Finite languages are always Regular.

Reg. Language
FA
RE
Finite Infinite

* Regular Expressions

→ Regular expressions are used for representing certain sets of strings in an algebraic fashion.

→ The language accepted by FA can easily be described by simple expressions called RE. It is the most effective way to represent any language.

→ The languages accepted by some regular expressions are referred to as Regular languages. (language accepted by DA)

→ RE is a way of showing how a regular language is built from base set of regular languages.

→ e.g. $L = \{ \varepsilon, a, aa, aaa \dots \}$ $[a^*]$
i.e. a^* {

FA: → (q0) a

→ For each regular expression [E] there is a regular language [L] {and vice-versa}.

Let R be a regular expression over alphabet Σ, if R is:

or

An expression is regular if:

→ ε is a regular expression denoting the set / regular language $\{\varepsilon\}$.

$R = \varepsilon$, $L(R) = \{\varepsilon\}$

→ ϕ is a RE denoting the regular language ϕ (empty set)

$R = \phi$, $L(R) = \{\}$

→ For each symbol $a \in \Sigma$, a is a RE denoting language / set $\{a\}$

$R = a$, $L(R) = \{a\}$

Note: These three are called primitives, as these form the minimum languages i.e. $\{\varepsilon\}, \{\}, \{a\}$.

→ The union of two regular expressions is also a RE.

$R_1 + R_2$ is a RE corresponding to language $L(R_1) \cup L(R_2)$ ($L(R_1 + R_2) = L(R_1) \cup L(R_2)$)

(R_1 & R_2 are two RE, $R_1 \cup R_2 \rightarrow$ RE)

→ Concatenation of two RE is also a RE.

$R_1 . R_2$ = Regular Expression

e.g. $a \cup b = \{a, b\}$

→ Kleene closure * of RE is also regular expression.

a^* = Reg. Exp $\{\epsilon, a, aa, aaa, \ldots\}$

$(RE)^* = RE$

→ If R is regular, (R) is also regular.

→ The regular expression over ϵ are precisely those obtained recursively by the application of the above rules once or several times.

If R^* is a RE then the language corresponding to it will be $L(R^*)$ where

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

Note: Any set that represents the value of Regular Expression is called a Regular Set.

Language
↓
Finite

Language
↓
Infinite

* RE for Finite (Automata)

→ If the language is finite, then it is regular.
→ If the language is finite, FA can be designed.
→ If the language is finite, RE can also be written.

Σ {a, b}

a) Language with no string
{ } ; ϕ (R.E)

b) Language with length 0
{ε} ; ε, λ

c) Language with length 1
{a, b} ; {a+b}
{a+b} → either a or b.

d) Language with length 2 : (a+b) (a+b) / a(a+b) b(b+a)
{aa, ab, ba, bb} ; (aa+ab+ba+bb)

e) Length 3
R.E (a+b) (a+b) (a+b)
lang: { aaa, aab, abb, aba, baa, bab, bba
bbb }

f) At most 1 : Length of the string should be atmost 1. (0 or 1)

$\{\varepsilon, a, b\}$; $(\varepsilon + a + b)$

g) Atmost 2 : $(\varepsilon + a + b)(\varepsilon + a + b)$

String with length not more than 2 b's and 1 a.

$\{\varepsilon, a, b, ab, ba, abb, bab, bba \ldots\}$

R.E = $\{\varepsilon + a + b + ab + ba + abb + bab + bba\}$

★ Regular Expressions for Infinite Language (*)

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

a^* → Star means it can have values 0, 1, 2 ---- $(a^0 \geq 0)$ (a^*, a^+)

$a^+ = aa^*$ $\quad [a^+ = a^* - \varepsilon]$

$a^+ = a$ $\{\because a^0 = \varepsilon\}$ $\{a\varepsilon = a\}$

1. All strings having a single 'b'

$a^* b a^*$ → Regular Expression

2. All strings having at least one 'b'

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

Note: $(a+b)^*$ is one of most important expressions i.e. all the strings possible on a and b.

