Regular grammar to regular expression  
Regular expression to regular grammar   
Regular language to regular grammar  
Regular grammar to regular language

Regular expression to finite automata  
Adren’s theorem (finite automata to regular expression)

**Regular Grammar to Regular expression**

1) S -> aSB L = { a^n b^n | n>=2 }  
 S -> aB  
 B -> b

2) S-> aS | epsilon L ={ a^n | n>0 }

**Regul­­ar Expression to regular grammar**

1) (a+b)\*a   
S 🡪 WX  
W 🡪 aW | bW | epsilon  
X 🡪 a

2) (a+1)\*  
S 🡪 aS | 1s | epsilon

3) a\*   
S 🡪 aS | epsilon

4) a+0  
S 🡪 a|0

5) (ab)\*   
S 🡪 abS | epsilon

6) (a/b)\*  
S 🡪 aS|bS|epsilon

7) (a\*) + (1)+ (b\*)

S 🡪 aS|1|bS|epsilon

8) (a\*) + (1) + (b+)   
S 🡪 aS|1|b|epsilon

9) (a/b) (a/b) (a/b)\*  
S 🡪 XXY  
X 🡪 a|b  
Y 🡪 aY | bY | epsilon

10) (a/b/epsilon) (a/b/epsilon)  
S 🡪 XX  
X 🡪 a|b|epsilon

11) a (a/b)\* b   
S 🡪 aXb  
X 🡪 aX | bX | epsilon

12) L = {a^n b^n} | n>=0  
S 🡪 aSb | epsilon

13) L = {a^n b^m} | n,m>=0  
S 🡪 aAbB  
A 🡪 aA|epsilon  
B 🡪 bB | epsilon

13) L = { (ab)^n } | n>=0  
S 🡪 abS

14) L = { a^n b^n c ^m | n,m>=0 }  
S 🡪 aXbY | epsilon  
X 🡪 aXb | epsilon  
Y 🡪 cY | epsilon

15) L = { a^n b^n c ^m | n,m>=1 }  
S 🡪 aXbcY  
X 🡪 aXb | epsilon  
Y 🡪 cY | epsilon

16) L = { a^n c ^m b^n | n,m>=0 }  
S 🡪 aSb | aXb |X | epsilon  
X 🡪 cX | epsilon

16) L = { a^n c ^m b^n | n,m>=1 }  
S 🡪 aSb | aXb  
X 🡪 cX | c

17) L = { a^n c ^m b^n | n,m>=1 }  
S 🡪 aSb | aXb   
X 🡪 cX

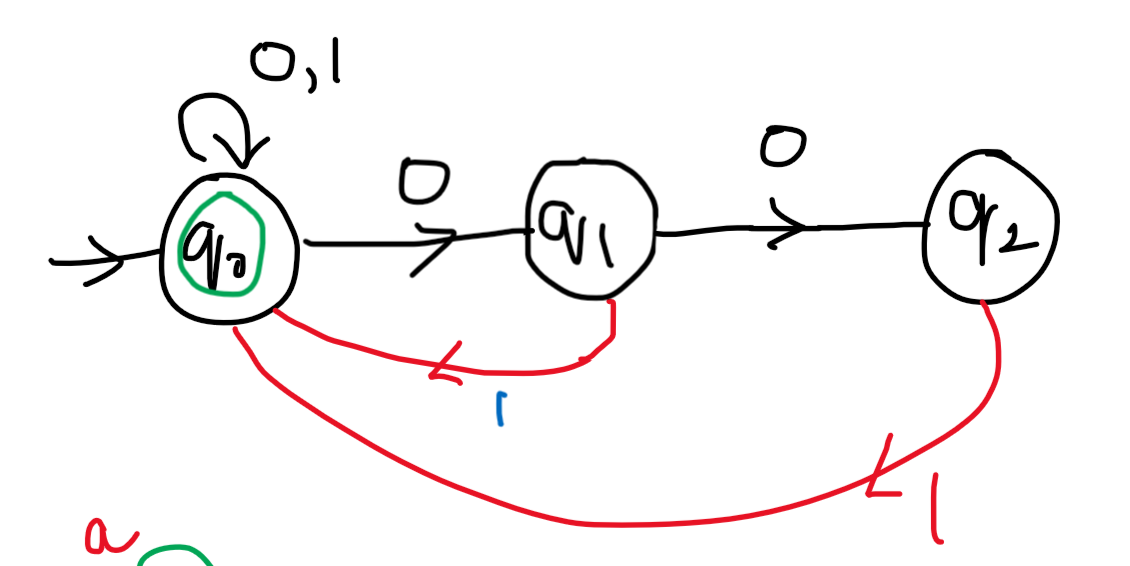
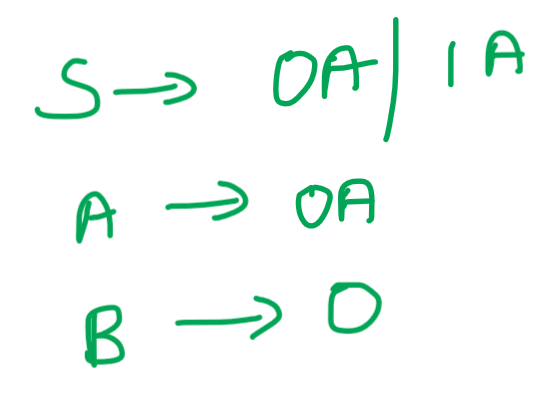
E = {a,b}  
L = { all non empty string starts and ends with the same symbol }

