
- if P_1 is undecidable then P_2 is also undecidable.
- it is only one way theorem.

if P_2 is undecidable, P_1 may or may not be undecidable.
- if P_2 is decidable surely P_1 is also decidable by (contra positive)
- if P_1 is decidable, P_2 may or may not be decidable)

~~AMBIG~~
- It is impossible to predict ~~when~~ whether a system will halt or not

Ambiguity - decidable for Regular language.

"A language is REC if the problem is decidable".

Ques I. $L = \{(w, M) \mid w \in L(M), M \text{ is FA}\}$ ↓ REC as Membership is decidable for Regular language

A problem is undecidable if it has infinite no of countable truth.

II. $L = \{M \mid L(M) = \phi, M \text{ is TM}\}$ ↓ Not REC as it is not decidable

$L = \{M \mid L(M) = \phi, M \text{ is HTM}\}$ - Not REC

EXCELLENT

Page No:
Date : / /

IV $L = \{(G_1, G_2) \mid G_1 \cong G_2$ if ~~M is FA~~ G_1 is Type 3$\}$

∀REC as it is decidable.

V $L = \{(M_1, M_2) \mid M_1 = PDA, M_2 = NFA, M_1 = M_2\}$

∴ NOT REC

VI $L = \{(M_1, M_2) \mid M_1 = PDA, M_2 = NFA, M_1 \neq M_2\}$

∴ NOT REC

as it is the complement problem.

VII $L = \{(M_1, M_2) \mid M_1 = PDA, M_2 = NFA, M_1 \neq M_2\}$

1. L is REC
2. ✓ L is RE but not REC
3. L is not RE

— both can be the answer so choose 2 option as it is very close to answer.

as No of 'not RE' problem is very less as compared to 'RE but not REC' problem.

EXCELLENT

Applications and Variations of FA-

Applications- 1. Lexical Analysis in compiler.

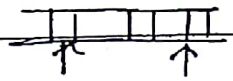
2. Pattern Matching but not string matching.
   ex- Lexical Analyzer
3. Text Editor
   - FIND/ SEARCH/ REPLACE
     ex- (starting with q)
   - Spell checker
     Ex- (R.E)
     ex- complete
     It will come up with menu have string ~~start~~ starting with compl.
   - GREP (Search command) in UNIX
     (General Regular Expression Processor
4. Finite storage
   Theorem - { if m words are there each of size n you will require 2^{mn} states for binary words.
   if it is ternary = 3^{mn} }
5. Sequential Circuit Design.
   (using mealy & moore)
   Ex- 1's complement, 2's complement

- FA is used to modelling a simple machine with less memory/ finite memory.
  Ex- Vending Machine

EXCELLENT

Variations of FA-

1. Multitape FA is equivalent to FA
   they both have same Power.
2. Multihead FA is equivalent to FA
   ie more than one head



it now only has finite memory.

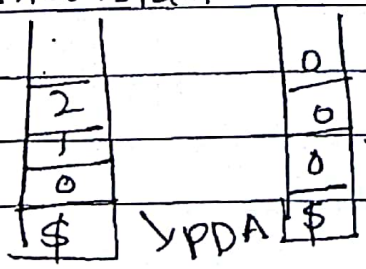
3. Two way automata also have same power to FA
   it is also known as 2-NFA & 2-DFA
   ie you can move left or right acc to your requirement.
   ie one can move to left and other can move to right
   but it can not accept non-regular language because of limited memory.

$\delta(q_0,1)=(q_1,1)(q_2,R)$

4. FA with stack.
   it has more power than FA as it become PDA of infinite size.

5. FA with finite stack has same power as FA as M/M is limited here.

6. FA + counter has more than FA
   it is known as CA (Counter Automata)
   it is similar to FA with stack but the basic difference is stack can have any symbol.
   but counter stack can have only one symbol



EXCELLENT

therefore, we can say that

FA < FA+counter < FA+stack

(it can do counting comparison → (it can do counting fn but cant do string matching)

Counter Automata has less power than PDA but has more than power than finite automata.

7. FA + two-stacks is equivalent is TM. it is & not equal to TM.

FA + 2-stack > FA + stack > ~~PDA~~ FA

2-PDA > 1-PDA > FA

So FA + two stack is more powerful than FA + 1-stack.
(it is equivalent to TM as it allows infinit Memory & movement in both direction)

8. FA + 3 stack has equal power as FA + 2 stack have, because Not machine is more powerful than TM.

Ques - n-stack PDA ≡ TM

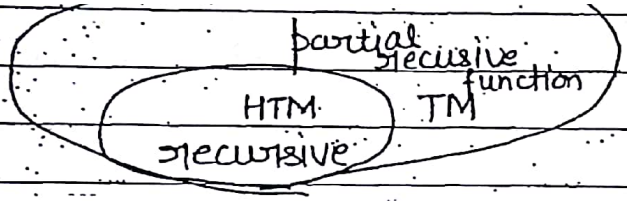
∴ n = {n ≥ 2}

9. FA + two counter is equivalent to Turing Machine.

FA < FA+1counter < FA+1-stack < FA+1-ccounter
or
FA+2-stack
|||
TM

To push FA to level of FA, min. required = 2 counters

EXCELLENT



1. Theory of Mealy & Moore.
2. Mealy/Moore + Input → output ?
3. Mealy/Moore → function
4. Mealy to Moore or Moore to Mealy.

Mealy & Moore Machine - It is a dfa with output.
it is not NFA as choice, ε, dc are not allowed but if is NFA as each dfa is NFA.

for Mealy - output fn. $\Delta_{Q \times \Sigma} \to \Gamma^*$ $\delta(q_0, a) = 001$

Moore Output fn. $\Delta_Q \to \Gamma^*$ $\Delta(q_0) = 001$

in Mealy O/P is when you exit but in Moore out is when you enter.

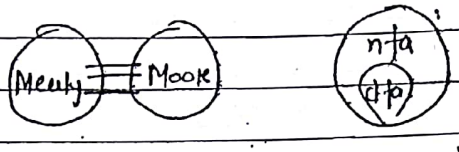
EXCELLENT

as $\Delta(q_0, a) = 001$

$\Delta(q_0, b) = 01$

but $\Delta(q_0) = 001$ (always same)

Mealy & Moore both have same powers & they are equal



- Each dfa is Nfa but Moore is not Mealy.

Formal Definitions:

Mealy/Moore $(Q, \Sigma, \Gamma, \delta, \Delta, q_0)$

- Q — set of state
- Σ — input symbols
- Γ — output alphabet
- δ — transition function
- Δ — output function
- q_0 — initial state

Differences-

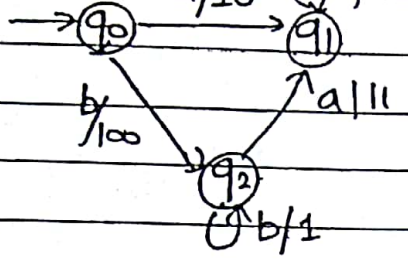
DFA/NFA	Mealy/Moore
• 5 tupled	• 6 tupled
• Final state as it work as acceptor.	• No final states are there.
• Γ, Δ are not there	• Γ, Δ are there Γ - as O/P is to be specified. Δ - output function.

Here, $\delta_{Q \times \Sigma \to Q}$

O/P function in Mealy- $\Delta_{Q \times \Sigma \to \Gamma^*}$ (O/P is fn of both present state & input alphabet)

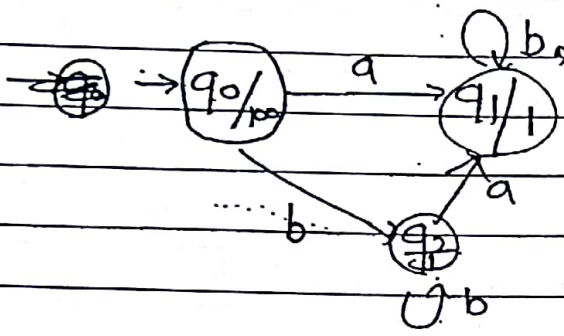
O/P function in Moore - $\Delta_{Q \to \Gamma^*}$ (O/P is fn of present state only

EXCELLENT



— Mealy Machine

if q_0, a comes goto q_1 and give o/p 10.



— Moore Machine

if $\Delta_{Q \times \Sigma^* \to \Gamma^*}$ then it may allow ϵ as input but it is not the case with transducer.

We can have o/p as Null as we have o/p alphabet as Γ^*

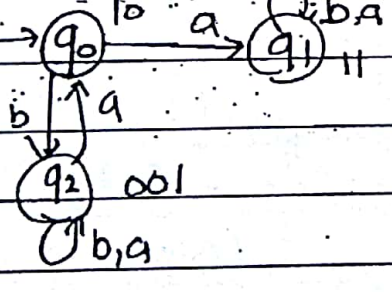
Ques - if you a Mealy machine with m state & n output then Moore will have atmost ~~mn~~ how many state?

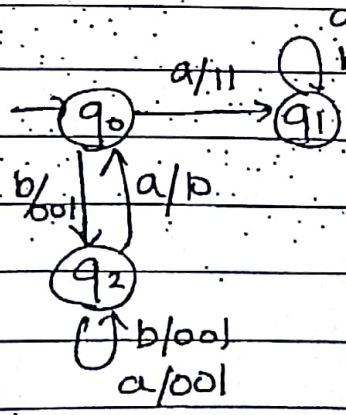
Ans - $\leq mn+1$ state

Ques Moore machine with m state & n output, equivalent Moore will have -

Ans - atmost mn states

Ex -





put the o/p with transition of the entering states.

EXCELLENT

some states as well.

	a	b	o/p
→q_0	q_1	q_2	10
q_1	q_2	q_1	11
q_2	q_2	q_0	001

← Moore Machine

Mealy Machine

	a	o/p	b	o/p
→q_0	q_1	11	q_2	001
→q_0	q_1	11	q_2	001
q_1	q_2	001	q_1	11
q_2	q_2	001	q_0	10

If it is

	a	o/p	b	o/p
→q_0	q_1	11	q_2	001
q_1	q_2	001	q_1	11
q_2	q_2	001	q_2	11

↳ both are equivalent-

∴ equivalent is

→q_0	q_1	11	q_2	001

∴ it will have atmost 3 states

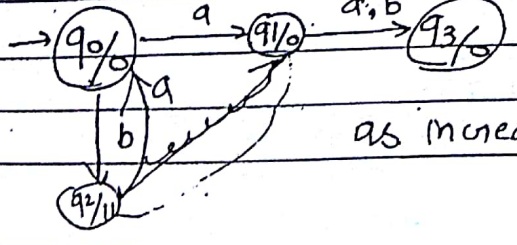
Moo Mealy to Moore-

	a		b	
q_0	q_1	0	q_2	11
q_1	q_2	0	q_1	0
q_2	q_0	0	q_1	11

Here m=3

n=2 ∴ worst case = 7 states

as it will result for many states

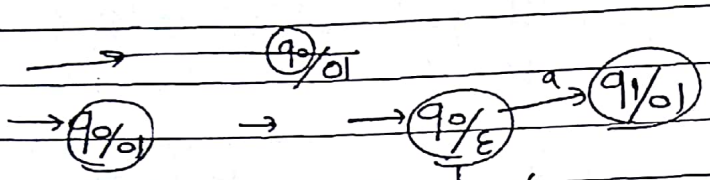


as increase of state occur.

mn cases can happen.

Page No.
Date : / /

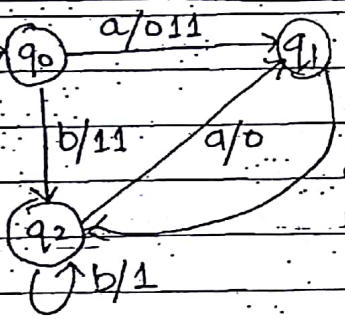
+1 occurs as when you enter Moore Machine will also produce o/p even at null state.
So you have keep initial state silent in begining so keep it output for ε at starting & add another state



+1 is keep initial state as silent state

∴ it can have atmost 'm+1' states

Ques-

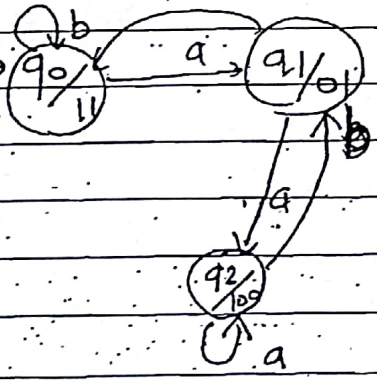


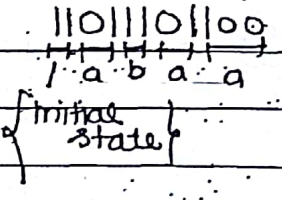
I/P = abaa
o/p = ?
0110001

for every I/P there is one o/p only

'abaa'

Ques





Note - if n size input string is given O/P will have n in Mealy
but in Moore - it is n+1.

EXCELLENT

Page No.
Date : / /

functions -

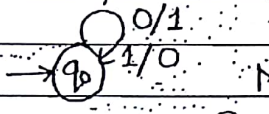
1. 1's complement
2. 2's complement
3. Binary Full Adder
4. Increment by 1

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

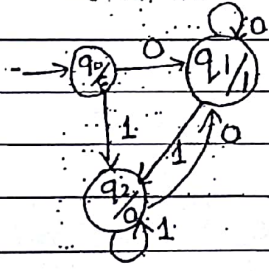
5. Change the sign bit
6. ~~Take~~ d(w) mod 3 (INTEGER DIVISIBILITY TESTER)
7. LOGICAL FUNCTION (NAND, NOR, XOR)
   (Simple Binary logical function)
8. Non-standard functions.

Moore is easier than Mealy if o/p are fixed ex- Mod n output is fixed.

1. 1's complement

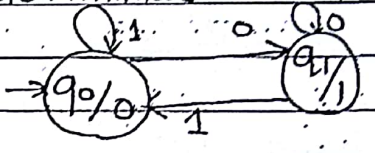


Mealy Machine



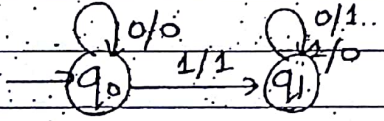
– Moore Machine

the Minimal machine is



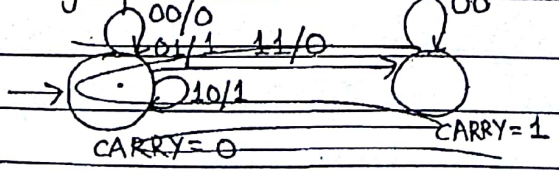
2. 2's complement-

- Input from LSB
- Remain as it is till first 1 came.
- Remain 1st 1 same & then complement

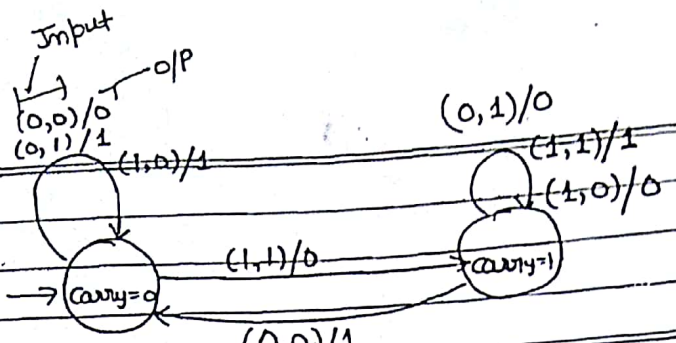


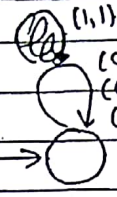
3. Binary full Adder

- I/P is pairs to full adder



EXCELLENT

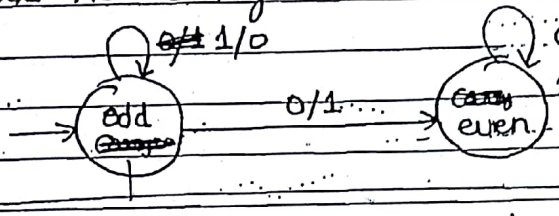


XOR - 

(half adder or XOR)

5. Increment by 1. even no- 1101**0**
odd no - 1101**1**

for even no - make lSB as 1. - Input from LSB.
for odd no - change has to done

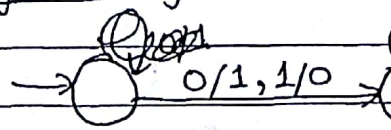


Ques - m state n o/p → total state in Moore - mn+1 or mn

because programmer can give instruction to user to not read starting state output so no need of putting silent twin.

if mn+1 mn choose mn+1

5. Change the sign bit - • Input from MSB



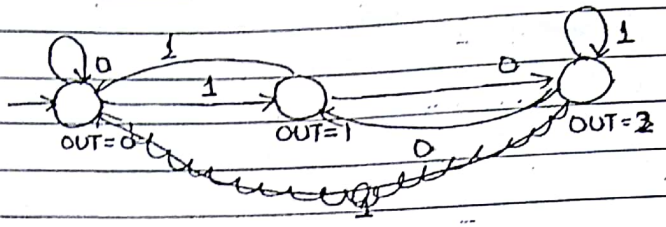
EXCELLEN

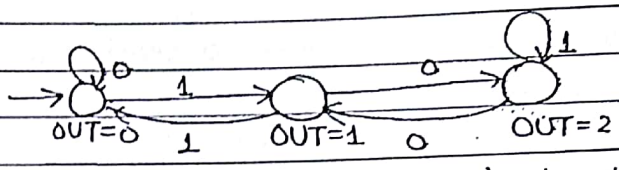
101101

0101011 0

Page No. 79
Date : / /

6. d(w) mod 3 -



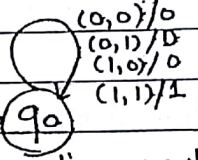


for 101101
0122210

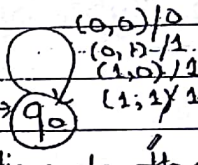
• instruction is given if last bit 0 then it is divisible by 3
If 1 → remainder is

10101
012221 0
it represent the residue of prefix

7. AND



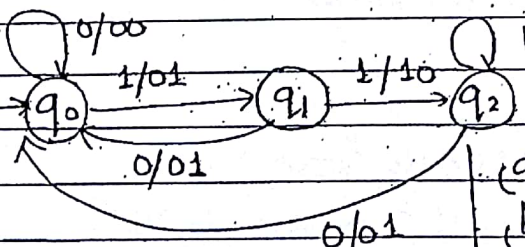
OR



for any logical function map value according to the truth table.

NON-STANDARD FUNCTION

Ques

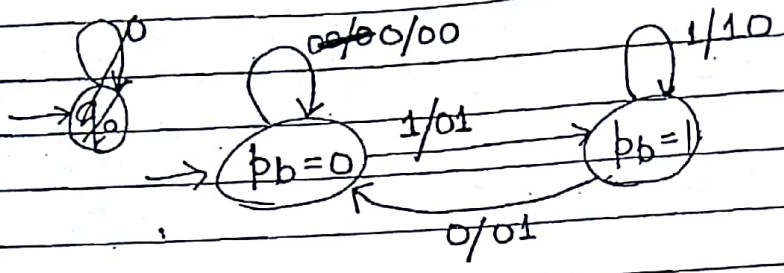


for such question, instead of drawing machine, just take an ~~artib~~ arbitrary input & check for o/p

(a) o/p as 01 for every 1 in I/P
(b) o/p as 00 ——— 0 in I/P
(c) sum of present & ~~current~~ previous bit of input
(d) None of these

EXCELLENT

Sum & present & previous bit-



{We have taken 2 states here as we have to remember only previous bit & it may be 0 or 1.}
∴ only two states required

Context Free Languages-

{Every Mealy Machine is moore machine - False
but for each Mealy, there exist an equivalent Moore - true}

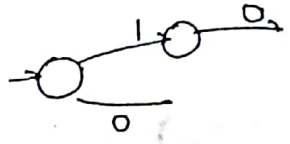
(1.) L → G & Types of CFL's (Standard CFL & grammar)
② G → L

(2.) Derivations: LMD, RMD, Derivation tree, partial DT
- Right Most Derivation
- Left Most Derivation

Ambiguity of grammar
Ambiguity of language

(3.) Membership algorithm- • CYK Algorithm — complexity - $O(n^3)$
• Brute Force parsing Algorith
- complexity - $O(k^n)$

• Properties of LL(k) & LR(k) Grammars.
↳ It will take O(n) time to check for membership.



(10+01)*

(4.) Algorithm in CFG:

(a) Removal of ε-productions.
(b) Removal of unit production.
(c) Removal of useless production.
(d) Removal of left recursion. } both causes ambiguity
(e) Removal of left factoring. }
(f) Conversion of CFG to CNF (Chomsky Normal Form)
(g) Conversion of CFG to GNF (Griebach Normal Form)

• Properties of CNF & GNF-

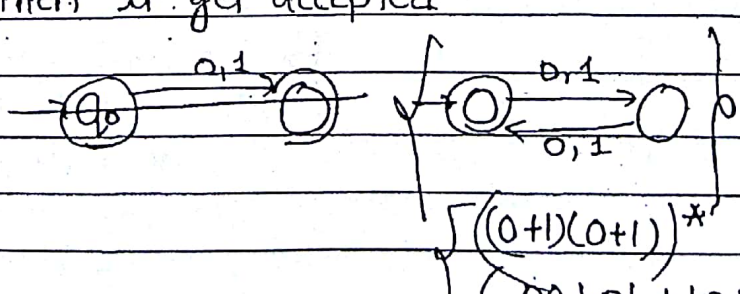
(5.) PDA programming: (a) DPDA & NPDA working
(b) Machine to Language conversion.
OR
Language to Machine conversion.

(a)

• Every Regular language is a CFL.
• PDA can do
  • counting comparison
  • string matching, but ~~the~~ FA cannot do both function

Ques I $L = \{ w_1 w_2 \mid |w_1| = |w_2| \}$ — Regular language
II $L = \{ w_1 \# w_2 \mid |w_1| = |w_2| \}$ — Not Regular but CFL as length of w_1 has to be stored.

I - as it is string of even length. if I/P is even length then it get accepted.



EXCELLENT

Page No.
Date : / /

Ques - All CFL's but not regular

1. $\{0^n 1^n \mid n \geq 0\}$

$S \to 0S1 \mid \epsilon$

$\{(01)^n (23)^n \mid n \geq 0\}$ $S \to 01S23 \mid \epsilon$

- $\{0^n 1^n \mid n \geq 1\}$

Ans = $S \to 0A1$, $A \to 0A1 \mid \epsilon$

or

$S \to 0S1 \mid 01$

- $\{0^n 1^n \mid n \geq 2\}$

$S \to 0S1 \mid 0011$

Ques $G_1 \to S \to 0S1 \mid \epsilon$

$G_2 - S \to 0S1 \mid 01$

$G_3 - S \to 0S1 \mid 01 \mid 0011$

$G_4 - S \to 0S1 \mid 0011$

G_2, G_3 are equivalent

Whenever check for equivalence of grammar initially check for equivalence of minimal string.

