

5-14

- DFA do not allow Null move

Minimal DFA is unique

NFA Design & Page Design
(Remove trap from DFA)
Date: / /

Chapter-2

Regular language & Finite Automata

1. FA Theory	
2. FA Design	$[L \rightarrow M, M \rightarrow L]$
3. Regular Expression	language corresponding to a expression
4. Regular Grammar:	$L \rightarrow \pi$ $L \rightarrow GL$ $G \rightarrow L$ $\pi \rightarrow L$

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 Constⁿ algorithm)
- DFA to min. dfa (minimization of Automata)
- E-closure
(NFA with E 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

language is regular but each regular language is not finite
ex - (a^*)

Page No.

Date : / /

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 fn (δ)
4. δ^* : extended (δ^*) transition fn
5. L(M) (Language representation of machine)
6. Theorem in FA

δ is used for mapping.

Ways to write transition fn

- 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(M) - Language accepted by complementing machine.

- finite language is always Regular.

- Pumping Lemma to show a lang. is not reg.

- MYHILL NERODE THEOREM

- KLEEN'S THEOREM

EXCELLENT

Pumping lemma is the property of regular language i.e. you cannot use pumping lemma to show whether a lang. is regular (one way theorem)

Page No. _____

Date : / /

- MYHILL NERODE theorem is used to prove whether a language is regular or not. (it is two way theorem)
- KLEENE'S Theorem - if a regular expression exist, you can make a (bidirectional) machine
- if a FA is not possible of given RL, then it is not RL.

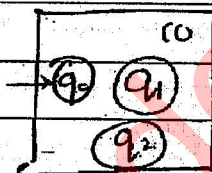
DFA & NFA specification - Automata

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

Q → finite non-empty set of internal states
 Σ → Input Alphabet finite & nonempty
 δ → transition function
 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 because it remembered only through moving of states.

(Ex)



$Q = \{q_0, q_1, q_2\}$
 $\Sigma = \{a, b\}$

Limitation -

- finite state
- finite M/M

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

EXCELLENT

δ -state transition function-

- Always finite
- No of command is finite

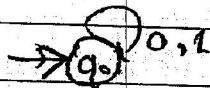
- In DFA, δ can ~~be~~ not be empty but in NFA, δ can be empty.

ex- for $L = \phi$

NFA



DFA



(because in DFA, each state must have 1 o/p for ~~DFA~~ each in symbol.)

NFA accepting $L = \epsilon$



- IN DFA, ϵ moves are not allowed.

Final state - $F \subseteq Q$

- accepting state
- represented by $\odot$
- finite state
- can be empty

Diff. b/w NFA & DFA-

δ for DFA- $\delta_{q \times z} \rightarrow q$

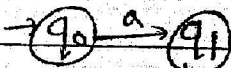
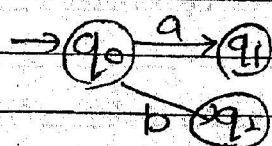
combination for current state with current I/P symbol will produce a resulting state

δ for NFA- $\delta_{q \times (z \cup \epsilon)} \rightarrow 2q$

{ Each DFA is also a NFA }

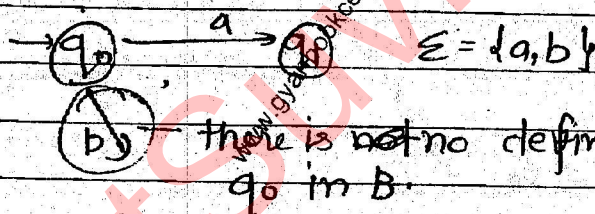
Page No.

Date : / /

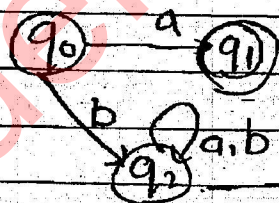
	DFA	NFA
1.	No choices allowed 	Choices are allowed 
2.	No Null Moves Allowed	2. Null Moves allowed, known as ϵ -NFA
3.	No Dead Configuration	Dead Configuration Allowed

