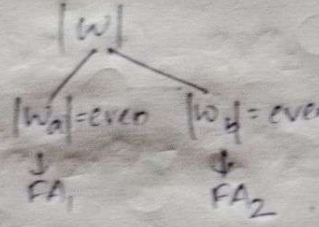
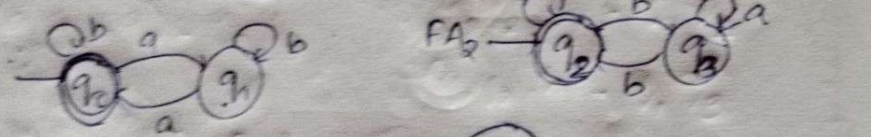
# Compound Automata

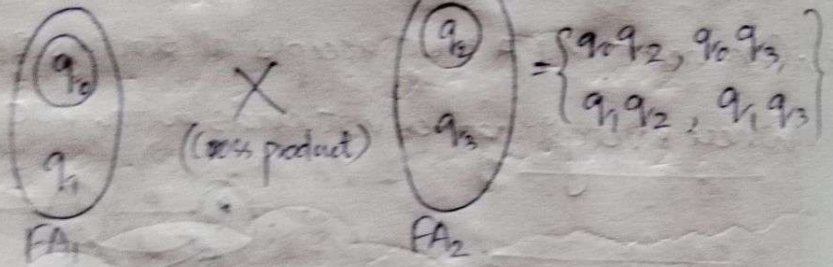
* The resultant FA obtained after performing operations like 'U', 'n', '-' on given FAs is called as compound Automata.

05/09

Q. Construct a NFA that accepts all the strings of a's & b's where every string contains, even no. of a's and even no. of b's




(Transition)

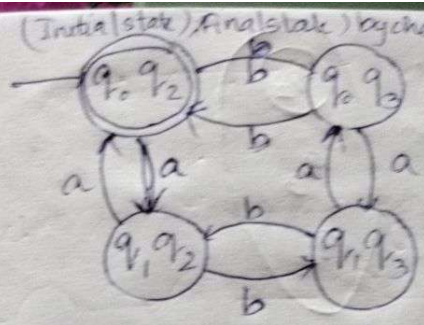
$\delta(q_0q_2, a) = q_1q_2$ ; $q_0$ on a → $q_1$, $q_2$ on a → $q_3$

$\delta(q_0q_2, b) = q_0q_3$ ; $q_0$ on b → $q_0$, $q_2$ on b → $q_3$

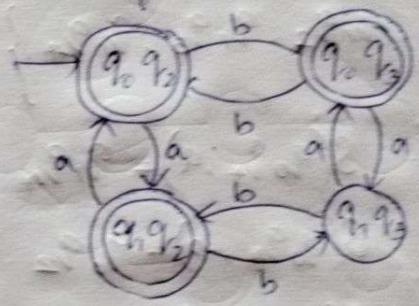
$\delta(q_0q_3, a) = q_1q_3$ ; $\delta(q_1q_2, a) = q_0q_2$ ; $\delta(q_1q_3, a) = q_0q_3$

$\delta(q_0q_3, b) = q_0q_2$ ; $\delta(q_1q_2, b) = q_0q_3$ ; $\delta(q_1q_3, b) = q_1q_2$

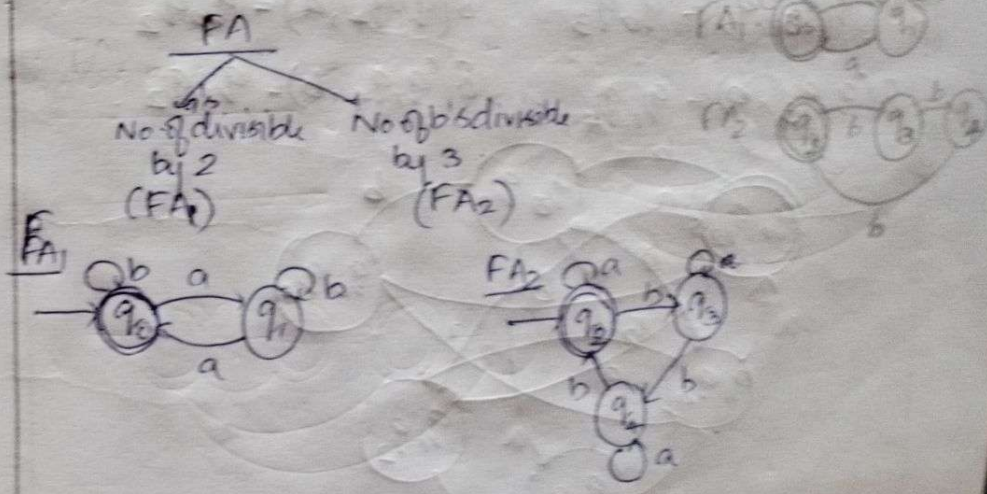
(Initial state) (Final state) by checking individual

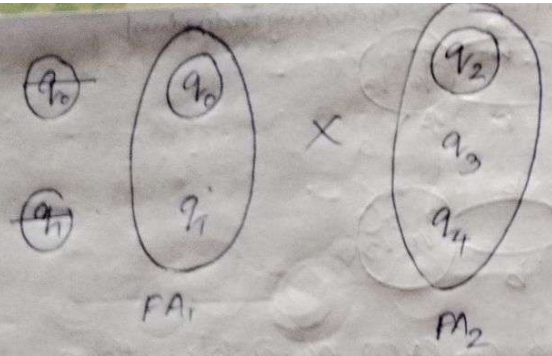



6. Construct NFA if even no. of a's & even no. of b's, then (of final state will be where $q_0q_2$)



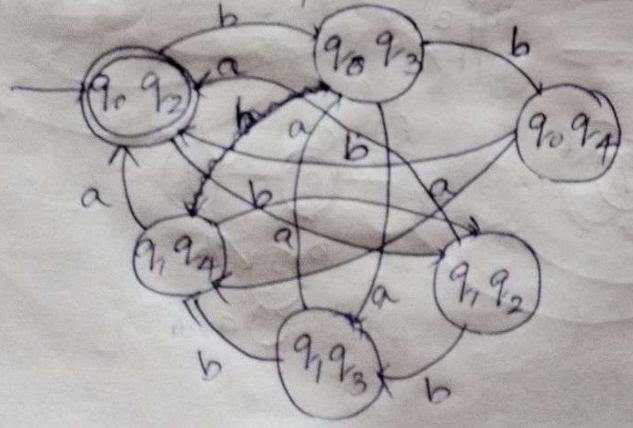

6. Construct NFA that accepts every string of a's & b's where every string has no. of a's divisible by 2 and no. of b's divisible by 3

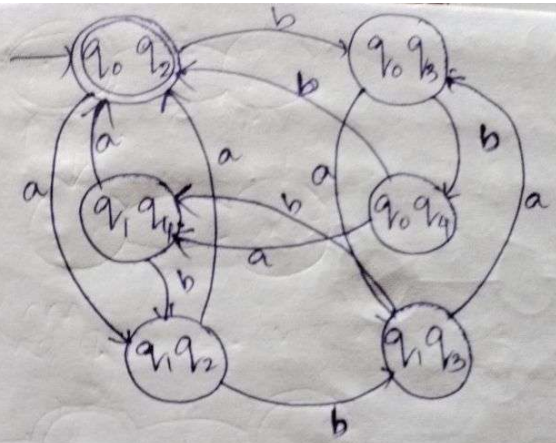




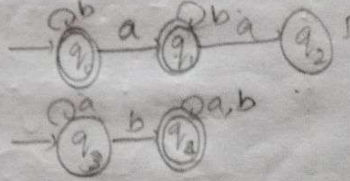

$= \{q_0q_2, q_0q_3, q_0q_4, q_1q_2, q_1q_3, q_1q_4\}$

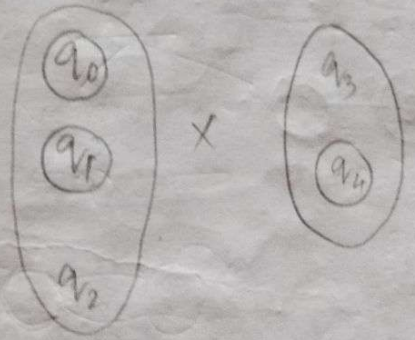
$\delta(q_0q_2, a) = q_1q_2$
$\delta(q_0q_2, b) = q_0q_3$
$\delta(q_0q_3, a) = q_1q_3$
$\delta(q_0q_3, b) = q_0q_4$
$\delta(q_1q_3, a) = q_0q_3$
$\delta(q_1q_3, b) = q_1q_4$
$\delta(q_0q_4, a) = q_1q_4$
$\delta(q_0q_4, b) = q_0q_2$
$\delta(q_1q_2, a) = q_0q_2$
$\delta(q_1q_3, b) = q_1q_3$
$\delta(q_1q_4, a) = q_0q_2$
$\delta(q_1q_4, b) = q_1q_2$