Ques - (a) $(0+1)^* \mid (0+1)^*$

(b) $(0+1)^* 11 (0+1)^*$

(c) $(0+1)^* 11 (0+1)^* + 1$

None is equivalent

(a) $(0+1)^* \mid$

(b) $(0+1)^* 11$

(c) $(0+1)^* \mid + (0+1)^* 11$

a, c are equivalent

To check for equivalence check for minimal & then start as 0, 1

00, 10, 11, 10

and so on-

Page No. 21
Date : / /

Ques- $(a+b)^*(b+ab) \equiv (a+b)^*b$

$\Leftarrow \{0^n1^n \mid \}$

- In CFL's, same language can be written in two ways-
  - conditional comparison - $\{0^m1^n \mid m=n\}$
  - Direct comparison $\{0^n1^n \mid n \geq 0\}$

for $\{ww^R \mid w \in \{0,1\}^*\}$
↳ cond^n comparison- $\{w_1w_2 \mid w_1 = w_2^R, w_1, w_2 \in \{0,1\}^*\}$

- $\{0^n1^{2n} \mid n \geq 0\}$
  ~~$S \to 00S11 \mid \epsilon$~~ $S \to 0S11 \mid \epsilon$
- $\{0^n1^{3n} \mid n \geq 0\}$
  ~~$S \to 0S11$~~ ~~$S \to 00S111 \mid \epsilon$~~
  ~~$S \to 000S1 \mid \epsilon$~~

* You can push any no of digit or symbol at a time in PDA but you can pop only one symbol at a time.

- $\{0^n1^{2n} \mid n \geq 2\}$ $S \to 00S1 \mid 001111$
- if $S \to 0^nS1^{2n} \mid 00111$
  ⇓
  $\{0^{n+2}1^{2n+3} \mid n \geq 0\}$
  then Let $n+2=K$ $n=k-2$
  $\Rightarrow \{0^K1^{2n+3} \mid n \geq 0\}$
  $\Rightarrow \{0^K1^{2K-1} \mid K \geq 2\}$

to do it normally start checking for minimal string only do for two function.

$S \to 00S1 \mid \varepsilon$

* $\{0^{2n}1^n \mid n \geq 2\}$ $S \to 00S1 \mid 000011$

* $\{0^{2n}1^{3n} \mid n \geq 0\}$ $S \to 00S111 \mid \varepsilon$

↳ it must be linear 0^{2n^2} will be CSL.

* $\{0^{2n+1}1^{3n+2} \mid n \geq 0\}$

$= \{0^{2n}011\,1^{3n} \mid n \geq 0\}$

↑ stopper, just separate the constt. powers.

$S \to 00S111 \mid 011$ · and make constant a stopper

If given

$S \to 00S111 \mid 011$

↓ solve it

$0^{2n}S1^{3n} \mid 011$

$0^{2n}0111^{3n}$

$\{0^{2n+1}1^{3n+2} \mid n \geq 0\}$

{PDA can solve linear function}

If given $\{0^m 1^n \mid m = \frac{2}{3}n\}$

↓

$3m = 2n$. Yes it is a CFL.

$\{0^m 1^n \mid m = 0 \cdot 1n\}$ - CFL

$\because m = n.$

$\{0^m 1^n \mid m = \sqrt{2}n\}$ - cannot be written as it must be integer.

$m = 2^n$ cannot done as it is not regular.

$m \geq 2n \quad m \neq 2n$

Ref • PDA can solve $\geq, \leq, <, >, \neq, =$

$m \not\geq 2n \quad m \leq 2n \quad m < 2n \quad m < 2n$

EXCELLENT

Page No. 82
Date : / /

Ques - $\{0^m 1^n \mid m \geq n\}$

~~$S \to 0S1 \mid 0 \mid \epsilon$~~

$\{0^m 1^n \mid m \leq n\}$

~~$S \to 0S1 \mid 1 \mid \epsilon$~~

$\{0^m 1^n \mid m > n\}$

~~$S \to 0S1 \mid 0$~~

$\{0^m 1^n \mid m < n\}$

~~$S \to 0S1 \mid 1$~~

Ques - $\{0^m 1^n \mid m = n\} = \{0^n 1^n \mid n \geq 0\}$

$S \to 0S1 \mid \epsilon$

$\{0^m 1^n \mid m \geq n\} = \{0^{m+n} 1^n \mid n \geq 0\}$

$S \to 0S1 \mid AS \mid \epsilon \quad A \to 0A \mid \epsilon$

$\{0^m 1^n \mid m \leq n\} = \{0^m 1^{n+m} \mid m, n \geq 0\}$

$S \to 0S1 \mid SA \mid \epsilon \quad A \to A1 \mid \epsilon$

$\{0^m 1^n \mid m > n\} = \{0^{m+n} 1^n \mid m \geq 0, n \geq 0, m \geq 1\}$

$S \to 0S1 \mid AS \mid \epsilon \quad A \to 0A \mid 0$

$\{0^m 1^n \mid m < n\} = \{0^m 1^{m+n} \mid m \geq 0, n > 0, n \geq 1\}$

$\{S \to 0S1 \mid SA \mid \epsilon \quad A \to A1 \mid 1\}$

$\{0^m 1^n \mid m \neq n\} = \{0^m 1^n \mid m > n \text{ or } n > m\}$

↳ DCFL

$S_1 \to 0S_1 1 \mid AS_1 \mid \epsilon$
$S_2 \to 0S_2 1 \mid S_2 B \mid \epsilon$
$A \to 0A \mid 0$
$B \to B1 \mid 1$
$S \to S_1 \mid S_2$

↳ Union of two language for $m > n$ and $n > m$

as ~~it~~ $m > n$ or $n > m$ is DCFL as it has only one comparison. there is no need of double comparison.

but if $m > n$ or $n > p$, it is CFL as it require two comp.

EXCELLENT

II. $\{0^m 1^n 2^p \mid m>n \text{ \& } m<p\}$ – CSL

III $\{0^m 1^n 2^p \mid m>n \text{ or } m<p\}$ – CFL

IV $\{0^m 1^n \mid m>n \text{ \& } n>m\}$ – Regular as it is $\emptyset$

Ques $S \to AS$

$S \to 000S1 \mid \varepsilon \to 0^{3n}1^n$

$A \to 0A \mid \varepsilon$ $0^m 1^n \mid m \geq 3n$

$S \to AS$

$S \to 0S1 \mid 01 \to \{0^m 1^n \mid m \geq n \geq 1\}$

$A \to 0A \mid \varepsilon$

$\{0^m 1^n \mid m \neq n\} \to$

$S \to S_1 \mid S_2$

$A \to 0A1 \mid \varepsilon$

$S_1 \to XA$

$X \to 0X \mid 0$

$S_2 \to AY$

$Y \to Y1 \mid 1$

or

$S \to XA \mid AY$

$X \to 0X \mid 0$

$Y \to 1Y \mid 1$

$A \to 0A1 \mid \varepsilon$

$S \to AX \mid AY$

$X \to 0X \mid \varepsilon$ — $m \geq n$

$Y \to 1Y \mid 1$ — $(0+1)^*$

$A \to 0A1 \mid \varepsilon$ — $m < n$

EXCELLENT

$S \to XA \mid AY$
$A \to 0A1 \mid \epsilon$
$X \to 0X \mid \epsilon$
$Y \to 1Y \mid \epsilon$

$(0+1)^*$ but Now the grammar became ambiguous.

{Ambiguity of a CFL is undecidable.}

Ternary CFL

- $\{0^m 1^n 2^p \mid m,n,p \geq 0\}$ — Regular

$S \to 0AB$
$A \to 0A \mid \epsilon$
$C \to 1C \mid \epsilon$
$B \to 2B \mid \epsilon$

- $\{0^m 1^n 2^p \mid m=n, p \geq 0\}$ — DCFL

$S \to AB$
$A \to 0A1 \mid \epsilon$
$B \to 2B \mid \epsilon$

- $\{0^m 1^n 2^n \mid m,n \geq 0\}$

$S \to AB$
$A \to 0A \mid \epsilon$
$B \to 1B2 \mid \epsilon$

— DCFL

- $\{0^m 1^n 2^m \mid m,n \geq 0\}$

~~$S \to AB$~~
$S \to 0S2 \mid 1A \mid \epsilon$
$A \to 1A \mid \epsilon$

— DCFL

- $\{0^m 1^n 2^m \mid m \geq 1, n \geq 2\}$

$S \to 0S1 \mid 0A1$
$A \to 1A \mid 11$

Date : / /

$\{0^m 1^n 2^p \mid m=n \text{ or } m=p\}$ CFL but not DCFL

as one variable comp. with or

$S \to S_1 \mid S_2$

$S_1 \to 0S_1 1 \mid \epsilon$

$S_1 \to AB$

$A \to 0A1 \mid \epsilon$

$B \to 2B \mid \epsilon$

$S_2 \to CB$

$C \to 0C2 \mid D$

$D \to 1D \mid \epsilon$

$\Rightarrow$

$S \to S_1 \mid S_2$ — for 'OR'

$S_1 \to AB$

$A \to 0A1 \mid \epsilon$ — $m=n$

$B \to 2B \mid \epsilon$

$S_2 \to 0S_2 2 \mid C$

$C \to 1C \mid \epsilon$ — $m=p$

$\Downarrow$

$S \to AB \mid S_2$

$\{\{0^n 1^n\} \cup \{0^n 1^{2n}\} \mid n \geq 0\}$

$\Downarrow$

$\{0^m 1^n \mid m=n \text{ or } m=2n\}$ — Not a DCFL but a CFL

$S \to S_1 \mid S_2$

$S_1 \to 0S_1 1 \mid \epsilon$

$S_2 \to 0S_2 11 \mid \epsilon$

{CFGs has power equivalent to NPDA}

$\{0^m 1^n \mid m \leq n \leq 2m\}$ → it will accept 0011, 00111, 001111 as well

$\Downarrow$

$\{0^m 1^n \mid m \leq n \text{ or } n \leq 2m\}$

$S \to S_1 \mid S_2$

$S_1 \to 0S_1 1 \mid X$

$X \to 1X \mid \epsilon$

$S_2 \to$

$\{S \to 0S1 \mid 0S11 \mid \epsilon\}$

EXCELLENT

- $\{0^m 1^n \mid m \le n \le 3m\}$

$\{S \to 0S1 \mid 0S11 \mid 0S111 \mid \epsilon\}$

You have to take all 1 to 3 because you can't generate any other from the given

$\{0^m 1^n \mid m \le n \le 4m\}$

$\{S \to 0S1 \mid 0S11 \mid 0S111 \mid 0S1111 \mid \epsilon\}$

Ques - $\{\omega \mid n_0(\omega) = n_1(\omega)\}$

$\{0^n 1^n \mid n \ge 0\}$ $\{\omega \mid n_0(\omega) = n_1(\omega)\}$

$\{0^n 1^n \mid n \ge 1\}$ $\{\omega \mid n_0(\omega) = n_1(\omega) \ge 1\}$

$\{0^n 1^{2n} \mid n \ge 0\}$ ~~$\{\omega \mid n_0(\omega) = 0 n_1(\omega)\}$~~

$\{\omega \mid n_1(\omega) = 2n_0(\omega)\}$

$\{0^n 1^{2n+3} \mid n \ge 0\}$ $\{\omega \mid n_1(\omega) = 2n_0(\omega) + 3\}$

$\{0^n 1^m 2^p \mid m = n, p \ge 0\}$ $\{\omega \mid n_0(\omega) = n_1(\omega),\ n_2(\omega) \ge 0\}$

$\{0^m 1^n 2^p \mid m = n \text{ or } m = p\}$ $\{\omega \mid n_0(\omega) = n_1(\omega), \text{ or } n_0(\omega) = n_2(\omega)\}$

$\{0^m 1^n \text{ or } 0^n 1^{2n}\}$ $\{\omega \mid n_0 = n_1 \text{ or } n_2 = 2n_0\}$

Date : / /

- $\{w \mid n_0(w) = n_1(w)\}$

if G: $S \to 0S1 \mid 1S0 \mid \epsilon$

↓

$\{0110 \notin G\}$

↳ so we can not use it.

~~$S \to 0S1 \mid 1S0 \mid 00S11 \mid 01S \mid 10$~~

~~$S \to 0S1 \mid 1S0 \mid 01S10$~~

this grammar will not generate

0110

1001

001100

∴ the o/p grammar is

$S \to 0S1 \mid 1S0 \mid SS \mid \epsilon$ ↳ to allow the production side by side.

for equal no. of zero and 1:

OR

$\{S \to 0S1S \mid 1S0S \mid \epsilon\}$ — equivalent to each other

OR

$\{S \to S0S1 \mid S1S0 \mid \epsilon\}$

but if $\{S \to 0S1S \mid S1S0 \mid \epsilon$

↓ it will miss the string ending with 0 & start with 1

1001

- $\{w \mid n_0(w) = 2n_1\}$

- do min. combination possible-

$\{S \to 0S0S1 \mid 0S1S0 \mid 1S0S0 \mid SS \mid \epsilon\}$

001 010 100 ↓ for side by side

↳ is putted to allow all 001100

EXCELLE

$\{ S \to SOS1SO \mid S1SOSO \mid SOSOS1 \mid \epsilon \}$ OR

$\{ S \to OS1SOS \mid 1SOSOS \mid OSOS1S \mid \epsilon \}$ OR

- $\{ w \mid n_0(w) = n_1(w) \}$ for $0^n 1^n$

$S \to OS1 \mid 1SO \mid SS \mid \epsilon$ $S \to OS1 \mid \epsilon$

counting comparisons are useful for calculation of parenthesized expression for matching paranthesis.
Matching of paranthesis comes with every looping structure.

$0^n 1^n$ → balanced

allow - $(^n)^n$ $(((\ )))$

$n_0(w) = n_1(w)$ → no restriction.
it will allow everything
( )) (

but we want left one before right one bracket so we need a grammar b/w these two grammar.

So →

$S \to (S) \mid SS \mid \epsilon$
↳ properly balanced paranthesis structure.

$S \to (S) \mid \epsilon \Rightarrow \{ \}^{n\ n}$

$S \to (S) \mid)S(\mid SS \mid \epsilon \to (()())))(()$ everything

$S \to (S) \mid SS \mid \epsilon \to (()())$ - correct properly paranthesized.

EXCELLENT

Page No.
Date : / /

G → S → (S) | ( ) | SS → (( )( )) → properly balanced but null never came.

- Instead of null it will give ( )

1. (( ))
2. ((( )))
3. ((( )( ))) - properly balanced.
4. (( )( ))

for balanced $\{w \mid n_L(w) = n_R(w)\}$

for properly balanced - $\{w \mid n_L(w) = n_R(w) \text{ & } n_L(v) \geq n_R(v)$ v is any prefix of w$\}$

as (( ))
→ prefix condn satisfied.

)) ((
→ not satisfied prefix condn
for prefix)) (L < R so it is not properly balanced.

Grammar for balanced one without null string.

$\{S \to (S) \mid)S(\mid SS \mid ( ) \mid)(\}$

$\Downarrow$

$\{w \mid n_L(w) = n_R(w) \text{ & } w \notin \epsilon\}$

EXCELL

TIT family

Even palindrome - $S \to 0S0 | 1S1 | \epsilon$ → CFL
$\{ww^R | w \in (0,1)^*\}$.

Odd palindrome - $S \to 0S0 | 1S1 | 0 | 1$
$\{wxw^R | w \in (0,1)^* x \in (0,1)\}$

All palindrome $S \to 0S0 | 1S1 | 0 | 1 | \epsilon$ → CFL

$\{ww^R \cup wxw^R\}$ → CFL

OR

$\{w | w = w^R\}$

palindrome -

$S \to 0S0 | 1S1 | \#$ → DCFL

$L(G) = \{w\#w^R | w \in (0,1)^*\}$

24-Apr

Page No.
Date : / /

Workbook chapter-1

1) d)
2) c
3) b
4) c
5) d
6) c
7) b
8) a
9) (b)
10. (b) (2×3) states
11. (d)
12. (b)
13 (b)
14 (c)
15) (a)
16 (a)
17) (d)
18) (c)
19) (a)
20) (d)
21) (d) → (c) is not there as $L(R_1) \cap L(R_2) = \phi$
22) (d)
23) (c)
24) (a)
25) (d)
26) (b)
27) (a)
28) (a)
29) (d)
30) b
31) b
32) (b)

33. (d)
34. (d)
35. (d)
36. (a)
37. (d)
38. (c)
39. (d)
40. (d)
41. (a)
42. (d) (start checking with smaller strings.
43. ~~(b) it is~~ (c)
44. (a)
45 (d) ~~it is true~~
46. (d)
47 (b)
48. (d)
49. (d)
50 (c)

51 (b)
52 (b)
53 (c)
54. b
55. c
56 b
57 a
58. b
59. c
60 b
61 a

62) c
63) d
64) a
65) a
66) b
67) d
68) a
69) b
70) a
71) d
72) c
73) c
74) a
75) (d)
76) (e)
77) (c)
78) (d)
79) (a)

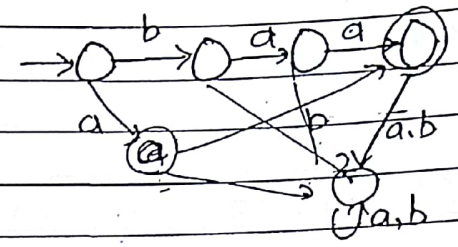
80) (c)
81) (c)
82) (a)
83) (c)
84) (e)
85) (d)
86) (c)
87) (d)
88) (d)
89) (d)
90) (d)

91 a
92 d
93) a
94) b
95) d
96) d
97) e
98) d
99) d
100) a

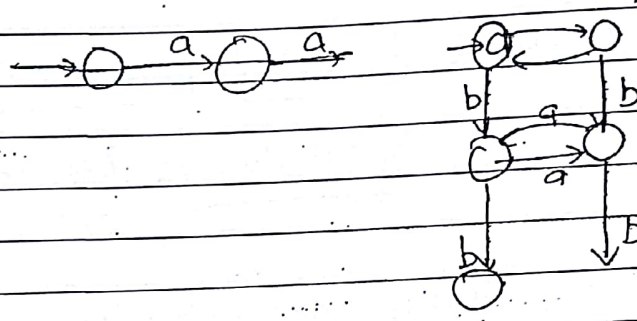
101) c
102) d
103) b
104) c
105 e
106 (a)
107 (d)
108 (c)
109 c
110. d
111 d
112 b
113 b
114
115
116
117
118
119
120.

EXCEL

$L = \{ba, ab, baa\}$



no of a less than 2 or
no of a less than 4
↓ union
no of a less than 4
as $A \cup B = A$
If $B \subseteq A$



In an ordinary DFA, one way to accept & one way to reject
2 DFA, one way to accept but two ways to reject a string

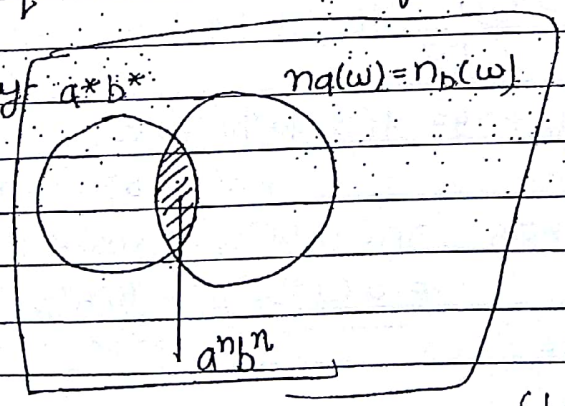
- non final state
- infinite loop

if a machine hangs, string is considered to be rejected

2-NFA to reject a string
- ~~Non final state~~
- DC
- Hang

if $\delta^*(q_0, 10100) = \infty$ (hang)

Basically a^*b^* $n_a(w) = n_b(w)$



if given
- $R_1 = a^*b^*$
- $R_2 = n_a(w) = n_b(w)$
- $R_1 \cap R_2 = a^n b^n$

$L_1 = a^n b^n$

$L_2 = n_a(w) = n_b(w)$

- $L_1 \cap L_2 = a^n b^n$
- $L_1 \cup L_2 = L_2$

EXCELLENT

Page No.
Date : / /

if 3a x 2 & 4a x 4
↓ then n is a is multiple of 2
as $A \cap B = B$ if $B \subseteq A$

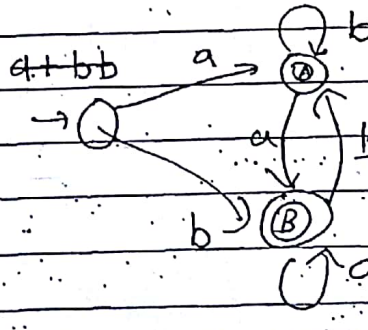
Ques - $\varepsilon + (11^*0)(11^*0+0)^*$
$= (11^*0+0)^*$ as $(\varepsilon + r_1 r_1^*) = r_1^*$
$= ((11^*+\varepsilon)0)^*$
$= (1^*0)^*$ → {ε, 0, 10, 110, ...}
$= (0^*1^*)^*$

↓ {1101 does not belong to it}

Ques $(a+ba)^*(a+ba)$ {as if it is ending with a its always having b before a as we are provided with $(a+b)^*$}
$= (a+ba)^*a$
$= (a+b)^*a$

When a re is given, check for its minimization.

(a+bb)



$r_A = a(b+aa^*b)^* + b(a+bb^*a)^*bb^*$

$r_B = a(b+aa^*b)^*aa^* + b(a+bb^*a)^*$

$r_A + r_B = a(b+aa^*b)^*(\varepsilon+aa^*) + b(a+bb^*a)^*(\varepsilon+bb^*)$

{$r_A + r_B = a(b+aa^*b)^*a^* + b(a+bb^*a)^*b^*$
$= a(a^*b)^*a^* + b(b^*a)^*b^*$}

EXCELLENT

if $a^*b^*a(a+b)^* \equiv (a+b)^*$ - false
it has (a)

$a^*b^*a(a+b)^* \equiv a(a+b)^*$ - false it force it to start with a

1. $a^*a\,b^*(a+b)^* = a(a+b)^*$
2. $c^*a\,b^*(a+b)^* = c^*a(a+b)^*$
3. $a^*a\,c^*(a+b)^* = a^*a(c^*)(a+b)^*$
4. $ab^*a^*(a+b)^* = a(a+b)^*$
5. $(a^*b^*)^* = (b^*a^*)^*$
6. $(0^*1^*2^*)^* = (0+1+2)^*$

if L_1 = Non-regular, $L_3 = L_1 L_2$ = Regular L_2 = ?
↓ Regular

* $L_1 = \{a^i b^j c \mid i,j \geq 0\}$
$L_2 = \{a\,b^i\,c^j \mid i,j \geq 0\}$

$L_1 \cap L_2 = \{ab^i c \mid i \geq 0\}$

allow only one c

b is independent

allow only one a

$L_1 \cap L_2$

* $L_1 = (ab)^* \cap (a^*b^*)$
$= \{\epsilon, ab\}$
$\therefore |L_1| = 2$

EXCELLENT

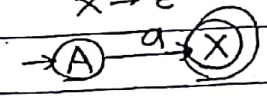
Conversion of Grammar to Machine=

- to convert Right Linear RG to Machine

$V \rightarrow T^* + T^*V$

for T^* – for $A \rightarrow \epsilon$

for $A \rightarrow ax$ put an arbitrary state, $X \rightarrow \epsilon$



$A \rightarrow a.bc...$

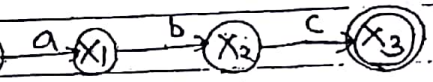
$\Downarrow$

$A \rightarrow aX_1$

$X_1 \rightarrow bX_2$

$X_2 \rightarrow cX_3$

$X_3 \rightarrow \epsilon$



for T^*V –

for ~~$S \rightarrow aB$~~

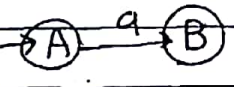
~~$B \rightarrow bc$~~

- $A \rightarrow B$ (Unit Production)

as $\{A \rightarrow \epsilon . B\}$

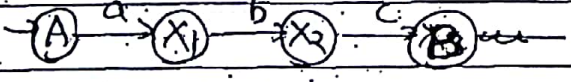
Every unit production corresponds to null move.