S 🡪 aXa | bXb | a | b  
X 🡪 aX | bX

E = {a,b}  
L = { Palindrome }   
S 🡪 aXa | bXb | a | b | epsilon  
X 🡪 aX | bX | epsilon

**Regular language to Regular grammar**

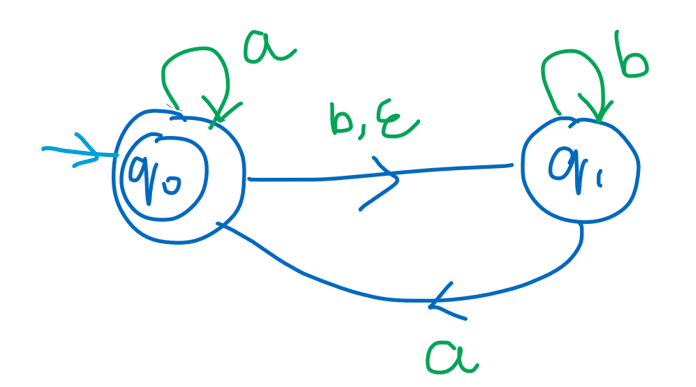
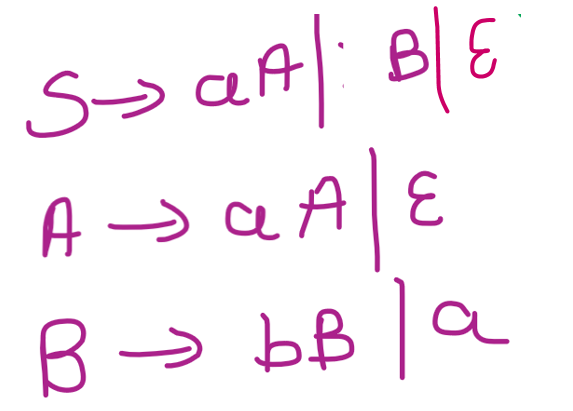
**1)L={w | w ends with 00}**



**2)L={0\*,1\*,0\*} with 3 states**

**3)L={a^n union b^na | n>=0}**



**\* L={ (a^n | n>=1) union (b^ma^k | m,k>=0) }**

**\* L(G) = { ancan / n ≥ 0 }**

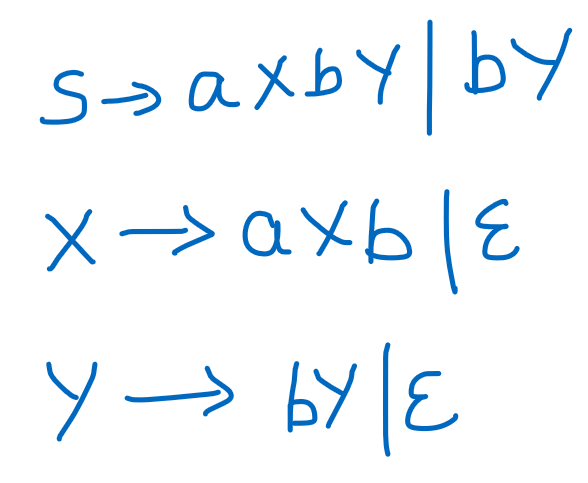
**\* L(G) = { anb2n / n ≥ 0 }**

**\* L(G) = { an+2bn / n ≥ 1 }**

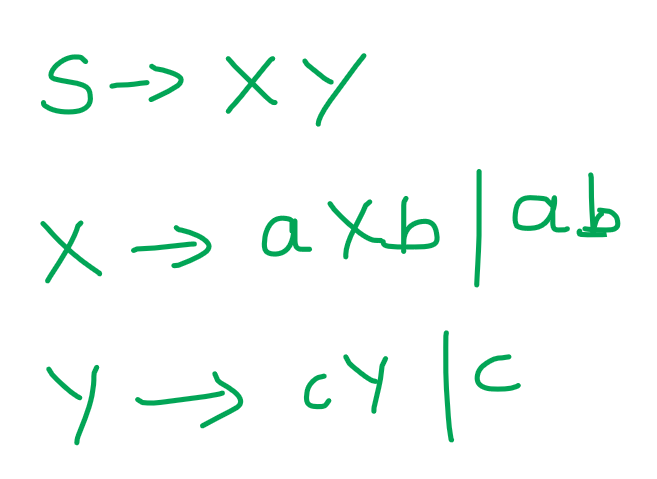
**\* L(G) = { anbn-3 / n ≥ 3 }**

**\* L(G) = { anbm / n ≥ 0, m ˃ n }**

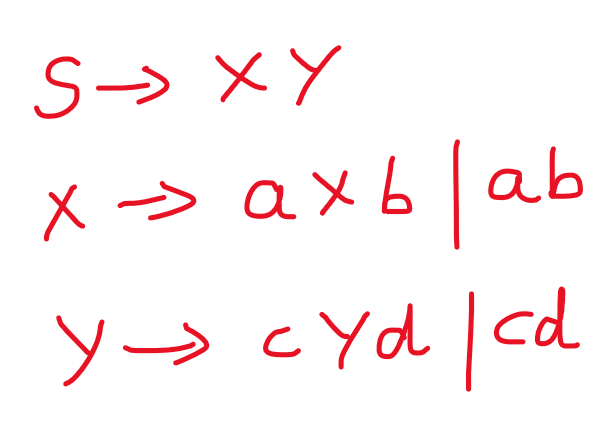
**The grammar can gone any way, but wrong string should not pass through that grammar.**