3. All strings having bbbb as substring

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

4. All strings ending with ab

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

5. Starting with ba

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

6. All strings starting and ending with a.

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

7. All strings containing 'a'.

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

8) All strings starting and ending with different symbols

$a(a+b)^*b$

or

$b(a+b)^*a$

Note If ques is, all strings with 2 b's

a^*bba^* → Wrong

Why?

It means only consecutive 2 b's.

e.g abba, aabbaa, ----

Correct method

$a^*ba^*ba^*$

ababa (not possible in a^*bba^*)

* Operations of RE

1. Union: If L and M are two regular languages then their union L∪M is also a RE.

Clearer

L = {(1+0)·(1+0)} = {00, 10, 11, 01}

M = {ε, 100}

then, L∪M = {ε, 00, 10, 11, 01, 100}

2. Concatenation: The concatenation of two regular languages L_1 and L_2, which are represented using $L_1.L_2$ is also regular and which represents the set of strings which are formed by taking any string in L1 concatenating it with any string in L2.

e.g $L_1 = \{0, 1\}$

$L_2 = \{00, 11\}$; then

$L_1.L_2 = \{000, 011, 100, 111\}$

3. Kleene closure: If L1 is a regular language, then the Kleene closure i.e. $L1^*$ of L1 is also regular and represents the set of those strings which are formed by taking a no. of strings from L1 and the same string can be repeated any number of times and concatenating those strings.

Example : ~~L1 = {0, 1}~~

L^* = zero or more occurences of language L.

* Properties of RE (Algebraic)

Kleene closure is a unary operator and union (+) and concatenation (.) are binary operators.

1. Closure: If R_1 and R_2 are two RE, then
   → $R1^*$ is a RE
   → R_1+R_2 is a RE
   → $R_1.R_2$ is a RE

Closure properties are those that are retained by the obtained when any operation is applied to a set.

Note:- Closure (closed) property means that if you apply any operation on any set, if the output obtained is a part of the same set or not

e.g. Set of Integer, Z

$1 \oplus 2 = 3$

$\frac{5}{2} = 2.5$ ✗

2. Closure Laws:

→ $\Rightarrow \boxed{(r^*)^* = r}$

Closing an expression that is already closed and does not change the language.

→ $\phi^* = \varepsilon$

i.e. It means a string formed by concatenating any no. of copies of an empty string is empty itself.

$$\boxed{r^+ = r \cdot r^*}$$

i.e. $r^* = \varepsilon + r + rr + rrr - - - -$

→ $r^* = r^* + \varepsilon$

3. Associative Property :- R_1, R_2 and R_3 are RE then,

∴ L.H.S = a+(b+c)
Regular set for RE = {a, b, c}
Hy R.H.S

i) $R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$

Note:- Associative property holds for Union operation

ii) $R_1 . (R_2 . R_3) = (R_1 . R_2) . R_3$ {L.H.S = a.(b.c) String accepted by RE ≠ abc Ukg R.H.S}

Note: Associative property holds for concatenation operator

Note: Associative property does not hold for Kleene Closure (*) as it is unary Operator.

4. Identity :-

i) if $r + x = r$
⇒ then $x = \phi$ { ϕ is the identity element for union }
$\therefore r \cup \phi = r$

ii) if $r . x = r$; for $x = \varepsilon$
$r . \varepsilon = r$ { ε is the identity element for concatenation }

5) Commutative Property : If R_1 and R_2 are RE,

i) $R1 + R2 = R2 + R1$ {$R_1 = a$, $R_2 = b$, RE a+b and b+a are equal}

ii) $R_1 . R_2 \neq R_2 . R_1$ {$R_1 = a$, $R_2 = b$, RE $a.b \neq b.a$}

6. Distributive Property: R_1, R_2, R_3 are RE, then

→ $(R_1 + R_2) \cdot R_3 = R_1 \cdot R_3 + R_2 \cdot R_3$ (Right distribution)

→ $R_1 \cdot (R_2 + R_3) = R_1 \cdot R_2 + R_1 \cdot R_3$ (Left distribution)

→ $(R_1 \cdot R_2) + R_3 \neq (R_1 + R_3)(R_2 + R_3)$

7. Annihilator: An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is annihilator.

e.g. In multiplication, $(0 * x = 0)$, Here, 0 is the annihilator.

