# Unit 1

DATE: __/__/____
PAGE: _____

## Theory of Automata & Formal Language

Theory of Automata ↔ Hardware

→ theory of abstract computing device

1. Finite Automata 1943
2. Pushdown Automata 1967
3. Linear Bounded Automata 1960
4. Turing Machine 1936

↓ powerful

Formal Language ↔ Software

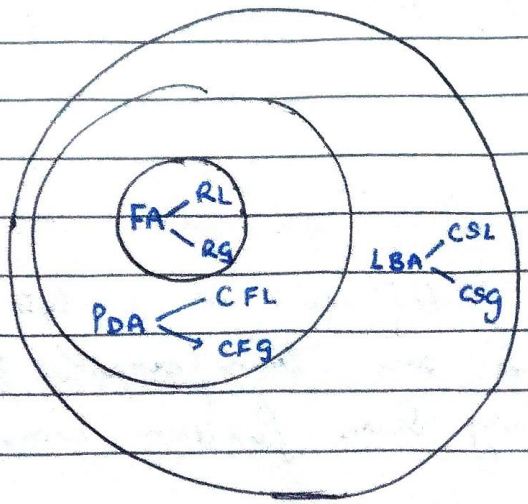
a Language consist of
→ grammar
→ character set

Types:

1. Regular language & grammar
2. Context free lang. & grammar
3. Context sens. lang. & grammar
4. Recursive enumerable lang

↓ powerful & grammar

~ by Noam Chomsky




## > Alphabets, String & Language:

Set of Symbols — denoted by Σ

String: list of symbols from alphabets

Set of strings

Say for binary $\Sigma = \{0, 1\}$

i) Empty string: $\epsilon$

ii) Length of string: length

iii) Power of an alphabet: $\Sigma^k$

Say for $\Sigma = \{a, b, c\}$

$\Sigma^1 = \{a, b, c\}$

$(len)^k = \Sigma^2 = \{aa, bb, cc, ab, bc, ac, ba, cb, ca\}$

$\Sigma^3 = \{aaa, aab, aba, baa, \ldots\}$

Set of all strings over $\Sigma$ is denoted by $\Sigma^*$

$= len^1 + len^2 + \cdots + len^{len}$

$= len^1 (len^{len} - 1) / (len - 1)$

DATE: __/__/____
PAGE: _____

> $\Sigma^*$ : All strings possible

Language: $L \subseteq \Sigma^*$

eg lang consist of $n$ 0's followed by $n$ 1's for some $n > 0$

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

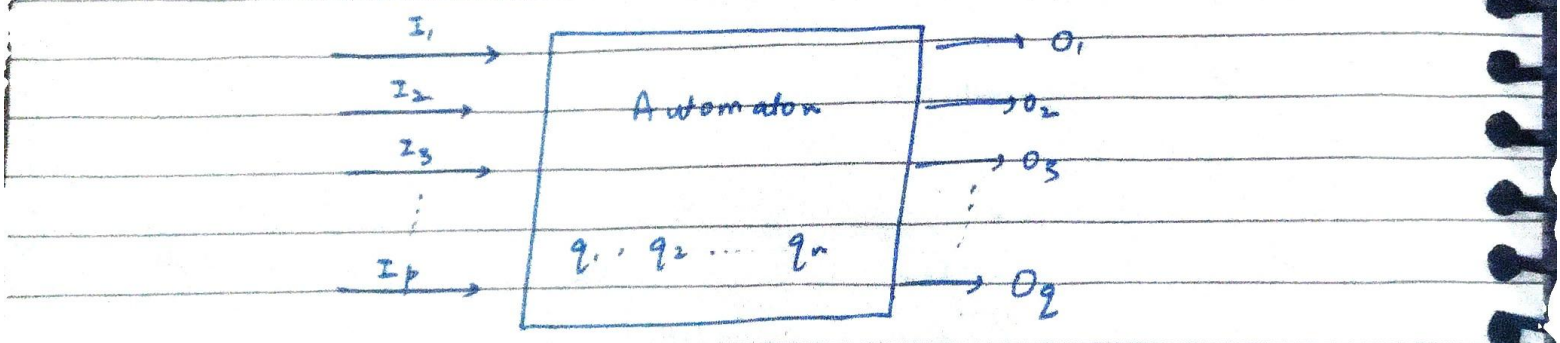
$L = \{ \epsilon, 01, 0011, 000111, \ldots \}$

eg set of binary numbers whose value is prime.

$L = \{ 10, 11, 101, 1011, \ldots \}$

## * Automata:

an automata is defined as a system where energy, materials, and info. are transferred, transmitted and used for performing some function without direct human participation eg: Automatic machine ; Automatic packing machine etc.



DATE: / /
PAGE:

## Characteristics of Automata

1. **Input**: finite number of input form $\Sigma$ (Alphabet set)
2. **Output**: finite number of Output
3. **State**: At a instant of time automaton can be in one of the states i.e. $q_1, q_2, \ldots, q_n$
4. **State relation**: Relation b/w present input and state.
5. **Output relation**: r/n b/w output & [either state only or both input & state]

| Output depends | input | State | machine | eg: |
|---|---|---|---|---|
| | ✓ | × | without mem. | DFA, NFA |
| | ✓ | ✓ | with mem. | PDA |
| | × | ✓ | moore machine | LBA |
| | ✓ | ✓ | melay machine | TM |

[at any instant of time]

→ **Grammar**: mathematical model given for computer.

↳ 4 tuples: $G = (V, \Sigma, P, S)$

i) V: finite non-empty set whose elements are called non-terminal

ii) $\Sigma$: finite non-empty set whose elements are called terminal

iii) $V \cap \Sigma = \phi$

iv) S: Special non-terminal $(S \in V)$ ∴ Start symbol

v) P: Set of production rule eg: $\alpha \to \beta$

$\alpha$: any combo of terminals & non-terminal (min - 1)

**Note**: i) Reverse Substitution not permitted

Good Write given $S \to AB \Rightarrow AB \to S$ ✗ Not allowed.

DATE: __/__/____
PAGE: _____

eg. $G = (\{S, A, B, E\}, \{a, b, c\}, P, S)$ & P: $S \to AB$, $A \to a$, $B \to b$, $E \to c$

| | | |
|---|---|---|
| {S, A, B, E} | Non-terminal | V |
| {a, b, c} | Terminal | $\Sigma$ |
| AB | S | S |

» BNF Notation

<Symbol> ::= _expression_

<Symbol> → non-terminal

_expression_ → consist of one or more symbols seperated by "|".

eg:

<number> ::= <digits> | <number> | <digit>

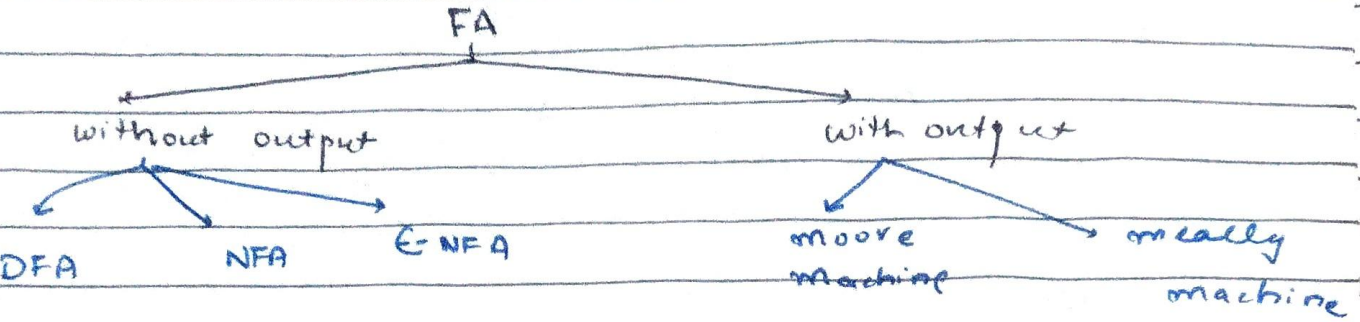
<digits> → 0|1|2...|9

---

» ~~Chomsky hierarchy~~

DATE : __/__/____
PAGE : _____

> Finite Automata has a set of states & its control move from state to state in response to external 'input'.

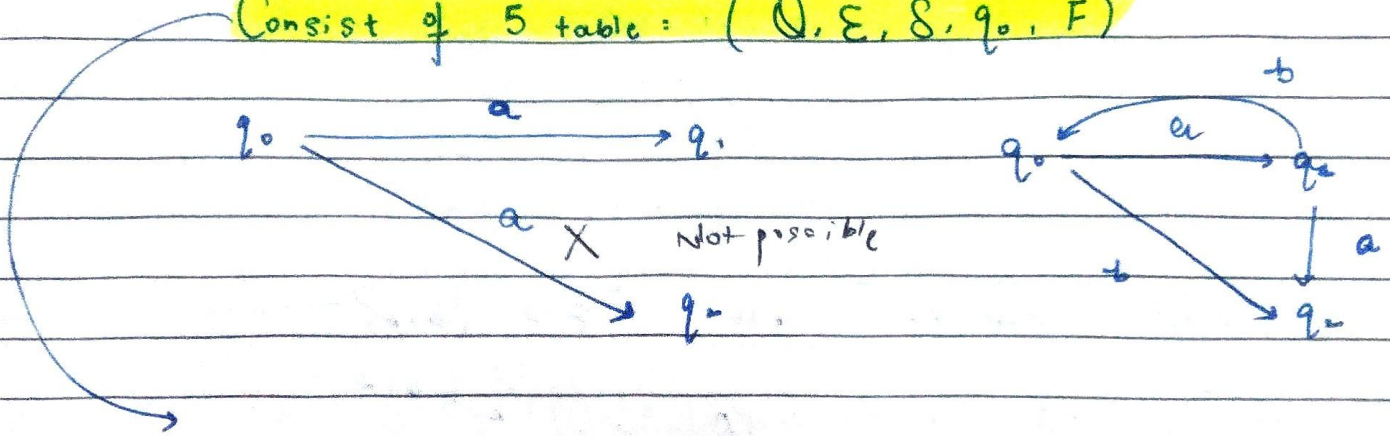



- **DFA**

[ Deterministic Finite Automata ]

In DFA on each input there is only one state which the automata can transition from its current state.