**\* L(G) = { anbn cm  / n, m ≥ 1 }**



**\* L(G) = { anbn cm dm  / n, m ≥ 1 }**



**\* L(G) = { anbm cm dn  / n, m ≥ 1 }**

**\* L(G) = { anbm / n, m ≥ 1 m ≠ n}**

**\* L(G) = { w {a, b}\* / na(w) = nb(w) + 1 }**

1)L = { a^n b^m | n,m>=1 }

2) **L = { a^n b^n c^m | n,m>=1 }** S 🡪 XY  
 X 🡪 aXb | ab  
 Y 🡪 cY | c

3) **L = { a^n c^m b^n | n,m>=1 }** S 🡪 aXb  
 X 🡪 aXb | CX | c

4) **L = { a^n b^m a^2n | n,m>=0 }** S 🡪 aXaa |epsilon  
 X 🡪 aXaa |bX | epsilon

5) **E = {a,b}**  
**All non empty strings start and ends with the same symbol (or)   
Palindrome**   
S 🡪 aAa | bAb | a |b | epsilon  
A 🡪 aA | bA | epsilon

6) **L = { w (0,1)\* | w^R and |w| is even }**  
S 🡪 epsilon | 1S1 | 0S0

7) **L = { w (0,1)\* | w contains at-least 3 ones }**S 🡪 A1|A1|A1  
A 🡪 epsilon | 0A | 1A



8) **L = { a b c | i,j,k>=0 and i+j = k }**  
S 🡪 aSc | X  
X 🡪 bXc | epsilon



9) **L = { a b c | i,j,k>=0 and i=j }**S 🡪 aXbcC  
X 🡪 aXb | epsilon  
C 🡪 cC | epsilon