Q. Construct MFA that accepts all strings of a's & b's where every string contain atmost one 'a' and atleast one 'b'.




$\{q_0q_3, q_0q_4, q_1q_3, q_1q_4, q_2q_3, q_2q_4\}$

$\delta(q_0q_3, a) = q_1q_3$
$\delta(q_0q_3, b) = q_0q_4$
$\delta(q_0q_4, a) = q_1q_4$
$\delta(q_0q_4, b) = q_0q_4$
$\delta(q_1q_3, a) = q_2$

23/09

# Non-Deterministic Finite Automata (NFA):

* Ambiguity in selecting state → $(q_0) \xrightarrow{0} (q_1)$

* In NFA, the FA which can have 0 or 1 or more than one transition from a given state for a given symbol is called as NFA :

* NFA can be defined by 5 tuples : $(Q, \Sigma, \delta, F, q_0)$

$Q \rightarrow$ set of states  $F \rightarrow$ set of final states

$\Sigma \rightarrow$ alphabets  $q_0 \rightarrow$ Initial state.

$\delta \rightarrow$ ~~transition state~~ transition function $Q \times \Sigma \rightarrow 2^Q$ (NFA). $Q \times \Sigma \rightarrow Q$ (DFA)

$Q \rightarrow$ single state

$2^Q \rightarrow$ set of states

## Properties of NFA

* The capabilities of NFA and DFA are same.

ie, L(NFA) = L(DFA)

* Construction of NFA is easier than DFA.

* NFA is more powerful than DFA, but DFA is more efficient than NFA.
(one string has multiple transition paths)
(no more path taken)

* NFA takes care of only valid construction and no need to take care of invalid constructions.

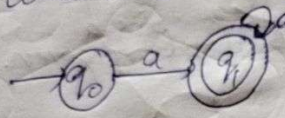
* No concept of Death State in NFA.

* No concept of compliment in NFA.

* Every DFA is a kind of NFA and NFA can be

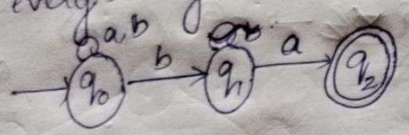
converted to DFA.

Q.1 Construct an NFA that accepts all strings of a's and b's where each string starts with 'a'.




transition for b : 0 or 1 or more than one

Q2. Construct an NFA that accepts all strings of a's & b's where every string ends with 'ba'.

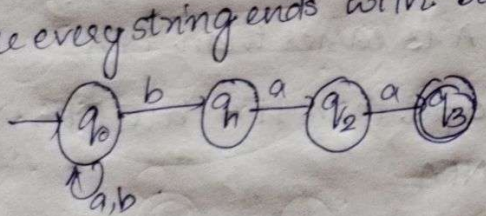



check only path exist:

abbba → ba will be accp

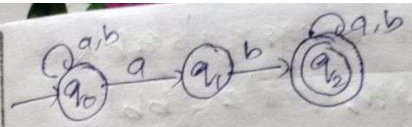
first loop

Q3. Construct an NFA that accepts all strings of a's & b's where every string ends with 'baa'.

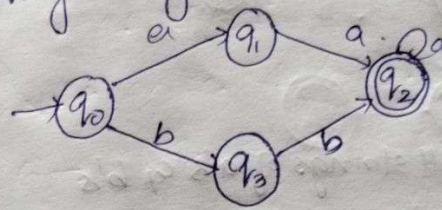



Q4 Construct an NFA that accepts all strings of a's & b's where every string contain substring 'ab'.

w = xabx




Q5. Construct an NFA that accepts all strings of a's & b's where every string starts with 'aa' or 'bb'



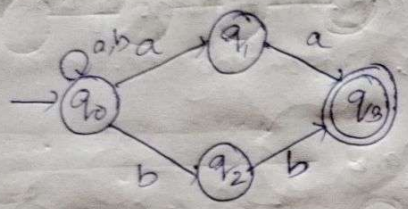

## Acceptance of NFA

* Let x be any string from alphabet Σ, corresponding to x there can be multiple transition path starting either from initial state.
* out of these transition paths if there exists transition paths which start at the initial state and ends at final state, then the string x is accepted by NFA.
* Set of the set of all strings which are accepted by NFA is called as language of NFA.

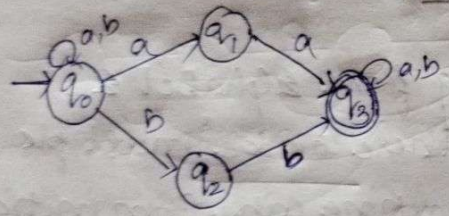
$$L(NFA) = \{ x \mid \delta(q_0, x) = F.S \}$$

24/09

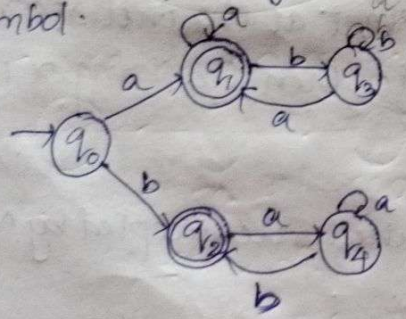
Q6. Construct NFA that accepts all strings of a's & b's where every string ends with 'aa' or 'bb'.



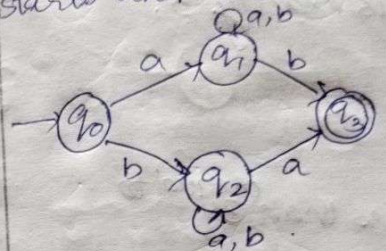

Q7. Construct NFA that accepts all strings of a's & b's where every string contain 'aa' or 'bb'.



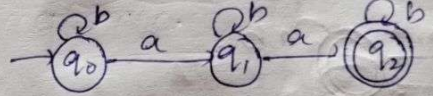

Q8. Construct NFA that accepts all strings of a's & b's where every string starts and ends with the same symbol.



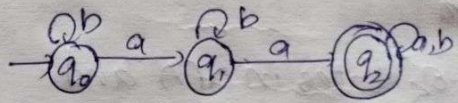

Q9. Construct an NFA that accepts all strings of a's & b's starts and ends with different symbol.



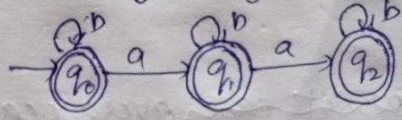

Q10. Construct an NFA that accepts all strings of a's & b's where every string contain exactly two a's.




Q11. Construct an NFA that accepts all strings of a's & b's where every string contain atleast two a's.



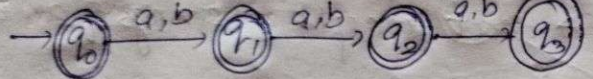

Q12. Construct an NFA that accepts all strings of a's & b's where every string contain atmost 2 a's.