In case of R.E

→ If $R + x = R$

$\Rightarrow \boxed{R \cup x = x}$

→ $R \cdot x = x$; (When $x = \phi$)

then, $\boxed{R \cdot \phi = \phi}$ [Also $\varepsilon \cdot R = R \cdot \varepsilon = R$]

$\therefore \phi$ is the annihilator for (.) operator.

8. Idempotent : For union,

$L + L = L$

An operator is idempotent if the result of applying it to two of the same values as arguments is that value

If we take union of two identical expressions we can replace them by one of the copy of expression

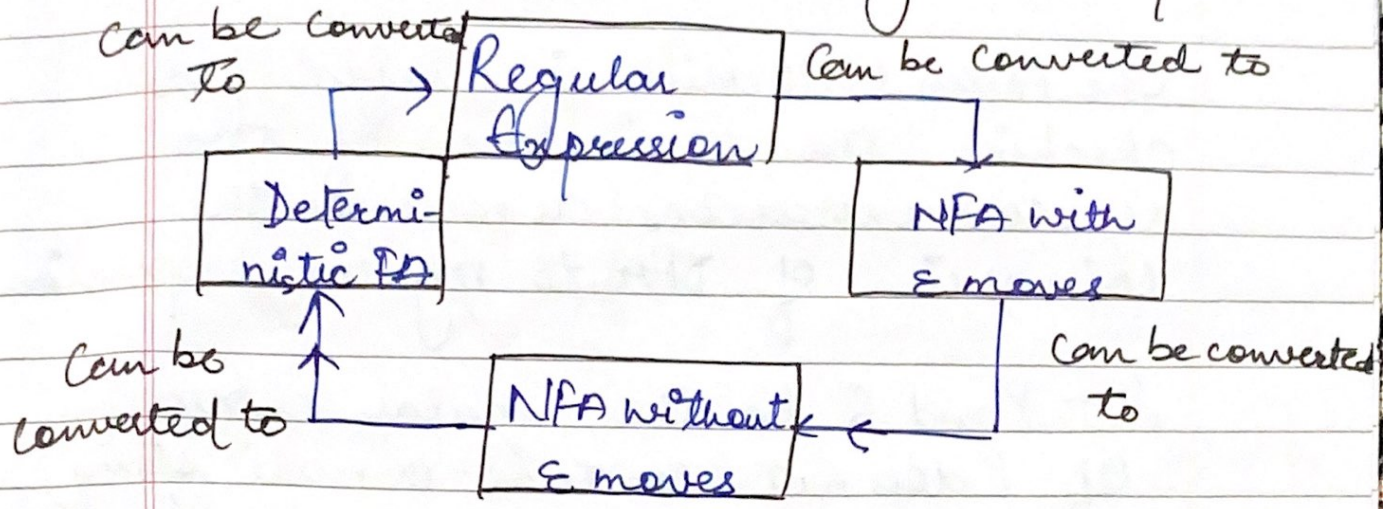
but $R \cdot R \neq R$

* Identities for RE

Let P, Q, R be RE

1. $\phi + R = R$
2. $\phi R = R\phi = \phi$ (or $\phi R + R\phi = \phi$)
3. $\varepsilon R = R\varepsilon = R$
4. $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$
5. $R + R = R$
6. $R^* R^* = R^*$
7. $RR^* = R^* R$
8. $(R^*)^* = R^*$
9. $\varepsilon + RR^* = \varepsilon + R^* R = R^*$
10. $(PQ)^* P = P(QP)^*$
11. $(P+Q)^* = (P^* \cdot Q^*)^* = (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$ and
    $R(P+Q) = RP + RQ$

Finite Automata and Regular Expression



The above fig. explains that it is easy to convert

→ RE into NFA with epsilon moves
→ NFA with epsilon moves to without ε moves.
→ NFA without ε moves to DFA.
→ DFA can be converted easily to RE.

Arden's Theorem

The Arden's Theorem is useful for checking the equivalence of two regular expressions as well as the conversion of DFA to regular expression.