10)   
S 🡪 aXc  
X 🡪 bX | aXc | epsilon

11) L = { a b c | i<j }

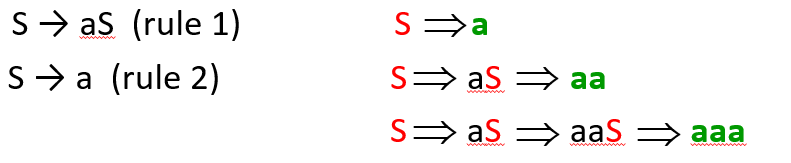


12) L = { a b c | i<k }

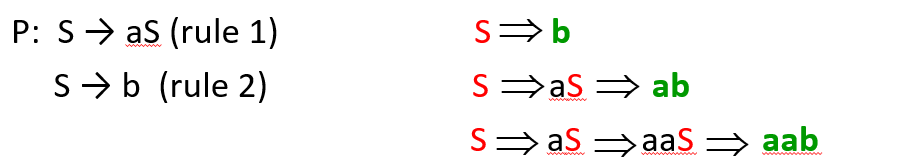


**Regular grammar to Regular language**

**1)**

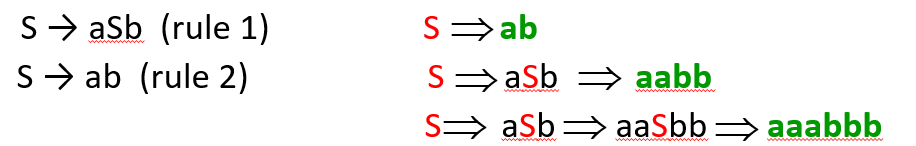
 **L(G) = { an / n ≥ 1 }**

**2)**

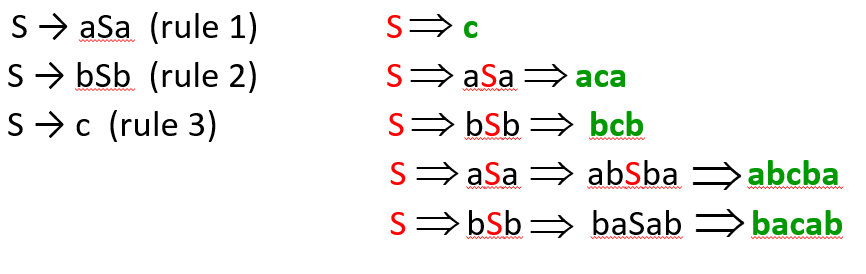
 **L(G) = { anb/ n ≥ 0 }**

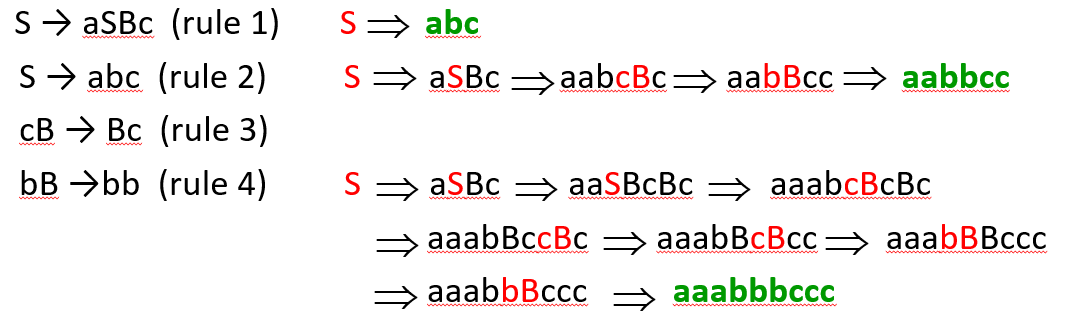


**3)**

 **L(G) = { anbn / n ≥ 1 }**

**4)**

  
**L(G) = { wcwR/ w {a, b}\* }**

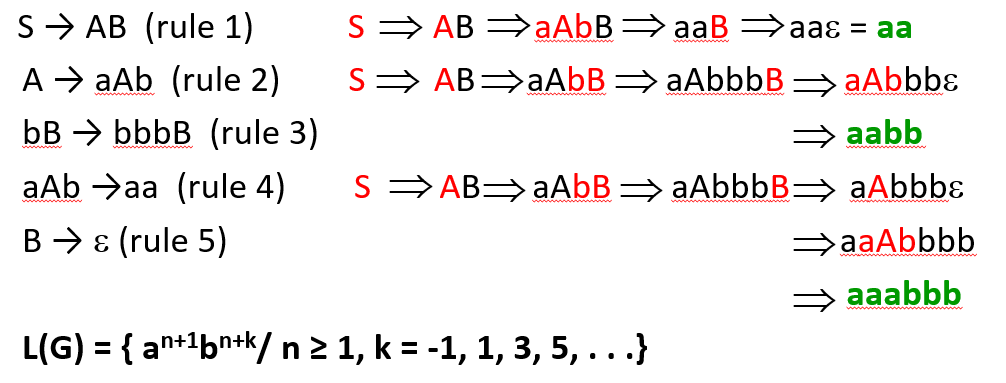
**5)**   
**L(G) = { anbnCn/ n ≥ 1 }**

**6)**



**L(G) = { an+1bn+k/ n ≥ 1, k = -1, 1, 3, 5, . . .}**

**7)**



**L(G) = { aa or a^nb^n | n>=2 }**

**8)**

**S 🡪 aSB  
S 🡪 aB  
B 🡪 b  
L(G) = { a^nb^n | n>=1 }**

**9)   
S 🡪 { aS | a }  
L(G) = { a^n | n>=1 }**

**10)   
S 🡪 aS | epsilon  
L(G) = { a^n | n>=0 }**

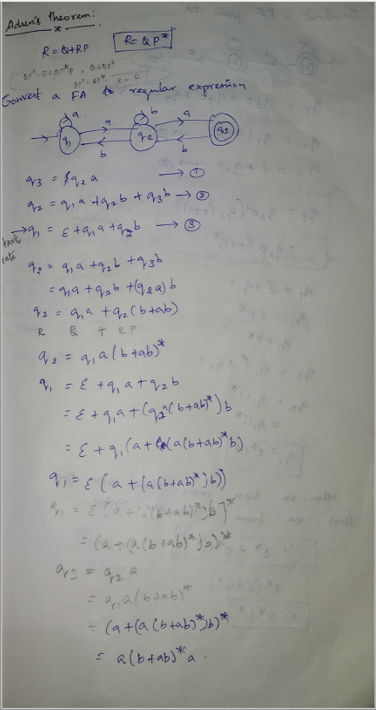
**11)   
A 🡪 aAb | ab**

**L(G) = { a^n b^n | n>=1 }**

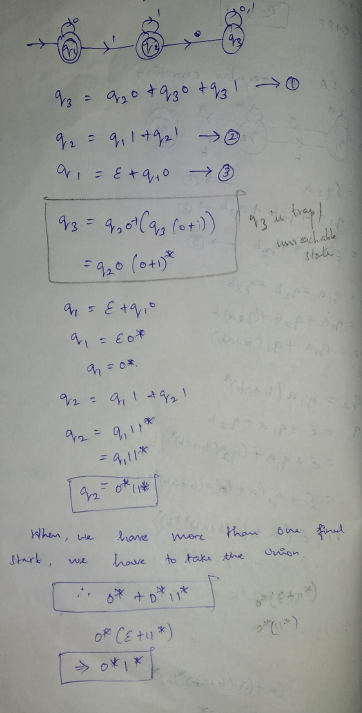
**12)   
A 🡪 aAb | epsilon**

**L(G) = { a^n b^n | n>=0 }**

1)

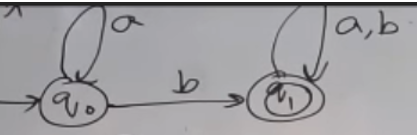


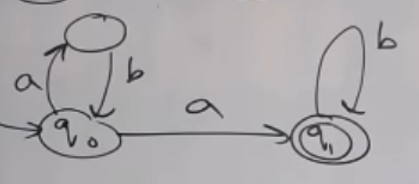
2)

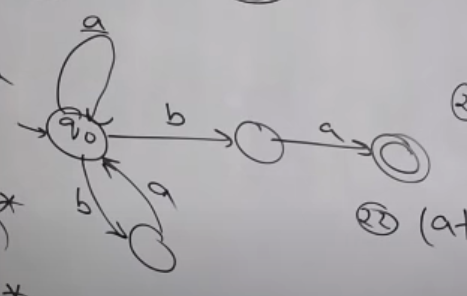


**Regular expression to finite automata**

**1) 0\*10\* L={ w|w contains a single 1 }   
2) (0|1)\* 1 (0|1)\* L = { w|w contains at-least a single 1 }  
3) (01) U (10)   
4) (0|1 0|1 0|1)\***  
**5) 0(0|1)\*0 + 1(0|1)\*1 + 0 + 1  
6) Null symbol  
7) Epsilon symbol  
8) a+b  
9) a.b  
10) a\*  
11) a+  
12) (a + b)\*  
13) a\*b\*  
14) (ab)\*  
15) a\*b  
16) ab\*  
17) a\*bc\***