Consist of 5 table : $(Q, \Sigma, \delta, q_0, F)$




$Q$ : finite non-empty set of states $\{q_0, q_1, q_2\}$

$\Sigma$ : finite non-empty set of input

$\delta$ : transition function eg: $\delta(q_0, a) = q_1$

$Q \times \Sigma = Q$

$q_0$ : Initial State

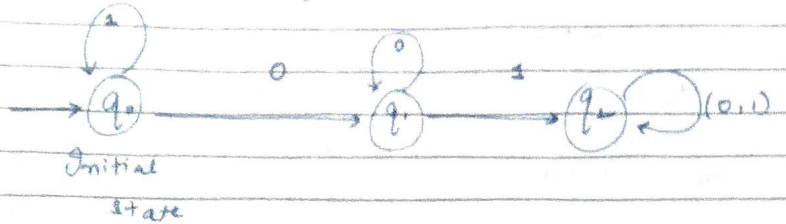
$F \in Q$ a set of final State called Acceptance State

Good Write

Ex:

$L = \{01, 0011, 101110, \ldots\}$

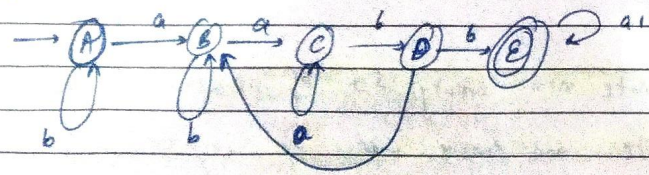
[Condition: [must contain 01]



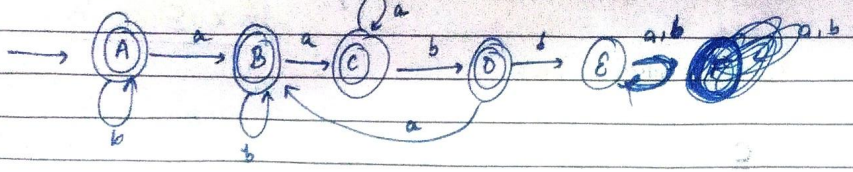

DFA $A = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, q_2)$

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

must contain 'aabb' $\Sigma = \{a, b\}$




must not contain 'aa bb'




Good Write

⇒ Notation of DFA

↳ 1. Transition dig (done before)

2. Transition table

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_2$ | $q_2$ |

⇒ Properties of transition function ($\delta(q, w) = q_2$)

⇒ $\delta(q, \wedge) = q$ [$\wedge$ : NULL]

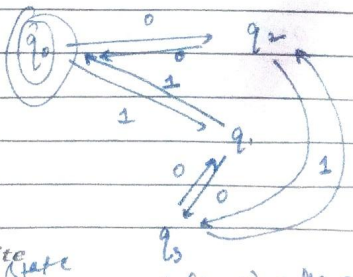
⇒ $= \delta(q, w)$ [if $w = xa$]

$= \delta(q, xa)$

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

eg: DFA $A = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, q_0)$

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_2$ | $q_1$ |
| $q_1$ | $q_3$ | $q_0$ |
| $q_2$ | $q_0$ | $q_3$ |
| $q_3$ | $q_1$ | $q_2$ |




for string $w = 11010$

$= \delta(q_0, 11010)$

$= \delta(\delta(q_0, 1), 1010)$

$= \delta(q_1, 1010)$

$= \delta(q_0, 010)$

$= \delta(q_2, 10) = \delta(q_3, 0)$

$\delta(q_1, 1) = q_0$

[printed]

Good Write

final state

## => Language of DFA

A lang of $A = (Q, \Sigma, \delta, q_0, F)$ is def as

$$L(A) = \{ w \mid \delta(q_0, w) \text{ is in } F \}$$

[i.e. last me final state pe phunch jae]

### Questions

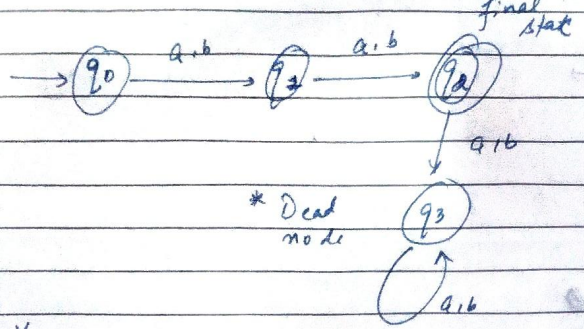
**Q1**

Construct a DFA that accepts all strings over {a,b}

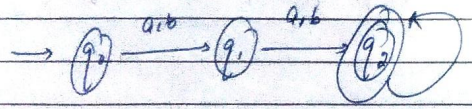
i) $|w| = 2$ ii) $|w| \geq 2$ iii) $|w| \leq 2$

Sol:

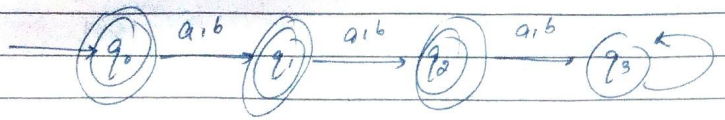
i) aa, ab, bb, ba




ii) ~~0,1~~ X




iii) $|w| \leq 2$



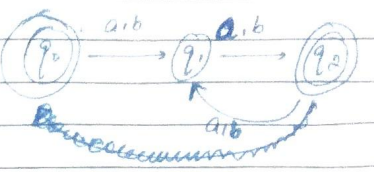

Good Write

**Q2** Construct DFA for $w \in \{a,b\}^*$

i) $|w| \% 2 = 0$ ii) $|w| \% 2 = 1$

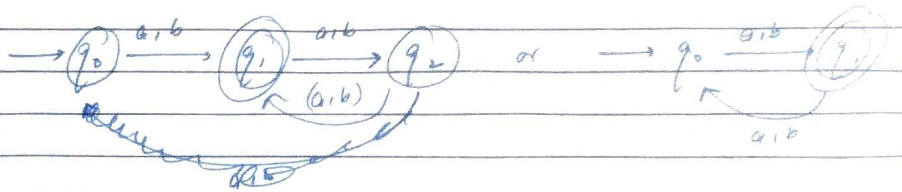
i) even count

$L = \{ \epsilon, aa, bb, ab, ba, aaaa, bbbb \}$




ii) odd count

$L = \{ a, b, aaa, bbb, aab, aba, baa, abb, bab, bba, \ldots \}$

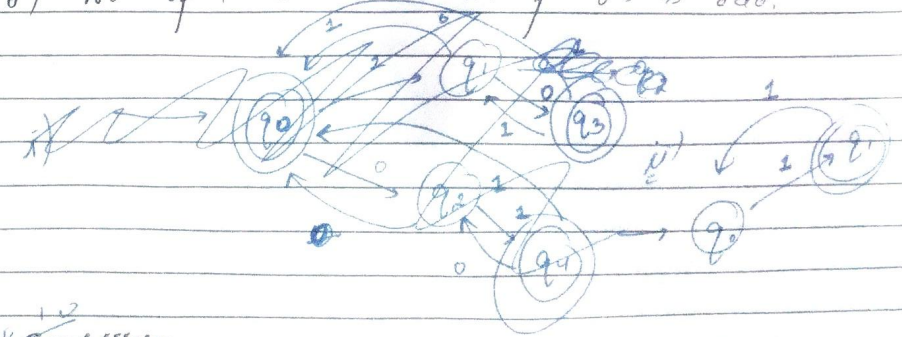



**Q3** Construct DFA accepting following lang. over {0,1}

i) No of 1's is even & no of 0's is even

ii) No of 1's is odd & no of 0's is odd.

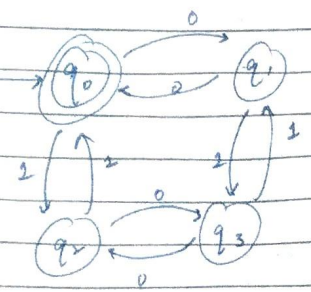
Sol:



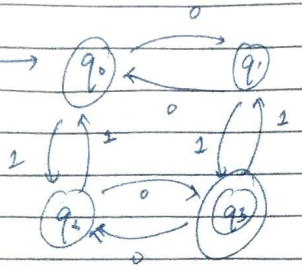

Good Write

11010 11001

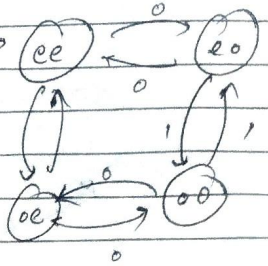
i)




ii)



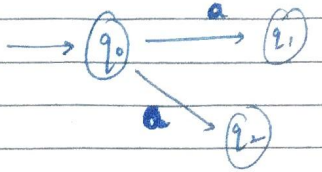

general




Good Write

## → NDFA

* It can be in few real states at once for NFA
* $\delta$ is a function that takes a states & input symbol as argument but return at zero, one or more state




in NFA: $\delta(q_0, a) \longrightarrow \phi, q_i, \ldots$ [more than 1].

