UNIT-III

Day-1

Regular Expressions:

A **Regular Expression** can be recursively defined as follows −

- **ε** is a Regular Expression indicates the language containing an empty string.

  (L (ε) = {ε})

- **φ** is a Regular Expression denoting an empty language. **(L (φ) = { })**
- **x** is a Regular Expression where **L = {x}**
- If **X** is a Regular Expression denoting the language **L(X)** and **Y** is a Regular Expression denoting the language **L(Y)**, then
  - **X + Y** is a Regular Expression corresponding to the language

    L(X) ∪ **L(Y)** where **L(X+Y) = L(X)** ∪ **L(Y)**.
  - **X . Y** is a Regular Expression corresponding to the language

    L(X) . L(Y) where **L(X.Y) = L(X) . L(Y)**
  - **R*** is a Regular Expression corresponding to the language **L(R*)**where **L(R*) = (L(R))***
- If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

Some RE Examples

Regular Expressions	**Regular Set**
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, … }
(0*10*)	L = {1, 01, 10, 010, 0010, …}

$(0 + \varepsilon)(1 + \varepsilon)$	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, …………..}
(11)*	Set consisting of even number of 1's including empty string, So L= {ε, 11, 1111, 111111, ……….}
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's , so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………..}
(aa + ab + ba + bb)*	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L = {aa, ab, ba, bb, aaab, aaba, …………..}

Regular Sets:

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

Properties of Regular Sets

Property 1. *The union of two regular set is regular.*

Proof −

Let us take two regular expressions

RE_1 = a(aa)* and RE_2 = (aa)*

So, L_1 = {a, aaa, aaaaa,.....} (Strings of odd length excluding Null)

and L_2 ={ ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

$L_1 \cup L_2$ = { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,.......}

(Strings of all possible lengths including Null)

RE ($L_1 \cup L_2$) = a* (which is a regular expression itself)

Hence, proved.

Property 2. *The intersection of two regular set is regular.*

Proof −

Let us take two regular expressions

$RE_1 = a(a^*)$ and $RE_2 = (aa)^*$

So, L_1 = { a,aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

$L_1 \cap L_2$ = { aa, aaaa, aaaaaa,.......} (Strings of even length excluding Null)

RE ($L_1 \cap L_2$) = $aa(aa)^*$ which is a regular expression itself.

Hence, proved.

Property 3. *The complement of a regular set is regular.*

Proof −

Let us take a regular expression −

RE = $(aa)^*$

So, L = {ε, aa, aaaa, aaaaaa, ...} (Strings of even length including Null)

Complement of **L** is all the strings that is not in **L**.

So, L′ = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

RE (L′) = $a(aa)^*$ which is a regular expression itself.

Hence, proved.

Property 4. *The difference of two regular set is regular.*

Proof −

Let us take two regular expressions −

$RE_1 = a (a^*)$ and $RE_2 = (aa)^*$

So, L_1 = {a, aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

$L_1 - L_2$ = {a, aaa, aaaaa, aaaaaaa, ...}

(Strings of all odd lengths excluding Null)

RE ($L_1 - L_2$) = a (aa)* which is a regular expression.

Hence, proved.

Property 5. *The reversal of a regular set is regular.*

Proof –

We have to prove $\mathbf{L^R}$ is also regular if **L** is a regular set.

Let, L = {01, 10, 11, 10}

RE (L) = 01 + 10 + 11 + 10

L^R = {10, 01, 11, 01}

RE (L^R) = 01 + 10 + 11 + 10 which is regular

Hence, proved.

Property 6. *The closure of a regular set is regular.*

Proof –

If L = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

i.e., RE (L) = a (aa)*

L* = {a, aa, aaa, aaaa , aaaaa,……………} (Strings of all lengths excluding Null)

RE (L*) = a (a)*

Hence, proved.

Property 7. *The concatenation of two regular sets is regular.*

Proof –

Let RE_1 = (0+1)*0 and RE_2 = 01(0+1)*

Here, L_1 = {0, 00, 10, 000, 010, ...} (Set of strings ending in 0)

and L_2 = {01, 010,011,.....} (Set of strings beginning with 01)

Then, L_1 L_2 = {001,0010,0011,0001,00010,00011,1001,10010,.............}

Set of strings containing 001 as a substring which can be represented by an RE – (0 + 1)*001(0 + 1)*

Hence, proved.