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

in DFA,

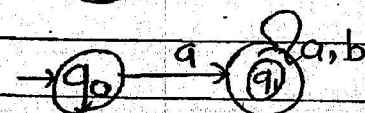


to do this, instead of Defining DC, use non final state



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

but in NFA,

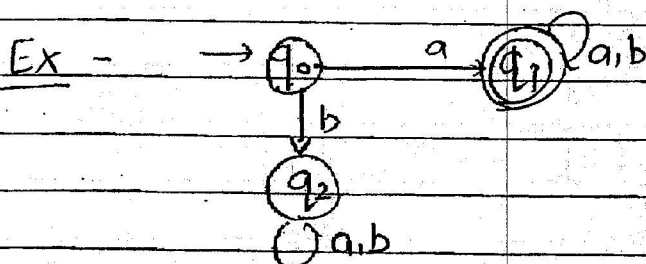


if "ba" comes

it get rejected in NFA

EXCELLENT

• (No of states in minimal NFA) $\leq$ (No of state in minimal DFA)



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

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

in DFA I can only give

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

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

$\Downarrow$

2^Q

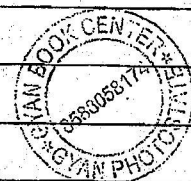
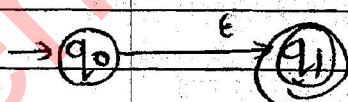
i.e., we use 2^Q here

as $2^Q = \text{power set of } Q$

$$= \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \dots\}$$

• Every DFA is a NFA but not vice versa.

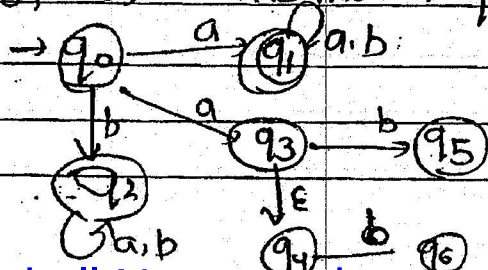
Null Moves



$\downarrow$ It means a spontaneous jump from one state to another without reading an input.

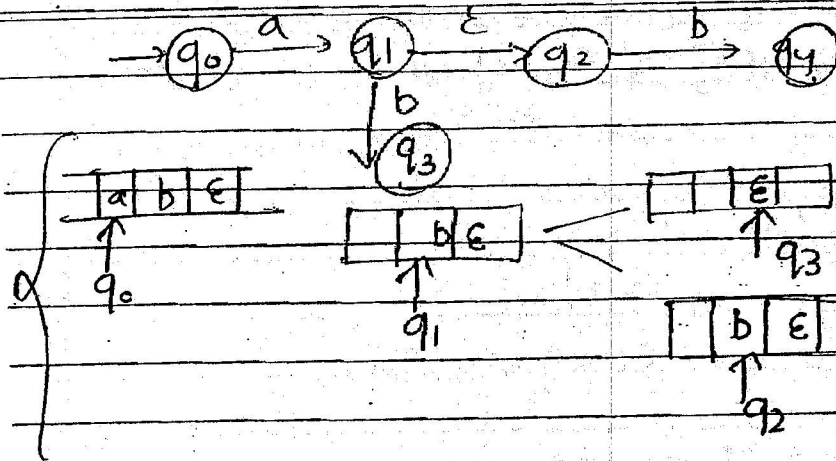
• Null cannot be the part of I/P tape (it can be only at last of tape for showing end of tape)

• $\delta^*(q_0, ab) \rightarrow$ 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 go to ϕ it can go to ϕ as $\phi \in 2^Q$

$$\delta(q_0, b) = \phi$$

can reject

δ^* for DFA

$\delta^* q x \leq^* \rightarrow q$ for DFA

as strings can be provided

$(\delta^* q x \leq^* \rightarrow 2^Q)$ for NFA

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

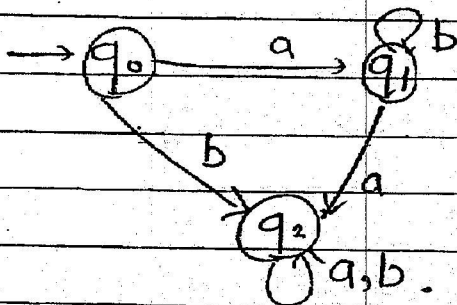
$$\delta(q_1, \epsilon) = \{q_1, q_2, q_3\} \rightarrow q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} q_3$$

if ϵ is not specified as a move

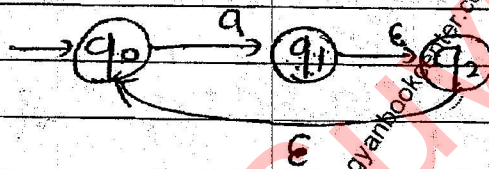
$$\delta(q_1, \epsilon) = q_1 \text{ as no move has done}$$