- $A \rightarrow aB$



Every null production corresponding to final state.

- $A \rightarrow abcB$ $A \rightarrow aX_1$, $X_1 \rightarrow bX_2$ $X_2 \rightarrow cB$



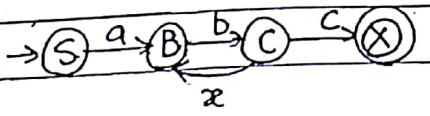
Ques 38- $S \to aB$

$B \to bC$

$C \to xB$

$C \to C \cdot \to C \to Cx$

$X \to \epsilon$



$a(bx)^*bc$

or

$ab(xb)^*bc$

as $\{(pq)^*p = p(qp)^*\}$

- $\Re + \phi = \Re$
- $\Re + \epsilon = \Re + \epsilon$ { if $\Re + \epsilon = \Re$ if $\Re$ contains ϵ }
- $\Re \cdot \epsilon = \Re$
- $\Re \cdot \phi = \phi$

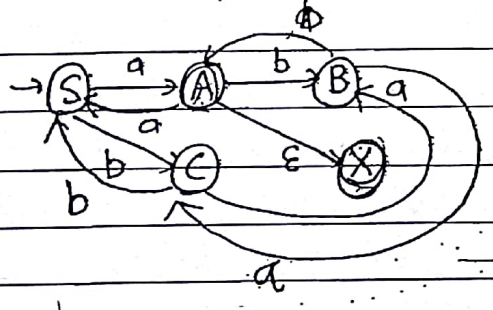
Ques- "for RLRG by to draw the machine."

$S \to aA \mid bC$

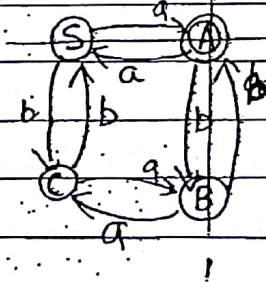
$A \to aS \mid bB \mid \epsilon$

$B \to bA \mid aC$

$C \to aB \mid bS$

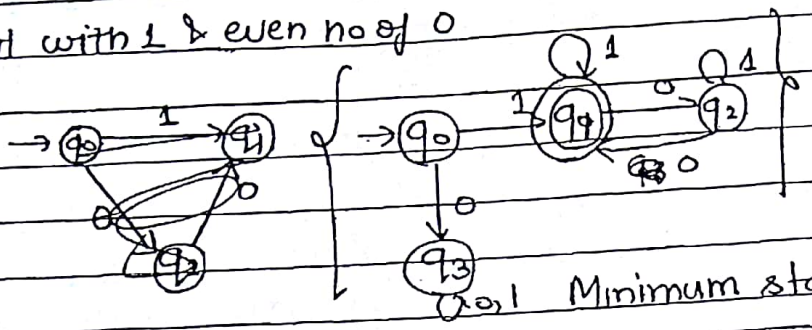


$a^* + b^* a^* +$



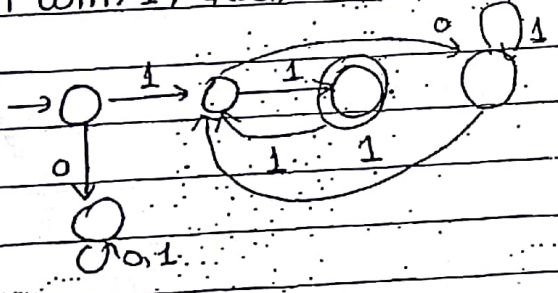
~~odd~~ even b odd a

w/ start with 1 & even no of 0



Minimum state for DFA = 4
NFA = 3

start with 1 & $d(w) \bmod 3 = 0$



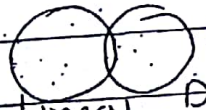
Ques
53. 3 possible answers -

$a(abb)^*ab$
$ba(bab)^*$

1. $a(abb)^*ab + ba(bab)^*$ loop at A
2. $aa(bba)^*b + ba(bab)^*$ loop at B
3. $aab(bab)^* + ba(bab)^*$ directly at C.

all are equiv ↓ bcoz loop can be resolved at 3 point A, B, C.

RL = LL < LINEAR < CFL < CSL
RL = LL < DCFL < CFL < CSL



Both P & NP problem are in REC

EXCELLENT

Page No. 90
Date : / /

Properties of Myhill Nerode Class-

1. ~~every myhill~~ $|E_i| \geq 1$ i.e each myhill-nerode ~~can~~ must have atleast one string.
2. Each myhill-nerode class must be unique.
3. ~~Each~~ two equivalence classes must have intersection as ϕ

$E_i \cap E_j = \phi$

(it must be disjoint)

because of min. dfa as in this, one string can perform single function only.

4. the union of all the equivalence class must be Σ^* ($\cup E_i = \Sigma^*$)

because in NFA, no Dead configuration are there.

Ques $0^*1^*2^* \cap 1^*0^*2^*$

(a) 2^*

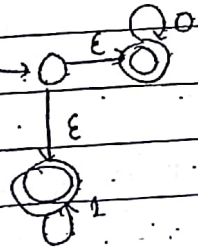
(b) $(0^*+1^*)2^*$ $\quad [0^*1^* \cap 1^*0^* = 0^*1^*]$

(c) ϕ

(d) $(0+1+2)^*$

$0^+1^*2^* \cap 0^*1^+0^+2^* = \phi$

0^*+1^* → Min. NFA

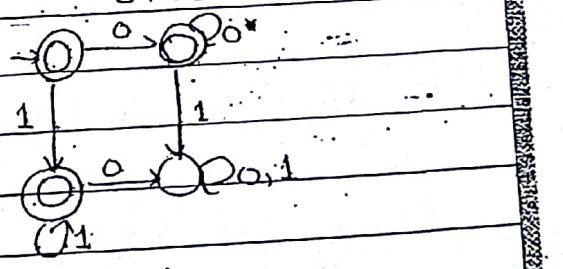


Min. DFA

0^*+1^*

$\varepsilon+00^*+\varepsilon+11^*$

$\varepsilon+00^*+11^*$



To generate a regular language, we only need $\{V \Rightarrow TV/\varepsilon\}$ because for RG, there is dfa if grammar are constructed from dfa. It will only of the form $\{V \rightarrow TV/\varepsilon\}$

- Unit production are luxury in Regular Grammar

EXCELLENT

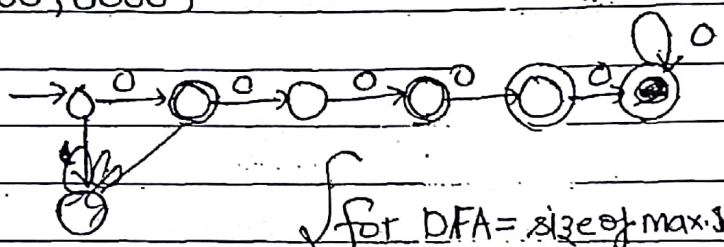
The class of Grammar like

$V \to T^* + \varnothing$ and

$V \to TV \mid \varepsilon$ have same powers.

Unary Design

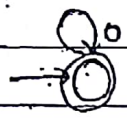
1. {0, 000, 0000}



for DFA = size of max. string + 2 (1 for state, 1 for trap)

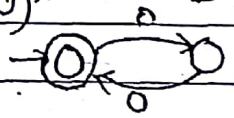
NFA = |w max| + 1

2. 0^*

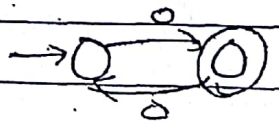


– Minimal DFA = 1

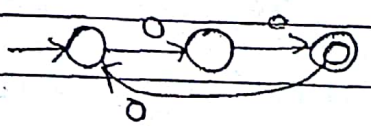
3. $(00)^*$ – even no of zero



4. $(00)^*0$ – odd no of zero



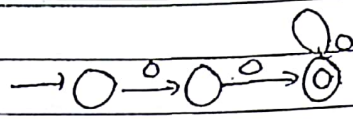
5) $(000)^*00$ – $n_0 \bmod 3 = 2$



$= 0(000)^*0$

$= 00(000)^*$

4. atleast 2 zero's



Ques - $\{0^{7n+4} \mid n \geq 0\}$ - how Many states in min. DFA ?

Minimum state = 7 as mod 7 machine

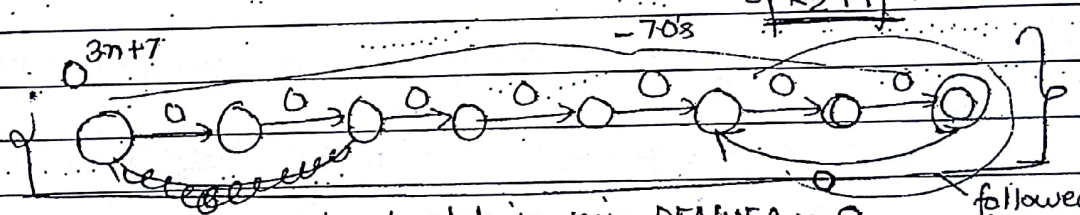
Ques $\{0^{k_1 n + k_2} \mid n \geq 0\}$

Case I - $k_1 > k_2$

then no of minimal state = k_1 as it become a mod k_1 machine

II - $k_2 \geq k_1$

then no of minimal state in M-DFA or NFA



No of state in min DFA/NFA = 8

Ques $\{0^{3n+7} \mid n > 0\}$ if $n > 1$

$k_2 + 2k_1 + 1$

Case II $k_2 > k_1$ & $n > 0$

No of states - $k_2 + k_1 + 1$ in case when $n > 0$

or $\{|\text{minimum string}| + 1 = \text{No of states}\}$

Here $n = 1$

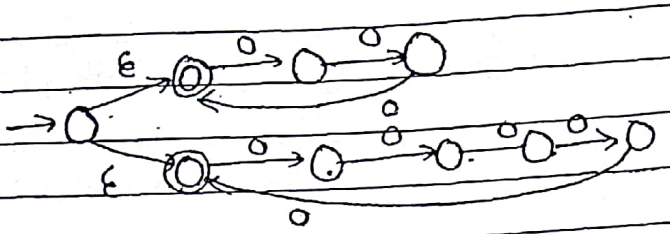
minimum string = 10

∴ No Of state = 11

EXCELLENT

Page No.
Date : / /

Ques $(000)^* + (00000)^*$
$0^{3n} \cup 0^{5n}$

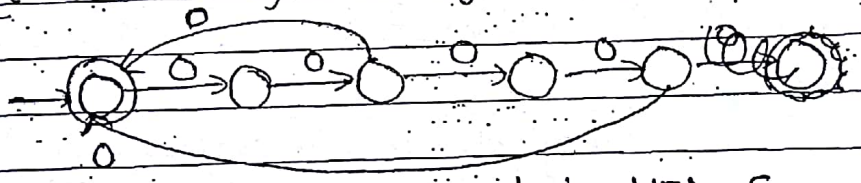


it is difficult to design DFA

No of states = in min NFA = 9
= 3+5+1

if $(000)^* + (000000)^*$
A B
$B \subset A = (000)^*$ No of states = 3

Ques - $(000 + 00000)^*$ No of state in Min. state DFA



↓ Min. NFA = 5

- Take the design of larger string
- Take to accumulate in it

(if the string are not multiple of each other smaller one will control the loop)

but if $(000 + 000000)$
↳ so (000) = 3 states are required.

EXCELLENT

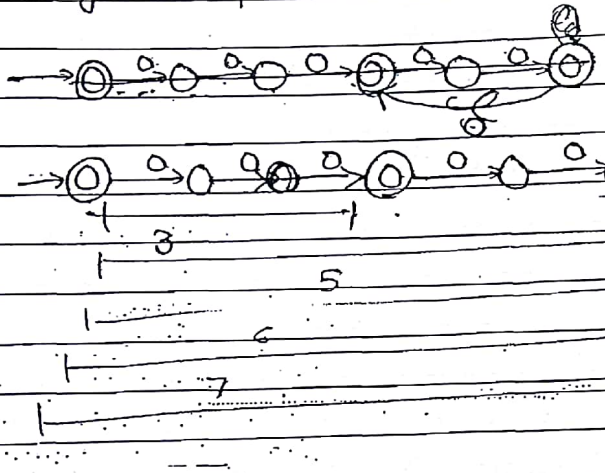
Page No. 09
Date : / /

Minimal DFA - Method-

if ε belong to language, accept it.

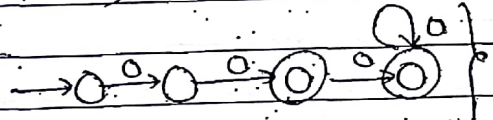
& goes on increasing the strings accept it if belong to language

but it goes to infinite.



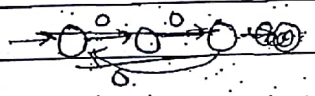
8 and so on after 8 no need to put more bcoz each string is accepted

$(00+000)^*$

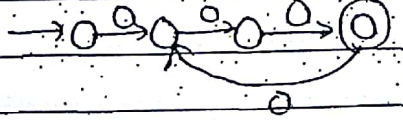


↳ after 3 it accept everything.

Ques - $\{0^{nk} \mid k>0, n \text{ is fixed integer}\}$ no of states in minimal FA = ?



if $n=3$ it 0^{3k} mod machine with residue 0.

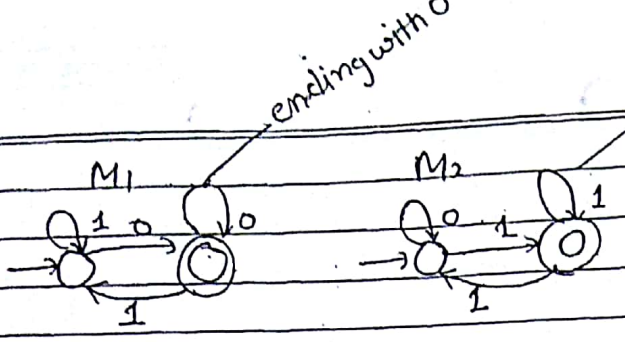


it would be n+1

if it is $k \geq 0$ then it would be n

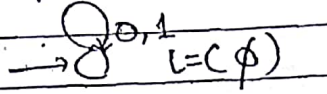
↳ as it also accept null.

Page No.
Date : / /

Ques- 

$L(M_1) \cap L(M_2) = \phi$

, so $M_1 \cap M_2$ require 1 state:



Sol (a) as it can result to less breakdown.

(b) more breakdown.

(c)

(d) $L = \{1010\}$

$h(a) = 00 \quad h(b) = 11$

$h^{-1}(L) = \phi$

$h(h^{-1}(L)) = \phi$

Q. $L_1 \in (ab + ba)^*$

$h(0) = aa$

$h(1) = bb$

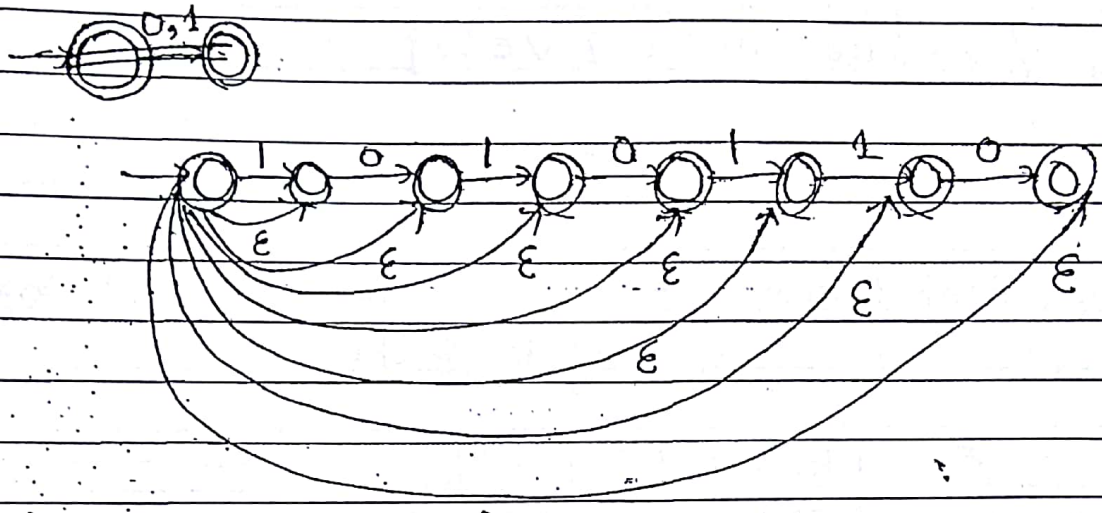
$h^{-1}(L_1) = \{\} = \{1\}$ for $*$. as 1 need no breakdown

$h(L_2) = \{bbaa + aabb\}^*$

$h^{-1}(L_3) = \{0^n 1^m \mid m \geq 0\}$

EXCELLENT

No of substring $= \frac{7(7+1)}{2} + 1 = 29$



Here, firstly all states are final state, so it can accept every prefix & then add ε move from initial state to starting & every state so that they can act as initial state

- So n length string, no of states for accepting every string $= n+1$

for n length string, no of state for accepting every prefix $= n+1$

for AP → find a & d

$a+(n-1)d$ & then find the equation as

If $a=3$ $d=2$ ...

$2n+1$

0

EXCELLENT

Page No.
Date : / /

Right Quotient & Left Quotient

Right Quotient - L_1/L_2

Right Quotient of L_1 with L_2

L_2/L_1 - Left Quotient of L_1 with L_2

$L_1/L_2 = \{u \mid uv \in L_1 \& v \in L_2\}$

$L_2/L_1 = \{v \mid uv \in L_1 \& u \in L_2\}$

for Quotient, Closure properties are only applicable for Regular & CFL.

	REGULAR	CFL
INIT(L)	✓	✓
L_1/L_2 L_2/L_1	✓	✓
CYCLE(L)	✓	✓
Suffix(L)	✓	✓

L_1/L_2 → if L_1 = regular

L_2 = CFL

$\therefore L_1/L_2$ → CFL

INIT(L) ≤ Initial or prefix of L → both are closed

SUFFIX(L) → last of L

CYCLE(L) = If L = 1010 • put each rotation of given language

CYCLE of L = {0101, 1010}

If L = {1234}

CYCLE of L = {1234, 2341, 3412, 4321}

EXCELLE

1. $L/\Sigma^* =$ Prefix of L
   $\Sigma^*/L =$ Suffix of L
   i.e right quotient of L with Σ^*, it will produce prefix of L

Example $L = \{ab, baa\}$
INIT of $L = \{\varepsilon, a, ab, b, ba, baa\}$

INIT of L contains all the prefix possible of the given string in language

$|INIT(L)| = 6$

$Suffix(L) = \{\varepsilon, baa, b, aab, ab, ba, a\}$

for Infinite language

- {INIT(L) will contains L, ε (always)
  INIT(L) will always contain ε, L}

for a^* → INIT(L) = a^* take everything possible

L	INIT(L)	Suffix(L)
a^*	a^*	a^*
$(ab)^*$ (as it can have aba as prefix)	$(ab)^* + (ab)^*a$	$(ab)^* + b(ab)^*$
a^*b^*	a^*b^*	a^*b^*
$a^nb^n \mid n>0$	$a^mb^n \mid m \geq n$	$a^mb^n \mid n \geq m$
$n_a = n_b$	$(a+b)^*$	$(a+b)^*$

$a^mb^n \mid m \geq n$ as it can have equal a & b as well a followed by b

↓ as no equality is required in prefix

$\{a^*/a = a^*\}$ $a/a = \{a, aa, aaa, \ldots\}$ $\{\epsilon, a, aa, aaa, \ldots\}$

Page No.
Date: / /

Ques $L_1 = \{100, 0100, 11\}$

$L_2 = \{1, 01\}$

L_1/L_2 = take every string of L_1 & divide it by L_2.

$\frac{100}{1} \times \quad \frac{100}{01} \times \quad \frac{0100}{1} \times \quad \frac{0100}{01} \quad \frac{11}{1} \checkmark \quad \frac{0111}{01}$

$L_1/L_2 = \{1\}$

$L_2/L_1 = \{00, 1\}$ as $\frac{100}{1}$ $\frac{1}{1}$

$L_1/L_2 \neq L_2/L_1$

Ques $L_1/L_2 = 0^*1/1^*0$ expand L_2

$= 0^*1\{0, 10, 110, \ldots\}$

$= \phi$

$L_2/L_1 = 1^*0/0^*1 = \phi$

$L_1 = a^*ba^*$

$L_2 = ba^*$

$L_1/L_2 = a^*ba^*/\{a, ba, baa, baaa, \ldots\}$

$\frac{a^*ba^*}{a} \times = a^*ba^*$

$\frac{a^*ba^*}{ba^*} = a^*$ as baa can never happen in end.

$L_1/L_2 = \{a^*ba^* + a^*\}$

$\{ L/\varepsilon = L \}$

$L_1 = a^*b$

$L_2 = b^*$

$L_1/L_2 = a^*b/\{\varepsilon, b, bb, \ldots\}$

$\frac{a^*b}{\varepsilon} = a^*b \qquad \frac{a^*b}{b} = a^* \qquad \frac{a^*b}{bb} =$ Not possible

$L_1/L_2 = \{a^*b + a^*\}$

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

~~baa*/a~~

$a^*/a = a^*$ — $\{a, aa, aaa, \ldots\}/a = a^*$

$baa^*/a = a^*$

$aaa^*/a = aa^*$ — $\{aa, aaaaaaa, \ldots\}/a = aa\{\varepsilon, a, aa, aaa\} = aa^*$

if L/Σ^* → prefix is obtained as Σ^* contains all the string.

Let $\{aba, baa\}/\Sigma^*$

$= \{a, ab, aba, \varepsilon, b, ba, baa\}$

↓ because Σ^* has each string for division.

$\{ |L/\Sigma^*| = |INIT(L)| = $ no of distinct prefix in L $\}$

	REGULAR	CFL
MIN	✓	✗
MAX	✓	X
HALF	✓	X → Half(1010) = 10)
ALT	✓	X if L = {10101, 100} Half(L) = {101} ↓ take only even

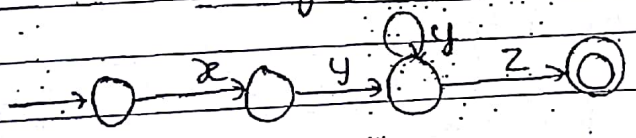
Page No.
Date : / /

Pumping Lemma for Regular Languages-

L is regular ⇒ L is satisfying pumping lemma for regular languages

There exist some language satisfy PL for RL which are not regular ⇒ True.
if L dont satisfy PL, then L is not Regular - True.

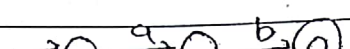
if L is regular then it has FA (M) with n states, let it accept $|w| \geq n$
then PL says
$w = xyz$ & $|y| \geq 1$
$\therefore xyz \in L(M)$
if it accepts xyz it must accept xy^*z
⇓
$xy^*z \in (L(M))$
i.e y can be pumped.



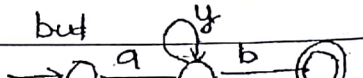
this the major weakness of FA that it cannot control the loops.

the pumping lemma says that if nfa accept a string of length having atleast equal to no of states (n) $|w|$
i.e $|w| \geq n$ then language is infinite
as it says xy^iz also belong to L(M)

proof - An state NFA, with loop can accept n-1 length string only



but



for $|w| \geq n$.