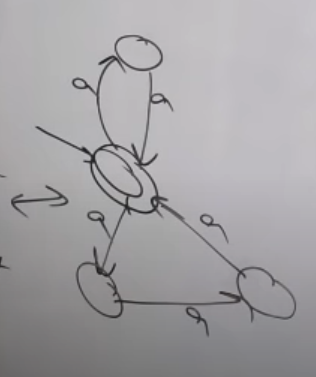
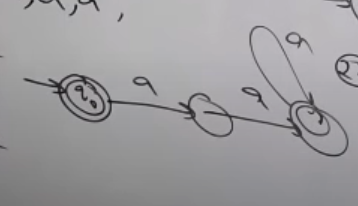
**18) a\*b(a+b)\***

**19) (ab)\*ab\***

**20) (a+ba)\*ba**

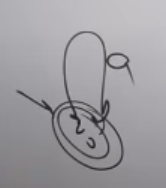
**21) (aa+aaa)\***



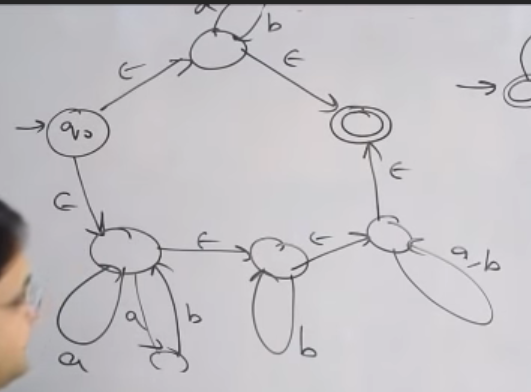
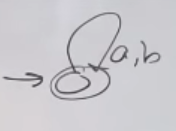
(or