Identity Rules:

Identities Related to Regular Expressions

Given R, P, L, Q as regular expressions, the following identities hold −

- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $RR^* = R^*R$
- $R^*R^* = R^*$
- $(R^*)^* = R^*$
- $RR^* = R^*R$
- $(PQ)^*P = P(QP)^*$
- $(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$
- $R + \emptyset = \emptyset + R = R$ (The identity for union)
- $R\,\varepsilon = \varepsilon\, R = R$ (The identity for concatenation)
- $\emptyset\, L = L\, \emptyset = \emptyset$ (The annihilator for concatenation)
- $R + R = R$ (Idempotent law)
- $L(M + N) = LM + LN$ (Left distributive law)
- $(M + N)L = LM + LN$ (Right distributive law)
- $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

Day-2

Equivalence of two Regular Expressions:

Manipulations of Regular Expressions:

Day-3

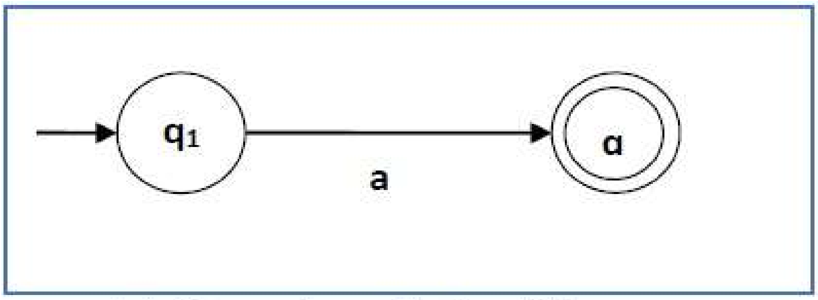
Finite Automata, and Regular Expressions:

Construction of an FA from an RE

We can use Thompson's Construction to find out a Finite Automaton from a Regular Expression. We will reduce the regular expression into smallest regular expressions and converting these to NFA and finally to DFA.

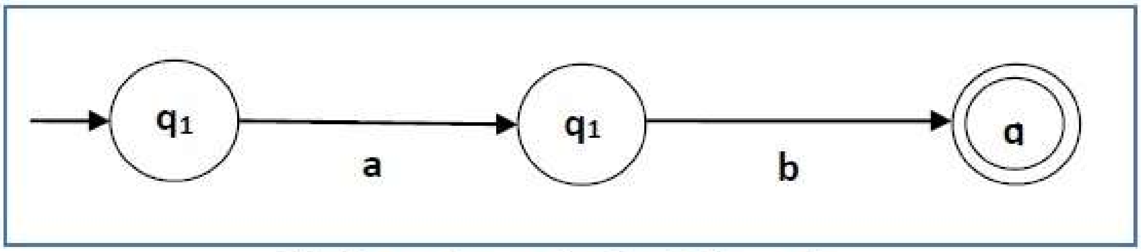
Some basic RA expressions are the following –

Case 1 – For a regular expression 'a', we can construct the following FA –



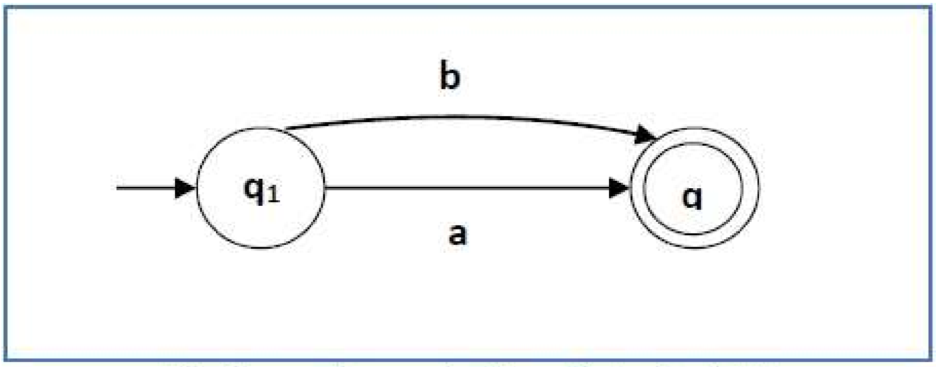
Finite automata for RE = a

Case 2 – For a regular expression 'ab', we can construct the following FA –



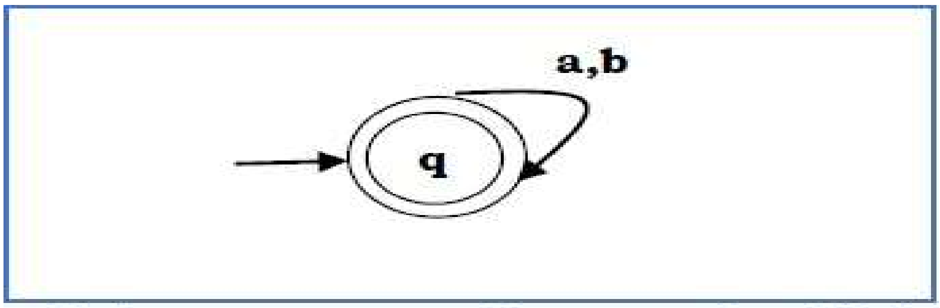
Finite automata for RE = ab

Case 3 – For a regular expression (a+b), we can construct the following FA –



Finite automata for RE= (a+b)