Applying PL

Ques - 5 state fa accept a string of length 4.
the fa is
(a) finite
(b) infinite
(c) finite or infinite (bcoz from given info, we cannot comment whether it has loop or not)

- 5 state fa accept a string of length 5 or 6
the language is
1. finite
2. infinite

- 5 state fa accepts a string of length 4 only
1) finite

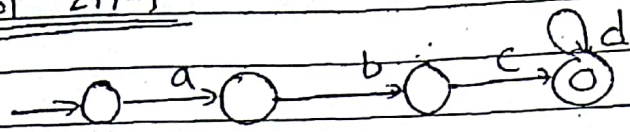
* A FA to be infinite of n state, it must accept a string of length n (atleast).

If n state FA is accepting infinite language it surely accept the string b/w length n & 2n-1

$|w| \geq n$ $\quad$ $|w| < n$

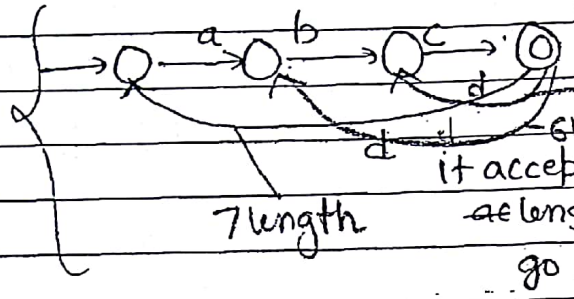
Date : / /

proof for 2n-1



4 length string

for 2n-1



7 length

it accept 6 length 7 string but you cannot go beyond it

for $|w| = 0$ & $n = 4$

5 length $n \leq |w| \leq 2n-1$

6 length $4 \leq |w| \leq 7$

I atleast one string above $n \geq n$

II atmost one string of $2n-1 \leq 2n-1$

If it is an infinite language

To check whether is a lang is infinite,
if any string in the range $n \leq |w| \leq 2n-1$
then the language is infinite otherwise finite.

The transition graph of infinite language surely contains a cycle.

if fa contains w of atmost 2n-1. is it ~~finite or~~ infinite?
No because it also cover the range below n. so it is possible if & only if
$n < |w| \leq 2n-1$

EXCELLENT

Myhill-Nerode Relation- It is applicable on a Minimal DFA.

$\{ u R v \Leftrightarrow \delta^*(q_0, u) = \delta^*(q_0, v) \}$ then u & v are Related.

- Reflexive uRu
- Symmetric uRv ⇒ vRu
- Transitive uRv, vRw ⇒ uRw

if u & v are related, they belong to same equivalance class in a given fa. (i.e which get pushed to same state)

No of equivalence class = No of states in minimal dfa.

97 (b) Union of some of equivalence class.
because in DFA if U of all class = Σ^*
but L = UF state

Myhill Nerode is right invariant equivalence relation as
if uRv ⇒ $\delta^*(q_0, u) = \delta^*(q_0, v)$
∴ uz ≡ vz (Right Variant)

index of relation = no of classes = it has to be finite as no of state in FA can be finite only.

- For a non regular language, No of class is infinite.

Ques $1^* \neq 1^*01^* + \_\_\_\_ = (0+1)^*$
$\{ = 1^*01^* 0(1+0)^* \}$

Ques I $(000)^*(\epsilon+0)$
II $(000)^*(\epsilon+0+00)$ II & IV are equivalent.
III $(000)^*$
IV $(0)^*$

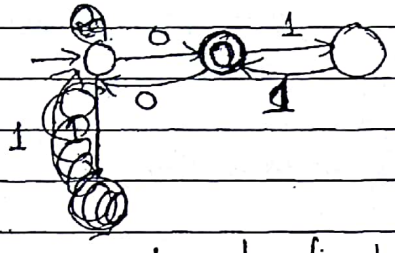
Page No.
Date : / /

105

Regular $0^n 1^m$: $n+m$ is odd

case 1- $0+E$ Odd no of 0 followed by even no of 1

$E+0$



(c) p_q by finding its complement

d) $0^p 1^p$ $p+q=10$ — R
$p \cdot q = 10$ — R
$p-q \neq 0$ — NR $(\leq, \geq, =, >, <, \neq)$
$p/q = 10$ — NR

these are regular with all the operators

↳ by taking complement.

for $\{0^p 1^q ; p \geq 0, q \geq 0, pq \geq 4\}$

↓ Do complement of machine..

$pq \leq 4$

Let $L = \{0^p 1^p ; p+q \geq 10\}$

the complement is

$\bar{L} = \{0^p 1^q ; p+q < 10\}$ ⇒ all the 1 must not follow zero & $p+q<9$

$(0+1)^* 10 (0+1)^* + \{0^p 1^q \nmid p+q<9\}$

↓ for all the $\bar{L}$ not following 0 ∴ $\bar{L}$ is regular then L is regular..

⇓ for less than 9

$0^4 1^1 + 0^3 1^1 + 0^2 1$

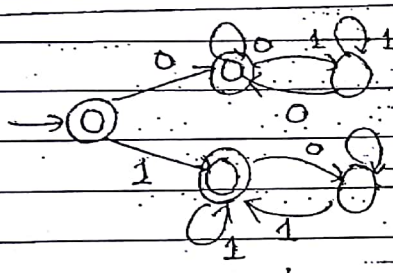
$0^4 1^1 + 0^2 1^2 + 0^1 1^4$ $\quad pq = 4$

$(0^4 0^* 1^1 1^* + 0^2 0^* 1^2 1^* + 0^1 0^* 1^4 1^*)$ $\quad pq \geq 4$

but only $(0^* 1^*)$

* In Regular language, it is not allow
  $T^* V T^* V$
  as it allow comparison)
  $T^* V / V T^*$ - it also allow comparison

Ques 12] -



$\Rightarrow \{ n_{01}(\omega) = n_{10}(\omega) \}$

It is basically not possible to compare no of 1 & 0. because it require infinite M/M (guideline violation)

but possible to comp. 10 & 01 bit require finite M/M

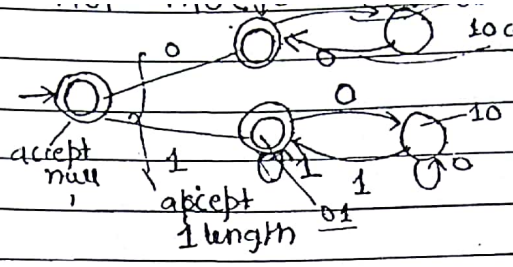
Ques. $n_{000}(\omega) = n_{111}(\omega)$ → NR as we can have infinite 000

$n_{011}(\omega) = n_{110}(\omega)$ - Regular but we cannot have the even

011011 (equal)

agar 011 ayega to possible nhi hai ki fir se 011 aaye kyuki it will lead to 110 110 so it is not possible to have infinite memory case.

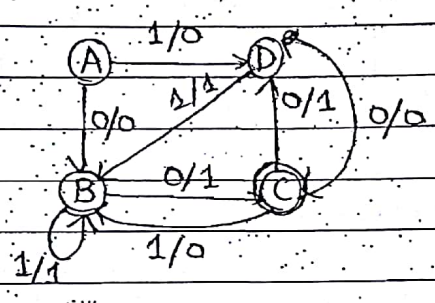
for $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ comparison (only two states are required)



No of states = 5

Unknown initial state –

13?



to find out starting state to reach C?

~~00 for A~~

~~0 for B~~

(a) 01 for A → O/P B

10 for A → O/P C

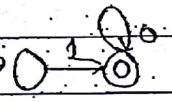
B → O/P C

D → O/P C

} Shortest path must work from each node A, B, D

139 – 1, 2, 4, 8, ... 2^n

1, 10, 100, 1000, ... 100...



Re = {10*}

1, 3, 9, 27, ... 3^n is not Regular in binary but in ternary

1, 10, 100, 100 ~~1, k, k² ... kⁿ~~

1 3 9 ↓ it is Regular

30-Apr

Page No.
Date :

∴ we can say that $1, k, k^2, \ldots k^n$ will be regular in base k alphabet set.

Ques - if a is divisible by 6 & a is divisible by 8

147

$\therefore a \div LCM(6,8)$

$\therefore a \div 24$

$\therefore$ 24 states

- $a \div 2$ & $a \div 8$
  $\therefore a \div 8$
- $a \div 2$ or $a \div 8$.
  $\therefore a \div 2$

No of final states $= 8 + 6 - 1 = 13$

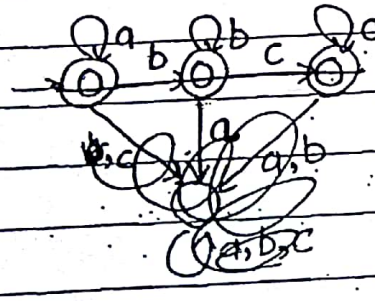
149 $((0+1)(0+1)(0+1))^*$

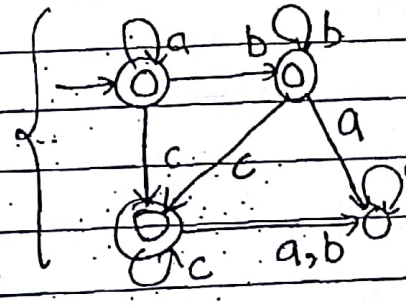
$((b^*ab^*ab^*ab^*)^* + b^*)^*$ - No of a divisible by 3

150 - both c & d

(d) $0^{m+n} + 1^{m+n} \Rightarrow 0^* + 1^*$

$a^*b^*c^*$





$= a^*b^*c^*$

No of states = 4

$\{ww^Rw^R \mid w \in (0,1)^*\}$ - CSL

$\{ww_1^Rw_2w_2^R$ — CFL but not DCFL

$ww^R\#w_2w_2^R$ — DCFL

$w_1\#w_2^R\#w_1^R$ CSL

$w_1\#w_1^R\#w_2\#w_2^R$ DCFL

EXCELLENT

{CFL can't do alternate comparison}

Page No. 100
Date : / /

$w_1 \# w_2 \# w_2^R \# w_1^R \rightarrow$ CFL

$w_1 \# w_2 \# w_1^R \# w_2^R \rightarrow$ CSL

{$w_1 w_1^R w_2 w_2^R$ ..... infinite matching - Not a CFL}

{NPDA can take finite no of copies ~~both~~ only.}

because

$\{a^n b^{2n}\} \cup \{a^n b^{3n}\}$ .....

↓ it requires infinite no of copies to be maintained.

↓

$\delta(q_0, a) = \{(q_0, a)\ (q_0, aa)\ (q_0, aaa) \ldots\}$

X it is invalid as it must be finite

$\{a^m b^n \mid m = kn \quad k \in z^+\}$ - it is not a CFL as

$m = n \quad m = 2n$ . . so on.

$\{a^n b^n\} \cup \{a^n b^{2n}\} \cup \{a^n b^{3n}\} \ldots$

FA can do finite storage only

{L ∩ L̄ is always regular}

{L ∪ L̄ is always Regular}

$(xyz)^R = z^R y^R x^R$

164 - (c) as $f(c) = \epsilon$ it can be placed anywhere

$a^n b^n$ - wont allow c

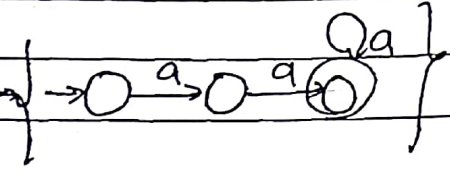
c) $a^n c^m b^n$ - it does not allow c to appear before a.

e) none of these.

EXCELLENT

$L = \{a^2, a^3, a^4, \ldots\}$

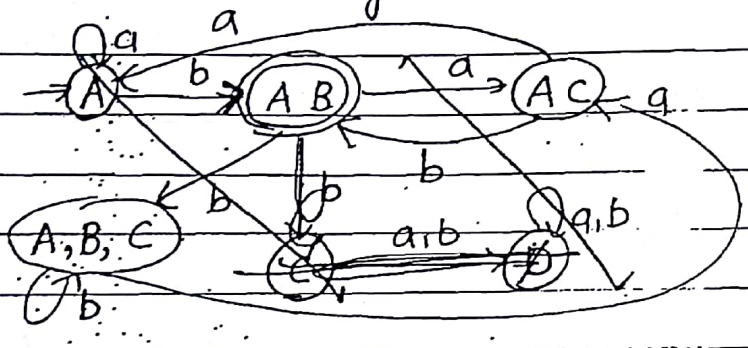
$= \Sigma^* - \{\lambda, a\}$



$\therefore$ RE = aa*

$\therefore$ No of states = 3

170



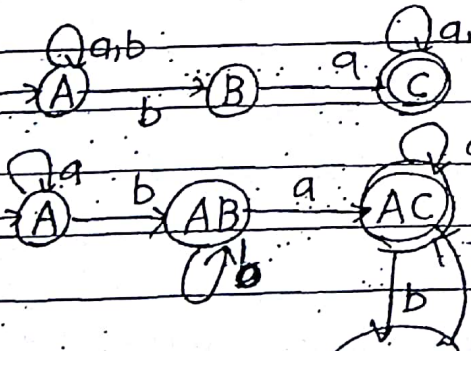
No of states = 4

$\{a^{m^n} \mid m < n \quad n \geq 1\}$ - Not a regular language as every no can not be written in this form

(170) ' language ending with b.

$\therefore$ No of state = 2

171 -



final

No of state = 2

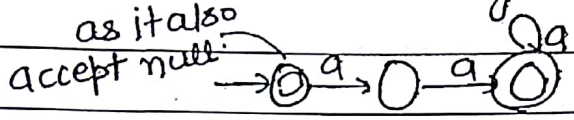
equivalent DFA - by subset construction

173 $L = \{(a^p)^* \mid p \text{ is prime no}\}$ $\Sigma = \{a\}$

prime no are atom & each no is broken down in term of prime no

$= \{a^2, a^3, a^5, \ldots\}^*$ → $(\varepsilon + aaa^*)$

∴ Minimal string is a^2



∴ O/P DFA has No of states = 3

174. $(a^* + a^*b^*) = a^*(\varepsilon + b^*) = a^*b^*$

175 – No of states = 2

as {A, C} {B, D}

176 – No of string is ϕ

$L_1 = 0^*1^*$

$L_2 = 1^*0^*$

$L_3 = (0+1)^*$

$L_4 = 0^*1^*0^*$

$L_1 \cap L_2 = 1, 0, 1, 00, \ldots, 11 = 0^* + 1^*$

$L_3 \cap L_4 = 0^*1^*0^*$

$(a^2 + a^3)^*$

$= a^2(\varepsilon + a)^*$

$(\varepsilon + aaa^*)$

$\therefore L = L_1 \cap L_2 - L_3 \cap L_4$

$= A - B$

$= A \cap \bar{B}$

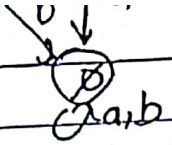
$= A - (A \cap B)$

$A \cap B = 0^* + 1^*$

$L = A - A \cap B$

$= \phi$

EXCELLENT



No of states = 6

~~{A, BC, ϕ, D, DC}~~ ED

178 L = {aba}

prefix(L) = {ε, a, ab, aba}

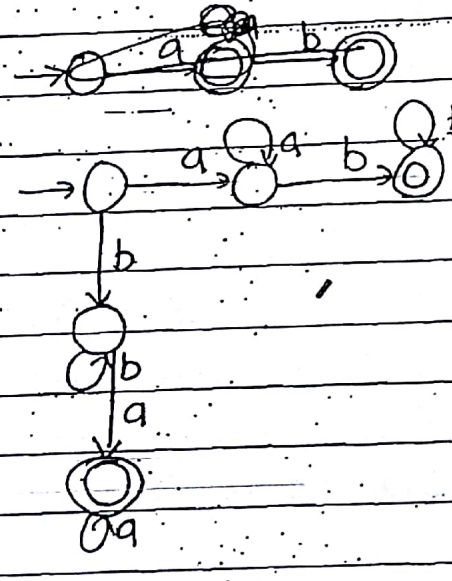
suffix(L) = {ε, a, ba, aba}

Ø = {ε, a, aba}/L

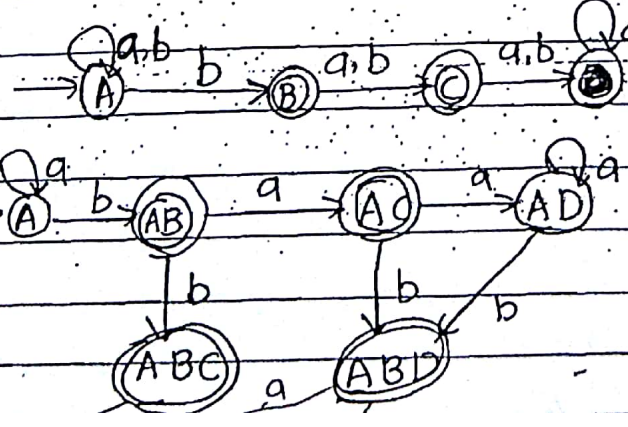
X = {ε}

D

179 -



180 -

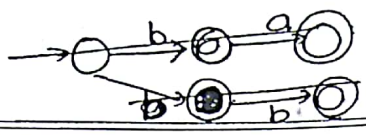


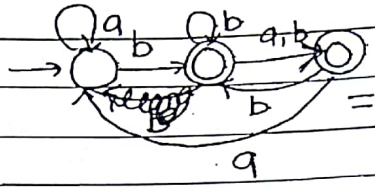
(3) $b(a+b)$

(3)

Page No. 102

Date : / /



180 -  = No of state = 3

181 - $L_1 = (0+1)^*$ $L_2 = (01^*0+1^*)$ $L_3 = (01+0+1+\epsilon)$

$L = L_1 \cap L_2 \cap L_3$

$= L_2 \cap L_3$

$= 01+0+1^*$

∴ Not containing $11 = \{\epsilon, 1, 010\}$

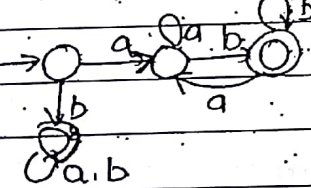
→ No of string = 3

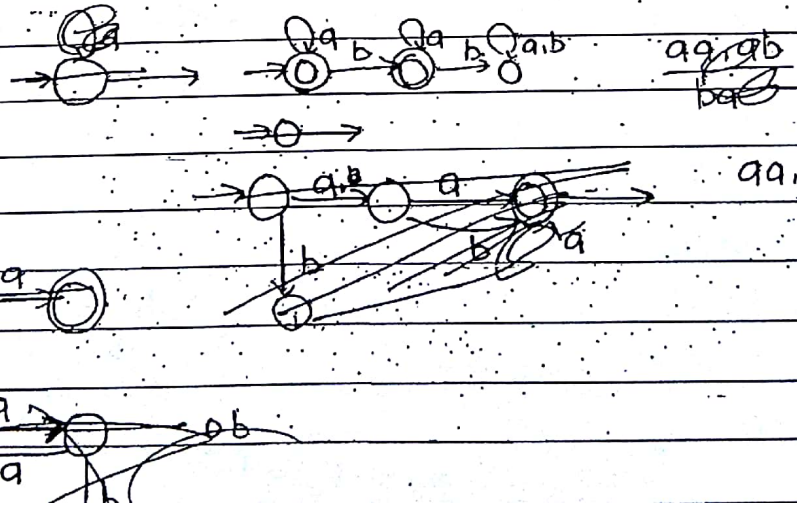
182 - ∴ No of states = 3

183 - $L = [made\ easy]$

$L_2 =$

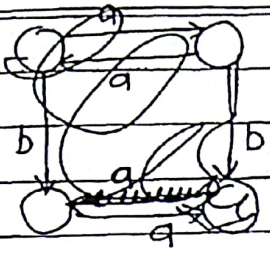
No of string in $L_2 = 8$

185 -  = 4 No of states = 4

186 - b 

Page No.
Date : / /

183 -



wx
0110
1010
0101 } yz
1001 } yz

184 - (w,x) can be placed two ways → 2 ways (w,x can be same
(y,z) can be placed two ways. - 2 ways y,z can be same)

↓↓

4 ways

186 - 5 states → two for exactly
two for even
1 * for trap
through product automata

[Any state combined with trap is also a trap in product automata.]

187 - e(S) = {S, T, A, E} 4

188 - No of states = 1 :- If from null move I can reach final state & final state is permanent accept.

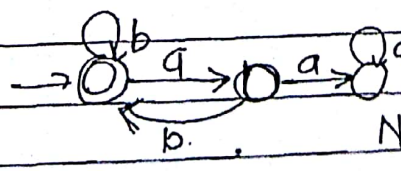
189 - R = (ab+b)*

equivalence class for $\Sigma^* = 1$

{b, ε, ab, ab, abab, bab, babab bb}

end with b -

ab+b



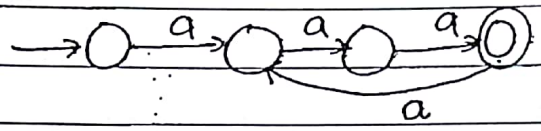
No of states = 3

EXCELLENT

$L_3 = \text{prefix}(L_1^* \cap L_2)$

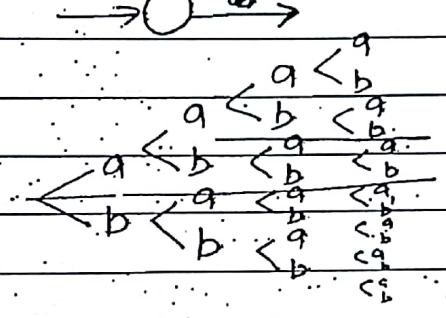
$= \{\epsilon, a, ab\} = 3$

Q1 - $L = \{a^{3(n+1)} \mid n \geq 0\}$



No of state = 4

Q2 -



aaaa
aaba
aaaa

abba
baab
baba
abb

out of 16 -

aaaa	baaa
aaab	baab
aaba	bbaa
aabb	baba
abaa	
abab	
abba	

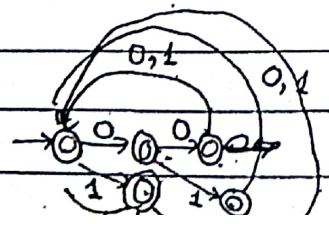
Draw for reject -

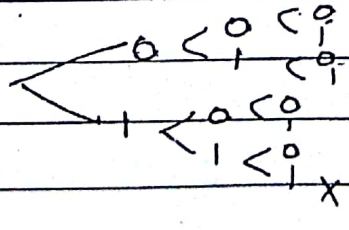
16 - 11 accepted

5 rejected

We will need 11 state for 11 accepted but 1 for showing reject.

Ques - 3 block - atleast 1 0

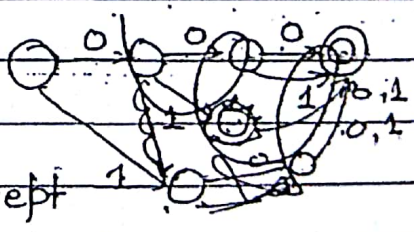




7 accept

+ 1 reject

∴ state = 8

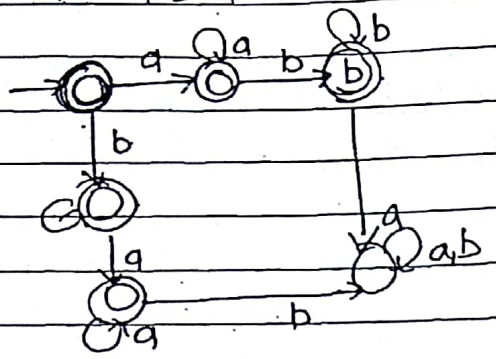


192. will go on ... string abaaba ab

Page No.
Date : / /

product automata is used when separate cond ~~is~~ given

193. $R = a^*b^* + b^*a^*$



No of final state = 5

194 - No of equivalence classes = 6

EXCELLENT

{ Top Down - LMD
Bottom Up - RMD }

CFG

1→ S→ 1S1 | 0S0 → palindrome

2. S→ 0S1 | 1S0 → counting comparison.

Derivations of Tree

Theorem - $w \in L(G)$ if and only if ∃ a derivation of w using the productions of G.