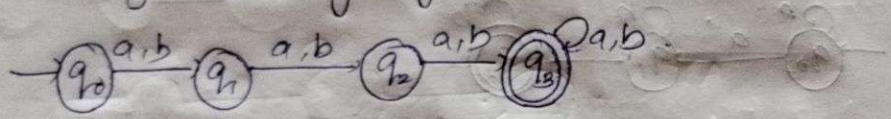
Q13. Construct an NFA that accepts all strings of a's & b's

a) where length of the string is exactly 3.


b) where length of the string is less than or equal to 3.




c) where length of string is greater than or equal to 3.




01/10 Equivalence between NFA and DFA

Process of conversion of NFA to DFA

* The process of conversion of NFA to DFA is also called as subset construction
* Let NFA has no of no. of states in NFA and m be no. of states in DFA.

Then Relation between NFA and DFA is

$$1 \leq m \leq 2^n.$$

* $m = 1$, is the best case
* $1 < m < 2^n$, is called as the ~~worst case~~ average case.
* $m = 2^n$ is called as the worst case.

Procedure

Let $M = (Q, \Sigma, \delta, q_0, F)$ for NFA and

$M' = (Q', \Sigma', \delta', q_0', F')$ for DFA.

Steps:

I: Initial state:

* There is no change in initial state in conversion of NFA to DFA.

$\therefore q_0' = q_0$.

II: Construction of $\delta'$

* Start the construction of $\delta'$ with the initial state and continue the process for every new state that appears in the input column and terminate the process whenever there is no new state appears in the input column.

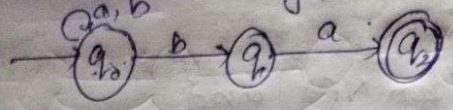
$$\delta'(q, x) = \delta(q, x)$$

$$\delta'(q_0 q_1 q_2 \dots q_n, x) = \bigcup_{i=0}^{n} \delta(q_i, x)$$

III: Final state

* Every subset which contain the final state of NFA is the a final state in DFA.

Q1) Ex: NFA (ending with ba)

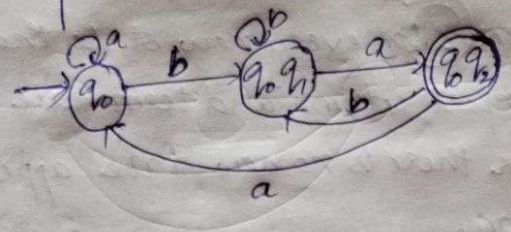


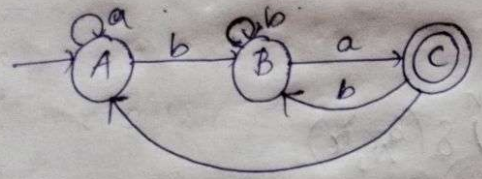

NFA to DFA.

① $q_0$ (Initial state)

② 

| $\delta'$ | a | b |
|---|---|---|
| → $q_0$ | $q_0$ | $q_0q_1$ |
| (new state) → $q_0q_1$ | $q_0q_2$ | $q_0q_1$ (old state) |
| $q_0q_2$ | $q_0$ | $q_0q_1$ |




③ $q_2$ appearing state as final state. (subset appearing containing $q_2$)

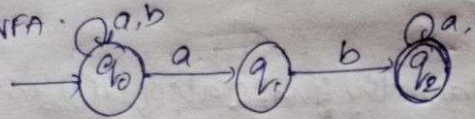
* DFA

A = $q_0$

B = $q_0q_1$

C = $q_0q_2$

Q2) NFA.



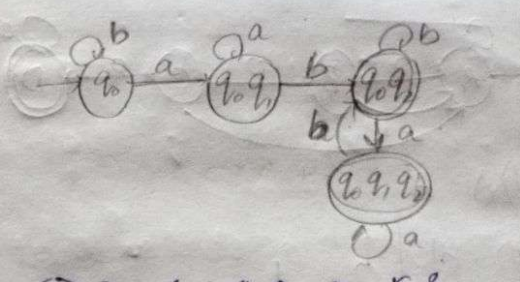

(containing ab)

NFA to DFA.

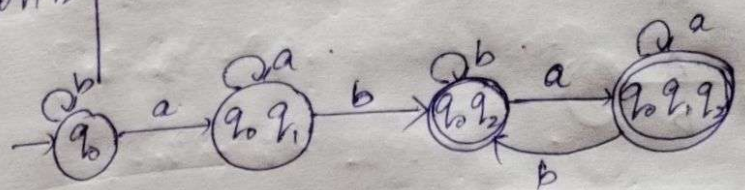
i) $q_0$ : $q_0$

② 

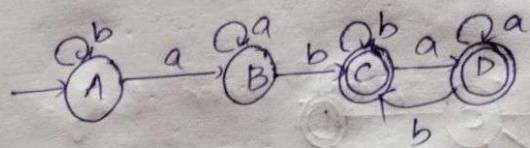
| $\delta'$ | a | b |
|---|---|---|
| $q_0$ | $q_0q_1$ | $q_0$ |
| $q_0q_1$ | $q_0q_1$ | $q_0q_2$ |
| $q_0q_2$ | $q_0q_1q_2$ | $q_0q_2$ |
| $q_0q_1q_2$ | $q_0q_1q_2$ | $q_0q_2$ |

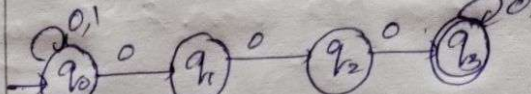



③ $q_0q_2$ & $q_0q_1q_2$ are F.S.




DFA.




Q3) 

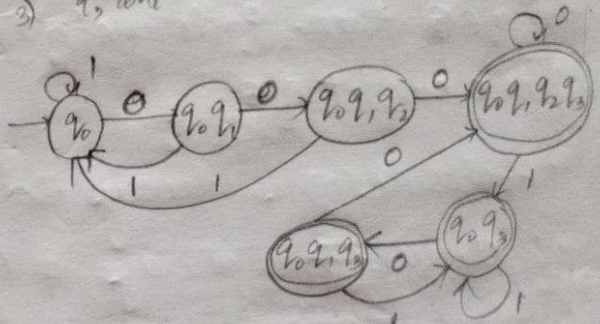

contains 000's

i) $q_0 = q_0$

ii) 

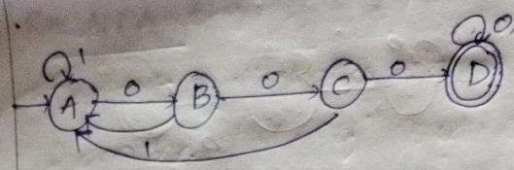
| $\delta'$ | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0q_1$ | $q_0$ |
| $q_0q_1$ | $q_0q_1q_2$ | $q_0$ |
| $q_0q_2$ | $q_0q_1q_3$ | $q_0$ |
| $q_0q_1q_2q_3$ | $q_0q_1q_2q_3$ | $q_0q_3$ |
| $q_0q_3$ | $q_0q_1q_3$ | $q_0q_3$ |
| $q_0q_1q_3$ | $q_0q_1q_2q_3$ | $q_0q_3$ |

iii) $q_3$ conf



Ans.

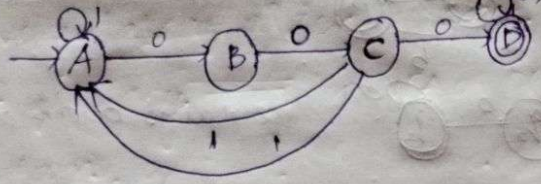
## NFA to DFA




1) A = A

2)

| $\delta'$ | 0 | 1 |
|---|---|---|
| A | B | A |
| B | C | A |
| C | D | A |
| D | D | D |




08/10

## ε-NFA

*NFA which gives transition for even empty string is called as ε-NFA.