Case 4 – For a regular expression (a+b)*, we can construct the following FA –



Finite automata for RE= (a+b)*

Method

Step 1 Construct an NFA with Null moves from the given regular expression.

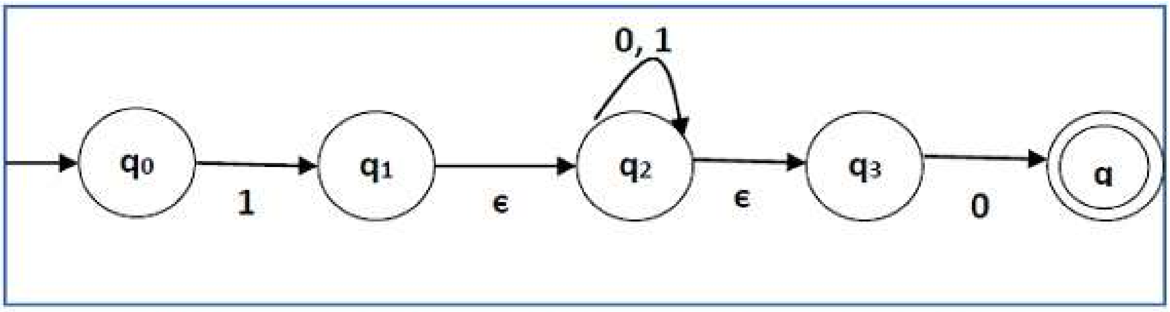
Step 2 Remove Null transition from the NFA and convert it into its equivalent DFA.

Problem

Convert the following RA into its equivalent DFA − 1 (0 + 1)* 0

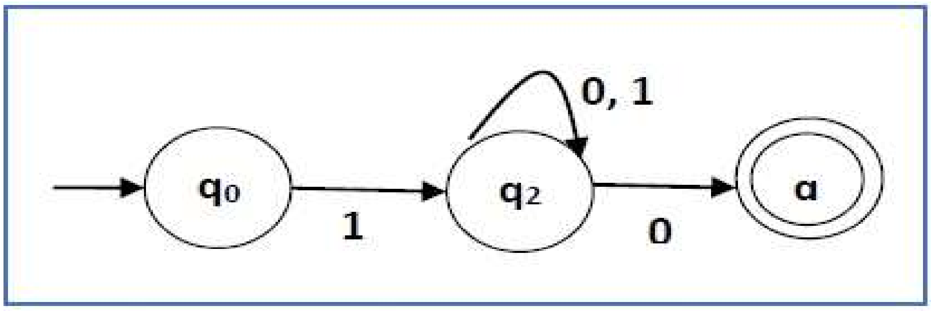
Solution

We will concatenate three expressions "1", "(0 + 1)*" and "0"



NDFA with NULL transition for RA: 1 (0 + 1)* 0

Now we will remove the **ε** transitions. After we remove the **ε** transitions from the NDFA, we get the following −



NDFA without NULL transition for RA: 1 (0 + 1)* 0

It is an NDFA corresponding to the RE − 1 (0 + 1)* 0. If you want to convert it into a DFA, simply apply the method of converting NDFA to DFA discussed in Chapter 1.

Applications of Regular Expressions:

Some Applications of Regular Languages
• Automata = finite state machines (or extensions thereof) used in many disciplines (the design of finite state systems)
• Efficient string searching
• Pattern matching with regular expressions (example: Unix grep utility)
• Lexical analysis (a.k.a. scanning, tokenizing) in a compiler (the topic of a lecture later in the course)

Example: String Searching

Designing Finite Automata from Regular Expression

In this article, we will see some popular regular expressions and how we can convert them to finite automata.

- **Even number of a's :** The regular expression for even number of a's is **(b|ab*ab*)***. We can construct a finite automata as shown in Figure 1.