**22) (a+aaaa)\***

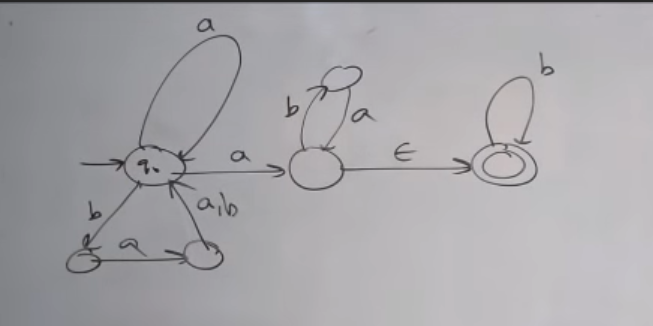


**21) (ab)\* + (a+ab)\*b\*(a+b)\***

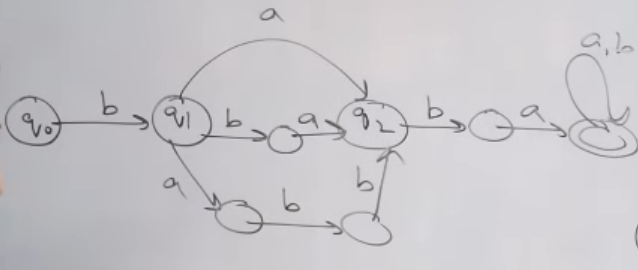
 

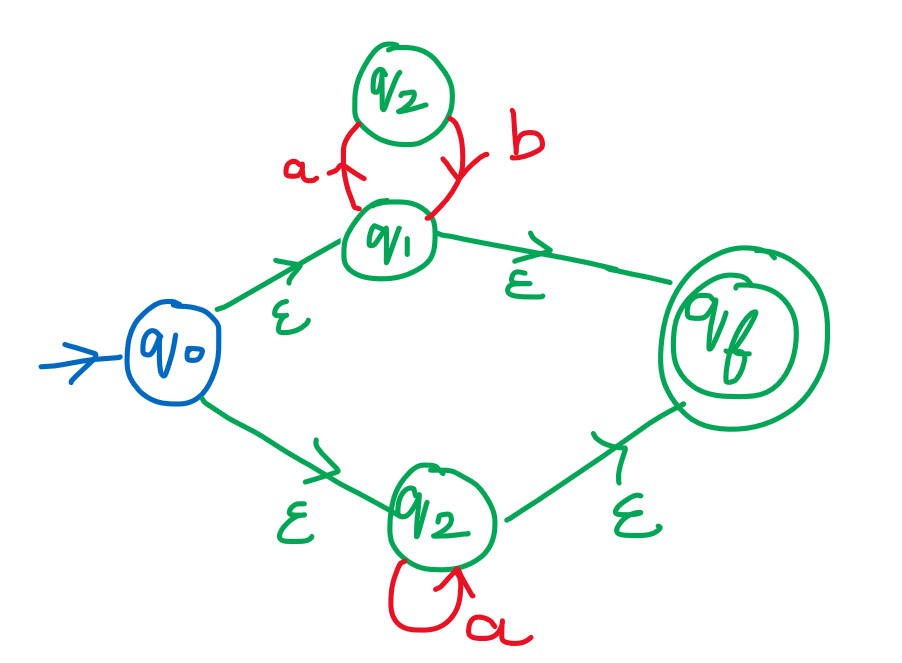


**22) [a+ba(a+b)]\* a(ba)\* b\***

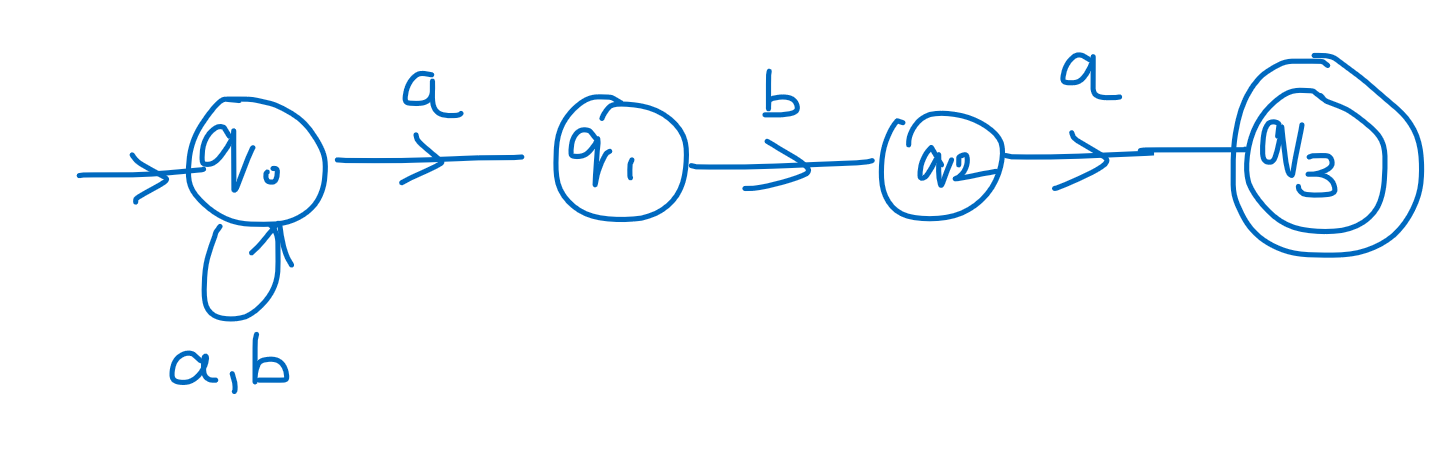


**23) b(a+ba+abb) (ba(a+b)\*)**

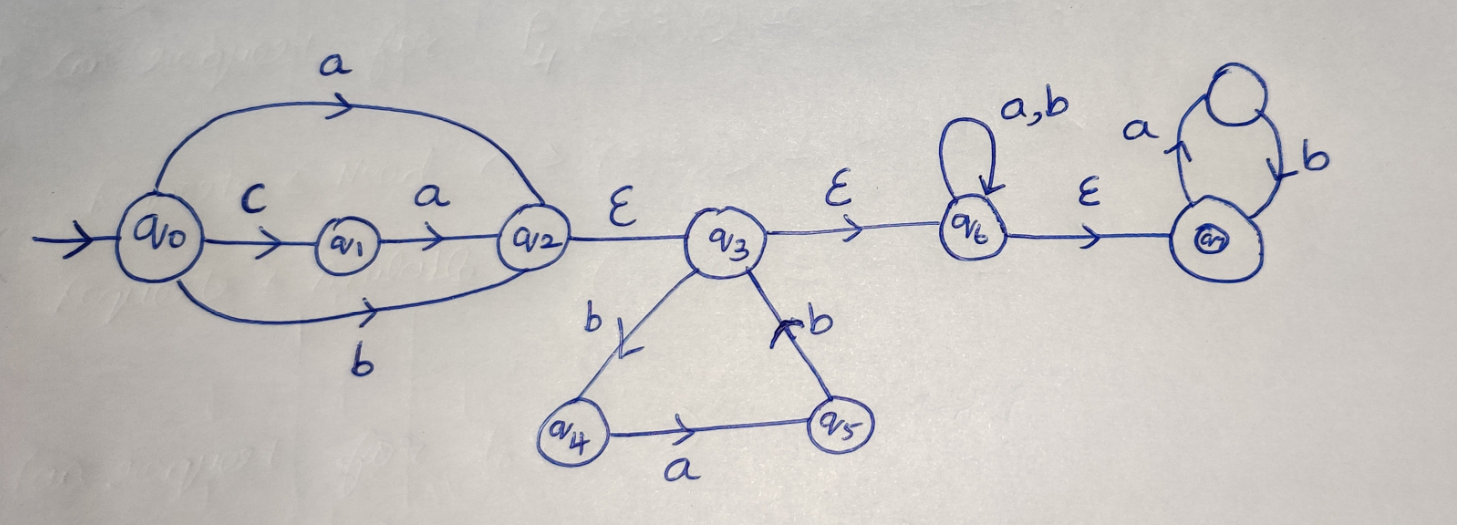


**24) (ab U a)\***

**25) (a U b)\* aba**



**26) (a+b+ca) [ (bab)\* + (a+b)\* ]\* (ab)\***



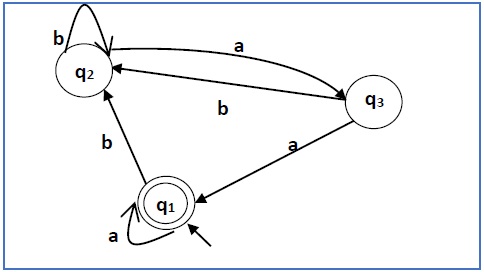
If the input alphabet is then the regular expression   
 🡪 describes the language consisting of all strings of length 1 over this alphabets