An ambiguous string is also a member of a language.

Each Even an unambiguous grammar have more than one derivation for a string. i.e LMD or RMD. so from here, concept of LMD & RMD arises. you have to specify LMD or RMD so that compiler dont get confused.

for each grammar, LMD & RMD are ~~is~~ unique, then grammar is unambiguous.

Ex - Let S→AB

A→aA|ϵ $\quad L(G) = a^*(bb)^*$

B→bbB|ϵ if w=abb to check whether it belong to L(G), check for derivation ?

S→AB → aAB → aAbbB → a bbB | abb

if there is a choice for substitution of variable, if left most variable is always substituted is LMD otherwise RMD (if Right variable is substituted)

if a string has derivation, it has both RMD as well as LMD

{A string has exactly 1D → false as it has two D's LMD & RMD}

Page No.
Date : / /

Ques - $S \rightarrow S+S \mid S*S \mid a \mid b \mid c$

$w = a+b*c$

I $S \rightarrow S+S \rightarrow a+S \rightarrow a+S*S \rightarrow a+b*S \rightarrow a+b*c$

II $S \rightarrow S*S \rightarrow S+S*S \rightarrow a+S*S \rightarrow a+b*S \rightarrow a+b*c$

for this grammar two LMD possible then it is ambiguous.

RMD - for a grammar, No of LMD = No of RMD for a given string w because each LMD corresponds to one Derivation tree & each derivation tree has one corresponding RMD.

RMD- ~~$S \rightarrow S+S \rightarrow S+c \rightarrow S*S+$~~

III $S \rightarrow S+S \rightarrow S+S*S \rightarrow S+S*c \rightarrow S+b*c \rightarrow a+b*c$

IV $S \rightarrow S*S \rightarrow S*c \rightarrow S+S*c \rightarrow S+b*c \rightarrow a+b*c$

Neither LMD nor RMD - $S \rightarrow S*S \rightarrow S*c \rightarrow S+S*c \rightarrow a+S*c \rightarrow a+b*c$ — mixture of LMD & RMD

LMD ~~is als~~ can be same as RMD if

$S \rightarrow aS \mid b$

$S \rightarrow aaS \rightarrow aab$

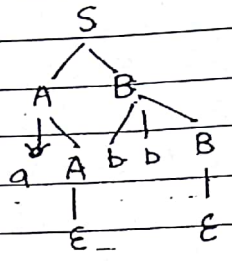
~~$S \rightarrow aab$~~

LMD is always similar to RMD for regular Grammars.

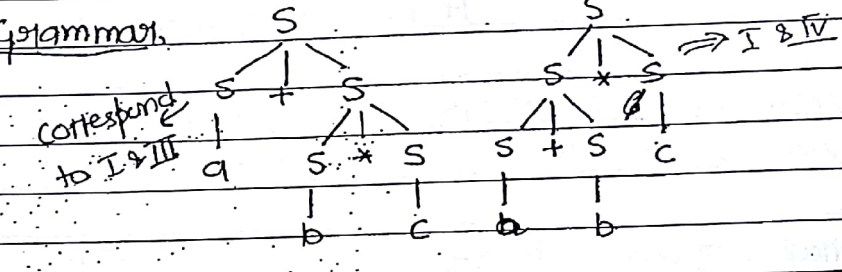
for every linear ~~it~~ Grammar, LMD is similar to RMD ie it has only one derivation. (bcoz it has single variable to be replaced)

EXCELLENT

for an unambiguous grammar, only one DT is possible for a particular string



for ambiguous Grammar,



- No of LMD = No of RMD = No of derivation tree.
- A derivation tree corresponds to one LMD & one RMD.

Problem caused by Multiple DT - { Syntax – Derivation, Semantics - Derivation tree }

- Multiple Meaning of a single stmt & meaning of ~~of~~ statement is confused & then compiler got confused, then it will create run time error.

Yield of tree - the leaf nodes ~~from left to right~~ abb in above.

if a string have two DT's then the yield of both tree must be same.

if a string has two or more DT's with same yield, then the grammar is ambiguous.

Page No.
Date : / /

Derivation Tree- A tree is said to be DT if

I) it must start with Start symbol i.e root

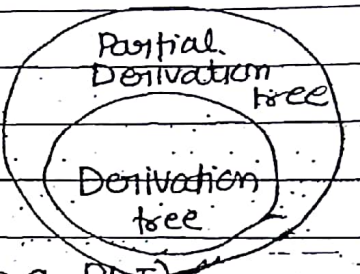
II) Every expansion of tree must corresponds to a production in Grammar.

III the yield must be a seten sentence belongi to L(G).

IV Yield must not be a sentential form i.e it must not variable.

- Every sentence is also a sentential form.
- Every DT is also a partial derivation tree.

Partial Derivation tree-



1. Yield may be a sentential form.
   (it may or may not be sentential form as each DT is also a PDT)
2. Every expansion must correspond to grammar in G.
3. the root of PDT need not be a start symbol.

- Derivation Tree is also known as Parse Tree.

for ambiguous Grammar- if atleast one string in G which have atleast 2 Derivation trees

UnAmbiguous- $\nexists\, w \in L(G)$ has exactly 1 derivation tree.

for $\forall\, w \in L(G)$

w has 0 DT - Not a Member
1 DT - UAG & a member
2 DT - Member with ambiguity

EXCELLENT

A Grammar is ambiguous if it has atleast 2 LMD or atleast 2 RMD's for $\exists\ w \in L(G)$

A Grammar is unambiguous if it has exactly 1 LMD for $\forall\ w \in L(G)$

- CFG ambiguity problem is ~~decidable~~ not decidable.

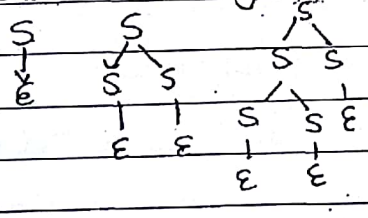
Reasons for Ambiguity -

1. Left Recursion: $S \to Sa \quad A \to A\alpha \mid \beta$
2. Left factoring $A \to \alpha\beta \mid \alpha\gamma$

not always cause ambiguity but sometime it can cause ambiguity.

If a Grammar has left factoring & left recursion then Grammar may or may not be ambiguous.

3. $S \to SS \mid \epsilon$ — it always causes ambiguity



4. $S \to aA \mid bA \mid ab \mid \epsilon$ → Never causes ambiguity

5. $S \to S_1 \mid S_2$ because it is possible if

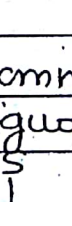
$$L(S_1) \cap L(S_2) = \phi$$

then S_1 & S_2 has common string then it causes ambiguity

Ex- $S \to S_1 \mid S_2$

$S_1 \to aA \mid \epsilon$ ⇒ have ϵ as common

$S_2 \to bB \mid \epsilon$ ∴ ambiguous



Ex - $S \to S_1 | S_2$

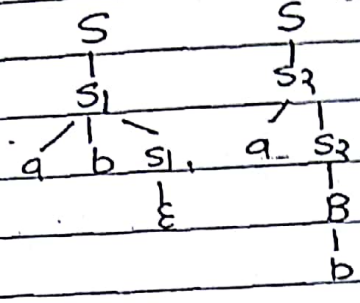
$S_1 \to abS_1 | \epsilon$

$S_2 \to aS_2 | B$

$B \to bB | b$

$L(S_1) \cap L(S_2) = ab \ldots$

$\therefore$ it is ambiguous



Ex - $S \to AB | BA$

$A \to Aa | \epsilon$ $\Rightarrow$ ambiguous

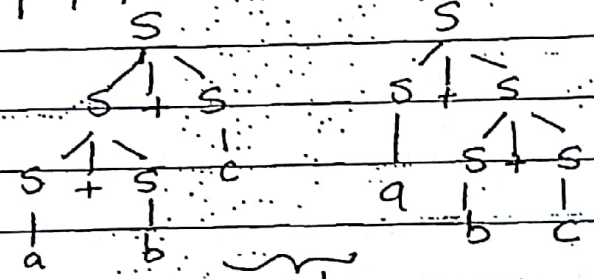
$B \to Bb | \epsilon$

$S \to AB | BA$

$A \to aA | a$ $\Rightarrow$ Not ambiguous

$B \to bB | b$

Ex - $S \to S+S | a | b | c$



↳ two different derivation tree

Hence it is ambiguous

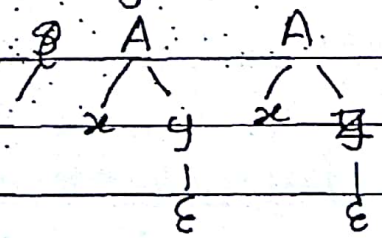
Ex - $S \to Sa | \epsilon$ → Left recursion but it is not ambiguous

Left factoring - $A \to xy | xz$

it may or may not cause ambiguity

Left factoring is ambiguous if

$L(y) \cap L(z) \neq \phi$



for $S \to aA|aB$

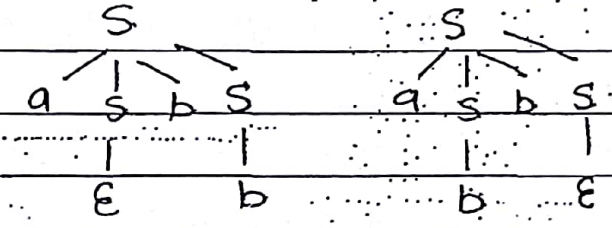
$A \to aA|\epsilon$ — Unambiguous as $L(A) \cap L(B) = \epsilon$

$B \to bB|\epsilon$

(6) - $S \to SaS$ it can cause ambiguity

Ex-

$S \to aSbS|b|\epsilon$ for abb



⇒ it is ambiguous because of presence of null

Ambiguity of language- A language is ambiguous if every Grammar to generate L is ambiguous. Such a language is also known as ~~inherited~~ INHERENTLY ambiguous language- this problem is known as Non-removable ambiguity:

A language is unambiguous if ∃ atleast a grammar which is unambiguous.

Ex- $S \to S+S|S*S|a|b|c$

(a) G is ambiguous

(b) L(G) is ambiguous

Example of ~~UnA~~ Ambiguous language-

Language which are ambiguous contain 3 ~~con~~ features -

Ex - $\{a^m b^m c^n\} \cup \{a^m b^n c^n\}$ — Ambiguous language
— CFL but Not DCFL

- it has no unambiguous grammar.

$S \rightarrow$ ~~$aSbAB \mid CD$~~
$S \rightarrow S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow AB$
$A \rightarrow aAb \mid \epsilon$
$B \rightarrow cB \mid \epsilon$
$S_2 \rightarrow CD$
$C \rightarrow aC \mid \epsilon \quad D \rightarrow bDC \mid \epsilon$

Condition for a language to be ~~condition~~ ambiguous.

1. two condition are in union
   A ∪ B
2. two condn must not overlap i.e $A \not\subseteq B$ or $B \not\subseteq A$
   i.e union is not Removable
3. $A \cap B \neq \phi$, the both condn must have something common to them.

Ex - $\{a^n b^n c^m d^m\} \cup \{a^n b^m c^m d^n\}$ — ambiguous

Ex - $S \rightarrow asb \mid bsa \mid ss \mid \epsilon$ → Not a ambiguous language as its ambiguity can ~~can~~ be removed.

Membership Algorithm-

A prodedure P is Membership algorithm for a language, if it can find $w \in L$ in finite amount of time.

$\forall w \in \Sigma^*$, if $w \in L$ → ~~say yes & Halt~~

if $w \notin L$ → No & Halt.
(it must not go in hang)

Membership is decidable for each language except RE.

Algorithm-

1. Brute Force Parsing - time Complexity - $O(k^n)$ • NP problem
(for CFL) where n is size of string.

2. CYK - TC - $O(n^3)$ • it is faster.
(for CFL) • P problem.
• optimum time Membership Algorithm

"P & NP is defined for problem but Not for Algorithm."

3. Require - $O(n)$ time for DCFL.
• It is used now a days.
• Run or. LL(k) or LR(k) compilers.

Grammar corresponding to DCFL - LR(k) or LL(k)
(most suited or best suited)

for every DCFL, there exist a LR(k) Grammar
but LL(k) can create DCFL but for each DCFL, it may
there may ~~not~~ or may not exist ~~DCFL~~ LL(k)

Date : / /

LL(K) uses ~~for ex~~ only one step of d

LR(K) uses 1 production (only) for one step of ∅ derivation.
n size string → n step → n production - O(n)

LR(K) grammars are always Unambiguous:
bcoz for each step only one derivation is possible so a single DT is possible.

Programming language Compiler require G to be
- Existence
- fast
- Unambiguous

CLR(K) satisfy all 3 condition.

LL(K) satisfy
- fast
- Unambiguous

only one production is processed at a single step but LL(K) donot guarantee the existence of grammar.

- both LL(K) & LR(K) can only create DCFL
- LL(K) is based on LMD but LR(K) grammar are based on RMD.
- LL(K) create top-down compilers & LR(K) create bottom up compilers.

EXCELLENT

Properties of LL(K) & LR(K)

(Decidable)

Brute Force Parsing- In this Method, all the possibilities are covered from start symbol each. No intelligence.

S → asb | bsa | as | aA

If no of production P

time = $|P| + |P| \cdot |P| + |P||P| \cdot |P| \cdots + |P|^{2n}$

$= |P|\,[|P|^{2n} - 1] / (|P| - 1)$

$= O(K^n)$

Input-

If will not CFG with ε production & unit production.

Stopping cond^n- length of string

We remove ε production & unit production as we have to satisfy non contracting condition.

2n- n time to replace by production like

S → ABCDE

n time to replace S → abcde

$n + n = 2n$

once n size is obtained it cant be reduced as ε & unit production are removed.

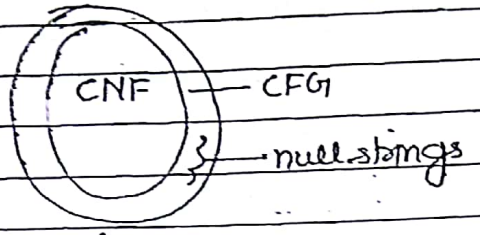
$\therefore O(P^{2n}) = O(K^n)$ $P^2 = K$

CYK Algorithm- Input- Grammar is in CNF (chomsky Normal form)

- Each CFG can be converted to CFG CNF.
- CNF & CFG both have same power.

[CNF allows- V → VV]

- CNF has same power as CFG practically but theoritically it has less power i.e CNF cannot generate Null string.



- for each CFG, is it possible to write CNF ? False
- for each CFL (null free CFL) or CFG without null string, is it possible to write CNF ? True.

- language generated by CNF is proper subset of language generated by CFG.

- CNF = Null free CFL's = GNF

∴ Griebach Normal Form- it also has same power as CFG

$$\begin{cases} V \to TV^* \\ V \to T \end{cases}$$

CNF & GNF has exactly same power

∀ CFG → ∃ CNF or GNF (false as it cannot generate null CFL)

for $\begin{cases} \forall \text{ CNF} \to \exists \text{ GNF} \\ \forall \text{ GNF} \to \exists \text{ CNF} \end{cases}$

GNF is used to convert Grammar to machine i.e PDA.

- to convert CFG to PDA, Input Grammar must be in GNF Form.

EXCELLENT

$B \to q$

3) LR(3)

4) None of these (as the grammar is ambiguous)

Ex - $S \to a|b$ → LL(1)

A Grammar G1 is LL(k) if and only if "k" symbol (including the current symbol) are presented to the compiler, the production that must be used in every step of derivation is 'UNIQUELY' determined & the derivation is used to be LMD.

LL(2) → 2 symbols are presented at a time to compiler.

[k-min-1 as current symbol has to be shown.]

LL(1) - $S \to a|b$

	a	b	ε
S	S→a	S→b	

LL(1) predictive parsing table

A Grammar is LL(K) if the table contain uniquely entries only.

Ex - $S \to qA|qB$, $A \to q$, $B \to b$

	a	b	ε
S	S→aA, S→aB	-	-
A	A→a		
B		B→b	

DCFL but not LL(1)

but in LL(2)

-	qa	bb	ab	ba	ε
S	S→aA	-	S→aB	-	
A					
B					

→ No confusion so it is LL(2)

EXCELLENT

Checking ambiguity of CFG is undecidable.

• Every UA is represented by LL(k) or LR(k) — False

LR(k)

Page No.
Date : / /

Ex- S → abA | abB
A → a
B → b

→ LL(3) as two symbol are not sufficient to remove confusion.

~~To ct~~
problem to check whether a grammar is LL(k) is decidable.

LR(K) Grammar- • k does not count current symbol. it does count of look ahead symbol only.
• LR(0) exist.
• the ~~dirie~~ derivation used is RMD.

• in LL(k), derivation is done from root to vert but LR(k) is done from Bottom to top. ie. Bottom-Up parser.

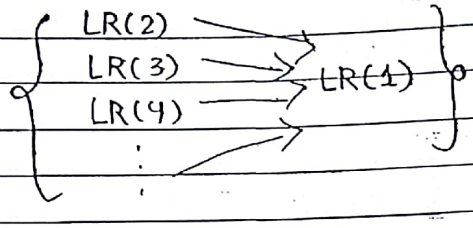
• LR(k) uses deduction ~~tries to eli~~ find out what can happen but ~~LR(0~~ LL(k) does predictive work.

~~Properties of LL(k) or LR(k)~~

1. LL(k) & LR(k) both are unambiguous
2. ——— both are O(n)
3. ∀ DCFL → ∃ LR(k)
∀ DCFL → LL(k) may or may not exist.
4. for each ~~DCFL~~ LL(k), LR(k) grammar there exist DCFL.
5. if a grammar is LL(k) then it is also LL(k') where k' ≥ k.
Ex- each LL(2) is also a LL(3) grammar.
6. ~~is~~ LR(k) ⇒ LR(k') for k' ≥ k
7. LL(k) ⇏ LL(k') if k' < k
8. Given a Grammar, to check whether it is LL(k) or LR(k) is decidable (for fixed k)

EXCELLENT

9. if a grammar is LR(K), for $K \geq 2$, there exist an equivalent grammar LR(1)



Conversion of Grammar from LR(2) to LR(1) is possible.

For each DCFL, there exist atleast LR(0) or LR(1)
∴ each DCFL will surely have LR(0) or LR(1) grammar.

Algorithm in CFG -

- Given CFG ⇒ ≡ CFG without Null production is possible false
- ——— w/o unit ——— True
- ——— w/o useless ——— True
- ——— w/o left Recursion ——— True
- ——— w/o left factoring ——— True
- ≡ CNF False
- ≡ GNF False

- Given CFG(ε-free) → ≡ CFG w/o Null production is possible — True
- ——— CNF → true
- ——— GNF true.

If a Grammar has no ε production, then ε does not belong language ∴ corresponds to the grammar

- if ε is present in Language, grammar must have null production.

- for all CFG G, G' with null production exist having
  $L(G) = L(G') - \{\epsilon\}$

An Grammar ~~with~~ when applied removal of useless production O/P will be removal of null, unit, useless production

Cover Conversion of CFG to CNF:-
1. Removal of null & unit production.
2. Conversion

CFG without any null production will be converted to equivalent CNF

Left factoring & Left recursion are removed to minimize the chance of ambiguity

- How to know that ambiguity is removed?
  check the obtained grammar for LL(k) & LR(k)
  Problem for checking for removal of ambiguity is undecidable because it is not possible to write LL(k) or LR(k) for each unambiguous grammar.

ε - Unit production are removed to convert a grammar to CNF or GNF
- or to decide the stopping cond^n.

- Useless production are written so that compiler can work faster.

CFG to CNF - for application of CYK
CFG to GNF - for conversion to PDA

EXCELLENT

Removal of Null Productions-

1. Identify all the nullable variable. $A \overset{*}{\Rightarrow} \epsilon$

Ques - $S \to ABa|aAb|aC$

$A \to aB|\epsilon$

$B \to bA|\epsilon$ No of Nullable variable = 2 (A, B, S)

$C \to a|b$

2) it create a production set $\hat{P} = P - \{\epsilon \text{ production}\}$

$S \to ABa|aAb|aC$

$A \to aB$

$B \to bA$

$C \to a|b$

3) Add all the production to $\hat{P}$ obtain by putting Nullable variable to Null

$S \to Ba|Aa|a|ab$

$A \to a$

$B \to b$

∴ New production set is

$S \to ABa|aAb|ac|Ba|Aa|a|ab$

$A \to aB|a$

$B \to bA|b$

$C \to a|b$

to check whether a grammar is Null free if S goes to null it is not null free.

EXCELLENT

$A \to aB/\varepsilon$
$B \to bA/\varepsilon$
$C \to a/b$

$\Rightarrow$ $A \to aB/a$, $B \to bA/b$, $C \to a/b$ $\Rightarrow$ So don't delete $C \to a/b$ & Grammar remains same.

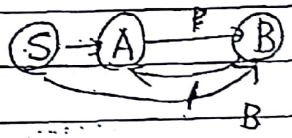
$L(G) = L(G') - \{\varepsilon\}$

(i) ~~A → a~~

Removal of Unit productions -

Ex - $S \to A/B$
$A \to aa/bb/B$ $B \to A/ab/ba$

1. Identify all the Unit production by using unit production dependency Graph



$S \overset{*}{\Rightarrow} A$
$S \overset{*}{\Rightarrow} B$
$A \overset{*}{\Rightarrow} B$
$B \overset{*}{\Rightarrow} A$

if G is $S \to A$
$A \to aa/bb/B$
$B \to ab/ba/A$

$\Rightarrow$ $S \overset{*}{\Rightarrow} A$, $S \overset{*}{\Rightarrow} B$ → indirect paths are also covered.
$A \overset{*}{\Rightarrow} B$
$B \overset{*}{\Rightarrow} A$

Ex - $S \to A/aAb/ac$
$A \to aa/bb/B$
$B \to ab/ba/A$
$C \to c/cC$

2. $\hat{P}$ = P - {unit production}

$S \to aAb/ac$
$A \to aa/bb$
$B \to ab/ba$
$C \to cC/c$

3. Add all the production obtained by mixing unit production with $\hat{P}$

$S \overset{*}{\Rightarrow} A$ $S \overset{*}{\Rightarrow} B$

$S \to aAb/ac/aa/bb/ab/ba$
$A \to aa/bb/ab/ba$ — $A \overset{*}{\Rightarrow} B$
$B \to ab/ba/aa/bb$ — $B \overset{*}{\Rightarrow} A$
$C \to c/cC$

EXCELLENT

Removal of the production which have atleast one useless variable either on right or left side.

- A variable is useful useless if it does never occur in any sentential form.
- If it does not generate a sentence i.e terminal string.

Ex- $C \to cC|dC$

or if generate a sentence which is no use of language

$$\left\{\begin{array}{l} S \to abB|C \\ C \to cC|dC \end{array}\right\}$$

Ex- $S \to aAb|bA$

$B \to aB|C|aC$

$A \to aA|b|B$

$C \to cC|dC$

Step1- Useful which can directly go to terminal

$U = \{B, A\}$

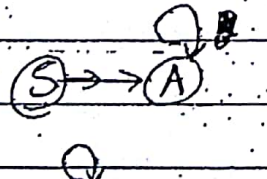
Step2- Try to include another variable by checking where they are one which are useful.