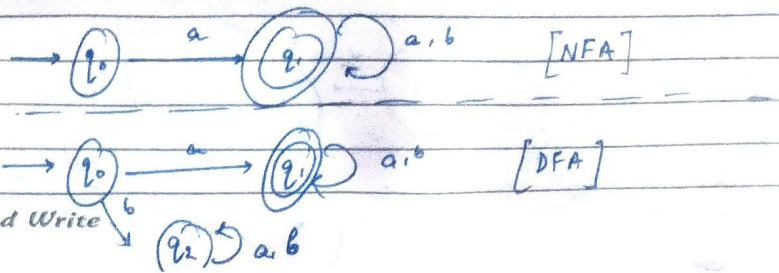
NFA consist of 5 tuple $(Q, \Sigma, \delta, q_0, F)$

$Q$: set of state
$\Sigma$: set of input
$q_0$: $q_0 \in Q$ initial state
$F$: $F \subseteq Q$ final state
$\delta$: $Q \times \Sigma \longrightarrow |2^Q|$

↳ power set of $Q$.

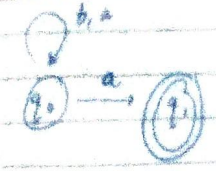
$\{\phi, \overline{1}, \overline{2}, \overline{3} \ldots\}$ ?

» Construct a NFA where it accepts all strings with $\Sigma = \{a, b\}$ starting with a.



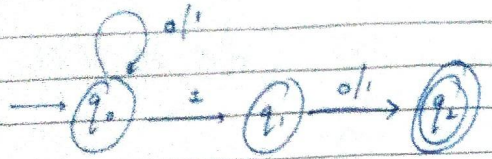

Good Write

i) end with a




Q) i) Construct NFA accepting the string with $\Sigma = \{0, 1\}$ & second last element is '1'

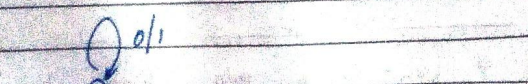
$L = (0|1)^* 1 (0|1)$



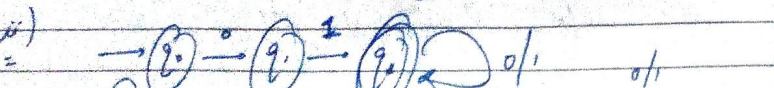

ii) Construct NFA that accepts all strings over $\{0,1\}$

i) String contain '1'
ii) Start with '0'
iii) Contains '01'
iv) ends with '0'

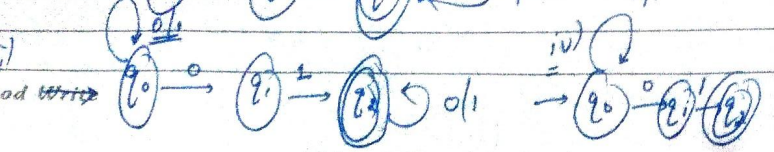
$\Sigma = \{0,1\}$: i) $(0|1)^* 1 (0|1)^*$




ii)




iii)




iv)



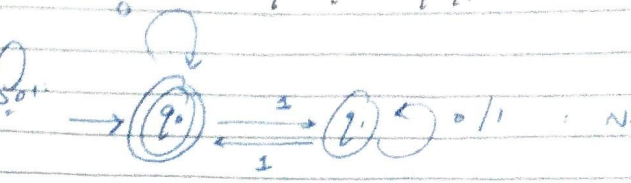

Good Write

# NFA to DFA

1. Construct DFA eq from NFA

$M = (\{q_0, q_1\}, \{0, 1\}, \delta, \{q_0\})$

| δ | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_1$ | $q_0 q_1$ |




: NFA

procedure:

1. $P(\{q_0, q_1\}) = [\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}]$

2. Identify initial state: $q_0$

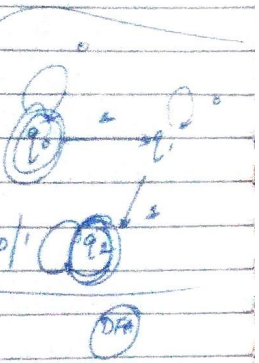
3. final state: $\{q_0\}, \{q_0, q_1\}$

[: all sets with $q_0$]

4. δ table:

| Σ | 0 | 1 |
|---|---|---|
| $\phi$ | $\phi$ | $\phi$ |
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_1$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_1\}$ |
| $q_0$ | $q_1$ | $q_1$ | $q_0 q_1$ |

union: $\{q_0, q_1\}$ union: $q_0 q_1$




Good Write

2. Construct DFA eq. of NFA given:

$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$

| $\delta$ | a | b |
|---|---|---|
| $q_0$ | $q_0, q_1$ | $q_2$ |
| $q_1$ | $q_0$ | $q_1$ |
| $q_2$ | – | $q_0, q_1$ |

Sol:

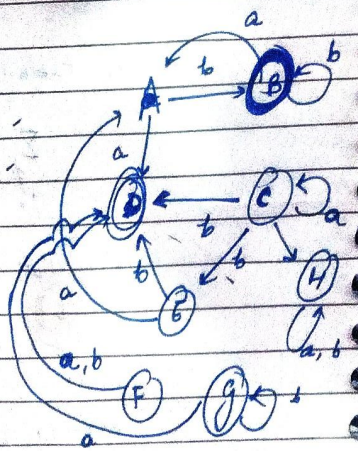
final state: $q_2$

$P[\{q_0, q_1, q_2\}] = [\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_1, q_2\}, \{q_2, q_0\}, \{q_0, q_1, q_2\}]$

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

| | a | b |
|---|---|---|
| $\phi$ | $\phi$ | $\phi$ |
| $\{q_0\}$ | $\{q_0, q_1\}$ | $q_2$ |
| $\{q_1\}$ | $q_0$ | $q_1$ |
| $\{q_2\}$ | $\phi$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_1, q_2\}$ |
| $\{q_1, q_2\}$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_2\}$ | $\{q_0, q_1\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ |

| | |
|---|---|
| h | |
| a | |
| b | |
| c | |
| d | |
| e | |
| f | |
| g | |




Good Write

## Minimization of DFA

Note: Only reachable states are involved here.

Q: Construct a min state automata eqv to finite automata given as

| | $\varepsilon$ | 0 | 1 |
|---|---|---|---|
| $\to q_0$ | | $q_1$ | $q_5$ |
| $q_1$ | | $q_6$ | $q_2$ |
| $(q_2)$ | | $q_0$ | $q_2$ |
| $q_3$ | | $q_2$ | $q_6$ |
| $q_4$ | | $q_7$ | $q_5$ |
| $q_5$ | | $q_2$ | $q_6$ |
| $q_6$ | | $q_6$ | $q_4$ |
| $q_7$ | | $q_6$ | $q_2$ |

Sol:

We make $\pi_0$

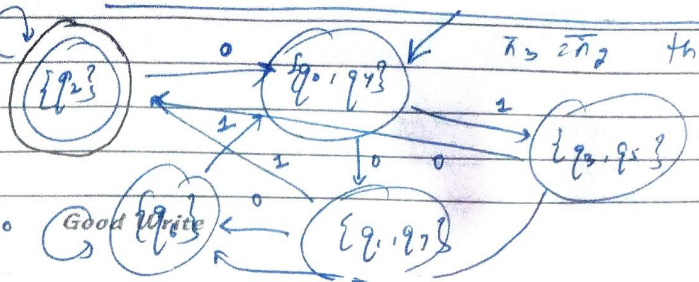
$\pi_0 = \{\overset{(1)}{\{q_2\}}, \overset{(2)}{\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}}\}$

$\pi_1 = \{\overset{(1)}{\{q_2\}}, \overset{(2)}{\{q_0, q_4, q_6\}}, \overset{(3)}{\{q_1, q_7\}}, \overset{(4)}{\{q_3, q_5\}}\}$

$\pi_2 = \{\overset{(1)}{\{q_2\}}, \overset{(2)}{\{q_0, q_4\}}, \overset{(3)}{\{q_6\}}, \overset{(4)}{\{q_1, q_7\}}, \overset{(5)}{\{q_3, q_5\}}\}$

$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

$\pi_3 = \pi_2$ thus states are



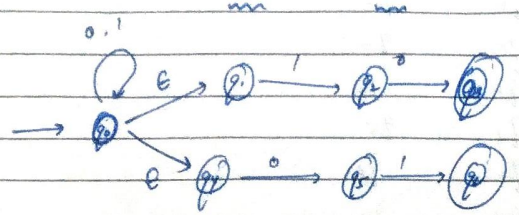

Good Write

» ε-NFA [used to combine diff machines]
(or)

↳ Null move

eg: $q_0 \xrightarrow{\epsilon} q_1$

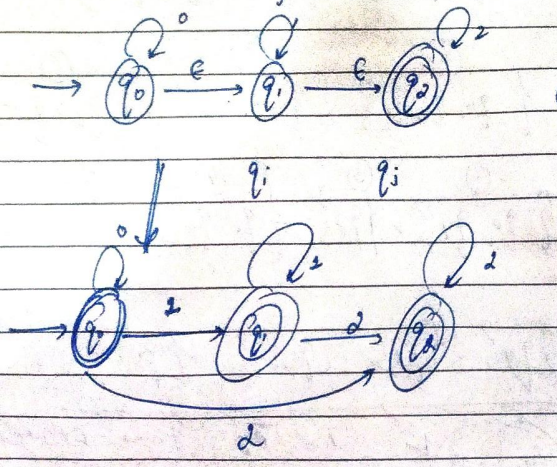
eg: ends with 10 or 01.