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 D



Ques



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

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

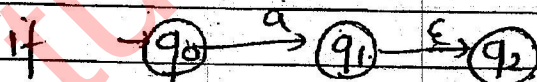
$$\delta^*(q_0, \epsilon) = \{q_0\}$$

$$\delta^*(q_0, aqa) = \{q_0, q_1, q_2\}$$

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

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

$$\delta^*(q_1, aqa) = \{q_0, q_1, q_2\}$$



$$\delta^*(q_1, a) = \emptyset$$

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

because it must have to be called on q_1

for equivalence of two strings, it must start from initial state

Page No.

Date: / /

Properties of δ^*

1. $\delta^*(q, \epsilon) = q$ (in DFA)
but in NFA

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

DFA

NFA

2. δ^*

$$1. \delta^*(q, \epsilon) = q$$

$$1. q \in \delta^*(q, \epsilon)$$

$$2. \delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$$

as only one path possible

if

$$2. \delta^*(q, u) = Uq_i \quad i=1 \dots n$$

moving to i state on u

$$3. \delta^*(q, uv) \neq \delta^*(\delta^*(q, v), u)$$

for v

$$\delta^*(q, uv) = \bigcup_{i=1}^n \delta^*(q_i, v)$$

$$= (q_1, v) \cup (q_2, v), \dots$$

$$4. \text{if } \delta^*(q, u) = \delta^*(q, v)$$

then,

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

i.e., DFA is Right invariant

but it is not left invariant

$$\text{i.e. } \delta^*(q, zu) \neq \delta^*(q, zv)$$

if implies

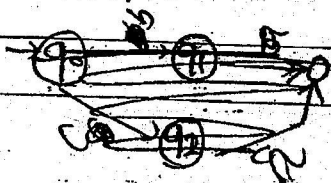
$$\{u \equiv v \Rightarrow uz \equiv vz\}$$

(it must start from initial state)

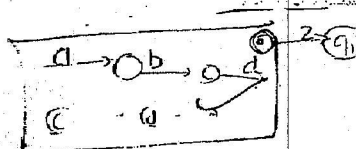
$$\text{if } \delta^*(q, uz) = \delta^*(q, vz)$$

then it is not always

$$\delta^*(q, u) \neq \delta^*(q, v)$$



EXCELLENT



Page No.

Date : / /

Ques -

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

$$\text{i.e. } qu \equiv v \text{ \& } uz \equiv vz$$

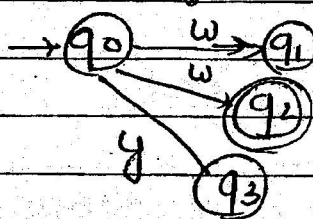
$L(M)$ - Language accepted by a machine-

For DFA, All string which push machine to final state

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

for NFA

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



$$\Rightarrow \{q_1, q_2\}$$

$$F = \{q_2, q_3\}$$

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

so we take intersection

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

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

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

EXCELLENT

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

$$\delta^*(q_0, w) = \emptyset \text{ (if DC occurs)}$$

$$\boxed{\emptyset \neq \emptyset \text{ (Rejected)}}$$

Theorems-

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

~~the language accep~~

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

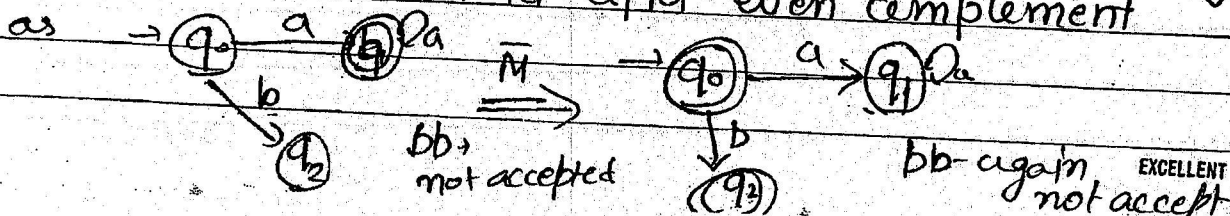
• It is useful for machine designing for complement of a condition.