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

C is not added then C is the useless symbol

$$\left\{\begin{array}{l} S \to aAb|bA \\ B \to aB|c \\ A \to aA|b \end{array}\right\}$$

Step3- Draw a variable dependency graph



the thing which are not reachable from S remove them so B is removed.

$$\left\{\begin{array}{l} S \to aAb|bA \\ A \to aA|b \end{array}\right\}$$

EXCELLENT

Removal of Left Recursion-

If any production is of the form

$$A \to A\alpha_1 | A\alpha_2 | A\alpha_3 | \cdots | A\alpha_m | \beta_1 | \beta_2 | \beta_3 | \cdots \beta_n$$

Its equivalent Grammar is

$$A \to \beta_1 A' | \beta_2 A' | \beta_3 A' | \cdots | \beta_n A'$$

$$A' \to \alpha_1 A' | \alpha_2 A' | \alpha_3 A' | \cdots | \alpha_m A' | \epsilon$$

We put β in starting as β are the stopping condn & then α can go on.

$A \quad A_1\alpha_2 \quad A_1\alpha_1 \quad \beta_1 \quad \beta_2 \quad \beta_3$

Ex $S \to S*S | S+S | a | b | c$

~~S → *~~

$$\left\{ \begin{array}{l} S \to aS' | bS' | cS' \\ S' \to *SS' | +SS' | \epsilon \end{array} \right\}$$

if $S \to S*S | S+S | a | b | \epsilon$

$$\Downarrow$$

$$\left\{ \begin{array}{l} S \to *aS' | bS' | S' \\ S' \to *SS' | +SS' | \epsilon \end{array} \right\}$$

Removal of Left factoring-

If the production is of the form

$$S \to \alpha\beta | \alpha\gamma$$

∴ Its equivalent grammar is

$$\left\{ \begin{array}{l} S \to \alpha X \\ X \to \beta | \gamma \end{array} \right\}$$

Ex- $S \to a0 | a1$

equivalent- $S \to aS'$

$S \to 0 | 1$

Ex $S \to ABa \mid ABCd \mid ADE$

∴ equivalent-

$S \to AX$

$X \to BY \mid DE$

$Y \to a \mid cd$

Conversion of CFG to CNF

Step1 - if CFG is ε-free, remove ε & unit production.

Step2 - Conversion to CNF

Ex - $S \to aABb \mid aB \mid BaC$

$A \to ab \mid bA \mid AB$

$B \to a \mid BA \mid b$

Its equivalent CNF is-

Step1 - identify all thing which are already CNF

$A \to AB$

$B \to a \mid BA \mid b$

Step2 - convert them to CNF by fully subscripted variable

i.e conversion of terminal

a)

$S \to D_aABD_b \mid D_aB \mid BD_aC$

$A \to D_aD_b \mid D_bA$

$D_a \to a$

$D_b \to b$

New Grammar is

$S \to D_aABD_b \mid D_aB \mid BD_aC$

$A \to D_aD_b \mid D_bA$

$D_a \to a$

$D_b \to b$

$A \to AB$

$B \to a \mid BA \mid b$

EXCELLENT

Date : / /

~~S → X_1X_2~~

~~S → D_aBD_2~~

S → D_aD_3 | D_aB | BD_1

A → D_aD_b | D_bA | AB

B → BA | a | b

D_a → a

D_b → b

D_1 → D_aC

D_2 → BD_b

D_3 → AD_3

Conversion of ~~a~~ CFG to GNF- ~~No~~

Properties of CNF & GNF-

1. CNF for CYK & GNF of PDA:
2. G is ~~a~~ CNF then every derivation of string length n with have exactly $(2n-1)$ steps

n $n-1$ steps, to obtain final state like n-1

↓steps to replaced from terminals

ABCDE

3. G is CNF then min. height of derivation tree for w of length n is $\lceil \log_2 n \rceil + 1$ & max. height can be n.

↓ bcoz each step will covers the 2 variable ie. 2 length string height /2 in each step.

3. In CNF, ~~eac~~ every derivation tree is a binary tree.
4. If G is in GNF, every derivation of string of length n will have exactly n steps as each time we will be having one terminal.

EXCELLENT

for same string, GNF always have less no. of steps.
GNF is not always a binary tree.

1-May-2016

Theorem - When conversion of CFG to CNF is done with production P in CFG & terminal T_0 & the size of longest string in RHS of CFG is :k. then Max no of production in $\hat{P}$ (for CNF) is $(k-1)|P| + |T|$

∴ Max. No of production $\leq (k-1)|P| + |T|$

(k-1) is taken as

Ex - $S \to ABCDE$

$S \to D_3E$
$D_3 \to D_2D$
$D_2 \to D_1C$
$D_1 \to AB$

} 4 production

|P| it may be possible that all production are of maximum length.

|T| - as it is possible that each variable has to be replaced

$D_a \to a$
$D_b \to b$
$D_c \to c$

EXCELLENT

- FA + stack

PDA programming :-

PDA - Machine M(Q, Σ, Γ, δ, q_0, Z_0, F)
- 7 tupled

- Q : finite set of states
- Σ : input alphabet
- Γ : stack alphabet (Γ can be same as Σ or may not be.)
- δ
- q_0
- Z_0 : start stack symbol / Empty stack symbol
- F

When stack is empty Z will be at top of stack.

$Z_0 \in \Gamma$

F can be empty.

PDA programming :-
1. final state Method
2. Empty stack Method
- Both have same power
- Both are interconvertable
- Both can represent any CFL

δ is partial function i.e in DPDA. i.e you need not to specific move for each combination ie why DC allowed but in DFA, δ is total function.

Empty stack Method - No final state is specified

if $F = \phi$

- it may be empty stack method or a language

δ for PDA -

DPDA - $\delta(q_0, a, b) = (q_1, ab)$

- q_0 : current state
- a : input symbol
- b : top of stack
- q_1 : Next state after inputting a
- ab : top of stack (push a)

~~δ for~~

- Both DPDA & NPDA allow ε - moves.

EXCELLENT

... making two copies of the machine

push a pop b

$\delta Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

Γ^* will allow infinite subsets as

$\{[a^n b^n] \cup [a^n b^{2n}] \cup [a^n b^{3n}] \cup \ldots\}$

it is written as

$\delta Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

or

$(\text{finite}) 2^{Q \times \Gamma^*}$

Infinite sets are not valid as PDA can only make finite no of copies.

$\delta(q_0, a^*, b) = (q_1, a^*)$] – invalid because it must have some limit

$\delta(q_0, \epsilon, b) = (q_1, a)$] – Valid as Null Moves are allowed.

	DPDA	NPDA
1.	Choices are not allowed	choices are allowed.
2.	Dead configuration is allowed	DC allowed.
3.	ϵ moves allowed with restriction.	ϵ- moves allowed freely.

EXCELLENT

In DFA, ε-moves are not allowed as it always ~~create choice~~
due to stack it is possible to avoid choice by using ε-move in DPDA.

Page No.
Date: / /

- DPDA, ε moves are allowed in such a way it does not create choices.

$\therefore \delta(q, \varepsilon, a) \neq \phi$

then $\delta(q, c, a) = \phi \quad \forall\, c \in \Sigma$

i.e if null move is specified for a state with specific symbol then you cannot do with input symbol otherwise choices will be created.

Ex - $\delta(q, \varepsilon, a) = (q', a)$
$\delta(q, a, a) = (q'', aa)$
↳ choice created Hence it is not a DPDA because

DPDA cannot create two copies together.

- the combination of $Q \times \Sigma \times \Gamma$ is not specified in δ, they are considered as ϕ which lead to Dead configuration.

Command-

- PUSH $\delta(q_0, a, b) = (q_1, ab)$
- POP $\delta(q_0, a, b) = (q_1, \varepsilon)$
- DoNothing (SKIP) $\delta(q_0, a, b) = \delta(q_0, b)$ or (q_1, b)
- REPLACE $\delta(q_0, a, b) = (q_1, c)$

any no of symbol can be pushed and only one symbol can be poped at a time.

(in skip, change of state can occur but no operation with stack)

- FA can be simulated by PDA as

$\delta(q_0, a) = q_1$
$\therefore \delta(q_0, a, Z) = (q_1, Z)$
↓ start stack symbol

- SKIP is used to simulate FA by PDA.

EXCELLENT

• $\{c^* a^n b^n \mid n \geq 1\}$

SKIP c like

$\delta(q_0, c, Z) = (q_0, Z)$

$\delta(q_0, a, Z) = (q_0, aZ)$

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

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

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

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

Every PDA can be programmed in three states only (Maximum states)

Ques- $(a^n b^n \mid n \geq 0)$

~~$\delta(q_0, a, Z) = ($~~

String rejection in DPDA is done in two ways-

- Non final state
- Dead configuration

• if a language accept ε, make start state as final state, if possible

push 1st b ← $\delta(q_0, a, Z) = (q_1, aZ)$ $F = \{q_0\}$

push all a ← $\delta(q_1, a, a) = (q_1, aa)$ $q_0 = q_0$

$\delta(q_1, b, a) = (q_2, \varepsilon)$ - pop first b

$\delta(q_2, b, a) = (q_2, \varepsilon)$ → pop all b

$\delta(q_2, \varepsilon, Z) = (q_0, Z)$ → final state

3 ways to program

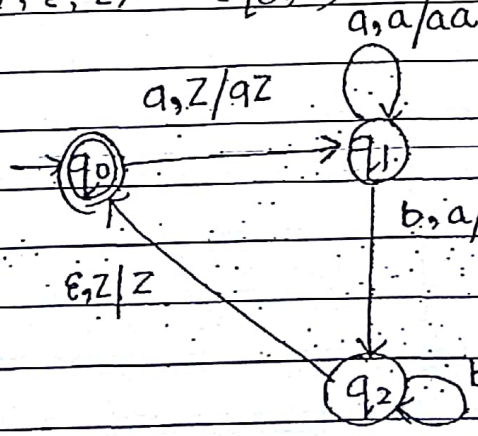
- state table
- Diagram
- Transition function

I/P stack Push to stack

$q, Z / qZ$ or q, Z, qZ

If more than 1 comment

$[q, Z, qZ ; b, Z, bZ$



In final state Method, need not to empty the stack but in empty stack method, stack must be emptied at the end of j^n i.e for acceptance of string stack must be emptied.

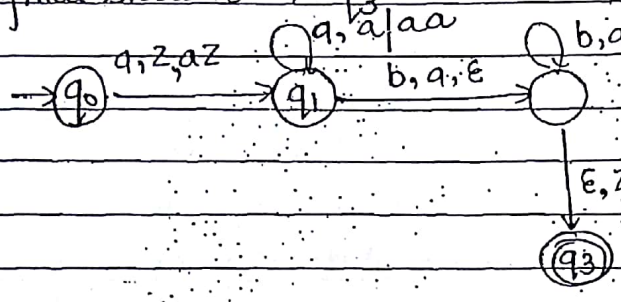
EXCELLENT

Page No.
Date : / /

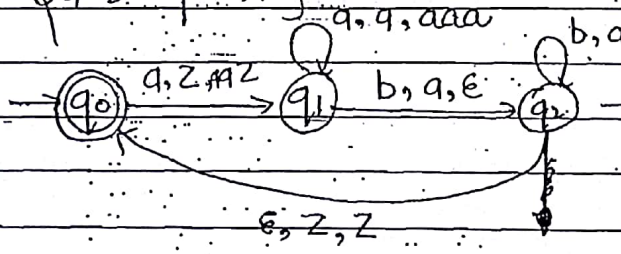
$\{a^m b^n \mid m, n \geq 0\}$

in final state - a will be there in stack
in empty state - you have to remove a from stack for its acceptance.

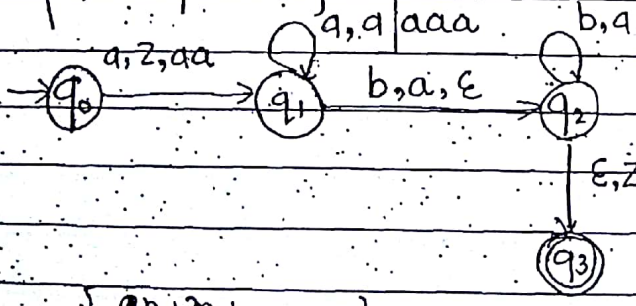
Ques - $\{a^m b^n \mid n \geq 1\}$
then acceptance will not be q_0
final state is at q_3



Ques - $\{a^n b^{2n} \mid n \geq 0\}$

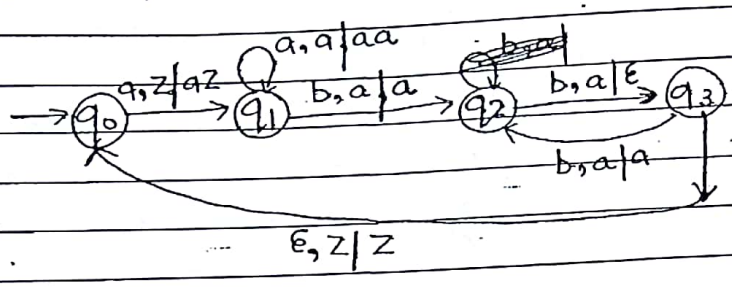


Ques - $\{a^n b^{2n} \mid n \geq 1\}$



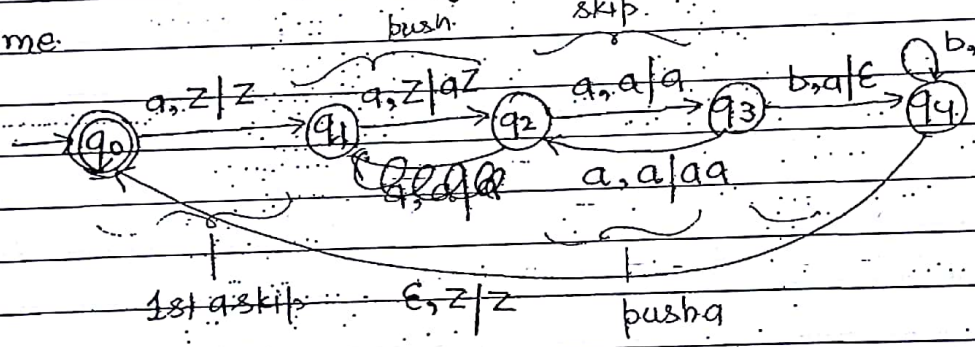
Ques - $\{a^{2n} b^n \mid n \geq 1\}$
↓ for popping b pop for 1 b & skip for another

$a^n b^{2n}$ - push 1 a for 1a & pop 2b's for 1a

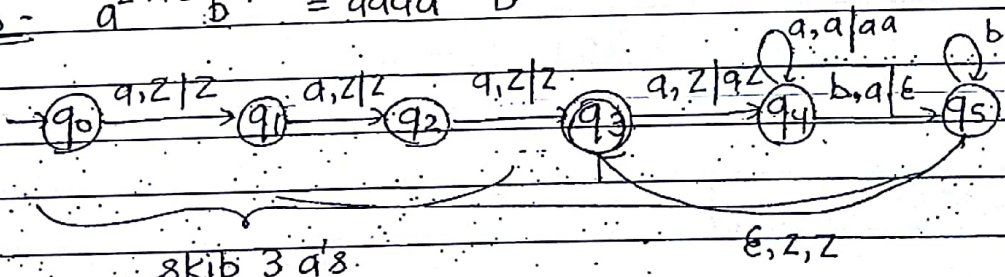


to see for machine
- check for the condition of both a & b.

Ques - $\{a^{2n} b^n\}$ only 1 program as it has condn of poping because you can pop only one symbol at a time



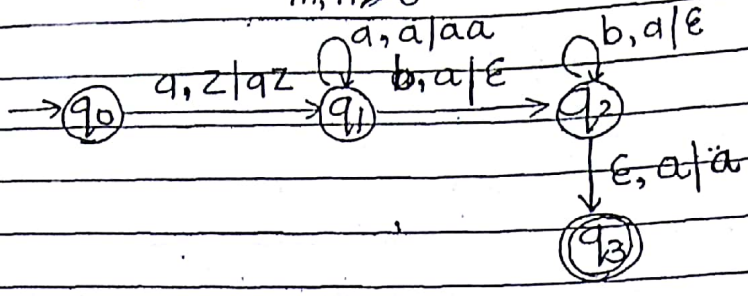
Ques - $a^{2n+3} b^n = aaa a^{2n} b^n$



$a^{2n} b^{n+3}$ → skip 3 b's in mid and also consider the case for b^3 when n=0

Date : / /

Ques - $\{a^m b^n \mid m > n\}$, $m, n \geq 0$



Ques - $\{a^m b^n \mid m < n\}$

last comment -

$\delta(q_2, b, z) = (q_3, z)$ it will go to final state if aabbbb

at 3rd b it go to q_f at another b it will go to dead so you have to write

$\delta(q_f, b, z) = (q_f, z)$

$\delta(q_f, \varepsilon, z) = (q_f, z)$

Ques - $\{a^m b^n \mid m \geq n\}$

$\delta(q_2, \varepsilon, a) = (q_f, z)$ → for $a > b$

$\delta(q_2, \varepsilon, z) = (q_f, z)$ → $a = b$

$\{a^m b^n \mid m \leq n\}$

$\delta(q_2, b, z) = (q_f, z)$

$\delta(q_2, \varepsilon, z) = (q_f, z)$

$\{a^m b^n \mid m \neq n\}$

(for $m < n$ & $m > n$)

combine them.

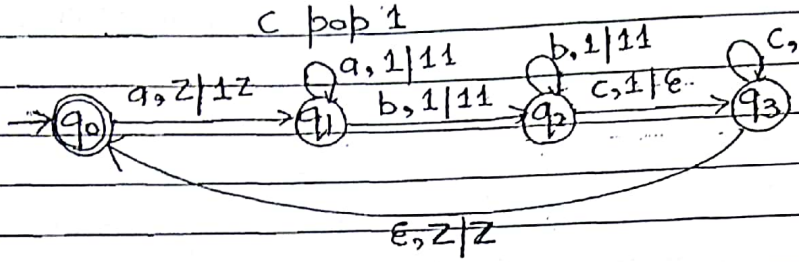
EXCELLEN

-for a you can push 1 also.

Page No. 119
Date : / /

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

for a push = 1
b push 1
c pop 1

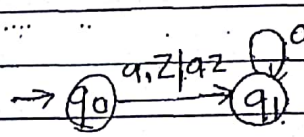


Ques - $\{a^{m+n} b^m c^n \mid m, n \geq 1\}$

a - push 1
b, c - pop 1

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

~~p = m+n~~ for a → push 1
b - pop until you reach end of stack.
if reach end just push it
c → pop 1



EXCELLENT

$\{a^n b^n\} \cup \{a^n b^{2n}\}$

$\delta(q_0, \epsilon, Z) = \{(q_1, Z), (q_2, Z)\}$ — if it is having any split, it might be an union.

Now q_1 will care for $a^n b^n$

It can do for $a^n b^{2n}$

- this thing is allowed only in NPDA

$\{a^n b^m \mid m \le n \le 2m\}$

$S \to aSb \mid aSbb$ — it can have any one required.

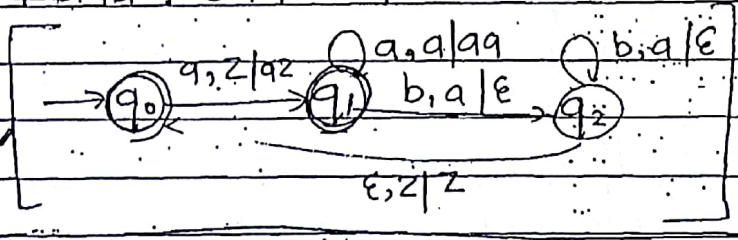
then

$\delta(q_0, a, Z) = \{(q_1, aZ), (q_1, aaZ)\}$

for $n \le m \le 2n$

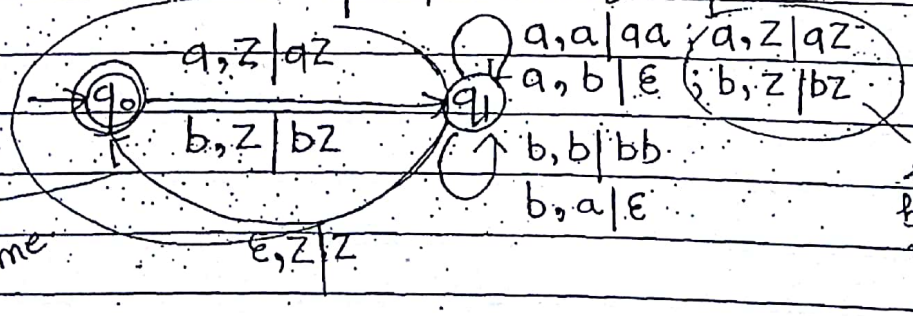
$\delta(q_0, a, Z) = \{(q_1, bZ), (q_1, bbZ)\}$

$a^n b \{a^n\} \cup \{a^n b^n\}$



II family $= n_a = n_b$

$\{w \mid n_a(w) = n_b(w)\}$



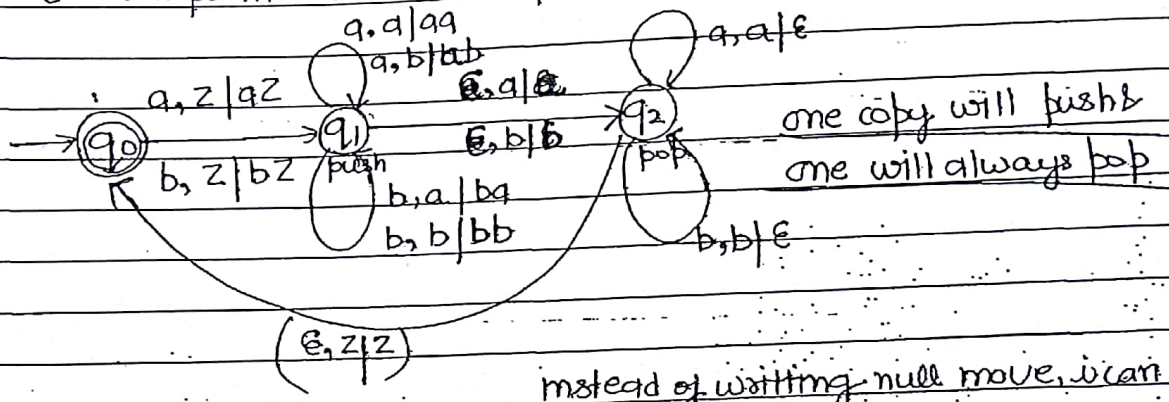
Page No. 120
Date : / /

III. family - Palindrome

→ $\{ww^R \mid w \in (a,b)^*\}$ · NPDA job as push to pop is not clear.