» ε-NFA to NFA

① all move $q_i$ should also be orig. from $q_i$

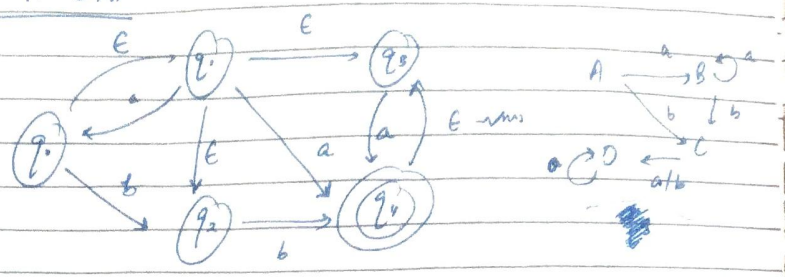
② if $q_i$ is final min $q_i$ as final




* Copy all arrow using $q_i$

Good Write

» ε-NFA to DFA




| NFA state | DFA state | a | b |
|---|---|---|---|
| $\{q_0\ q_1\ q_2\ q_3\}$ | A | B | C |
| $\{q_0, q_1, q_2, q_3, q_4\}$ final | B | B | C |
| $\{q_2, q_4, q_3\}$ final | C | D | D |
| $\{q_4, q_3\}$ final ($q_4$ common) | D | D | ϕ |

ε-closure $(q_0) = \{q_0, q_1, q_2, q_3\} = A$

D-trans[A, a] = ε-closure (move (A, a))

ε-closure $(q_0, q_4)$

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

D-trans [A, b] = ε-closure (move (A, b))

= ε-closure $(q_2, q_4)$

$= \{q_2, q_4, q_3\}$

D-trans [B, a] = ε-closure (move (B, a)) = B

[B, b] = ε-closure (move (B, b)) =

Good Write

DATE: __/__/____
PAGE: _____

# Finite Automata with Outputs

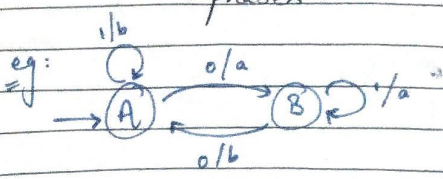
(Output size = Input + 1)

## ⇒ Mealy machine (same output length as input)

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ — $q_0$: Initial state

- $Q$: finite set of states
- $\Sigma$: Inputs
- $\Delta$: set of output alphabets
- $\delta$: transition function
- $\lambda$: Output function $[\Sigma \times Q \to \Delta]$

eg:




$Q = \{A, B\}$

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

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

$\delta$:

| | 0 | 1 |
|---|---|---|
| A | B | A |
| B | A | B |

$\lambda$:

| | 0 | 1 |
|---|---|---|
| A | a | b |
| B | b | a |

$q_0 = \{A\}$

A gives output as a when 0 is fed & also moves to B.

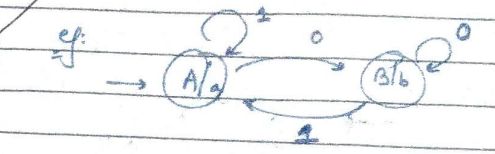
* Output dep. on both input & state

## ⇒ Moore machine

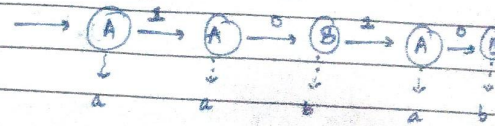
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

$Q \to \Delta$

eg:




Lets feed it with 1010.



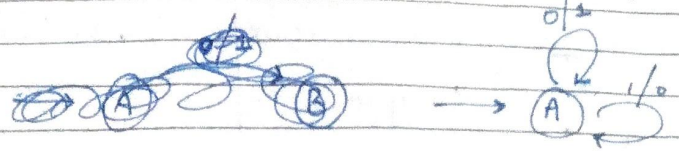

* Output dep. only on state



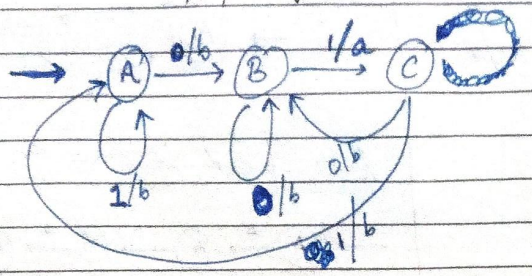
Good Write

eg:1 Construct a mealy machine that produces 1's complement of any binary input string.

Sol:

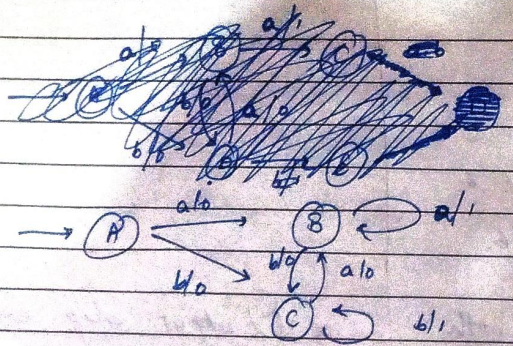



eg:2 whenever '01' is detected it must print 'a' else keep printing 'b'




eg:3 make a mealy machine accepting the language where $\Sigma = \{a, b\}$ ending with aa or bb.

Sol: we need to get outputs so that we know if there is aa or bb.




Good Write

mealy machine to get 1's complement

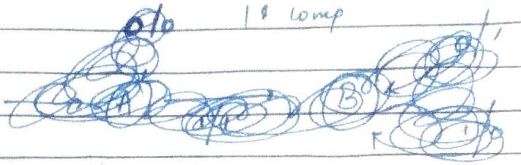
1's complement

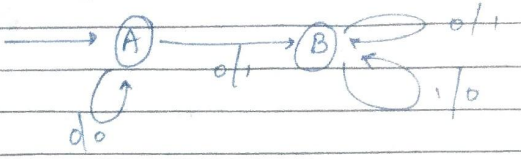
(1010100 100)' +1

as it is

01010111100

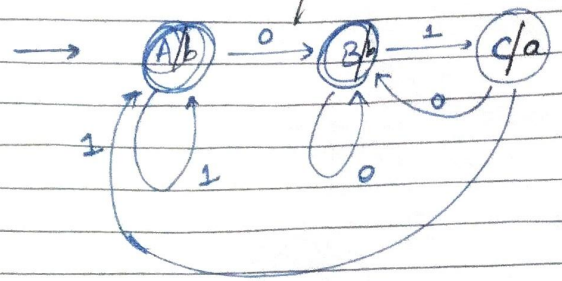
1's comp



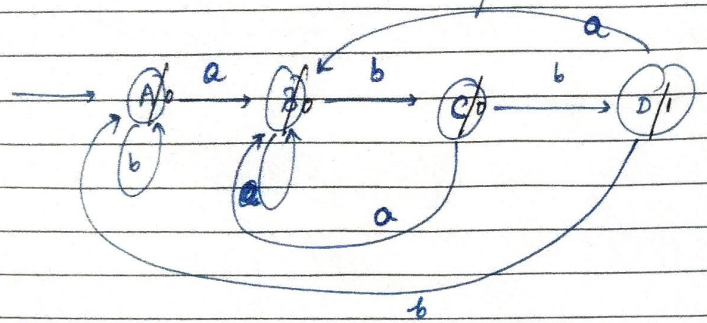

Good Write

→ Construct a moore machine that prints 'a' whenever the seq '01' is encountered

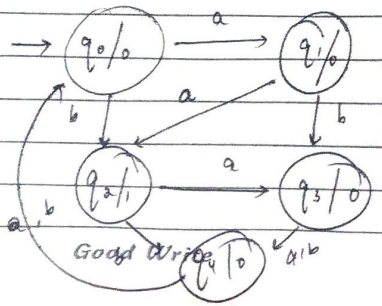



→ counts the occurrence of 'abb' in any input string over {a,b}

say we make an output string where number of 1 will tell the count




→ for the flwing moore machine the input alphabet is $\Sigma = \{a, b\}$ and the output alphabet is $\Delta = \{0, 1\}$ find the outputs for i) aaabab ii) abbb iii) ababb




i) 001001
ii) 00000
iii) 000001

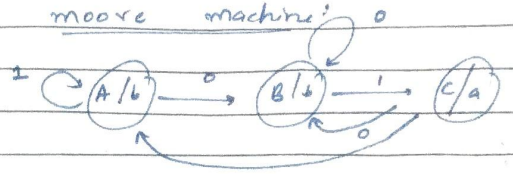
=



# Conversion to moore to mealy machine

eg: Construct a moore machine that prints 'a' whenever the seq '01' is encountered in a binary string & then convert to mealy machine ✓

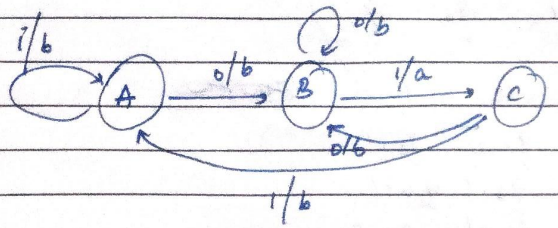
Sol: moore machine:




| | 0 | 1 | Output |
|---|---|---|---|
| A | B | A | b |
| B | B | C | b |
| C | B | A | a |

* all we have to do is:

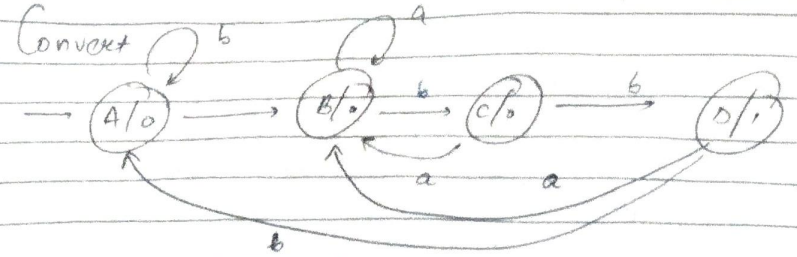
check each & every arrow that ends in a specific state & get all the arrows the output value of the state in moore



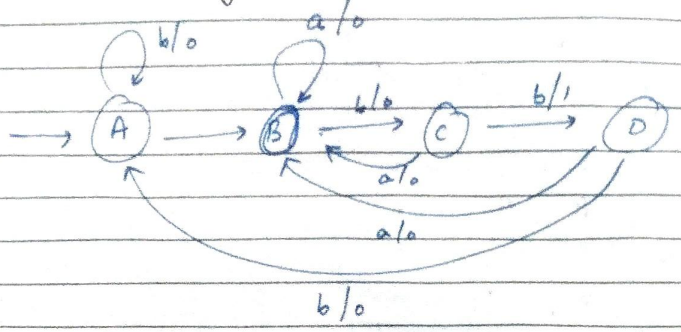

| | 0 | 1 |
|---|---|---|
| A | B/b | A/b |
| B | B/b | C/a |
| C | B/b | A/b |

DATE: / /
PAGE:

Q
= Convert




to mealy.




You can do this directly via table too

Ex convert

| State | 0 | 1 | Output |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_2$ | 1 |
| $q_1$ | $q_3$ | $q_2$ | 0 |
| $q_2$ | $q_2$ | $q_1$ | 1 |
| $q_3$ | $q_0$ | $q_3$ | 1 |

to mealy

Sol

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1, 0$ | $q_2, 1$ |
| $q_1$ | $q_3, 1$ | $q_2, 1$ |
| $q_2$ | $q_2, 1$ | $q_1, 0$ |
| $q_3$ | $q_0, 1$ | $q_3, 1$ |

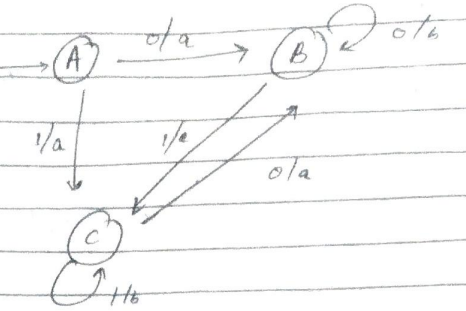
Good Write

# Mealy to Moore

DATE: / /
PAGE:

Similarly to convert a moore machine to mealy machine

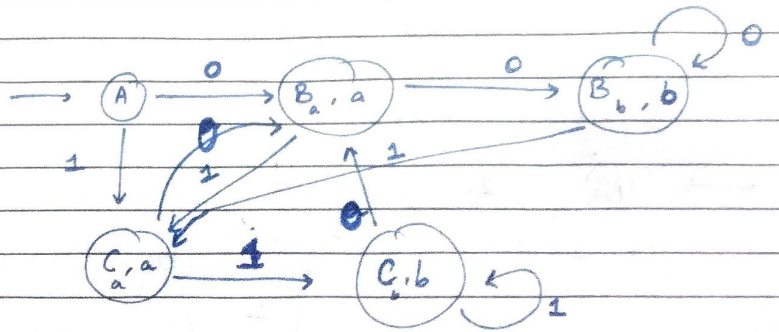
Ex.




Sol: $\Sigma = (0, 1)$

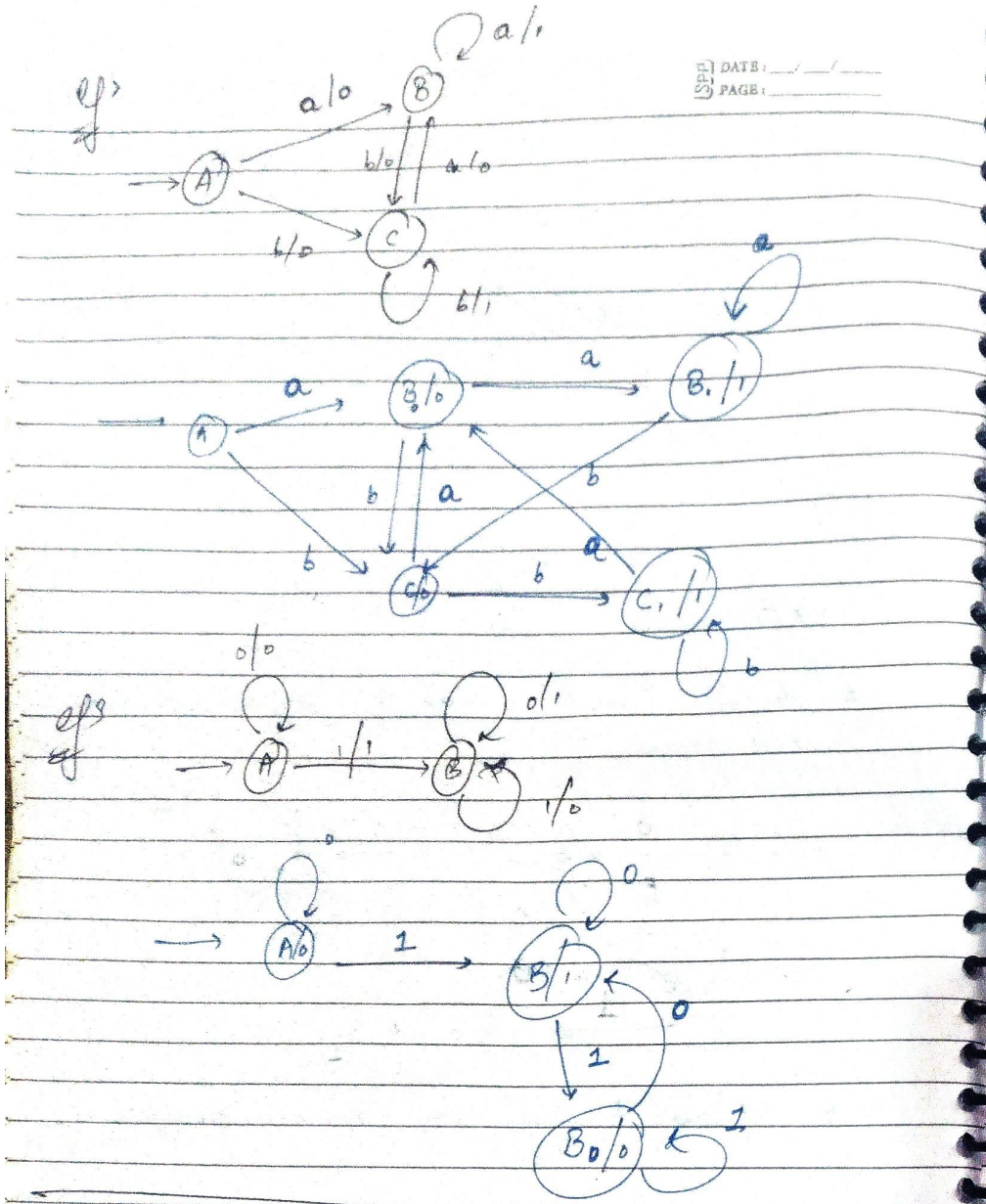
$\Delta = (a, b)$


* whenever there is a clash of states we make new states.




* we have to make sure no connections/cases are left.

Good Write




# Mealy to Moore

using transition table

| State | a | b |
|---|---|---|
| → $q_0$ | $q_3$ : 0 | $q_1$ : 1 |
| $q_1$ | $q_0$ : 1 | $q_3$ : 0 |
| $q_2$ | $q_2$ : 1 | $q_2$ : 0 |
| $q_3$ | $q_1$ : 0 | $q_2$ : 1 |

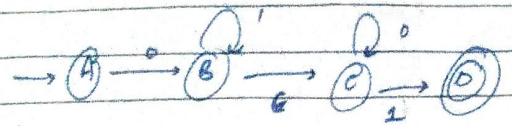
| | a | b | Output |
|---|---|---|---|
| $q_0$ | $q_3$ | $q_1$ | 1 |
| $q_{10}$ | $q_0$ | $q_3$ | 0 |
| $q_{20}$ | $q_{21}$ | $q_{10}$ | 0 |
| $q_3$ | $q_{10}$ | $q_0$ | 0 |
| $q_1$ | $q_0$ | $q_3$ | 1 |
| $q_{21}$ | $q_{21}$ | $q_{20}$ | 1 |

Good Write

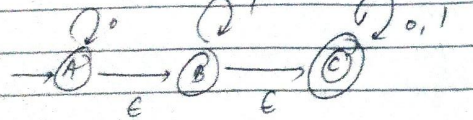
Good Write

# ✳ ε - NFA

You can use ε for moving from one state to another too.