Let P and Q be two regular expressions. If P does not contain a null string, then $R = Q + RP$, has a unique solution i.e. $\boxed{R = QP^*}$

Proof:-

$$R = Q + (Q + RP)P \quad (\because R = Q + RP)$$
$$= Q + QP + RPP$$

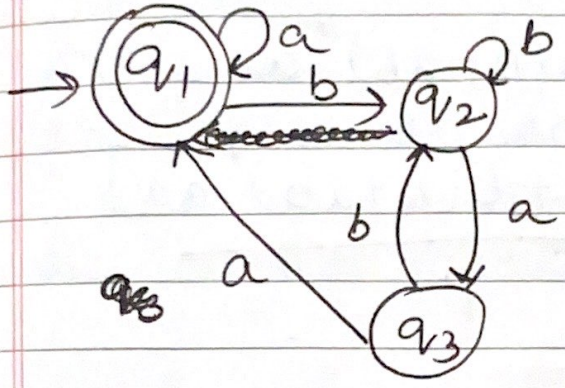
When we recursively put the value of R, we get the following eq:

$$R = Q + QP + QP^2 + QP^3 \ldots\ldots$$

$$R = Q(\varepsilon + P + P^2 + P^3 + \ldots)$$

$$\therefore \boxed{R = QP^*} \quad [\text{As } P^* \text{ represents } (\varepsilon + P + P^2 + P^3 + \ldots)]$$

Finite Automata to RE



(see how to reach q_1) $(q_1, a) \to q_1$
$(q_3, a) \to q_1$

$$q_1 = q_1 a + q_3 a + \varepsilon \quad \text{— (1)}$$
$$q_2 = q_1 b + q_2 b + q_3 b \quad \text{— (2)}$$
$$q_3 = q_2 a \quad \text{— (3)}$$

Find the value of final state i.e. q_1.

Substitute values

$$q_2 = q_1 b + q_2 b + q_2 ab$$
$$\underbrace{q_2}_{R} = \underbrace{q_1 b}_{Q} + \underbrace{q_2}_{R} \underbrace{(b + ab)}_{P}$$

$$\Rightarrow \boxed{q_2 = q_1 b (b + ab)^*} \quad \text{— (4)}$$

Now, $q_1 = q_1 a + q_2 a a + \varepsilon$

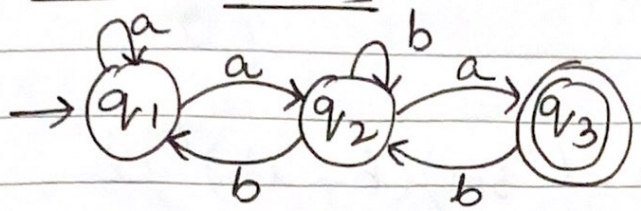
$q_1 = q_1 a + q_1 b (b + ab)^* aa + \varepsilon$

$\underset{R}{q_1} = \underset{Q}{\varepsilon} + \underset{\bar{R}}{q_1} \underbrace{[a + b(b + ab)^* aa]}_{P}$

$q_1 = \varepsilon [a + b(b + ab)^* aa]^*$

$\Rightarrow \boxed{q_1 = [a + b(a + ab)^* aa]^*}$

* NFA to RE



$q_3 = q_2 a$ — (1)

$q_2 = q_1 a + q_2 b + q_3 b$ — (2)

$q_1 = \varepsilon + q_1 a + q_2 b$ — (3)

$q_3 = q_2 a$ (Substituting values of q_2)
$= (q_1 a + q_2 b + q_3 b) a$
$= q_1 aa + q_2 ba + q_3 ba$ — (4)

$q_2 = q_1 a + q_2 b + q_3 b$ (Putting value of q_3 (1))
$= q_1 a + q_2 b + (q_2 a) b$
$= q_1 a + q_2 b + q_2 ab$

$q_2 = q_1 a + q_2 (b + ab)$
R Q R P

{ Using Arden's theorem
$R = Q + RP$, then
$R = QP^*$

$\therefore \boxed{q_2 = (q_1 a)(b + ab)^*}$ — (5)