*It is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$
- Q - set of states
- Σ - alphabet
- δ - transition function
- $q_0$ - initial state.
- F → set of F.S

*Differs in δ (transition f^n)

ie, $\delta \rightarrow Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$ (lies in powerset of Q)

DFA
$Q \times \Sigma \rightarrow Q$
NFA
$Q \times \Sigma \rightarrow 2^Q$
ε-NFA
$Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$

### Properties of ε-NFA

*The capabilities of ε-NFA & NFA and DFA is same; ie language

*ie, L(ε-NFA) = L(NFA) = L(DFA)

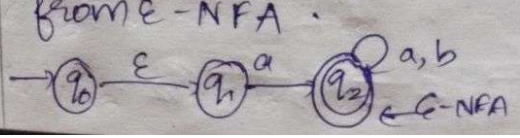
*Inclusion and exclusion of ε-transition to the NFA will not affect the language of NFA.

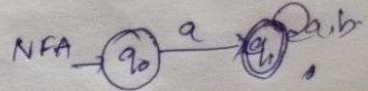
*ε-NFA will provide additional programming convenience compared with NFA.

ie ε-NFA → NFA →(100%) DFA, ε-NFA → DFA (50%)

[NFA →(100%) DFA requires 100% minimisation while ε-NFA to DFA requires 50% minimisation]

ε-NFA to NFA only required to remove ε from state transition



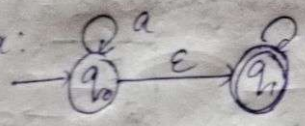



## ε-closure(q)

* The set of all states which are at zero distance from the state 'q' is called as ε-closure(q).

or

* The set of all states that can be reached from the state 'q' along ε labeled transition paths.

Ex: 


ε-closure($q_0$) = { $q_0$, $q_1$ } (zero distance from $q_0$)

ε-closure($q_1$) = { $q_1$ }.

Notes:

* Every state is at zero distance from itself.
* If 'q' is any state in NFA or DFA then the ε-closure(q) contains q.
* If $q \in Q$ then ε-closure(q) is a non-empty (one state will be there) finite subset of Q.
* ε-closure(q) is always a closed set.
* ε-closure(ϕ) is always ϕ (empty set).

## Equivalence between ε-NFA and NFA

I * No change in the initial state.

II * No change in the total number of states.

III * May be change in the final states.

Procedure

Let $M = (Q, \Sigma, \delta, q_0, F)$ for ε-NFA, then $M' = (Q', \Sigma', \delta', q_0', F')$ for NFA.

(1) Initial state

$q_0' = q_0$

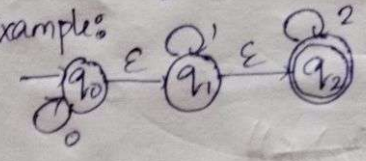
(2) Construction of transition function (δ')

$\delta'(q, x) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q), x))$

(3) Final state

Every state whose ε-closure contains the final state of ε-NFA is a final state in NFA.

Example:

1) 


→ ε-NFA $\Sigma = \{0, 1, 2\}$.

ε-closure($q_0$) = { $q_0, q_1, q_2$ }.

ε-closure($q_1$) = { $q_1, q_2$ }

ε-closure($q_2$) = { $q_2$ }

$\delta'(q_0, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0), 0))$

$\varepsilon\text{-closure}(q_0, 0)$

$= \varepsilon\text{-closure}\{\delta(q_0, q_1, q_2, 0))$

$= \varepsilon\text{-closure}\{q_0\}$

$= q_0 q_1 q_2$ $\quad \delta'(q_0, 2)$

$\delta'(q_0, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0), 1))$

$= \varepsilon\text{-closure}(\delta(q_0 q_1 q_2, 1))$

$= \varepsilon\text{-closure}\{q_1\}$

$= q_1 q_2$

$\delta'(q_0, 2) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0), 2))$

$= \varepsilon\text{-closure}(\delta(q_0 q_1 q_2, 2))$

$= \varepsilon\text{-closure}\{q_2\}$

$= q_2$

$q_1 \xrightarrow{\varepsilon}$

$q_2 \xrightarrow{\varepsilon}$

~~$\delta'(q_1, \varepsilon) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1), \varepsilon))$~~

~~$= \varepsilon\text{-closure}(\delta(q_1 q_2, \varepsilon))$~~

$q_1 \xrightarrow{\varepsilon} q_1 q_2$ $\quad q_1 q_2 \quad q_1 q_2$

$\downarrow 0 \to \phi \xrightarrow{\varepsilon} \phi \quad \downarrow 1 \to q_1 \xrightarrow{\varepsilon} q_1 q_2 \quad \downarrow 2 \to q_2 \xrightarrow{\varepsilon} q_2$

$q_2 \xrightarrow{\varepsilon} q_2$ $\quad q_2 \quad q_2$

$\downarrow 0 \quad \downarrow 1 \quad \downarrow 2$

$\phi \quad \phi \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_2$

$\downarrow \varepsilon$

$\phi$

07/10

# Minimization of FA

* The process of detection and elimination of equal and unreachable state in the FA is called as minimization of FA.
* If DFA contain deathstate, then we need to maintain Death state in MFA, ie MFA can contain deathstate.
* The FA which is free from equal and unreachable states is called as MFA.
* A language can be represented by more than one DFA but it can be represented only one ~~DFA~~ MFA.
* That is, FA is not unique ~~and~~ but MFA is unique.

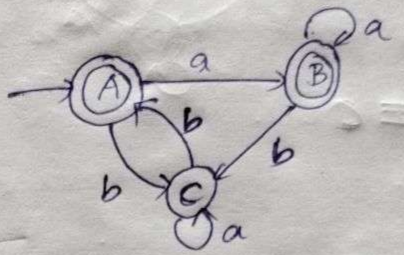
Methods of Minimization of FA

1. State equivalence Method
2. Table filling Algorithm

State equivalence Method

* Define the set Q of all states into mutually disjoint subsets by defining the equivalence relation over the set Q.
* The number of equivalence class is equal to number of disjoint subset equal to number of states in MFA.

Ex: 1.



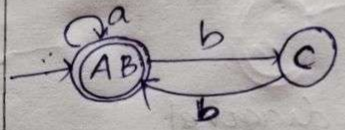

$Q = \{A, B, C\}$ (set of states)

$\pi_0 = \{(A, B), (C)\}$ (set of F.S), (set of Non-F.S)

$G_1 \quad G_2$

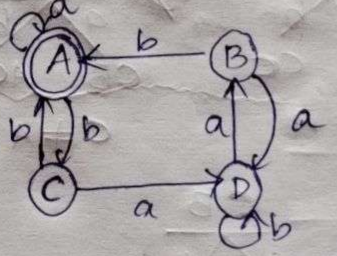
| | a | b |
|---|---|---|
| A | $G_1$ | $G_2$ |
| B | $G_1$ | $G_2$ |

$\therefore A \equiv B$




→ M.F.A

2. Reduce DFA to MFA




$Q = \{A\ B\ C\ D\}$

$\pi_0 = \{(A), (B\ C\ D)\}$

$G_1 \quad G_2$

| | a | b |
|---|---|---|
| ~~A~~ | ~~$G_1$~~ | ~~$G_2$~~ |
| B | $G_2$ | $G_1$ |
| C | $G_2$ | $G_1$ |
| D | $G_2$ | $G_2$ |

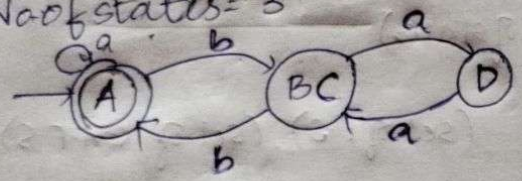
B C combine

$\pi_1 = \{(A), (BC), (D)\}$
$G_1 \quad G_2 \quad G_3$

| | a | b |
|---|---|---|
| B | $G_3$ | $G_1$ |
| C | $G_3$ | $G_1$ |

$\therefore B \equiv C$

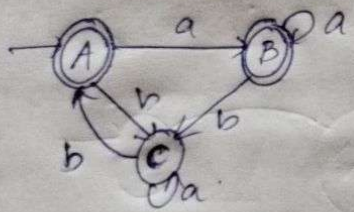
No of states = 3




M.F.A

Table Filling Algorithm

$Q = \{A, B, C\}$.