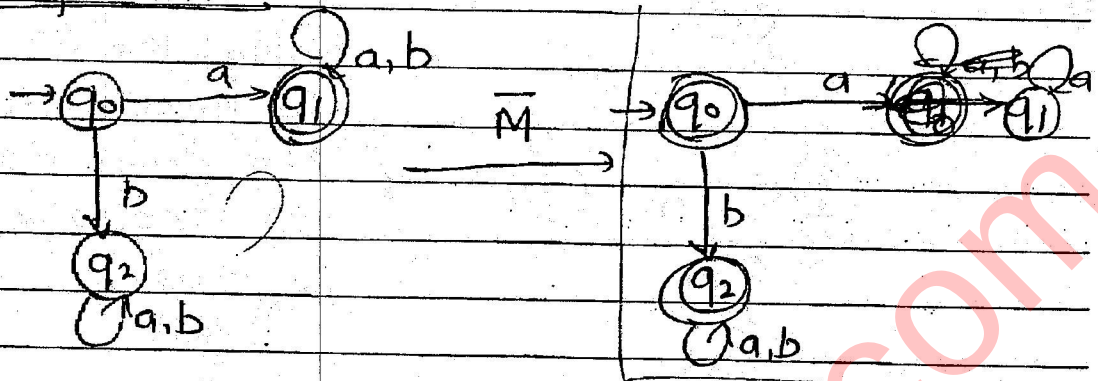
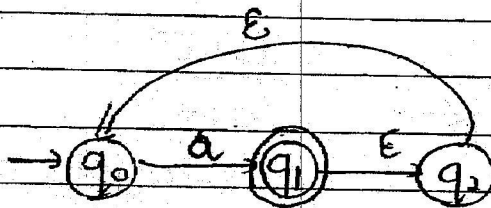
Ex - not starting '001'

[for Complementing a Machine - Convert all final state to Non final state & non final state to final state]

It is possible becoz in DFA because Acceptance & Rejection is one way through final state

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



Example for DFAQues -

What is complement of language accepted by machine

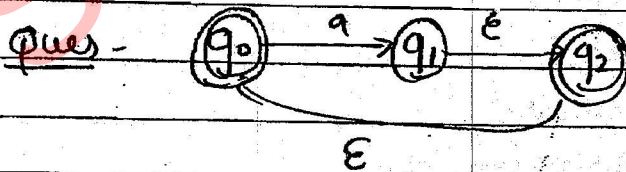
[Not possible directly as it is NFA, First convert in to DFA] as complement is possible for DFA by using that theorem.

You can find the language & then complement it.

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

$$\begin{aligned} \overline{L(M)} &= a^* - a^+ \\ &= a(\epsilon) \end{aligned}$$

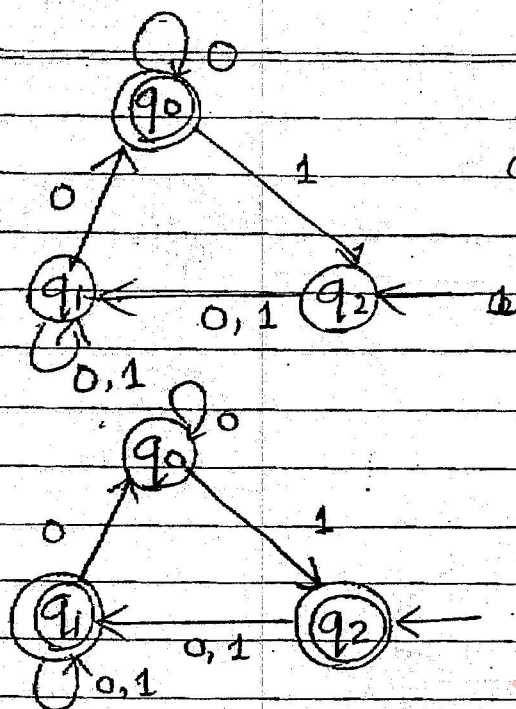
ϵ is not accepted as not input ϵ is given.
as q_0 is not final state



$$\begin{aligned} \overline{L(M)} &= a^* \\ \overline{L(M)} &= \emptyset \end{aligned}$$