$q_1 = \varepsilon + q_1 a + q_2 b$ (Putting value of q_2 from (5))

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

R Q R P

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

$\boxed{q_1 = \varepsilon((a+a(b+ab)^*)b)^*}$ $\{\varepsilon \cdot R = R\}$

$\boxed{q_1 = (a+a(b+ab)^* b)^*}$ — (6)

Final State (q_3)

$q_3 = q_2 a$ (Putting value of q_2 from (5))

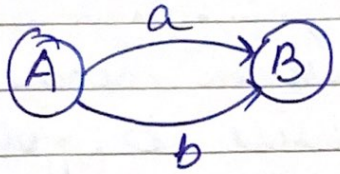
$= q_1 a (b+ab)^* a$ [value of q_1 from (6)]

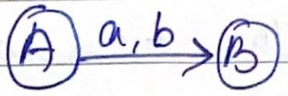
$\boxed{q_3 = (a+a(b+ab)^* b)^* a (b+ab)^* a}$

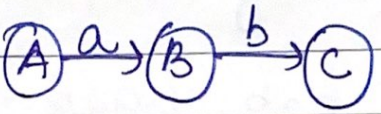
Required RE for given NFA.

RE to FA (Basic Examples)

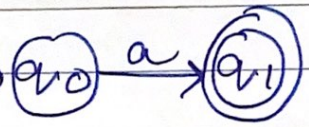
→ Given a RE, (a+b)

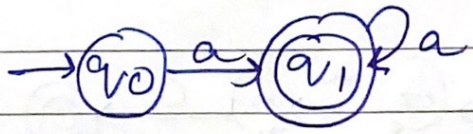
= 

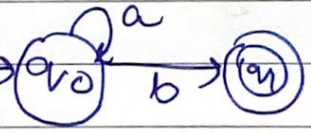
or 

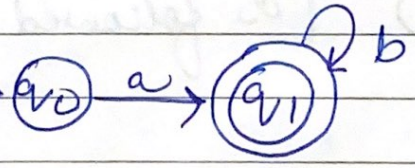
→ (a.b) = 

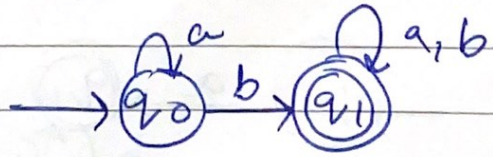
→ a^* = 

→ a → 

→ a^+ = 

→ a^*b → 

→ ab^* → 

→ $a^*b(a+b)^*$ = 

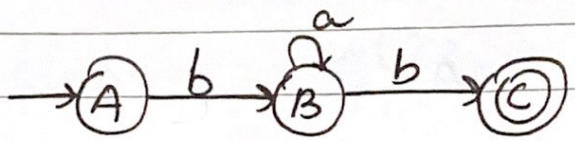
* Conversion of RE to FA

1) ba^*b

The regular expression means that the strings accepted will b, followed by zero or more no. of as, then again b.

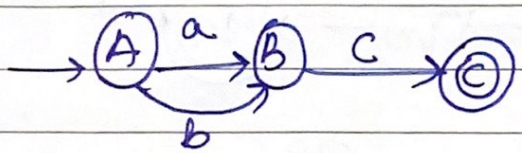
$L = \{bb, bab, baab ---\}$

FA =



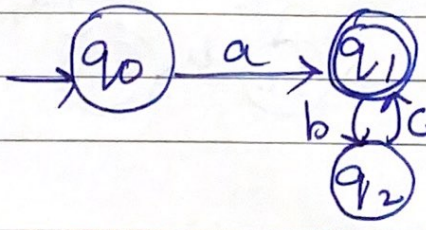
2) $(a+b)c$

$\{L = \{ac, bc\}\}$



3) $a(bc)^*$ [a followed by zero or more 'bc']