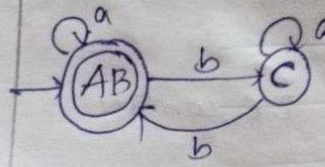



Draw matrix & consider only lower diagonal

| | A | B | C |
|---|---|---|---|
| A | | | |
| B | BA | | |
| C | CA | CB | |

BA
a b
BB CC
(same) $B \equiv A$

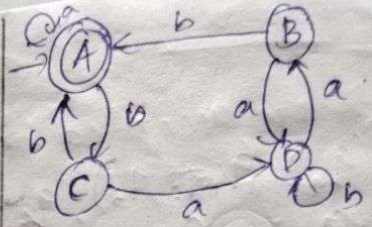
CA CA
a b
CB AC
$C \neq A$




MFA

CB
a b
CB AC
$C \neq B$

2.




$Q = \{A, B, C, D\}$.

| | A | B | C | D |
|---|---|---|---|---|
| A | | | | |
| B | BA | | | |
| C | CA | CB | | |
| D | DA | DB | DC | |

BA
a b
DA AC
$B \neq A$

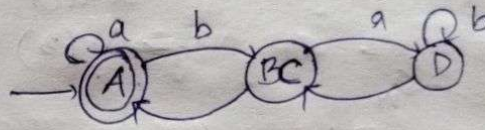
CA
a b
DA AC
$C \neq A$

CB
a b
DD AA
$C \equiv B$

DA
a b
BA DC
$B \neq A$

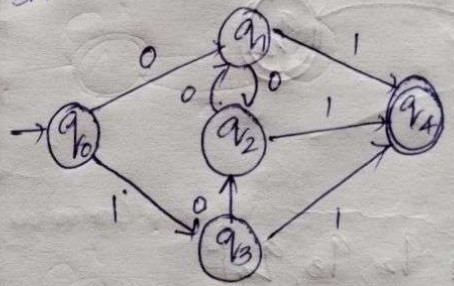
DB
a b
BD DA
$D \neq B$

DC
a b
BA DA
$D \neq C$




M.F.A
no of states = 3

3.




- Reduce this DFA to MFA using state equivalence & table filling.

State-equivalence

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

$\pi_0 = \{(q_4), (q_0, q_1, q_2, q_3)\}$
$G_1 \quad G_2$

| | 0 | 1 |
|---|---|---|
| $q_0$ | $G_2$ | $G_2$ |
| $q_1$ | $G_2$ | $G_1$ |
| $q_2$ | $G_2$ | $G_1$ |
| $q_3$ | $G_2$ | $G_1$ |

$\therefore \pi_1 = \{(q_4), (q_1, q_2, q_3), (q_0)\}$
$G_1 \quad G_2 \quad G_3$

| | 0 | 1 |
|---|---|---|
| $q_1$ | $G_2$ | $G_1$ |
| $q_2$ | $G_2$ | $G_1$ |
| $q_3$ | $G_2$ | $G_1$ |

$\therefore q_1 \equiv q_2 \equiv q_3$

States: $q_0, q_1, q_2, q_3, q_4$

(3)

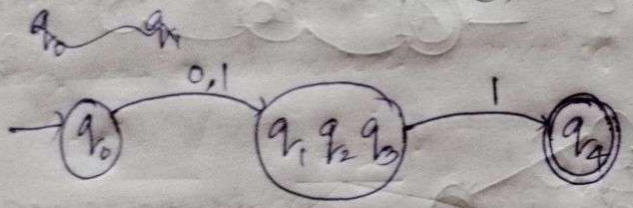
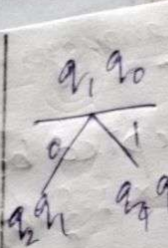



Table fill Algorithm

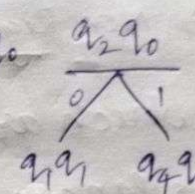
$q_0 \quad q_1 \quad Q = \{q_0, q_1, q_2, q_3, q_4\}$.

| | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ |
|---|---|---|---|---|---|
| $q_0$ | | | | | |
| $q_1$ | $q_1 q_0$ | | | | |
| $q_2$ | $q_2 q_0$ | $q_2 q_1$ | | | |
| $q_3$ | $q_3 q_0$ | $q_3 q_1$ | $q_3 q_2$ | | |
| $q_4$ | $q_4 q_0$ | $q_4 q_1$ | $q_4 q_2$ | $q_4 q_3$ | |

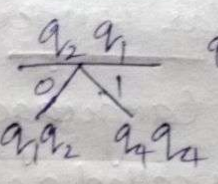
$q_1 \neq q_0$

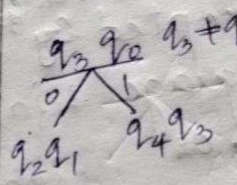



$q_2 \neq q_0$

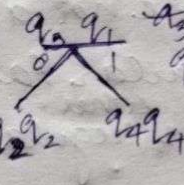



$q_2 \neq q_1$



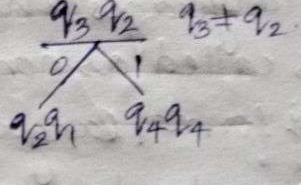




$q_3 \neq q_0$

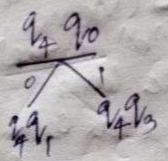


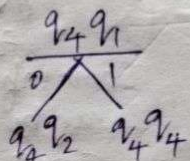

$q_3 \neq q_1$

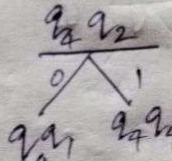
$q_3 = q_1$

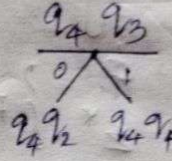



$q_3 \neq q_2$



8/10. **Regular Languages & Non-Regular Languages**

(RL) * The languages which are accepted by FA is called as regular language. Ex: $L=\{ab\}$, $\{aa, ab\}$

(NRL) * The languages which are not accepted by FA is called as non-regular language. $L=\{a^n b^n\}$ (infinite)

* Every formal language is either regular or non-regular language. $\{ab, aabb, aaabbb, \ldots\}$

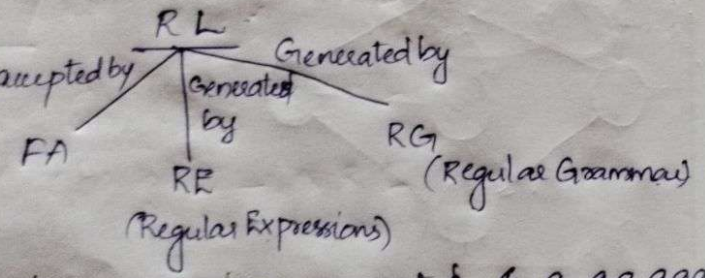
Ex: $L=\phi$ } RL
$L=\Sigma^*$ (infinite lang)

(NRL) (Memory element for $a^n b^n$ required)

Some infinite lang can be regular lang. Ex: $\Sigma^*$

* If L is finite language then its a regular language

* Every regular language is accepted by ~~FA~~ a FA.




Ex: $a^*$ → Regular expression, generates $\varepsilon, a, aa, aaa \ldots$

$(a+b)$ → RE → generates $\{a, b\}$

$(a+b)^*$ → generated by any combinations of $(a+b)$.

Q. Verify which of the following languages are regular?

(1) $L=\{a^m b^n \mid m, n \geq 0\}$ ∴ R.L ~~Since no. of are not fixed, additional memory requirement~~

(2) $L=\{a^m b^n \mid m, n \geq 1\}$ ∴ R.L

(3) $L=\{a^m b^n \mid mn \text{ is finite}\}$ → R.L

(4) $L=\{a^m b^n \mid m < n < 2^n\}$ → R.L

(5) $L=\{a^m b^n \mid m, n \geq 3\}$ → R.L

* R.L ~~from~~ $a^n b^n$: as n size is not fixed, it may require additional memory requirement.