aabbaa

ε - even palindrome push until you reach mid

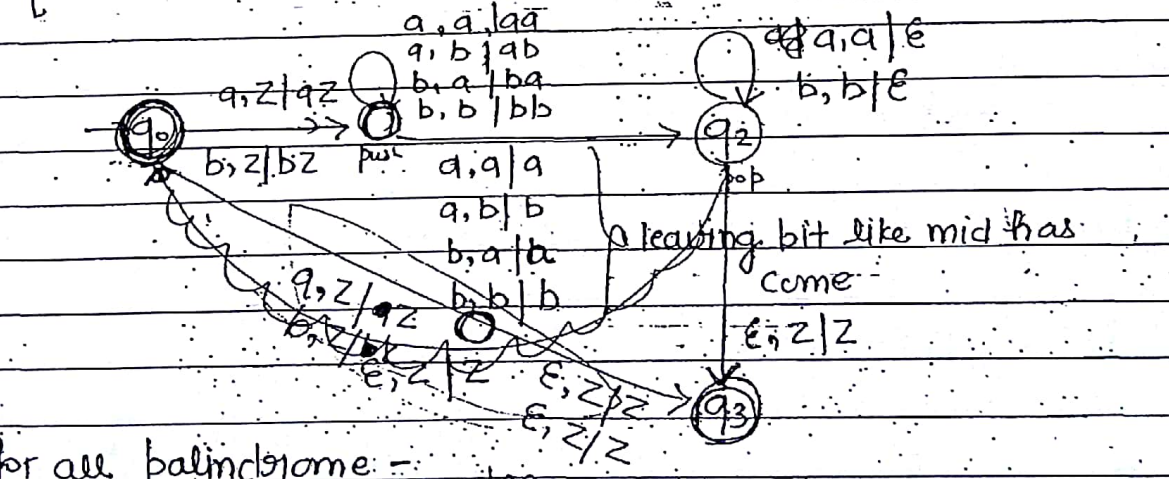


instead of writing null move, i can also write

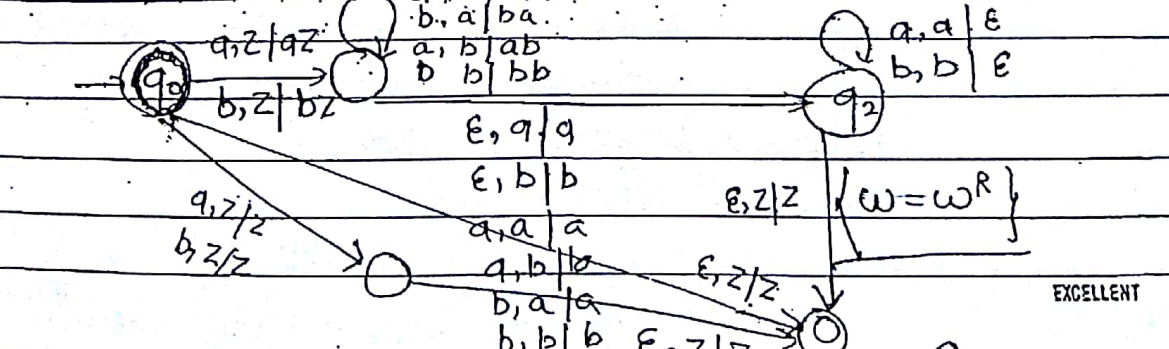
$\{a,a|\varepsilon\}$
$\{b,b|\varepsilon\}$

q,z
q,q |aa| ab
a,a

$\{wxw^R \mid w \in (a,b)^*\}$

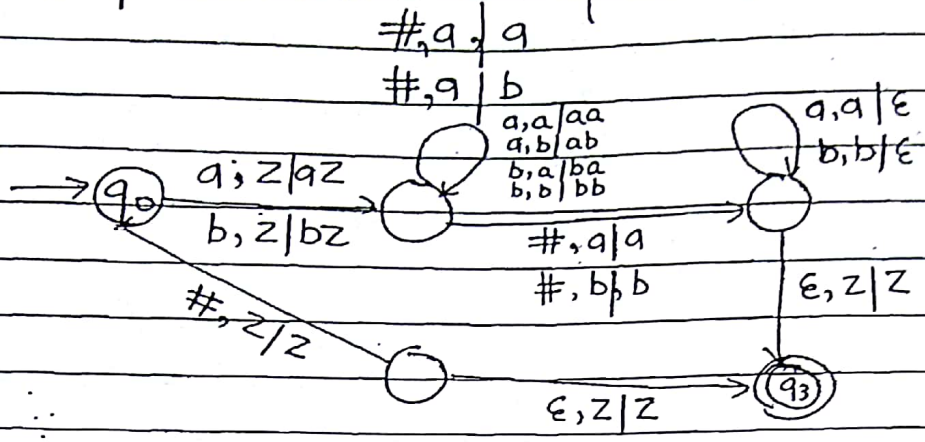


for all palindrome -



EXCELLENT

for # palindrome. $\{w\#w^R\}$



GODEC, TURING, POST

Turing Machine, REC, RE, Not RE

Turing Machine - 1. Defmition.

2. Machine to Language

3. L - M ? $\{a^n b^n c^n \mid n \geq 0\}$

4. Turing Machine as a transducer

$f(x,y) = x+y$

5. Church turning thesis

Imp 6. Variation of Turing Machine.

"No Logical Machine can have more power than turing machine."

RE & REC language - • Definition of REC & RE.

• facts related REC & RE.

- Enumeration Procedure (EP) property
- Membership Algorithm (MA) property
- Lexigraphical ordering property
- Closure properties
- $L, \bar{L}$ theorem

• Examples of REC, RE, Not RE

RE but not REC - L_u, Self A (self Accepting)

Not RE → NSA, $\bar{L_d}$ (Non Self Accepting)

fair good EXCELLENT

- Some Undecidable problems related to turing machine:
  - HP (Halting Problem)
  - BTHP (Blank Tape Halting problem)
  - SEP (State Entry problem)
  - PCP (Post Correspondence problem)
  - MPCP (Modified Post Correspondence problem)
  - REM (RE Membership)
- Reduction theorem's ($P_1 \leq P_2$)

Turing Machine- [FA + R/W + R ↔ L]

[FA + R/W ⇒ equivalent to FA]

- finite states
- single starting state

$(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

- $\square$ → Blank symbol (b)
- Γ → tape alphabet
- Σ → input alphabet

tape tape alphabet will contain input alphabet as well as output alphabet & blank symbol as well

if o/P are 0, 1
Input a, b
$\Gamma = \{a, b, 0, 1, \square\}$

tape alphabet is required for Write operations here so that storage can be done.

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

B is used to show end of tape.

Σ can never contain $\square$.

$\Sigma \subseteq \Gamma$ (false it can contain $\square$)

$[\Sigma \subset \Gamma - \{\square\}]$

Page No.
Date: / /

Deterministic Turing Machine

δ format for DTM & NTM

DTM: $\delta(q_0, a) = (q_1, b, L)$ → Movement of Head

- q_0: current state
- a: I/P
- q_1: Next state
- b: change the I/P ie Write

$\delta(q_0, B) = (q_1, b, R)$

$\therefore \delta \quad \boxed{\delta_{Q \times \Gamma} \rightarrow Q \times \Gamma \times \{L, R\}}$

↑ Γ is here as tape is acting as storage too, output symbol can also occur while mapping.

Command -

1. Do Nothing $\delta(q_0, a) = (q_1, a, R)$
2. Replace $\delta(q_0, a) = (q_1, b, L)$

NTM: $\delta_{Q \times \Gamma} \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$

Both NTM & DTM have same power & are interconvertable

Standard Turing Machine

1. Deterministic
2. Single tape
3. Infinite in both direction
4. Read Write
5. Left Right Movement

NTM- $\delta(q_0, a) = \{(q_1, a, L), (q_2, b, R)\}$

EXCELLENT

DTM	NTM
• It does not allow choices	• It allow choices
• ε-moves are not allowed	• ε-move are not allowed
• DC allowed	• DC allowed

{δ in Turing Machine is a partial function}.

the configuration of TM is also known as ID (instantaneous description)

it give information ~~which~~ about parameter which can change (state, Movement of head, content of tape)

□ a a b b □ ID-
↑ q_0 string representation-
$\square q_0 aabb\square$
q_0 will move to right.

Ex - ID is $\square q_0 aabb\square$
and $\delta(q_0, a) = (q_1, b, R)$
$\delta(q_1, a) = (q_2, a, L)$

After two move what is the configuration of M/C →

$\square q_0 aabb\square \vdash \square b q_1 abb\square \vdash \square q_2 babb\square$

$\square q_0 babb\square$ (replace & then move right) — $\square b q_2 abb\square$ (Move left)

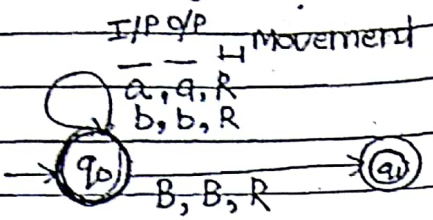
Ques - $\square(q_0, aabb)\square \vdash \square q_0 baabb\square \vdash^* \infty$? check whether it is true.
it mean after some step machine will go to hang.

↳ False here.

EXCELLENT

Machine to Language-

Ex 1 -



Ques - $L(M) = (a+b)^*$

What type of string it halt & hang? It always halt but never hang.

What is content of tape at end of ~~string~~ expression. ⟫ Same as Input

Turing Machine — HALT — Accept / Reject

Turing Machine — HANG (Hang is always considered as Rejection)

To reject a string in TM

1) Non final state
2) DC
3) Hang

Acceptance in TM - final state acceptance only.

- In a turing Machine, final states are permanent accept, so No moves on final states are defined. FA can also halt without processing the whole Input. If a dead final state occur it is also accept

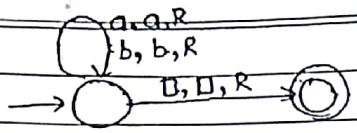
aab
→ a lead to final state. No move from a b then dead but final so accepted.

Turing Machine halts only at dead configuration.

- TM with only left & right move, can never go into hang.

EXCELLENT

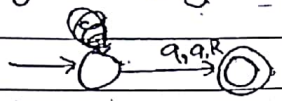
Ex 1 -



$L(M) = (a+b)^*$

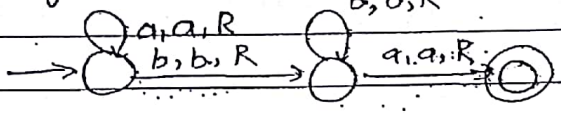
- To convert a FA to turing machine use blank moves only.

for string starting with a.



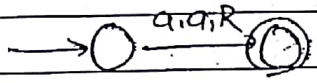
No Need to decide for all as it get accepted as it one reach to final state ~~& no need to show the~~

substring ab



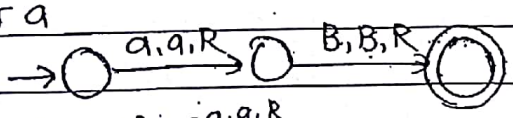
$a^*bb^*a(a+b)^*$

to accept 'a' only



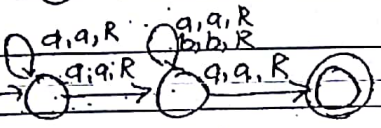
→ it will accept $a(a+b)^*$

for a

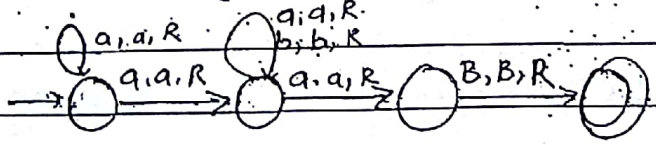


- Use blank symbol

for



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



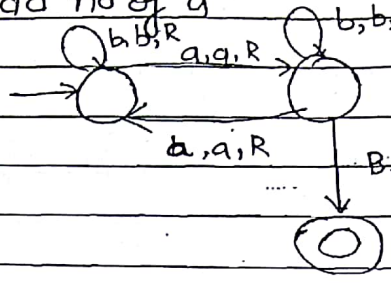
→ $a^*a(a+b)^*a$.

EXCELLENT

Null string are not considered in TM.

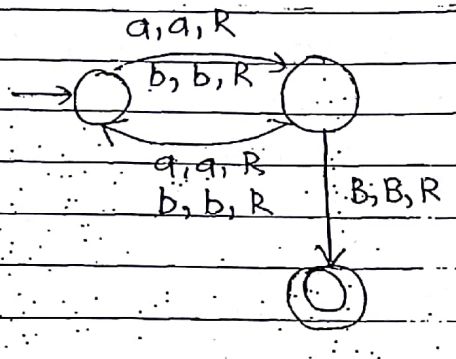
Page No.
Date : / /

for odd no of a

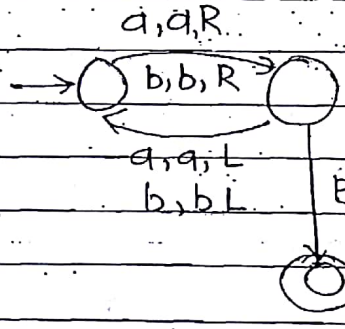


in turing machine after reaching final state it will accept everything so if specific o/p use B,B,R move.

Ques



L(M) = odd length string
Halt - Halt on everything
Hang - ϕ
Content of tape - same as I/P



~~B a B~~ □a□
□b□
□aa□
□ab□

When L & R are there, try to cover with taking string.

□a□ |- accept □b□ |- accept □aa□ |- hang (reject)

□ab□ |- reject due to hang

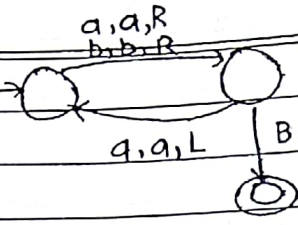
□ba□ |- hang □bb□ |- hang

□aba□ - ~~accept~~ always reject as two bit itself hanging

Language = {a,b}

Halt - {a,b}

Ques -



□aq□ - accept
□b□ - accept
□aa□ - hang
□ab□ - halt & reject
□ba□ - hang
□bb□ - halt reject

Language - {a, b}

Halt = {a, b, ab, bb}

~~Hang~~ (~~a+b~~)~~b~~ ⇓

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

Hang → $(a+b)a(a+b)^*$

↓ when 2nd bit is a

content of tape → it will remain same.

Language to Machine -- $L=\{a^n b^n c^n \mid n \geq 1\}$ as ε is not considered in turing machine.

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

TM can do any no of comparison

TM for $a^n b^n \mid n \geq 1$

- tick for a & search for its b & tick it too
- if finally not a & b equal $a^n = b^n$
replace a by x & then one b by y & so on.

□	□	a	a	a	b	b	b	□	□

$\delta(q_0, a) = (q_1, x, R)$ — q_1 - will search for b & make it y.

$\delta(q_1, a) = (q_1, a, R)$

~~$\delta(q_1, y)$~~

$\delta(q_1, y) = (q_1, y, R)$ — because at second round you have to leave it

$\delta(q_1, b) = (q_2, y, L)$

$\delta(q_2, y) = (q_2, y, L)$

$\delta(q_2, a) = (q_2, a, L)$

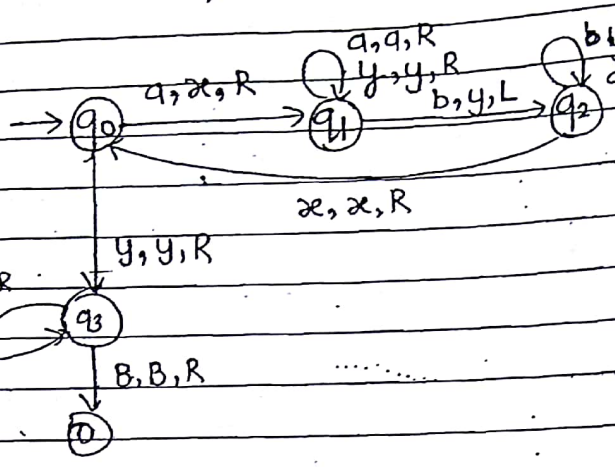
$\delta(q_2, x) = (q_0, x, R)$

$\delta(q_0, y) = (q_3, y, R)$ $\quad \delta(q_3, □) = \delta(q_f, □)$

$\delta(q_3, y) = (q_3, y, R)$

EXCELLENT.

~~$\delta(q_0, a) = (q_1, x, R)$~~



$\{a^n b^n c^n \mid n \geq 1\}$

$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, y) = (q_1, y, R)$

~~$\delta(q_1, b) = (q_2, y, R)$~~ $\delta(q_1, b) = (q_2, y, R)$

$\delta(q_2, b) = (q_2, b, R)$

$\delta(q_2, z) = (q_2, z, R)$

$\delta(q_2, c) = (q_3, z, R)$

$\delta(q_3, z) = (q_3, z, L)$

$\delta(q_3, y) = (q_3, y, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, x) = (q_0, x, R)$

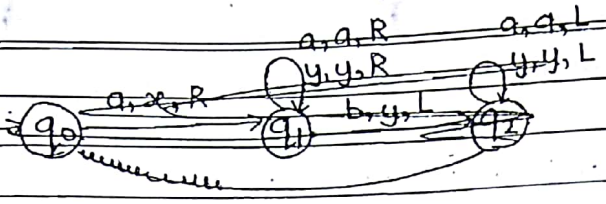
$\delta(q_0, y) = (q_4, y, R)$

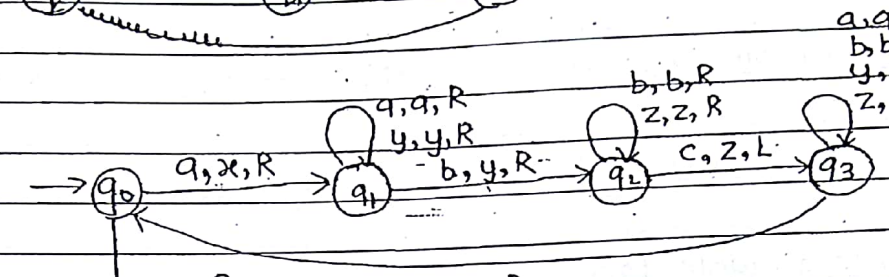
$\delta(q_4, y) = (q_4, y, R)$

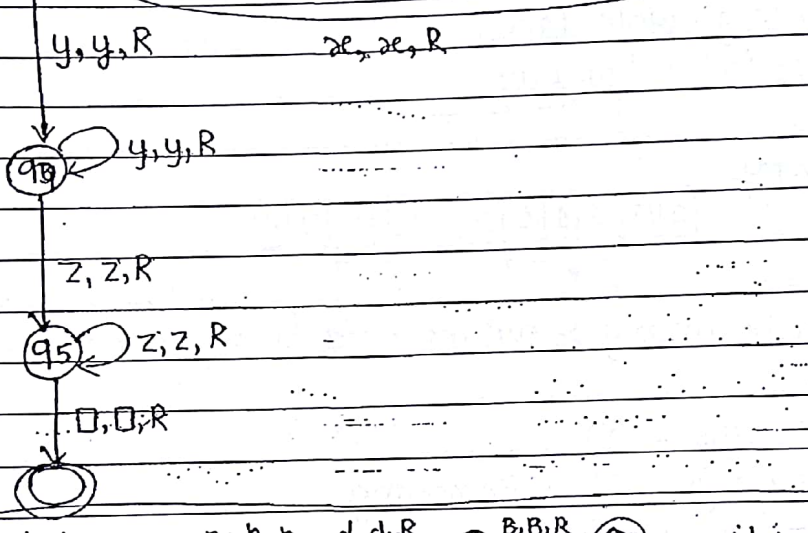
$\delta(q_4, z) = (q_5, z, R)$

B, B, R specify end of string as B is only placed at the end of tape

Page No. 125
Date : / /



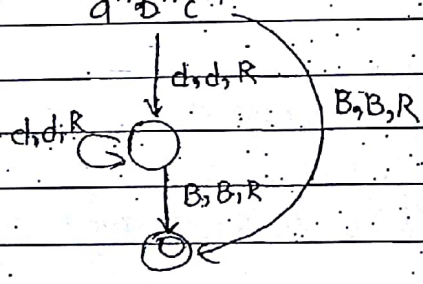




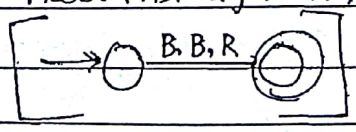
$a^n b^n c^n d$ – $a^n b^n c^n$ — d, d, R → B, B, R → ((q)) it is specify important to specify Blank move at final state

$a^n b^n c^n d d^*$ $a^n b^n c^n$ — d, d, R → B, B, R → ((q)), d, d, R (loop)

$a^n b^n c^n d^*$



to accept null through TM



EXCELLENT

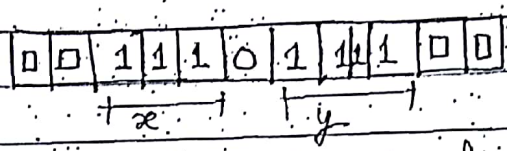
Turing Machine as a transducer - A set of function accepted by turing machine is partial recursive function.

A set of function represented by TM is called Recursive-fn.

"EVERY LOGICALLY COMPUTABLE FUNCTION IS PARTIAL RECURSIVE."

RE - Turing recognizable language
REC - Turing decidable language

$f(x, y) = x + y$



- I/P is given in unary & o/P may be binary or unary

$\delta(q_0, 1) = (q_0, 1, R)$
$\delta(q_0, 0) = (q_1, 1, R)$ // Removing of 0
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q_1, B) = (q_2, B, L)$
$\delta(q_2, 1) = (q_3, B, L)$
$\delta(q_3, 1) = (q_3, 1, L)$ // parking of head at initial state
$\delta(q_3, B) = (q_3, B, R)$

o/p- B 1 1 1 1 1 1 1 1 B // Removing of 0

B 1 1 1 1 1 1 1 B B B // Removing of 1 (extra

(all PL)
Fortran, C, C++, C#, Java - are not CFL because it has two feature not CFL. So they are CSL

the following feature of Programming language cannot be handled by CFL -

1) Variable declared before use.

real ~~ww~~ abc;

abc = 1.5;

as it corresponds to $\{ww \mid w \in (0,1)^*\}$

w for abc & again at the time of use w has to be checked.

The C language is fully parsable only by turing machine — True

2. Matching the formal and actual parameters of function.

It corresponds to

$[a^n b^m c^n d^m \mid m, n \geq 0]$

as ~~fun(abc, xyz)~~ f(a, b, c, d, e) → $x^2 + 3$

(real real int int int)

~~fun(int abc, int xyz)~~ f(x, y, z, w, 4)

$x^2 + 3$

$x^2 + 3$ $x^2 + 3$ which cannot be done by PDA.

- Mainly part of PL are DCFL but some features cant be handled so we require TM.

Page No.
Date : / /

Ques - $f(w) = ww$

if 111

O/P - 111111

1 change 1 to x & and go on & change 1 □ to y
2. Again come back and so on
3 finally change all x & y to 1.

$\delta(q_0, 1) = (q_1, x, R)$
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q_1, □) = (q_2, y, L)$ $\delta(q_1, y) = (q_1, y, R)$
$\delta(q_2, y) = (q_2, y, L)$
$\delta(q_2, 1) = (q_2, 1, L)$
$\delta(q_2, x) = (q_0, x, R)$
$\delta(q_0, y) = (q_3, y, R)$
$\delta(q_3, □) = (q_4, □, L)$
$\delta(q_4, y) = (q_4, 1, L)$
$\delta(q_4, x) = (q_4, 1, L)$
$\delta(q_4, □) = (q_4, □, R)$

Church Turing Thesis - "Every logically computable function can be represented by turing machine"

λ-calculus Model - it has a set of fn which covers all logical property & can be represented by TM.

If a turing machine can do any logical function they logic & TM has same power.

λ-calculus has same power as turing machine, post system also have same power as turning machine
Rewritting system has same power as turing Machine.

EXCELLENT

Any Machine based on logic can never exceed the power of TM.