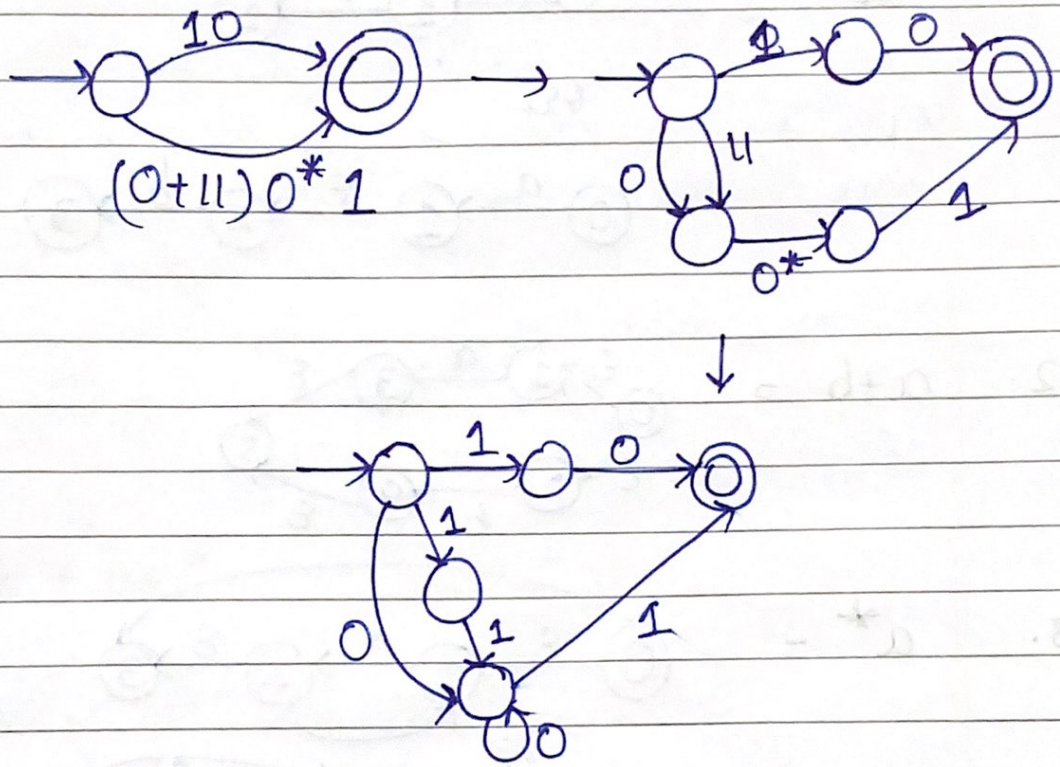
$L = \{a, abc, abcbc, --\}$



[q_1 will be the final state as the string are ending with input c, then accepted]

RE to FA - Example (NFA)

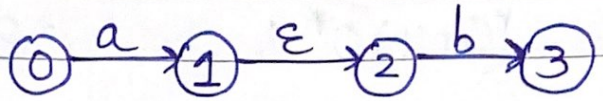
1 Given RE = $10 + (0+11)0^*1$



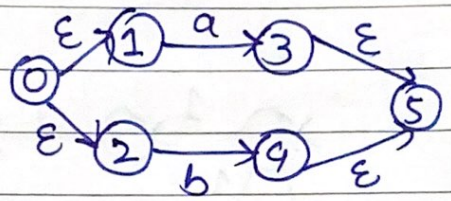
* Common RE used in ε-NFA

1. ab =

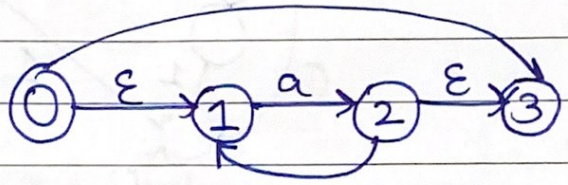
Or



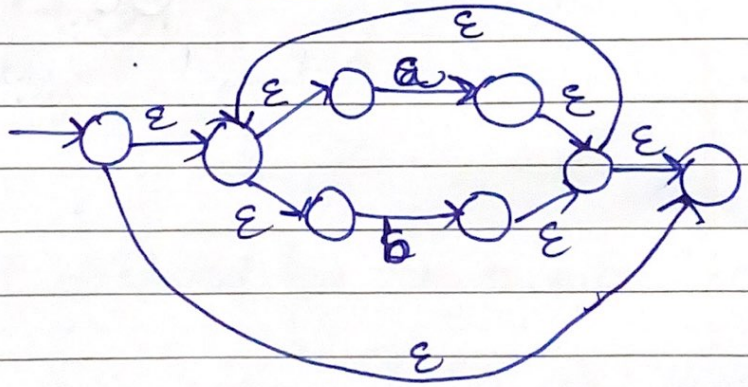
2. a+b =



3. a^* =



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



* Equivalence of two finite Automata

→ The two FA perform the same kind of functions.