(as ϵ is accepted)

Ques -



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

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

$(0+1)^*$

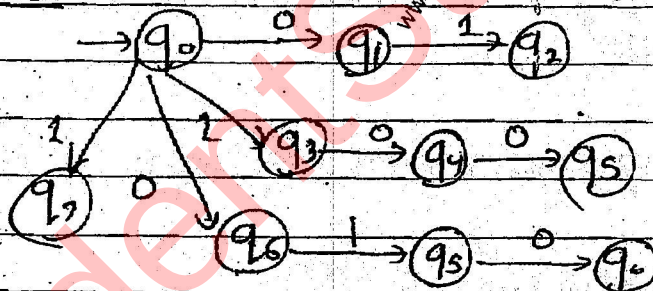
Do not apply that
theorem as it is NFA.

Theorem

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

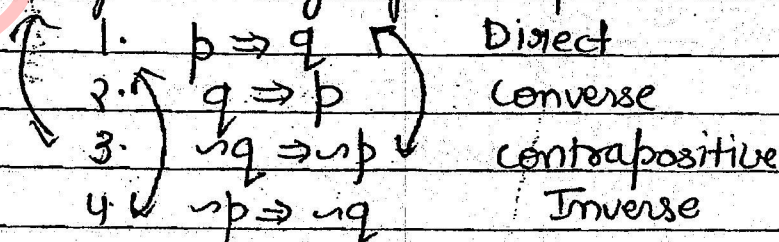
01

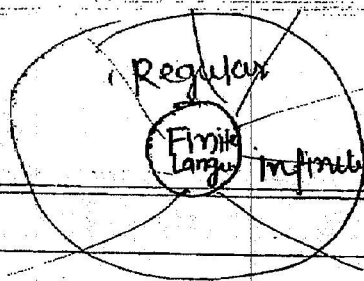
$L = \{01, 100, 010, 1\}$



$\{ \text{FINITE} \Rightarrow \text{REGULAR} \}$

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





Page No.

Date : / /

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

Theorem 2 for every regular language FA is possible.

$$REG \iff \exists FA$$

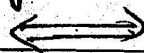
- for every non reg. FA is not possible
- for every reg. FA is possible.

Theorem 3 Pumping Lemma

if L is Regular surely L will satisfy the pumping lemma for Regular lang.

- it is one way theorem.
- 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.

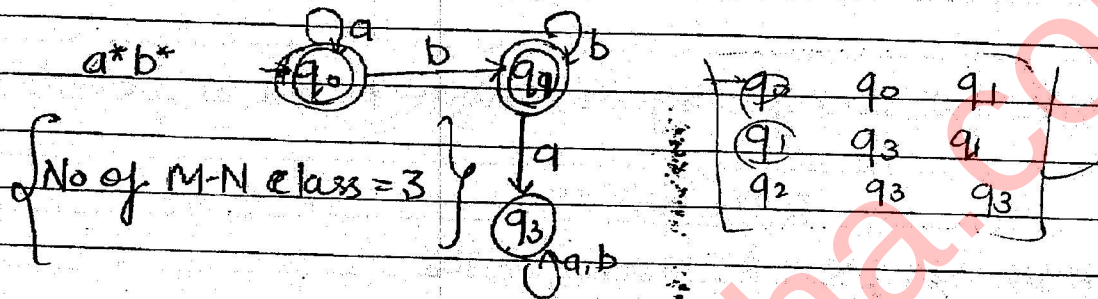
- two way theorem

- Each state in minimal DFA corresponding to Myhill-Nerode equivalence class
- So since no of state is finite L is regular becoz 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 (∞)
 as it is not regular



[Myhill Nerode class cannot be connected with NFA.]

Myhill-Nerode class accept string & 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 $\Leftrightarrow \exists$ FA which accept L .

$\Leftrightarrow \exists$ dfa which accept L .

- Every Machine that accept ~~FA~~ regular is FA.