Page No. 127
Date : / /

Variations of TM-

all Machine has same power as TM

1. TM with stay option
2. TM with seminfinite tape
3. offline TM
4. Multitape TM
5. Multidimensional TM
6. Non deterministic Turing Machine
7. Universal turing Machine

1. HTM — Less power than TM
2. LBA — LBA < HTM < TM
3. R/O TM Read only TM
3. 1way Turing Machine
5. TM with finite tape

→ same power as FA
one way ∞-FA+R/W+ left or right only

1. TM stay option- it has option to Move left right or remain on same state

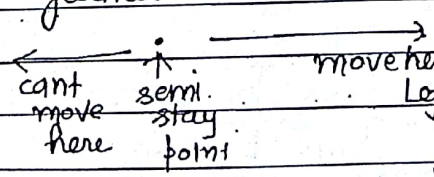
$\delta: Q \times \Gamma \to Q \times \Gamma \times (L, R, S)$

both have same power as for stay in simple {L,R} Machine you can write a R & then L Move

↑ → R
↑ ← L
stay

to simulate it Γ no of production are to be added

2. Semi-Infinite - one side you can not move further
• if you reach a posn, you can't move further



they both have same power

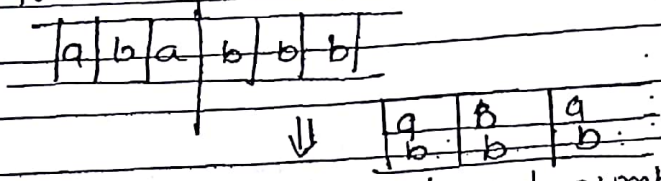
EXCELLENT

Proof- ← L | R → both side ∞

cut it out & keep two tracks

→ L
← R so infinite is simulated to semiinfmite

to maintain two track, use two symbols as



it is done with the use of compound symbol.

when you are doing program

let say you want to use lower track.

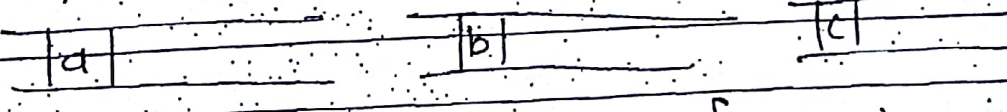
use

$\delta(q_0, bb) = (q_1, \rightarrow)$
$\delta(q_0, bb) = (q_1, -)$
$\delta(q_0, bb) = (q_1, \leftarrow)$

and define for each symbol

3. off Line TM- Input & output are kept separately in a tape.

4. Multitape TM - it has same power



(q_1, a, b, c) as you can check for various input on diff. tape at a time but single tape will simulate it in some finite step

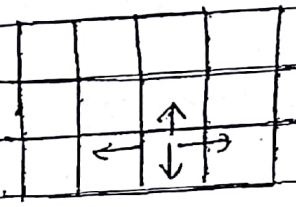
Multitape is faster

EXCELLE

if a k-tape TM can solve problem in $O(n)$ time but STM will solve the Machine will solve in $O(n^2)$ time.

if k-tape solve in $O(n^4)$ → STM will solve in $O(n^4)$ time.

5. MultiDimensional TM- Input tape is a Multidimensional



Movement possible are Right Left, up, down

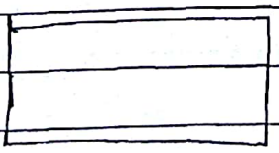
MultiDimensional TM has same power as TM.

NTM - same power as DTM
as each NTM can be converted to DTM.

UTM - • it is reprogrammable but TM is not because TM can not modify its program.
TM can ~~they~~ never corrupt itself as it cannot see its program

UTM - • 3 tapes Machine
• each tape has R/W head
• an UTM can simulate any other TM as it can read the software.

TM program Co-tape ID storage.



EXCELLENT

A tape will read the software & co-tape will store the change accordingly & 3 tape will store the current configuration

To store the software in tape, it is converted into binary encoding

T - s/w

e(T) - binary software

In this manner computer executed

e(T) → ex object code of program
co-tape - memory
current tape - run time M/M.

Let program is given

$\delta(q_0, a) = (q_1, a, R)$

$\delta(q_1, b) = (q_2, a, b)$

e[T] =

$Q = \{q_0, q_1, q_2\}$ Let $q_0 - 1$

$\Gamma = \{a, b, \square\}$ $q_1 = 11$

(L, R) $q_2 = 111$

$a \rightarrow 1$ $\square - 111$

$\therefore e(T)$ $b = 11$

1010110111 and so on. $L \rightarrow 1$

it get stored in tape. $R = 11$

In this manner, machine executes.

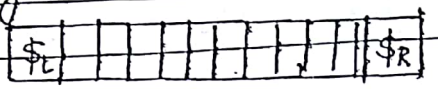
∴ It is agreed that UTM is model of computation.

1 way TM is same as FA as no need of writing as you cant access it by moving backward.

Page No. 89
Date : / /

HTM - it has less power as TM as it must have to halt.

LBA - Less power than HTM
because the working space (input tape) is restricted by left end marker & Right end marker



- M/M is bounded but not finite. it is dependent on input size
- It cannot solve Recursive f^n because require size of a program is depending not depending upon input size.

Meaning of context Sensitive - it is context sensitive because it must be non-contracting

$$\alpha \to \beta \Rightarrow |\alpha| \leq |\beta|$$

if a grammar has $\alpha \to \beta$
and $|\alpha| \to |\beta|$

then every production can be converted to the form

$$zxy \to zty$$

ie x is converted to t but left context must be z & right context must be y. that why, it is context sensitive

CFG are context free
z, y must be null. ∴ it is context free.

3. R/Write TM ≡ FA + R↔L ≡ Two way automata
power is similar to FA

EXCELLENT

Page No.
Date : / /

REC & RE Language

Definition - A Language is RE if & only if ∃ TM which accept L & halts for $w \in L$.

There are some language for which no turing machine exist

Undecidable - which can not be automated. i.e No turing machine exist.

semi-decidable - which can be automated but can go into hang.

"A Language is REC if & only if ∃ HTM which accept L & which halts for all $w \in \Sigma^*$."

Turing Machine is a software but HTM is an algorithm as it must halt in finite time.

A Language is REC if & only if ∃ HTM which accept L & which halts for all $w \in L$ (false as it has to halt for $w \in \Sigma^*$)

- For a language ; No HTM exist - may be RE or Not RE
  No TM exist - Not RE

- RE is also known as turing recognizable language or turing acceptable language or semi-decidable language.
- REC is turing decidable language or decidable language

- for a language, if finiteness is decidable, infiniteness is also decidable.
  if a problem is decidable then its complement is also decidable.

- Every Undecidable problem is Not REC but not Not RE

- Not RE language are Not even Semidecidable.
- Semidecidable will also cover all decidable problem.

EXCELLENT

(turing enumeration procedure)

if a turing ~~is~~ language has enumeration procedure, it is RE but may or may not be REC.

RE languages are also known as turing enumeration language

* L is REC iff both L & $\bar{L}$ has TEP.

• [If a L is RE but not REC then $\bar{L}$ donot have TEP.]

2 L $\bar{L}$ Theorem- if L is RE & $\bar{L}$ is also RE ⇒ both L, $\bar{L}$ must be REC

→ L is RE but not REC

i.e if L is RE but not REC than $\bar{L}$ must be Not RE (outside)

→ L is REC

if L is REC as L & $\bar{L}$ are RE (inside)

If a language L donot have TEP - Not RE but complement of that language i.e $\bar{L}$ may have TEP (i.e may or may not be)

~~Membership~~

• Enumeration procedure is one which can list member of language in finite amt of time. A language is RE which has TEP as members of RE can be listed.

• Not RE donot have TEP as No logic for its automation. but Not RE ha EP but not TEP.

• A language is REC as we can list all the member of L & $\bar{L}$ ie (all $w \in \Sigma^*$) i.e REC will always halt.

• Every language is enumerable

(RE) TEP \ EP (Not RE)

EXCELLENT

Date : / /

Every language is enumerable

proof-

Sets → Uncountable set - UC Infinite (complex No, Irrational no. (Set of Real No))

Sets → Countable set → Countable Infinite, countable finite (finite)

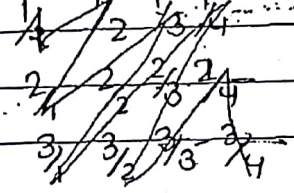
(N, Z, P) Set of natural No (as discret set i.e discrete number we can find distance & can map)

Language is countable (discrete) iff it has EP i.e list of member can be obtained in finite time. (if someone give a natural no we can print it out)

Set of real No cannot be printed so it is limited from logic

Irrational No are uncountable infinite

Rational No are countable infinite because



go to Zigzag and so on you can find it out

UCI × UCI = UCI

Uncountable infinite set are continous but $C \times C \rightarrow C$..

not discrete

- logic can take discrete objects only.

6 facts-

1. Every language is countable $L \subseteq \Sigma^*$ is C
2. Every subset of countable language is countable

proof- Σ^* is

for Σ^* EP - list To device EP we use properi ordering by length

Σ^* 0 - ε

1 - a, b

EXCELLENT

so in finite amount, string of length n, inputted by user can be listed so they can be listed.
. list is discrete too
. Σ^* is countable Hence all language are countable

Page No. 131
Date

as $L \subseteq \Sigma^*$

Σ^* is countable then it is discrete so why each logic can be written?
bcoz recognizing a language is a task of accepting something & rejecting something ie a set of language which is a more deeper concept as it can lead to Not RE

2. L_{reg}, L_{CFL}, L_{CSL}, L_{RE}, L_{REC} is Countable Infinite.
3. L_{NOTRE} is Uncountable Infinite
4. 2^{Σ^*} is UCI.
- Every axiomatic system (probability, geometry) is incomplete set of Not RE is uncountable infinite bcoz problem set is not discrete then soln set is not discrete as well hence we can not count the problem hence no soln can also be devised.

- A 'Not RE' language is countable but set of Not RE is not countable as problem set is infinitely continous but a finite set of Not RE is countable.
L_{reg} - it represent set of regular language
- 2^{Σ^*} - Set of all languages on an alphabet is UCI
5. Set of formal language is Countable Infinite.

6. Set of Non-regular language is UCI as it also contains Not RE
7. $L_{Non-reg}$, $L_{Non-CFL}$, $L_{Non-CSL}$, $L_{Non-REC}$ is also UCI.

- $2^I, 2^Z, 2^P$ is UCI

EXCELLENT

as well as Non-Member.

L is RE iff L has TEP.

We donot have MA for RE has it donot guarantee to halt on every input.

If L has MA surely $\bar{L}$ has MA.

If L has TEP & $\bar{L}$ may or may be TEP.

if $\bar{L}$ has MA, then $\bar{\bar{L}}$ is also have MA.

If L donot have MA it may or may not be RE but Not REC.

3. Lexicographic ordering - L is REC iff L has lex ordering.

iff a language can be enumerated through lex ordering, it is surely REC.

4. Closure properties - if L is RE then $\bar{L}$ may or may not be RE

if L is REC then $\bar{L}$ is surely REC

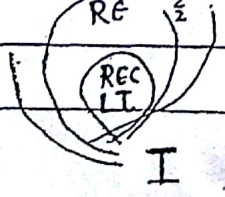
if $\bar{L}$ is RE L may or maynbe RE

if $\bar{L}$ is REC L must be REC.

5. L, $\bar{L}$ theorem - if both L, $\bar{L}$ are RE then both are REC as it halt on all string of L, $\bar{L}$

Ques if L, $\bar{L}$ are two complementary languages

1. Both are REC
2. one is not RE & other is RE but not REC
3. both are not RE



II case signify that one is RE but not REC so you can not put $\bar{L}$ in REC as it must not REC also L can not be placed in RE as EXCELLENT if L, $\bar{L}$ both is RE it became RE.

III

Cantor's theorm

Page No. 132
Date: / /

$2^S \rightarrow UCI$

for L, $\bar{L}$ only three case

- if L is RE but not RE, then $\bar{L}$ is surely 'Not RE' bcoz if $\bar{L}$ became RE, then L, $\bar{L}$ became REC
- if L is Not RE then $\bar{L}$ May be RE or not RE

for L, $\bar{L}$

- one is RE & other is REC - possible as RE is also REC
- both are RE as RE is also REC
- both are REC - possible
- both are Not RE - possible
- both are RE but not REC - impossible if both are RE then L is REC.
- if L is not DCFL then $\bar{L}$ is surely not DCFL

ExaMpLeS oF ReC, RE, NoT RE -

RE but not REC - L_u = Semidecidable but not decidable

~~Not RE~~ L_u - universal language of Turing Machine

Not RE - L_d - diagonalization language (as based on Cantor's Diagonalization theorem)

$L_d = \{ T \mid e(T) \notin L(T) \}$ set of all turing machine which have $e(T) \notin L(T)$

$e(T)$ - encoding of T

L_d - Not RE - Machine which rejects in own coding.

$L_u = \{ T \mid e(T) \in L(T) \}$ - Machine can accept it own encoding

So a machine even not accepting its encoding, so no machine is designed as no logic exist for its encoding.

Ques - A turing machine which can list the machine which rejects its own code exist ?-

No it is L_d Not RE

EXCELLENT

$\overline{L_4} = L_d$

as L_4 is RE but not REC so its complement is Not RE only because if it became RE then both become REC.

L_d = Not self accepting. And L_4 = self accepting.

$\overline{T_d} = L_u$

as all machine which accept its coding will be complement of L_d.

Ques- 1. This sentence is true. — True (logical)

2. This sentence is false. — Not true as not logical)

Reduction of Algorithm- if $P_1 \leq_p P_2$ if P_1 is reducible to P_2 in polynomial time if P_1 is decidable then P_2 may or may be decidable

If P_1 is UD then P_2 is UD

If P_2 is decidable then P_1 is also decidable

If P_2 is decidable then P_1 can be decidable or semidecidible

$\Leftarrow$ $P_1 \leq_p P_2$	$\Rightarrow$ (contrapositive) is
P_1 decidable $\Leftarrow$ P_2 decidable	P_1 UD $\Rightarrow$ P_2 UD
P_1 RE $\Leftarrow$ P_2 RE	P_1 Not RE $\Rightarrow$ P_2 Not RE
P_2 SD $\Leftarrow$ P_2 SD	P_1 Not SD $\Rightarrow$ P_2 Not SD
P_1 REC $\Leftarrow$ P_2 REC	P_1 Not REC $\Rightarrow$ P_2 Not REC
$P_1 \in P$ $\Leftarrow$ $P_2 \in P$ (i.e. polynomial time problem)	$P_1 \notin P$ $\Rightarrow$ $P_2 \notin P$
$P_1 \in NP$ $\Leftarrow$ $P_2 \in NP$	$P_1 \notin NP$ $\Rightarrow$ $P_2 \notin NP$
	P_1 is NP Hard $\Rightarrow$ P_2 also is NP Hard

contrapositive & direct statement

$p \Rightarrow q$

&

$\neg p \Rightarrow \neg q$

1. Halting Problem - Given a turing machine T, $w \in \Sigma^*$ can you comment whether machine will halt or not?

It is undecidable as can not be predicted whether it will halt or not.

2. Blank Tape Halting Problem (BTHP) - it is also undecidable.

Given a blank tape to TM, to ~~wt~~know/predict whether it will halt or not is undecidable.

3. State Entry Problem - Given a Turing machine T, a string w & a state q. is it possible to decide that T will enter in q while processing w? it is undecidable

4. Post Correspondence problem (PCP) - Given two set of strings with equal No of strings

$u_1 u_2 u_3 \quad u_4$

A = (100, 11, 0, 000)

$v_1 \quad v_2 \quad v_3 \quad v_4$

B = (0, 1000, 1, 1100)

Can you develop analgorithm to obtain PCP problem?

it is undecidable.

PCP solution - to check whether a combination of string in A is equivalent to string in B.

$u_2 = v_2$

$u_1 u_2 = v_1 v_2$

1000 = 1000

the combination cant be predicted as there exist infinite no of sol^n.

EXCELLENT

5. MPCP (Modified PCP) - it is fixed that combination will start with u_1 & v_1 respectively
if PCP sol^n is there MPCP may or may not be there
if MPCP sol^n is there PCP will always exist.
it is also undecidable.

- PCP & MPCP are decidable on unary alphabet because algorithm can be deviced easily.

- If every string in second set is bigger than all string in set A - No PCP exist
- If string in set A are smaller than some string in set B - then PCP exist.

- is whether will halt in finite no of steps? It is decidable (Turing)
ie Halting of Machine in finite steps is decidable.

Ques - Whether a s/w will produce o/p or not is
1. Decidable
2. Undecidable ~~or~~ but not Semidecidable
3. ~~Or~~ Semidecidable

- A Machine which print Π is Undecidable as it is a s/w.
- An Algorithm will produce o/p is - trivial decidable.

6. RE Membership - It is ~~decidab~~ undecidable.

RICE'S THEOREM - Every ~~Any~~ Non-trivial Question about RE is undecidable.

ie it is undecidable for all 8 properties.
Non-trivial question - where logic require.

EXCELLENT

$= \{a^m b^n a^p \mid m = n \text{ \& } n = p\}$

$\overline{L}$ ⇓

$\{a^m b^n a^p \mid m \neq n \text{ or } m \neq p\}$

⇓ it is CFL as double comparison with OR.

if a CSL with double comp. with OR ⇒ $\overline{L}$ will be CSL

——— AND ⇒ $\overline{L}$ will be CFL.

$L = \{a^{n_a} = n_b = n_c$ ∴ its complement

$\overline{L} = \{n_a = n_b \text{ or } n_b = n_c\} = CFL$

$ww \rightarrow CSL$

$\overline{ww} \rightarrow CFL \Rightarrow \{w_1 w_2 \Rightarrow w_1 \neq w_2\}$

$ww^R \rightarrow CFL$ PDA can do = as well as ≠

$\overline{ww^R} = CFL$

In straight order Matching is not possible ie ww but mismatching is possible $\overline{ww}$

- A grammar i.e regular is unambiguous then conversion to machine will give NFA
- Ambiguous grammar will always generate NFA but NFA may or may not be ambiguous grammar. DFA will produce unambiguous grammar.
- DPDA will always generate unambiguous grammar.

43- $L = \{a^n b^n \mid n \geq 0, n \neq 13\}$

$a^n b^n - \{a^{13} b^{13}\}$

$L - R = L$ as R is regular

∴ it is a DCFL

44. $\{a^n b^n\} - \{a^{3n} b^*\}$

$L - R$ = Regular

EXCELLENT

(id ⊕ (id ⊕ id) - right

$X = (X \oplus Y)$

$= ((X \oplus Y) \oplus Y)$

$= ((X \oplus Y) \oplus (Z * Y)$ ⟶ ⊕-on (left-bracket - LA

$= ((X \oplus Y) \oplus (Z * (Z * Y))$ *-RA

If both side bracket are allowed
↳ it is both
LA & RA

78- $E \to E*F \mid F+E \mid F$

$F \to F \$ F \mid id$

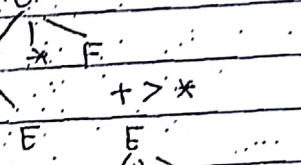
1. * > +
2. − > *
3. + = −
4. + > *

id * id − F

* > − > +

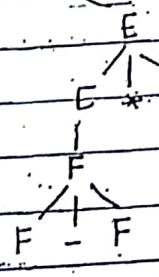
(id * id) + id

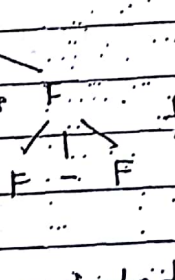
E → E * F



∴ * = +

(id − id) * id





for * you have to go to

−>*

99 Rank of a Non terminal - only related to CNF

to find range rank of B

$S \to aA$ $S \to CaA$ ⇒ $S \to CaA$

$A \to BC$ $A \to BC$ $A \to BC$

$B \to bAd$ $B \to CbACd$ $B \to CbD$

$B \to CbD$ $Ca \to a$

$Cb \to b$

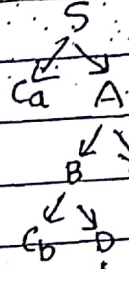
$Cd \to d$

$D \to ACd$

CNF

1) convert to CNF

Rank of B = longest path to leaf from that



finite. Here A=∞ so it is

If Rank of a ~~language~~ some variable is infinite, language may or may not be infinite as L=∅ for S→AA, A→AA

103- S→aSa|B

B→ε|bBa

⇒ $a^m b^n a^n a^m$

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

∴ length of every efficient string = 2m+2n

2m+2n ≤ 9

m+n ≤ 4.5

0	4	= 4 = 5 so 1^n
1	3	for = 3 ⇒ 4 so 1^n
2	2	for 2 ⇒ 3 so 1^n
3	1	1 ⇒ 2 so 1^n
4	0	

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

2n+2m < 10

n+m < 5

n+m ≤ 4

n+m ≤ 4

0 1 2 3 4

1 2 2

0 1

Ques- S→Aa|Bb|Cd|AB

A→aA|bB|eC|BC|d|E find nullable variable

B→A|aA|B|ε B, C, A, S

C→aC|d|ε

E→fg

$(\varepsilon + 1 + 111^*) = 1^*$

Decidability - $(L_1 = L_2\ \ L_1 \subseteq L_2\ \ L_1 \supseteq L_2\ \ L_1 \subset L_2\ \ L_1 \subset L_2)$ for regular & undecidable for every non-regular.

$L_1 \subseteq L_2$ iff $L_1 - L_2 = \phi$

~~$L_1 \oplus \phi$~~ $L_1 = L_2$ iff $L_1 \oplus L_2 = \phi$.

$L_1 \subset L_2$ ⇒ iff $(L_1 - L_2 = \phi)$ & $(L_1 \oplus L_2 \neq \phi)$

Intersection over CFL is not decidable.

∩-in closure property means undecidable.

- DCFL decidability- equivalence is decidable

subset & ~~ambiguity~~ is undecidable

ambiguity is decidable

- Disjointness is not decidable in DCFL. EXCELLENT

$L_1 \cap L_2 = \phi$ (X)

Date: / /

- Regular & Grammar & DCFL grammar
- Grammar Ambiguity in RL is decidable.
- DCFL Grammar Ambiguity is ~~not~~ decidable in CFL Ambiguity of Grammar & language both are undecidable

- Palindrome over unary alphabet is regular language.

Ques: which of following is CFL?
1. palindrome over unary
2. $\{a^m b^n c^p \mid m<n \text{ or } n>p\}$

Ans - both are CFL.

Ques $w\#w^R \quad w \in \{1\}^*$
↓ Not a Regular language but DCFL
as u have to count no of 1's in given w.

$w\#w \quad w \in \{1\}^*$
↳ ~~CSL~~ CFL not a CSL as w is on unary alphabet.

ww - regular on $w \in \{1\}^*$

- Both P. & NP comes under REC problem.
- So, Every P & NP ∅ is decidable.

Reduced CFG - (without useless Symbol)
Simplified CFG - CFG without ε, unit production & useless symbol

Proof - for $L_{RE}, L_{REC}, L_{reg}, L_{DCFL}, L_{CFL}, L_{CSL}$ is CI
to prove L_{RE} is CI
$L_{RE} = \{L_1, L_2, \ldots L_3\}$ prove it is discrete.
to prove it is CI prove set of L_{RE} is also a language.
$L_1 \Rightarrow e(T_1)$ (binary encoding of T_1 which accept L_1
$L_{RE} = \{e(T_1), e(T_2), \ldots\}$
this set of program is discrete as encoding get convert to string & set of string can belong to L.
L_{RE} is a language & a language is CI so L_{RE} is CI & Hence RG, CFL, CSL are also RE
Hence they are also CI.