## Converting ε - NFA to NFA



for each state we have we have to check that what states it attain by $\epsilon^*$ i.e. all the states that can be gone using ε only.

| | 0 | 1 |
|---|---|---|
| → A | {A,B,C} | {B,C} |
| B | {C} | {B,C} |
| C | {C} | {C} |

A --> ε* 0 ε*
A A A
B ϕ ϕ
C C C

A --> ε* 1 ε*
A ϕ B
B B C
C 0 C

B --> ε* 0 ε*
B ϕ ϕ
C C C

B --> ε* 1 ε*
B B B,C
C C C

C --> ε*
C

Good Write

e)

| | 0 | 1 |
|---|---|---|
| A | {B,C} | ϕ |
| B | {C} | {B,C} |
| C | {C} | {B,C} |
| 0 | ϕ | ϕ |

A --> ε* 0 ε*
A B B
C

ε* 1
A ϕ

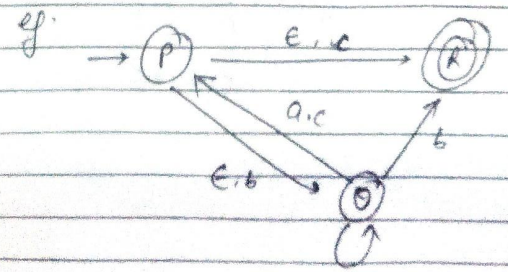
B --> ε* 0 ε*
B ϕ ϕ
C C C

ε* 1 ε*
B B B,C
C D D

C --> ε* 0 ε*
C C C

ε* 1 ε*
C D D

f)



| | a | b | c |
|---|---|---|---|
| P | {P,Q,R} | {Q,R} | {P,Q,R} |
| Q | {P,Q,R} | {R} | {P,Q,R} |
| R | Q | ϕ | ϕ |

P --> ε* a ε*
P P {P,Q,R}
Q ϕ {P,Q,R}
R ϕ {P,Q,R}

P --> ε* b ε*
P Q --> Q
Q R R
R ϕ R

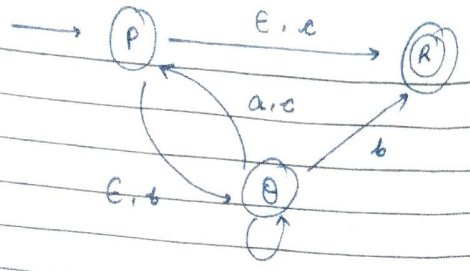
P --> ε* c ε*
P P {P,Q,R}
Q R R
R ϕ R

Q --> ε* a ε* R
Q P {P,Q,R}

ε* b ε*
B R R

ε* c ε*
C C
R R

Good Write




| | a | b | c |
|---|---|---|---|
| P | {P, Q, R} | {Q, R} | {P, Q, R} |
| Q | {P, Q, R} | {R} | {P, Q, R} |
| R | ϕ | ϕ | ϕ |

Q... ε* a ε*
Q P P, Q, R

ε* b ε*
Q R R

ε* c ε*
Q {P, Q} {P, Q, R}

P... ε* a ε*
P ϕ ϕ
Q P {P, Q, R}
R ϕ ϕ

ε* b ε*
P Q Q
Q R R
R ϕ ϕ

ε* ε ε*
P R R
Q Q, P → {P, Q, R}
R ϕ ϕ

{P, Q, R}



## Regular Expression

used to representing certain sets of strings in an algebraic function.

1. Any terminal symbol i.e. symbols ∈ [Σ] including ∧ & ϕ are regular expressions.
2. The union of 2 R.E. is also a R.E.
3. Concatenation of 2 R.E. is a R.E.
4. The iteration (or closure) of a R.E. is also a R.E.

∧ (Empty) ; ϕ (Null)

Closures (*) & +

Do

{∧, 0, 00, 000, ...} | {0, 00, 000 ...}

= 0* | = 0+

$$0^+ = 0^* - \{\wedge\}$$

## Identities of RE

$\phi$: empty set

$\epsilon$ or $\wedge$ : without consuming an input

1. $\phi + R = R$
2. $\phi R = R\phi = \phi$
3. $\epsilon R = R\epsilon = R$
4. $\epsilon^* = \epsilon$ & $\phi^* = \epsilon$
5. $R + R = R$
6. $R^* R^* = R^*$
7. $RR^* = R^* R = R^+$
8. $(R^*)^* = R^*$
9. $(\epsilon + R)^* = \epsilon + R^* R = R^*$ * Imp!
10. $(PQ)^* P = P(QP)^*$
11. $(P+Q)^* = (P^* \cdot Q^*)^* = (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$

## Arden's theorem

If P and Q are two R.E. over $\Sigma$, and if P doesn't contain $\epsilon$, then the following equation in R is given by

$R = Q + RP$ has a unique solution i.e. $\boxed{R = QP^*}$

$R = Q + RP \longrightarrow (1)$

replace $R = QP^*$

$R = Q + QP^* P$

$= Q + QP^* P$

$= Q[\epsilon + P^* P] = Q[P^*] = R$ proved

Good Write

DATE: / /
PAGE:

Prove that $(1 + 00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$ equal to $0^*1(0+10^*1)^*$

Sol:

$= 1(\epsilon + 00^*)1 + 1(\epsilon + 00^*)1(0+10^*1)^*(0+10^*1)$

$= 0^*1 + 0^*1[(0+10^*1)^*(0+10^*1)]$

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

$= 0^*1(0+10^*1)^*$

Q: Design a RE for the following lang. over $\{a,b\}$

i) length 2 exactly
ii) atleast length = 2
iii) atmost length = 2

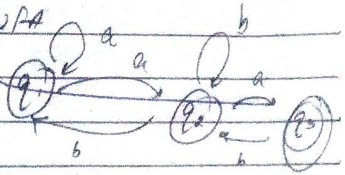
Sol:

$= (a/b)\cdot(a/b)$ (i)

$= (a/b)(a/b)(a/b)^*$ (ii)

$= (\epsilon + a/b)(\epsilon + a/b)$ (iii)

Q: ~~find RE for the given NFA~~

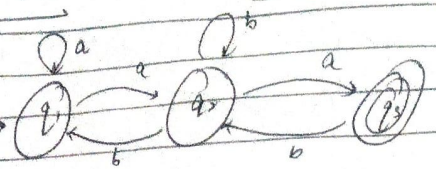



~~$R = Q^* a (bQ^* a / b^*) \cdot a$~~

Good Write

# RE using NFA

Q: find RE from




Solution:

Write eq for each state

final state $q_3 = q_2 a \longrightarrow (1)$

$q_2 = q_1 \cdot a + q_2 \cdot b + q_3 \cdot b \longrightarrow (2)$

$q_1 = q_1 \cdot a + q_2 b + \epsilon \longrightarrow (3)$

Solve them:

$q_3 = (q_1 a + q_2 b + q_3 b) a$

$= q_1 aa + q_2 ba + q_3 ba$

$q_2 = q_1 a + q_2 b + [q_2 a] b$

$q_2 = q_1 a + q_2 b + q_2 ab$

$= q_1 + q_2 [\epsilon + a] b$

$q_2 = q_1 a + q_2 (b + ab)$ → ~~R = Q + RP, QP*~~

$R = Q + R(P) = QP^*$

$= (q_1 a) \cdot (b+ab)^* \longrightarrow (4)$

$q_1 = q_1 a + q_1 a (b+ab)^* + \epsilon$

$q_1 = \epsilon (a(b+ab)^*)^*$

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

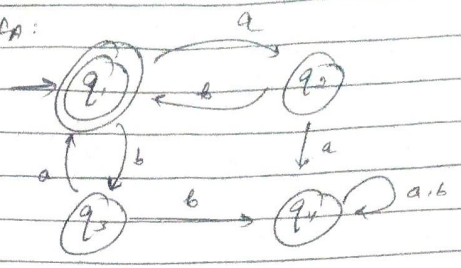
(1) (2) (3) (4)

Good Write

$q_1 \; a\,(b+ab)^* + \epsilon = Q$

# RE from DFA

Q: DFA:




$q_1 = \epsilon + q_3 a + q_2 b$

$q_2 = q_1 \cdot a$

$q_3 = q_1 b$

$q_4 = q_2 a + q_2 b + q_4 (a+b)$

Solving

$q_1 = \epsilon + q_1 ba + q_1 ab$

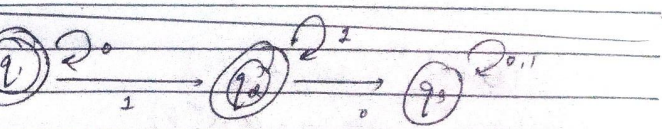
$q_1 = \epsilon + q_1 (ba + ab)$ — $R = Q + RP$, $R = QP^*$

$q_1 = \epsilon (ba+ab)^*$

$q_1 = (ba+ab)^*$

Step:
- Write eq
- Put final state eq
- Solve to finally get the final state eq
- RE = 0 or the final state eq

Q: DFA:




Sol:

$q_1 = \epsilon + q_1 \cdot 0 = \epsilon 0^* = q_1 = 0^*$

$q_2 = q_1 \cdot 1 + q_2 \cdot 1 = q_2 = 0^* 1 + q_2 1$

$q_2 = 0^* 1 \cdot 1^*$

$q_3 = q_2 \cdot 0 + q_3 (0+1)$

$q_3 = 0^* 1 1^* 0 (0|1)^*$

Good Write

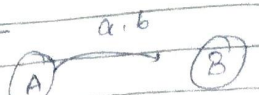
$RE = (0^*) + (0^* 1 1^*) = 0^* (\epsilon + 1 \cdot 1^*)$

$= 0^* 1^*$

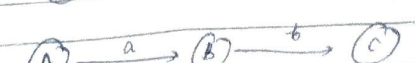
# RE to FA

Rules

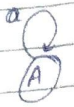
a+b :



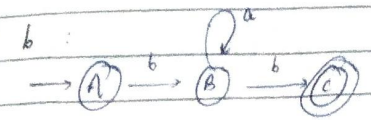
a.b :