Here its $a^m b^n$, both may vary and ~~can~~ no FA can be constructed.

(6) $L=\{a^m b^n \mid m=n\}$ ∴ N.R.L (Mem requirement)

(7) $L=\{a^m b^n \mid m<n\}$ ∴ NRL (a & b no store) (comparison store)

(8) $L=\{a^m b^n \mid m>n\}$ ∴ NRL

(9) $L=\{a^m b^n \mid m \neq n\}$ ∴ NRL

(10) $L=\{a^m b^n \mid m=2n\}$ ∴ NRL

(11) $L=\{a^m b^n \mid m \text{ divides } n\}$ ∴ NRL (for calculating relation there should be memory element)

1) $L=\{w \in \{a,b\}^* / |w|_a \equiv 0 \bmod 2\}$ ($|w|_a$ → No of a)

(remainder 0 when divided by 2)

* R.L (even no of a)

2) $L=\{w \in \{a,b\}^* / |w| \equiv r \bmod n\}$ → * RL

(length of string $r>n$)

3) $L=\{w \in \{a,b\}^* / |w|_a \equiv |w|_b\}$ → * N.R.L $\{ab, abab, \ldots\}$

4) $L=\{w \in \{a,b\}^* / |w|_a \leq |w|_b\}$ → * NRL

5) $L=\{w \in \{a,b\}^* / |w|_a = 2|w|_b\}$ → * N.R.L

6) $L=\{ww^R / w \in \{a,b\}^*\}$ → NRL

$ww^R$ → ~~reverse~~ R → reverse

$ww^R$ {abbbba}

~~7 L= {wcw}~~

7) $L=\{wcw^R / w \in \{a,b\}^*\}$ → NRL

c → fixed symbol

8) $L=\{ww^Rc / w \in \{a,b\}^*\}$ → NRL

9) $L=\{cww^R / w \in \{a,b\}^*\}$ → N.R.L

Properties of Regular languages

* Every finite language is regular.
* An infinite language can be either regular or non-regular.
* A regular language can be finite or infinite.

* Non-regular language is always infinite.
* Every subset of a regular set need not be regular.

Ex $L_1=\{a^m b^n / m,n \geq 0\}$ R.L, $L_2=\{a^m b^n / m=n\}$ N.R.L

Here $L_2$ is subset of $L_1$ but $L_2$ is NR.L

* Regular languages is also called as regular sets.
* Every finite subset of a regular set is regular.
* Every subset of a non-regular set need not be non-regular.
* Every finite subset of a non-regular set is regular (Finite)
* Superset of a regular set need not be regular.
* Superset of a non-regular set need not be non-regular.
* Finite union of regular set is always regular.
* Infinite union of regular sets need not be regular.

$L=\{\{ab\},\{aabb\},\{aaabbb\}\ldots\}$ U = $L=\{a^m b^n / m=n\}$ NRL

* Finite intersection of regular set is always regular.
* Infinite intersection of regular sets may not be regular.

10/10.

## Regular Expression

* An expression which generates regular language is called regular expression.

(OR)

* An expression which is constructed over the string using the operators *, + and . is called as regular expressions.

* Ex: $r = a + b = \{a, b\}$ (OR +)

$r = ab = \{ab\}$ (AND .)

$r = a * b = \{$

$r = a^* = \{\varepsilon, a, aa, \ldots\}$

## Regular Operators

* Operators are: *, +, .

* → Kleen (or c) closure

+ → Union

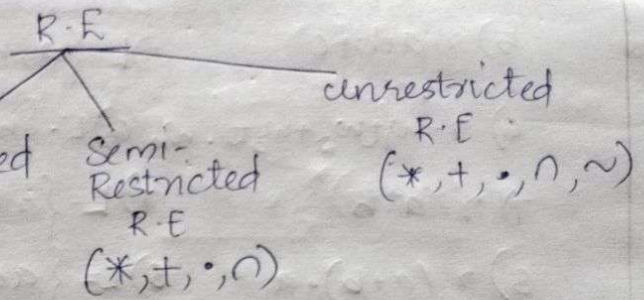
. → Concatenation

* The order of precedence: * (1) . (2) + (3)

## Types of regular expressions

Three types:

1. Restricted R.E
2. Semi Restricted R.E
3. Unrestricted R.E




## Properties of Regular Expressions

* A regular expression always generate regular language
* A non-regular language cannot be represented by regular expression.
* Regular expression cannot be constructed for non-regular languages
* Algebraic expressions always represents numerical value. Ex: $(0+1)0 = 00, 10$ (AE)

$(0+1)0 = \{00, 10\}$ (RE)

(* → 2, . → 1)

Examples of R.E and corresponding R.L:

| | R.E | R.L |
|---|---|---|
| 1) | $r = \phi$ | 1) $L = \{\} = \phi$ |
| 2) | $r = \varepsilon$ | 2) $L = \{\varepsilon\}$ |
| 3) | $r = a$ | 3) $L = \{a\}$ |
| 4) | $r = a + b$ | 4) $L = \{a, b\}$ |

(1), (2), (3): R.E without Regular operators

R.E R.L

5) $r = a+b+c$ 5) $L = \{a, b, c\}$.

6) $r = w_1 + w_2 + w_3 + \ldots w_n$ 6) $L = \{w_1, w_2, w_3 \ldots w_n\}$

7) $r = a \cdot b$ 7) $L = \{ab\}$

8) $r = (a+b) \cdot a$ 8) $L = \{aa, ba\}$.

* Prove that every finite language is regular.

Let L be a finite language

$\Rightarrow L = \{w_1, w_2, w_3 \ldots w_n\}$.

Inorder to prove that L is a regular language, it is sufficient to prove that L is generated by some regular expressions.

Now, the regular expression can be constructed by inserting the + operator between the strings.

$r = w_1 + w_2 + w_3 + \ldots + w_n$

∴ Evergn We generated regular expression for the above regular language

$\Rightarrow$ L is a regular language.

$\Rightarrow$ Every finite language is regular.

Properties:

* If R is the r is the regular expression, then $L(r)$ is the language generated by regular expression

* If $r_1$, $r_2$ be two regular expression, then both $r_1 + r_2$ and $r_1 \cdot r_2$ are also regular expressions.

* $L(r_1) + L(r_2) = L(r_1 + r_2)$

* $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$

* If r is a regular expression both $r^*$ and $r+$ are also regular expressions.

* $L(r^*) = [L(r)]^*$

* $L(r+) = [L(r)]^+$

* $(r^*)^* = r^*$ $r^*, r^* = \{\varepsilon, r, rr \ldots\}$.

* $(r+)^+ = r^+$ $r+ = \{r, rr, rrr \ldots\}$

* $r^* \cdot r = r+$ r

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

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

Q.1 Construct regular expression that generates all the strings of a's & b's including $\varepsilon$.

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

$\rightarrow r = (a+b)^*$

Q2. Construct R.E that generates all strings of a's & b's excluding $\varepsilon$.

$\rightarrow r = (a+b)^+$

Q3 Construct R.E
where every string starts with 'a'

$r = a \cdot (a+b)^*$

Q4
where every string ends with 'ba'

$w = xba$

$r = (a+b)^* \cdot ba = (a+b)^* ba$

Q5
where every string starts and ends with 'a'

$w = axa, a$ $\quad x \to a+b$

$r = a \cdot (a+b)^* \cdot a + a$

$= a(a+b)^* a + a$ $\quad L = \{a, aa, aba, aaa\}$

$+ \to OR.$

14/10
Q6. construct RE
where every string ~~strah~~ starts and ends with same symbol.

$w = axa, bxb, a, b$

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

14/10.
Q7 Construct R.E that generates all strings of a's & b's where every string starts and ends with different symbol.

$w = axb, bxa$

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

Q8
where every string contains exactly two a's.

$x \to b^*$

$w = xaxax$

$r = b^* a b^* a b^*$

Q9.
where every string contains atleast two a's.