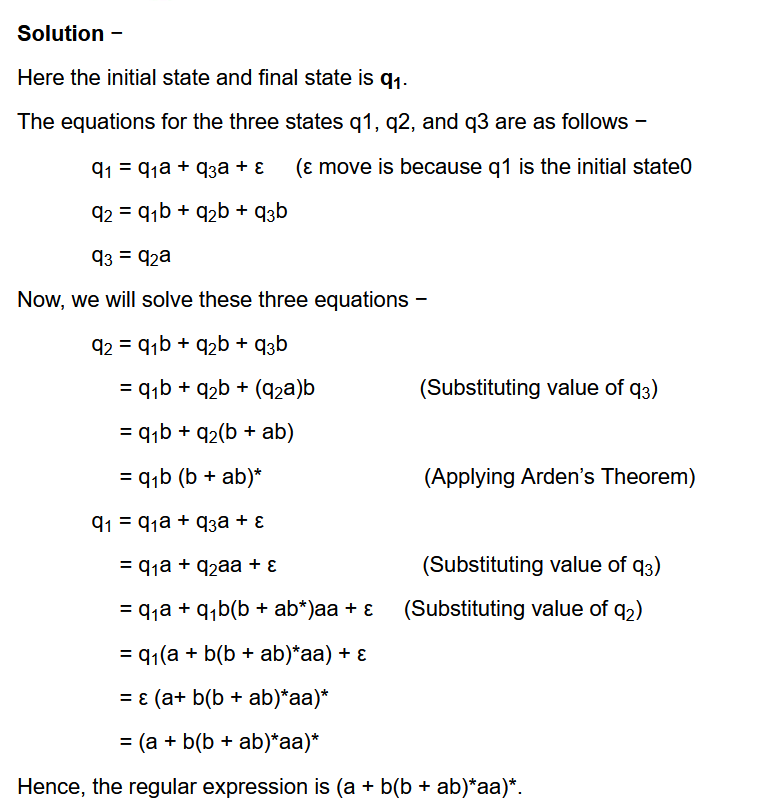
\* 🡪 describes the language that contains all strings

\* 1 🡪 describes the language that contains all the strings ending with 1

**Adren’s theorem(finite automata to regular expression)**

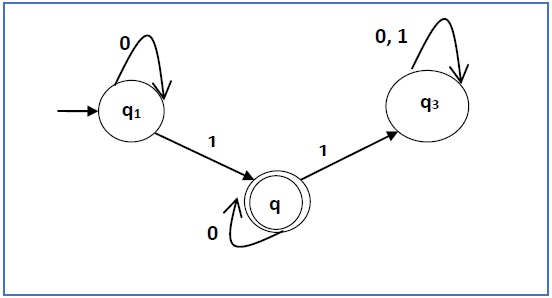
1)

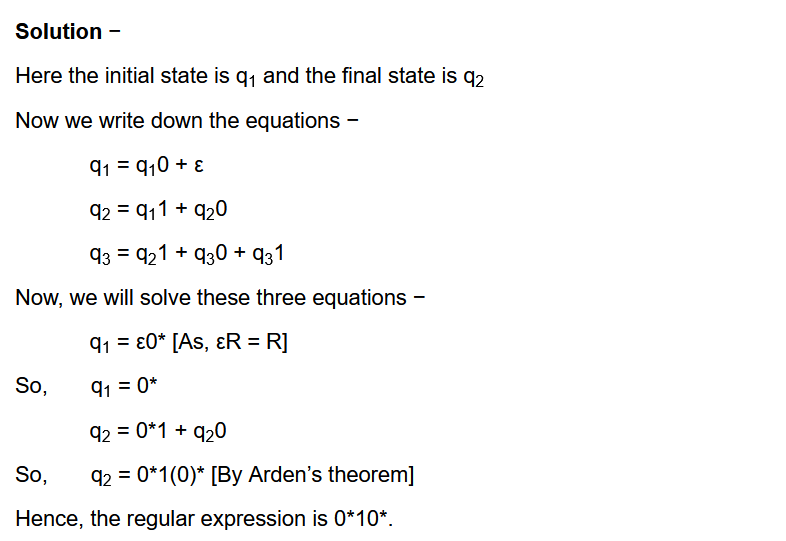






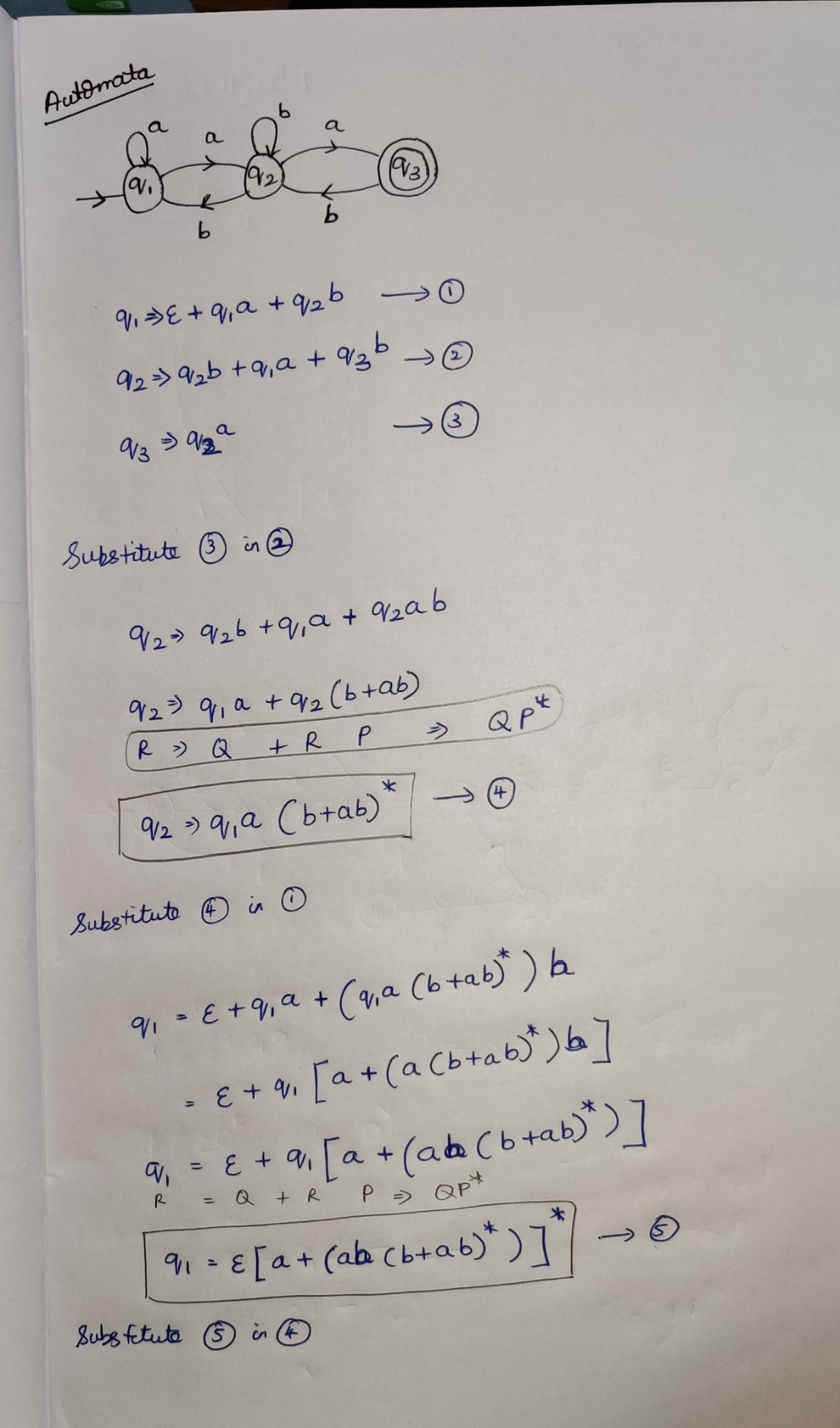
2)