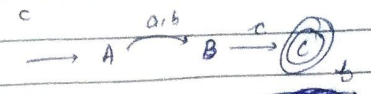
a* :



eg :) ba* b :



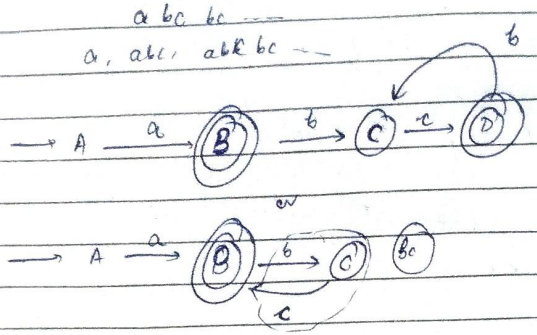
ii) (a+b) c



iii) a(bc)*

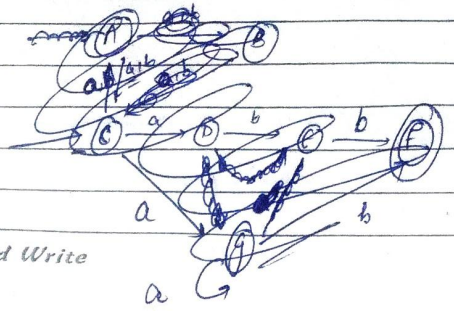
abc bc ——

a, abc, abc bc ——

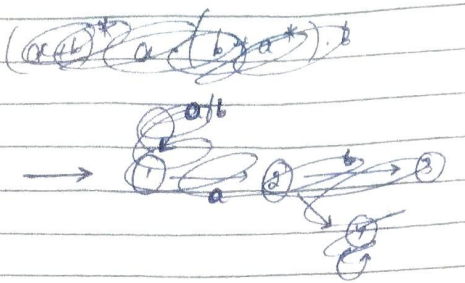


eg: (a+b)* (abb + a+b)

(a+b)* (abb + a.a*b)

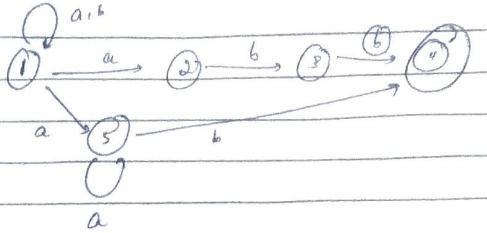


Good Write

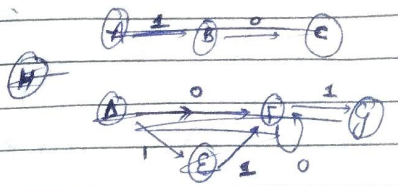


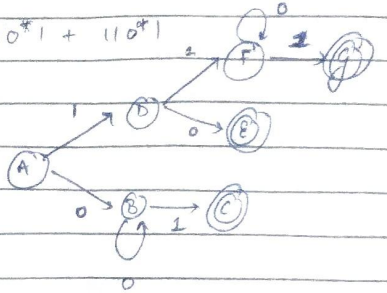
= (a+b)* (abb + aa*b)

=



RE : 10+(0+11) 0* 1 = 10+ 00*1 + 110*1





Good Write

# Equivalence of 2 finite automata

Steps:

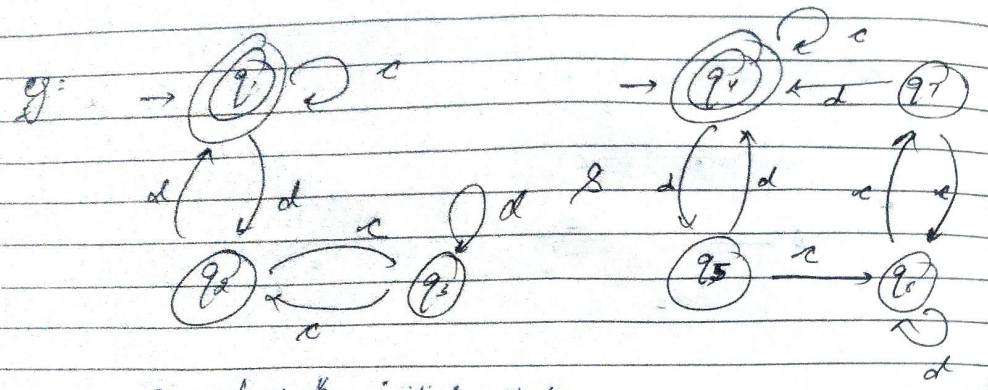
for any pair of states $\{q_i, q_j\}$ the transition for input at $\Sigma$ is defined by $\{q_a, q_b\}$ where

$\delta(q_i, a) = q_a$ and $\delta(q_j, b) = q_b$

$A_1$ $A_2$

The 2 automata arent equivalent if for a pair $\{q_a, q_b\}$ one is INDETERMINATE state & other is final state.

2. If Initial state = final state for $A_1$ then same must hold for $A_2$.




$q_1$ → final & initial state

$q_4$ → final & initial state

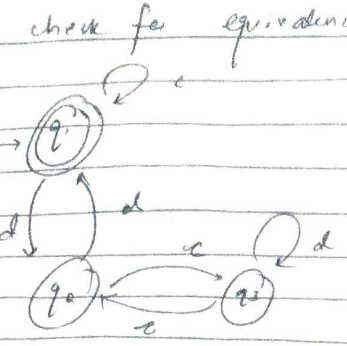
now start making pairs

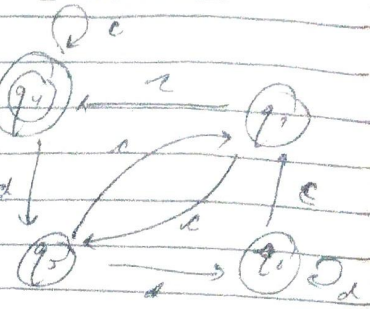
| | ε | d |
|---|---|---|
| $q_1, q_4$ | $\delta(q_1, q_4)$ $\{q_1, q_4\}$ (F) (F) | $\{q_2, q_5\}$ NF, NF |
| $q_2, q_5$ | $(q_3, q_6)$ NF NF | $(q_1, q_4)$ F F |
| $q_3, q_6$ | $\{q_2, q_7\}$ | $\{q_3, q_4\}$ |
| $q_2, q_7$ | $\{q_3, q_6\}$ | $\{q_1, q_4\}$ |

[A & B are equivalent]



If check for equivalence




Sol: $q_1$ & $q_4$ both are final & Initial states

| | ε | d |
|---|---|---|
| $\delta(q_1, q_4)$ | $q_1, q_4$ IS FS | $q_2, q_5$ IS IS |
| $\delta(q_2, q_5)$ | $q_3, q_7$ IS IS | $q_1, q_6$ FS IS |

error [Not equ$^n$]

# Pumping Lemma

→ Its a method to prove that a lang. is not regular

→ It cant be proved if a lang is Regular

⇒

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

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

ii) $|y| > 0$

iii) $|xy| \leq p$

## prove irregular language

(we prove by contradiction)

- Assume A is regular
- It has to have a pumping length (say p)
- All strings longer than p can be pumped
- Now find a string s in A such that $|s| \geq p$
- Divide S into xyz
- Show that $xy^iz \notin A$ for some i
- Then consider all ways that S can be div into xyz
- Show that none of these can satisfy all 3 conditions at a time
- S cant be pumped

→ Contradiction



Eg: Using pumping lemma prove $A = \{a^n b^n \mid n \geq 0\}$ is not regular

Sol: Let $a^n b^n = A$ be regular

let pumping length = p

$S = a^n b^n$

x y z

say $p = 7$

| $xy^iz$ say $i = 2$ |
|---|
| aa aaaaaa aabbbbbb $\notin A$ |
| $\|xy\| \leq p$ ✗ |
| $xy \leq p$ ✗ |

(C1) $S = \underbrace{a}_{x}\underbrace{aaaaaa}_{y}\underbrace{abbbbbb}_{z}$

(C2) $S = \underbrace{aaaaaaa}_{x}\underbrace{bbbbbb}_{y}\underbrace{b}_{z}$

(C3) $S = \underbrace{aaaaaaa}_{x}\underbrace{bbbbbbb}_{y}\underbrace{}_{z}$

Q: Using pumping lemma prove $A = \{yy \mid y \in \{0,1\}^*\}$ isnt regular.