↳ False

(as PDA, TM can accept FA)

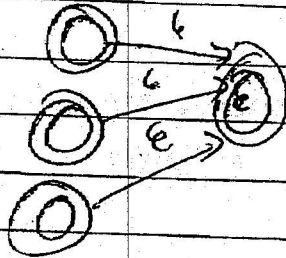
~~as~~ L is regular if exist dfa :- true

as NFA can be converted to dfa.

• L is Regular $\Leftrightarrow \exists$ nfa with single final state

$\hookrightarrow$ Yes

(bcz each Nfa with many final state can be converted to NFA with single final state by using ϵ -move)



• L is Regular $\Leftrightarrow \exists$ dfa with single final state

$\hookrightarrow$ False

• L is Regular $\Leftrightarrow \exists$ NFA without Null Moves

$\hookrightarrow$ Yes (from given NFA, null move can be removed)

• L is Regular $\Leftrightarrow \exists$ NFA with DC

$\hookrightarrow$ Yes (it is equivalent NFA by increasing the state)

• L is regular $\Leftrightarrow \exists$ NFA without choice

$\hookrightarrow$ Yes

choice, Null, DC are the luxury to NFA

• DFA, NFA both have same power

• L is regular $\Leftrightarrow \exists$ g.e. for $L \rightarrow$ (Yes)

• $nfa \equiv dfa \Leftrightarrow$ g.e. $\exists$ g.e. (have same power)

• L is regular $\Leftrightarrow$ g.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

→ Formal Grammar

↳ also known as regular 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 RE - Not Machine no grammar (Informal Language)

RE 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.

for $u \rightarrow v$

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

✓ U must contain atleast one variable

bcuz only terminals cannot be expanded.

$$aa \rightarrow x$$

$$QA \rightarrow \checkmark B$$

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

Type 1 Grammar - at each step, there must be an expansion or it may be same but can not contract.

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

2 | 4 | 5 | 6 |

as $BaA \rightarrow aB$
 $|4| = 3$ IV

So Not a type 3

* Contraction is not allowed in type 1, but it allows $S \rightarrow \epsilon$

for type 1-

$$\underline{u \rightarrow v}$$

$$y, v = (v u T)^*$$

but $v = \epsilon$ if $u = \text{single terminal}$
(start symbol)

ie $qS \rightarrow E$ (not allowed)

$S \rightarrow \epsilon$ is allowed only if S not appear in RHS of production

production $S \rightarrow OBA$

Ex - $AB \rightarrow qS$

3-5

false

EXCELLENT

as compiler set limit ~
if expand to will be reach
then terminate

an
a b c d e / term.

Page No.

Date : / /

- Compiler will not work if there is contraction, properly because if contraction occur compiler will not find or satisfy the condition to terminate.

agar har baar expansion hoga to zaror compiler ki limit tak pahuch jayega

Type 2 Grammar -

for $u \rightarrow v$

$u \in V$

$v \in (V \cup T)^*$

$v \in (V \cup T)^*$

$\left\{ \begin{array}{l} S \rightarrow aAB/\epsilon \\ A \rightarrow ab \\ B \rightarrow c \end{array} \right\} + (\text{Type 1}) + (\text{Type 0})$

- it must also be non contracting

Type 3 Grammar -

Type 2 + Type 1 + Type 0

for $u \rightarrow v$

$v \in T^*v + T^*$

$v \in VT^* + T^*$

Right Linear Regular Grammar

Left Linear Regular Grammar

RLRG $A \rightarrow aaB/aa$

LLRG $A \rightarrow Baa/aa$

ie right hand side can contain only one variable.

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

$\left\{ \begin{array}{l} S \rightarrow aaB/aa \\ B \rightarrow ccD \\ D \rightarrow Ha \end{array} \right\} \rightarrow \text{Not regular Grammar as both LLRG \& RLRG}$

$A \rightarrow B \rightarrow \text{allowed in Type 3}$

EXCELLENT

• Every regular grammar is right linear or left linear

cont....

• L is regular $\Leftrightarrow \exists$ Right linear regular grammar on L

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

It implies Right Linear RL & Left Linear LL both have same power.

• L is regular $\Leftrightarrow \exists$ reg without recursion-

False because recursion is * (it is necessity)