i.e. the languages they accept will be the same.

→ The two FA are said to be equivalent if they accept same set of strings over an input set Σ.

Steps for Equivalence

1) For any pair of states $\{q_i, q_j\}$, the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where

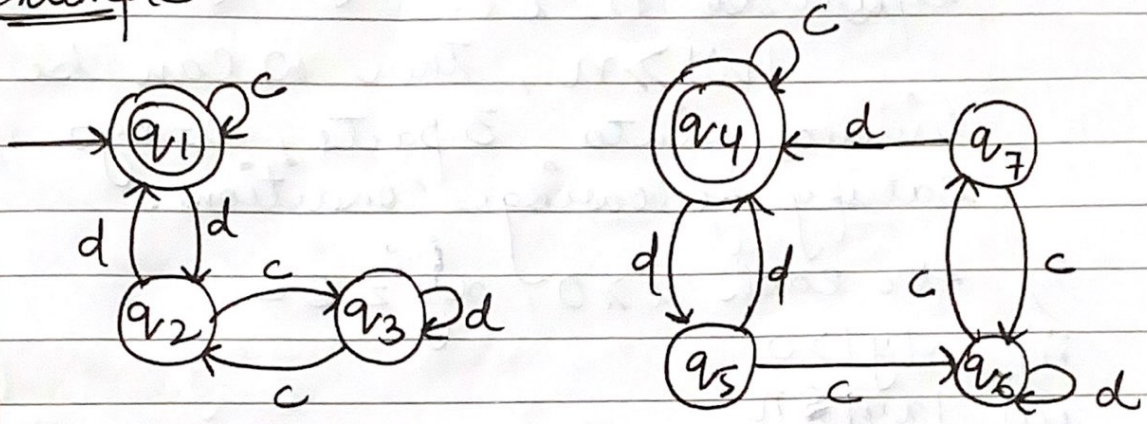
$$\delta\{q_i, a\} = q_a \text{ and}$$
$$\delta\{q_j, a\} = q_b$$

2) While combining the states with inputs, the pair results must be either both final states or intermediate states. (if not, then the two automata are not equivalent)

Note: If at any transition, you find the pair of state in different set, then it is concluded that the automata are not equivalent

3) If initial state in one automaton is the FS as well, then the initial state must be the final state in the second automaton as well for them to be equivalent.

Example



A

B

States	c	d
(q_1, q_4)	(q_1, q_4)	$(q_2\ q_5)$
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

FS = {q1, q4}

IS = {q2, q3, q5, q6, q7}

* Pumping Lemma for RE

If L is an infinite language then there exists some positive integer 'n' (pumping length) such that any string $w \in L$, has length greater than equal to 'n' i.e.

$|w| \geq n$, then w can be divided into 3 parts, $w = xyz$ satisfy following conditions:

i) for each $i \geq 0$, $xy^iz \in L$
ii) $|y| > 0$
iii) $|xy| \leq n$

→ If the language is finite, then exist a FA, it is a regular language (RE)

→ Infinite language may or maynot be regular

→ Pumping lemma is a negative test, used as a proof for irregularity of a language.

→ If PL is applied to a language L, and the language satisfies all the conditions

of the Pumping Lemma, then regularity of the language is undecidable. (i.e. It still may or may not be regular)

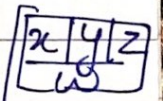
→ On the other hand, if the language fail to satisfy the Pumping Lemma test, then the language is said to be NOT REGULAR.

→ Hence the Pumping Lemma is used to prove that a language is NOT regular.