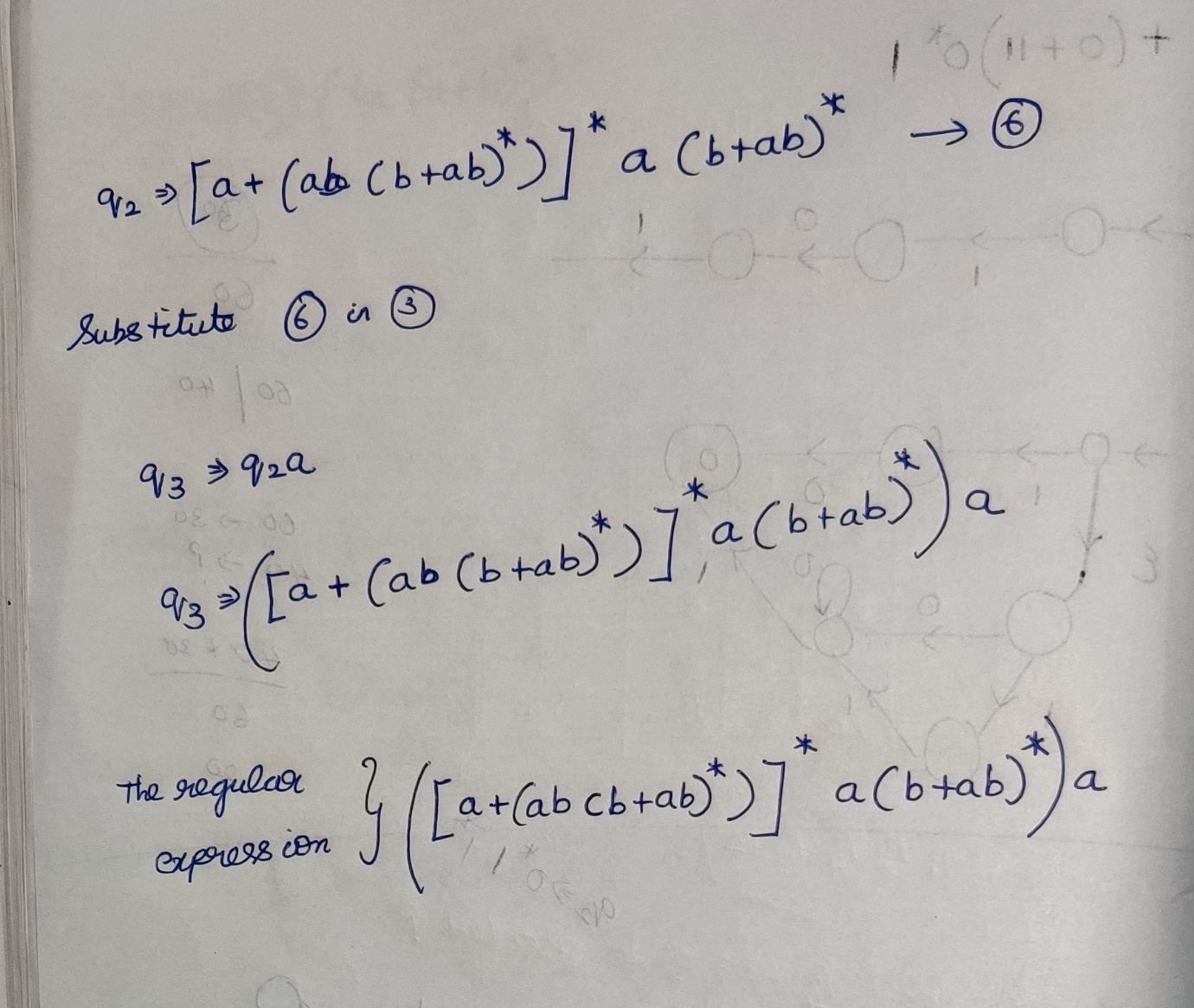






3) **Question**





**4) Question**