$w = xaxax$

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

Q10
where every string contains atmost two a's

$w = x(a+\varepsilon)x(a+\varepsilon)x$

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

Q11 Construct R.E(r)
where every string contains number of a's is even.

$w = \{(xaxax)^*$

$r = (b^*ab^*ab^*)^*b^*$

Q12 where every string contains number of a's is odd.

$w = (xaxax)^*xax$

$r = (b^*ab^*ab^*)^*b^*ab^*$

Q13 where length of the string is 3.

$w = xxx$

$r = (a+b)(a+b)(a+b) = (a+b)^3$

Q14 where length of the string is atleast 3.

~~$w = (xxx)^*$~~ $w = xxxy$

$r = (a+b)^3(a+b)^*$

Q15 where length of the string is atmost 3.

~~$w = (a+b+\varepsilon)^3$~~ $w = (x+\varepsilon)(x+\varepsilon)(x+\varepsilon)$ $(x+\varepsilon)(x+\varepsilon)(x+\varepsilon)$

$r = (a+b+\varepsilon)^3$

Q16 where the length of the string is even.

~~$w = (xx)^*$~~ $w = [x^2]^*$ ~~$(a+b+\varepsilon)^*$~~

$r = r = [(a+b)^2]^*$

Q17 where the length of the string is odd.

$w = [x^2]^*x$

$r = [(a+b)^2]^*(a+b)$

Q18 where the length of the string is 2 mod 3 2, 5, 8, 11...

$r = (a+b)^2[(a+b)^3]^*$

∴ If the length of string is r mod n, then: $r = (a+b)^r[(a+b)^n]^*$

Q19 where each string starts with a and length is odd.

$w = a, aba, ax$

$r = a[(a+b)^2]^*$

Q20 where each string starts with b and length is odd

$w = b, bx$

$r = b[(a+b)^2]^*$

Q21 where each string starts with b and length is even

$w = b[xx]^*x$

$r = b[(a+b)^2]^*(a+b)$

Q22 where every string contains fourth symbol from the left end is b.

$w = xxxbx$

$r = (a+b)^3 b(a+b)^*$

Q23 where every string contains fourth symbol from the right end is a.

$w = yaxxx$

$r = (a+b)^* a (a+b)^3$

Q24 where every string starts with a and do not contain 2 consecutive b's.

$w = ax$ → no 2 consecutive b's

$a^+ \rightarrow a, aa, aaa$

$ab^+ \rightarrow ab, abab$

$r = (a^+ + ab^+)^+ = (a+ab)^+$

$a^* ax b x b x$

$a + a^* a^* b a^*$

$(ba)^2$

$(ba)(ba)$

$baba$

$ab$

Q25 Construct R.E $a^m b^n$ such that $m+n$ = even

$m$ = even & $n$ = even → $a^{2x}$ $b^{2x}$

$m$ = odd & $n$ = odd → $a^{2x+1}$ $b^{2x+1}$

$r = [(aa)^*(bb)^* + (aa)^* a (bb)^* b]$

Q26 where $m+n$ = odd for $a^m b^n$

$m$ = even & $n$ = odd → $a^{2x}$ $b^{2x+1}$

$m$ = odd & $n$ = even → $a^{2x+1}$ $b^{2x}$

$r = (aa)^*(bb)^*b + (aa)^*a(bb)^*$

15/10 Equivalence between FA and RE

1) Arden's Lemma (can be used for DFA, NFA)
2) State Elimination Method (can be used for DFA, NFA, ε-NFA, Transition graph)

Arden's Lemma

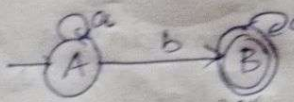
*Is used to find out the RE for a FA:
*It is not suitable for converting ε-NFA to RE.

Rule: Let P, Q be the two regular expressions, then

a) If 'P' doesn't contain 'ε', then the equation $R = Q + RP$ has unique solution.
∴ $R = Q + RP$ can be written as $R = QP^*$.

b) If 'P' contains 'ε', then the equation $R = Q + RP$ has infinite solutions.

Ex: 1)




P contains ε → comes when initial state is the F.S

$A = Aa + \varepsilon$ (Initial state) (A)

$B = Ab + B(a+b)$ (To state B)

Mapping $R = Q + RP$ to $A = Aa + \varepsilon$
R RP Q

$\therefore A = Aa + \varepsilon = \varepsilon + Aa$

Rhs

If P doesn't contain ε: P has no ε, then: $R = QP^*$

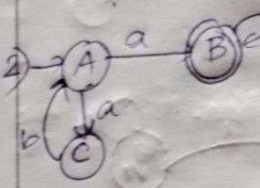
$\therefore A = \varepsilon a^* = a^*$

$B = Ab + B(a+b)$ Since P doesn't contain 'ε', so
R Q R P $R = QP^*$

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

$B = a^*b(a+b)^*$ R.E for FA

Ex: 2)




$R = Q + RP$

$A = \varepsilon + Cb$

$B = Aa + Bb$

$C = Aa$

R Q R P

$\therefore A = \varepsilon + Aab$ ($\because C = Aa$)

∴ RP does not contain ε, $R = QP^* = \varepsilon(ab)^* = (ab)^*$

$C = Aa = (ab)^*a$

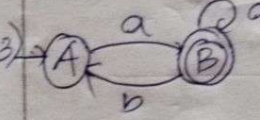
R Q RP

$B = Aa + Bb$ = ∴ $R = QP^*$ (P not contain ε)

$B = Aab^* = (ab)^*ab^*$

$\therefore RE = (ab)^*ab^*$

3)




$A = B\varepsilon + Bb$

$B = Aa + Ba$
Q RB

~~$A = \varepsilon + Aaa^*b$~~
~~$B = Q + RP^*$~~

~~$Q = R = QP^*$~~
~~$= \varepsilon(aa^*b)^*$~~

$A = \varepsilon + (Aa + Ba)b$
$= A = \varepsilon + Aab + Bab$
$Q = R = QP^*$
$= Aaa^*$

$A = \varepsilon + Aa\,Bb$

$A = \varepsilon + Aaa^*b$ $\qquad R = QP^*$

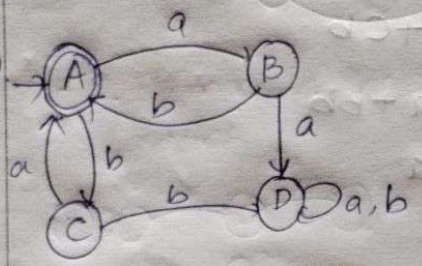
$R \quad Q \quad RP$ $\qquad A = \varepsilon(aa^*b)^*$

$A = (aa^*b)^*$

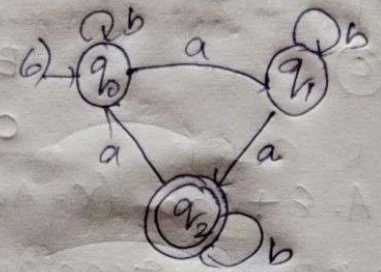
$B = (aa^*b)^*aa^*$

$R.E = (aa^*b)^*aa^* = (a+b)^*a+$

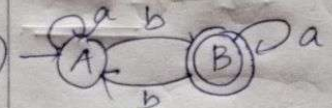
4)



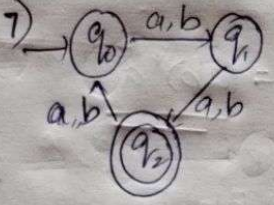

6)




5)




7)



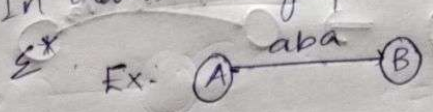

17/10. State Elimination Method

* This mechanism is normally used for FA's but can also be used for transition graph

* The transition graph is a graph that can have more than one initial state

* In transition graph, the edge table can be from $\Sigma^*$ (any combinations of letters)

Ex:

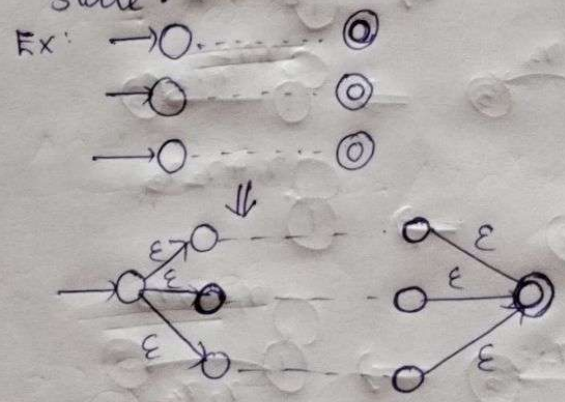



* $\delta : Q \times \Sigma^* \to 2^Q$ is the transition function

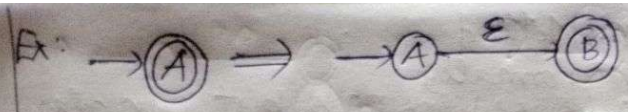
Algorithm for State Elimination Method

StepI: Simplify the transition graph such that it has only one initial state and one final state:

Ex:



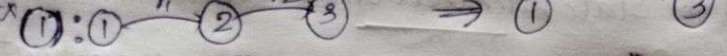

StepII: Simplify the system in a such a way that, it has different initial and final state.

Ex: 


Step III: If more than one edge exists between a pair of states in the same direction, they are called parallel edges. Parallel edges can be combined into a single edge using '+' operator.

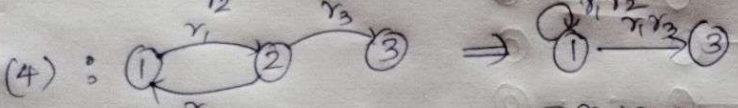
Ex: 


Step IV: Eliminate the state (2) from the following graphs

Ex (1): 

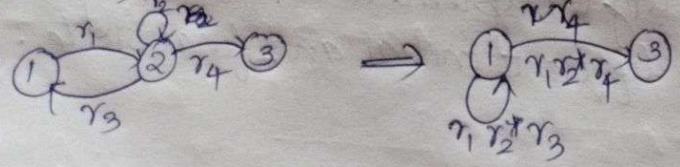

(2): 


(3): 

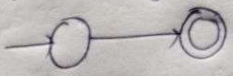

(4): 


(5): 


(6): 


(7): 

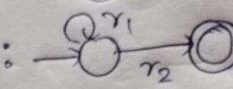

Step V: Continue the process of state elimination until the transition graph takes any one of the following forms

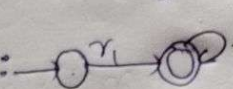
(1):  , RE = r

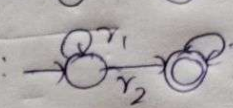
(2):  , RE = r*

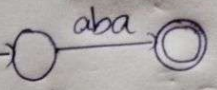
(3):  , RE = (r1 r2)*

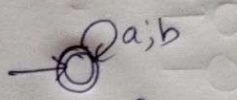
(4):  , RE = (r1 + r2)*

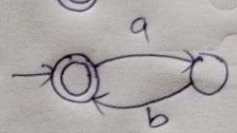
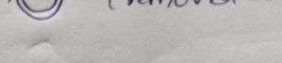
(5):  , RE = r1* r2

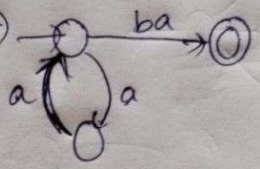
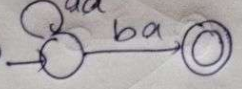
(6):  , RE = r1 r2*

(7):  , RE = r1* r2 r3*

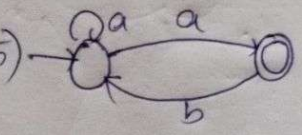
Ex: 1)  r = aba

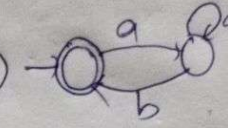
2)  r = (a+b)*

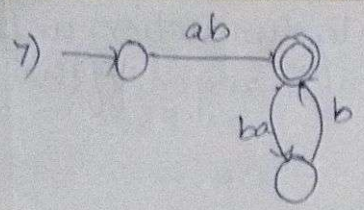
3)  ⇒  (removed st 2) ∴ r = (ab)*

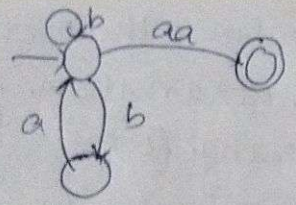
4)  ⇒  r = (aa)* ba

HW

5) 

6) 

7) 


8) 


21/10 Algebraic properties of Regular Expressions

I *closure property:
- RE is closed with respect to *, ·, +, ie if r is a RE, r* is also RE.
- If $r_1, r_2$, are RE then $r_1 + r_2$, $r_1 \cdot r_2$ are also RE.

II *Associative property:
- RE satisfies associative property ~~with respect~~ with respect to (+, ·) (union, concatenation).
- Ex: $(r_1 + r_2) + r_3 = r_1 + (r_2 + r_3)$.
- ~~Ex: $(r_1 \cdot r_2) r_3 = r_1 (r_2 \cdot r_3)$~~
- Ex: $r_1 (r_2 \cdot r_3) = (r_1 \cdot r_2) r_3$

III *Identity property:
- $r + x = r$
- $r \cdot x = r$

} x is called identity.

Ex: $a \equiv a + \phi$ (empty lang) (no empty string)
- $\phi$ is the identity element with respect to union
- $r \cdot x = r \Rightarrow x = \varepsilon$ is identity element respect to concatenation.

IV *Annihilator:
- $r + x = x$
- $r \cdot x = x$

} x is called annihilator.

- $r + x = x \Rightarrow x = ?$ No annihilator with respect to union
- $r \cdot x = x \Rightarrow x = \phi$, is annihilator with respect to concatenation

V *Distributive property:-
- RE satisfies ~~both~~ distributive properties with respect to concatenation and then on union.
- RE is not distributive with respect to union and then concatenation.

Ex: $(r_1 + r_2) \cdot r_3 = r_1 \cdot r_3 + r_2 \cdot r_3$
$r_1 \cdot (r_2 + r_3) = r_1 \cdot r_2 + r_1 \cdot r_3$
} satisfies

$r_1 + (r_2 \cdot r_3) \neq (r_1 + r_2) \cdot (r_1 + r_3)$ } Not satisfies.

$r_1 \cdot r_2 + r_3 \neq (r_1 + r_3) \cdot (r_1 + r_3)$ } Not satisfies.

VI *Commutative Property:-
- RE is commutative with respect to union (+) but not with respect to concatenation (·).
- Ex: $r_1 + r_2 = r_2 + r_1$
$r_1 \cdot r_2 \neq r_2 \cdot r_1$
$a^* \cdot b^* \neq b^* \cdot a^*$

VII* Idempotent Properties:

- RE satisfies idempotent properties w.r.t. union(+) but not w.r.t concatenation(.)
- Ex: $r \cdot r \neq r$

  $r + r = r$

Closure properties of Regular Languages

* RL satisfies the closure properties w.r.t the following operators:

1) Compliment
2) Klean Closure
3) Positive closure
4) Reverse operator
5) Prefix operator
6) Concatenation
7) Union
8) Intersection
9) Difference
10) Symmetric Difference
11) Substitution
12) Homomorphism
13) Inverse Homomorphism
14) Max (L)
15) Min (L)
16) Cycle (L)
17) Compliment OR
18) Compliment NOR.

* Substitution: Let $\Sigma$ and $\Delta$ be two ap alphabets then substitution is a kind of mapping from $\Sigma \rightarrow \Delta$ where the symbols of $\Sigma$ is replaced by RL of another alphabet $\Delta$.

* Homomorphism:- It is a kind of substitution from $\Sigma$ to $\Delta$ where the symbols of $\Sigma$ is replaced by a single string of another alphabet $\Delta$.