Explanation

i) for each $i \geq 0$, $xy^iz \in L$

This means, first we have to divide the string, say w, into 3 parts, x, y, z, we need to increase the values of y, such that $i \geq 0$, to check if it belongs to the language or not.



ii) Note Length of y should be greater than 0.

iii) and length of xy i.e. $|xy|$ should be collectively less than or equal to n.

classmate
Date
Page

i) for each $i \geq 0$, $xy^iz \in L$
ii) $|y| > 0$
iii) $|xy| \leq n$

Example

$a^n b^{2n}$; $n \geq 0$

Taking any string that $\in$ to the RE.

$aabbbb \in L$ $\{n=2\}$

$x = aa$, $y = bb$, $z = bb$

$(bb)^1$ Pumping $(bb)^{i+1}$

$aa\ bbbb\ bb \notin L$

x y z

because, in the given language, no. of b's should be twice times no. of a ($a^n b^{2n}$)

Since the string does not belong to the language, it means it has failed the Pumping lemma test, as it is not satisfying the condition; $i \geq 0$, $xy^iz \in L$.

Also, If you change the values for x, y, z, then

Again, for $a^n b^{2n}$

$aabbbb$

x y z

$y = (ab)^{i+1}$ (Pumping)
$y = (ab)^{i+1}$

$\underset{x}{a}\ \underset{y}{aba}\underset{z}{b}\ \underset{z}{bbb} \notin L$ [The given language was 'a' followed by b]

Again, it doesnot belong the language, as it is failing the condition.
So, the language is NOT REGULAR.

classmate
Date
Page

* Myhill-Nerode Theorem

→ The Myhill-Nerode Theorem was proven by John Myhill and Anil Nerode in 1958.

→ Another method of Minimization of DFA.

→ Also called as table-filling method.

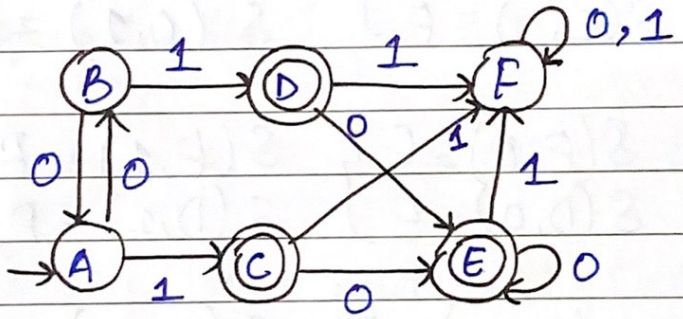
Steps

1. Draw a table for all pairs of states. We make the pairs of different states.
2. Mark all pairs where $P \in F$ and $Q \in f$, means, while making the pairs where one of the state is a final state and other is not
3. If there any unmarked pairs $[P, Q]$ such that $[\delta(P, x), \delta(Q, x)]$ is marked, then mark $[P, Q]$ where 'x' is an input symbol.

Repeat this process, until no more marking can be done.

4. Combine all the unmarked pairs and make them a single state in the minimized DFA.

Example



	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓	✓	✓		
F			✓	✓	✓	

Check transitions for unmarked pairs:

(B, A) – $\delta(B,0) = A$, $\delta(B,1) = D$
$\delta(A,0) = B$, $\delta(A,1) = C$

(D, C) – $\delta(D,0) = E$, $\delta(D,1) = F$
$\delta(C,0) = E$, $\delta(C,1) = F$

(E, C) – $\delta(E,0) = E$, $\delta(E,1) = F$
$\delta(C,0) = E$, $\delta(C,1) = F$

(E, D) – $\delta(E,0) = E$, $\delta(E,1) = F$
$\delta(D,0) = E$, $\delta(D,1) = F$

(F, A) = $\delta(F,0) = F$, $\delta(F,1) = F$
$\delta(A,0) = B$, $\delta(A,1) = C$
(Since (E,C) is marked, Mark (F,A) also)

(F, B) = $\delta(F,0) = F$, $\delta(B,0) = A$
(Since (F,A) is marked, mark (F,B) as well).

Resultant DFA