Sol: Let $yy = A$ be regular $y \in \Sigma^*$ where $\Sigma = \{0,1\}$

let pumping length = p

$S = 0^p 1 0^p 1$

for $xy^iz$ $|xy| \leq p$

{0, 00, 000, 0000, 00000 etc}

000...0 1 000...0 1

x y z

if we do $y^i$ i=2 $\notin A$ ~~000 0 0000 1~~ irregular

# Regular Grammar

Noam Chomsky gave a mathematical model of grammar which is eff for writing computer language

| | | |
|---|---|---|
| Type 3 | Regular grammar | Reg. Lang. |
| Type 2 | Context free grammar | Context free lang. |
| Type 1 | Context sensitive grammar | Context sens. lang. |
| Type 0 | Unrestricted grammar | Recursively Enumerable language |

## defining a grammar:

A grammar G can be formally described using 4 tuples as $G=(V,T,S,P)$ where

- (V) = Set of variables or Non terminal symbols
- (T) = Set of terminal symbols.
- (S) = Start symbol
- P = production rules for terminals & Non-terminals

(A production rule has the form $\alpha \longrightarrow \beta$ where $\alpha$ & $\beta$ are strings on $V \cup T$ and atleast one symbol of $\alpha$ belongs to V.

eg: $G = (\{S,A,B\}, \{a,b\}, S, \{S \to AB, A \to a, B \to b\})$

(V) (T) (S) (P)

Good Write

# Reg. grammar

## Right linear grammar

A grammar is said to be right linear if all production are of form

$A \to xB$ (B: right)

$A, B \in V$ $\quad x \in T$

eg $S \to (abS) / b$ (right linear) $\qquad S \to Sbb/b$ (left linear)

## Left linear grammar

$A \to Bx$ (B: left)

## * Derivations from a grammar

The set of all the strings that can be derived from a grammar is said to be the LANGUAGE generated from that grammar

eg: $G1 = (\{S,A\}, \{a,b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\})$

Sol:

$S \to aAb$

$A \to \epsilon$

$S \to a\epsilon b$ or $S \to ab$

$aA \to aaAb$

$S \to a[a[aAb]b]b$

$S \to aaaAbbb$

$S \to aaaaAbbbb$

or

$S \to a^n A b^n$

$S \to a^n b^n$ *

Good Write

Ex 2

$G_2 = \{\{S, A, B\}, \{a, b\}, \varepsilon, \{S \to AB, A \to a, B \to b\}\}$

$S_2$: $S \to ab$

eg:?

$G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA/a, B \to bB/b\})$

$S \to AB$

$A \to aA$ or $a$

$S \to aAB$ → $S \to aabAB$, $S \to aAb$

$S \to ab$ → $S \to ab$, $S \to abB$

$S \to aAbB$ → $aabbB$, $aaAbB$

$S \to aAb$ → $aaAb$, $aab$

$S \to ab$

$S \to abB$ → $abb$, $abbB$

$[[ab], [a^2b^2, abbb \cdots], [aab, aaab \cdots aa \cdots ab], a^nb^n]$

$[a^nb^n, ab^n, a^nb]$ $(n \geq 1)$



## Context free Language

In formal languages theory, a context free language is a language gen by context free grammar

The set of all CFL is identical to the set of lang. accepted by Pushdown Automata.

In Context free grammar has production rule of form

$A \to \alpha$

where $\alpha \in (V \cup T)^*$ & $A \in V$

eg: for gen. a language that generates equal number of a's & b's of form $a^n b^n$, the context free grammar is

$G = \{(S, A), (a, b), (S \to aAb, A \to aAb/\epsilon)\}$

Sol: production rule

$S \to aAb$, $A \to aAb/\epsilon$

∴ context free grammar

$S \to aAb$

$S \to a(aAb)b$ or $S \to ab$

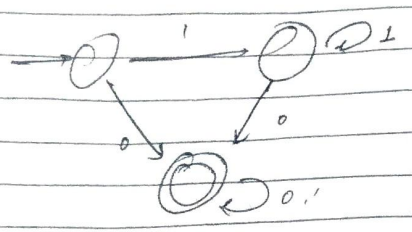
$S \to a^2(aAb)b^2$ or $S \to a^2b^2$

$\vdots$

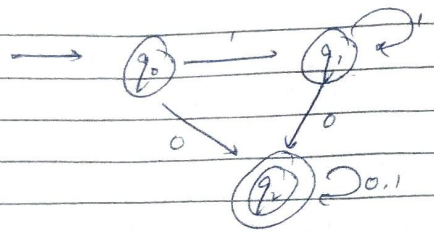
$a^nAb^n$ or $S \to a^nb^n$

## problems on Reg. Lang. & FA

1. Consider the DFA




find its RE




$q_2 = q_1 \cdot 0 + q_0 \cdot 0 + q_2 (0+1)$

$q_1 = q_1 \cdot 1 + q_0 \cdot 1$

$q_0 = \epsilon$

$q_2 = q_1 \cdot 0 + 0 + q_2 (0+1)$

$q_1 = q_1 \cdot 1 + 1$ $\quad [R = Q + RP]$ $\quad R = QP^*$

$q_1 = 1 \cdot 1^*$

$q_2 = 1 \cdot 1^* \cdot 0 + 0 + q_2 (0+1)$

$= (1 \cdot 1^* + \epsilon) 0 + q_2 (0+1)$

$q_2 = \underset{Q}{1^* \cdot 0} + \underset{R}{q_2} \underset{P}{(0+1)}$

$q_2 = 1^* 0 (0+1)^*$

Good Write

2. Consider $L_1 = \phi$ $L_2 = \{a\}$ which one rep $L_1 L_2^* \cup L_1^*$

Sol: $\phi \cdot a^* \cup L_1^*$

$= \phi \cup L_1^*$ $\quad \because \phi^* = \epsilon$

$= \phi \cup \epsilon$

$= \epsilon$

Good Write

* Method to find whether a specific string belongs to that grammar or not

1. Start with Start symbol & choose the closest prodⁿ that matches to the given string.
2. Replace the variables with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

Q: Verify whether $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA \mid \varepsilon$, $B \to 1 \mid 1S \mid 0BB$ generates 00110101.

Sol: Start with S

(starts with 0)

$S \to 0B$

0BB (Reads in 0)

$S \to 00BB \to 1$, $1S$, $1$, $1S$

00 11 X
00 11 S
00 1S 1
00 1S 1S
} possible cases

0011S → 00110B → 1 X, 1S → 0011010S → 00110101B

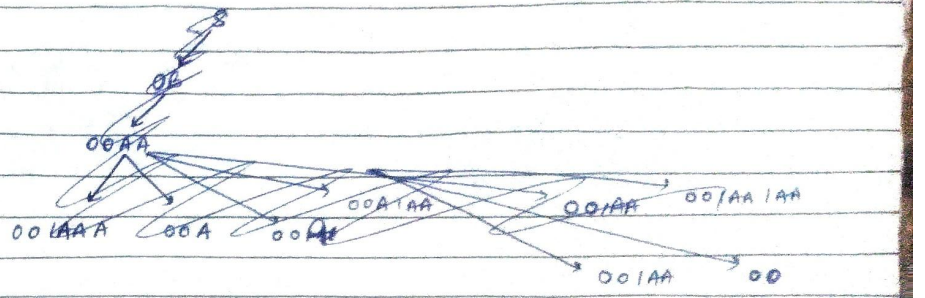
accepted 00110101



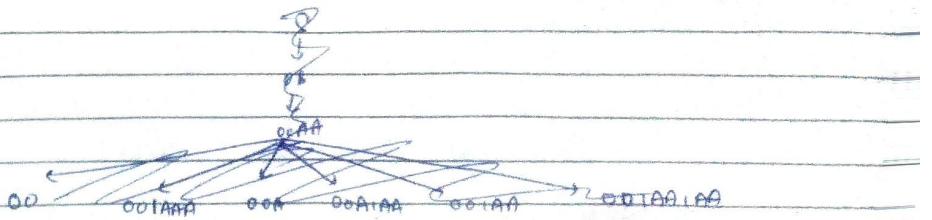
Derivation tree

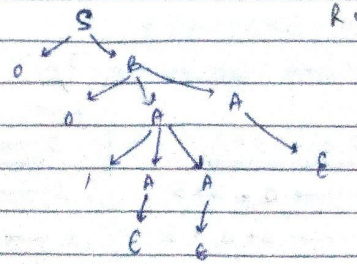
It is a ordered rooted tree that graphically represents the semantic info of strings derived from a context free grammar

eg: $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA \mid \varepsilon$, $B \to 0AA$




i.e.




Root vertex: must be labelled by the start symbol

Vertex: Labelled by Non-terminals (V)

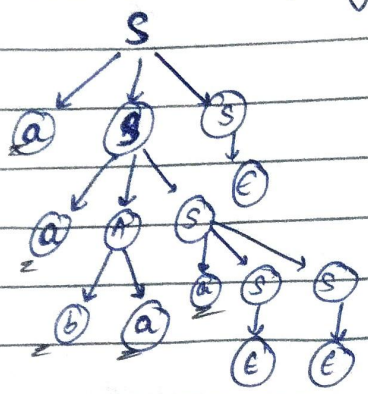
Leaves labelled (T).

DATE: __/__/____
PAGE: _____

## left deⁿ tree

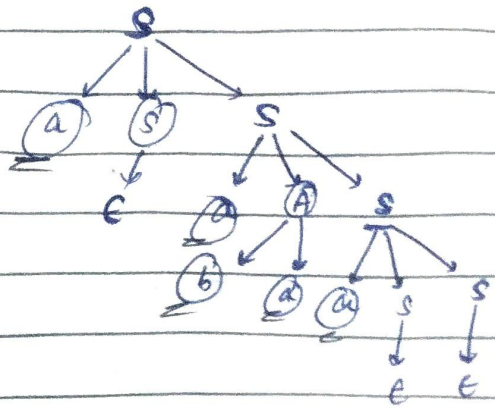
applying prodⁿ to left most variable in each step.

eg: $S \to aAS \mid aSS \mid \epsilon$ , $A \to SbA \mid ba$
(for gen aa b aa)




## Right deⁿ tree

apply prodⁿ to right most var in each step




## # Ambigous grammar

a grammar is said to be ambigous if there exists 2 or more derivation trees for a string w (that means 2 or more left derivation tree)

eg: $G = (\{S\}, \{a+b, +, *\}, P, *)$ where P is such that
$S \to S+S \mid S*S \mid a \mid b$

the string a+a*b can be gen. by

| left | left |
|---|---|
| $S \to S+S$ | $S \to S*S$ |
| $S \to a+S*S$ | $S \to S+S*b$ |
| $S \to a+a*b$ | $S \to a+a*b$ |

(2 diff methods)