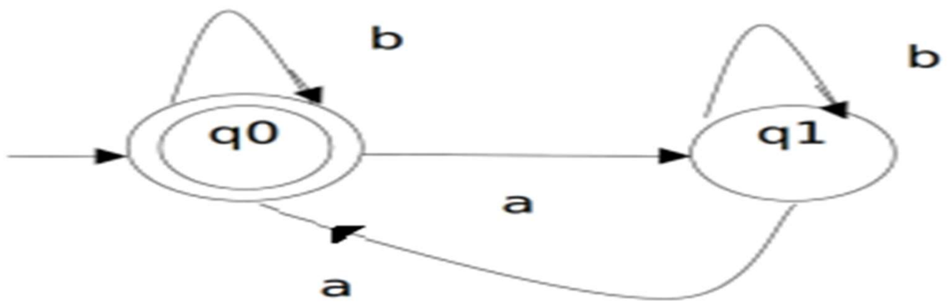


Figure 1

The above automata will accept all strings which have even number of a's. For zero a's, it will be in q0 which is final state. For one 'a', it will go from q0 to q1 and the string will not be accepted. For two a's at any positions, it will go from q0 to q1 for 1st 'a' and q1 to q0 for second 'a'. So, it will accept all strings with even number of a's.

- **String with 'ab' as substring :** The regular expression for strings with 'ab' as substring is **(a|b)*ab(a|b)***. We can construct finite automata as shown in Figure 2.

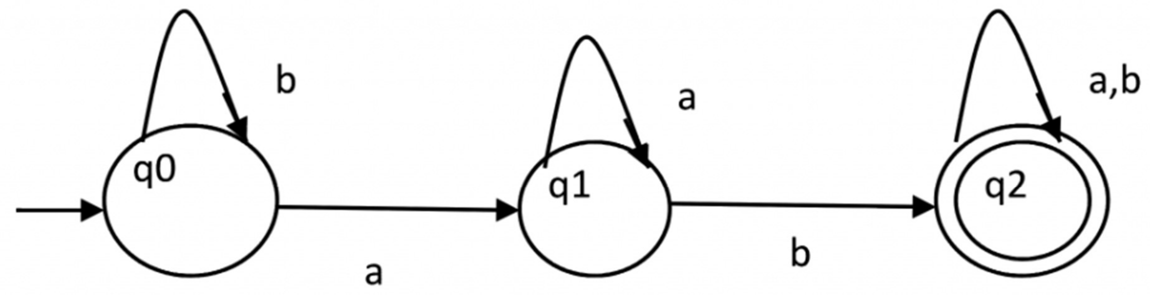


Figure 2

The above automata will accept all string which have ‘ab’ as substring. The automata will remain in initial state q0 for b’s. It will move to q1 after reading ‘a’ and remain in same state for all ‘a’ afterwards. Then it will move to q2 if ‘b’ is read. That means, the string has read ‘ab’ as substring if it reaches q2.

- **String with count of ‘a’ divisible by 3 :** The regular expression for strings with count of a divisible by 3 is $\{a^{3n} \mid n >= 0\}$. We can construct automata as shown in Figure 3.

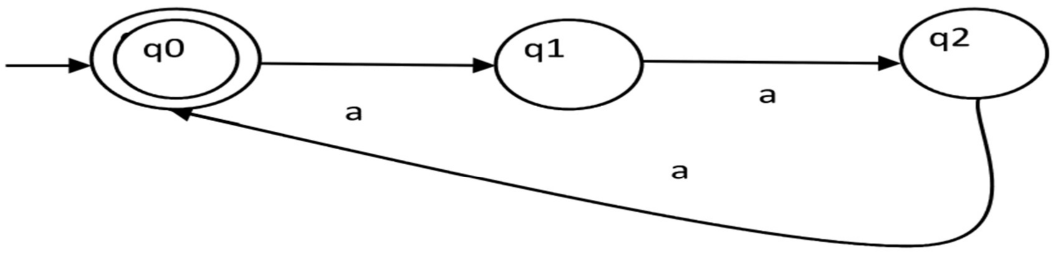


Figure 3

The above automata will accept all string of form a^{3n}. The automata will remain in initial state q0 for ε and it will be accepted. For string ‘aaa’, it will move from q0 to q1 then q1 to q2 and then q2 to q0. For every set of three a’s, it will come to q0, hence accepted. Otherwise, it will be in q1 or q2, hence rejected.

Note : If we want to design a finite automata with number of a’s as 3n+1, same automata can be used with final state as q1 instead of q0. If we want to design a finite automata with language $\{a^{kn} \mid n >= 0\}$, k states are required. We have used k = 3 in our example.

- **Binary numbers divisible by 3 :** The regular expression for binary numbers which are divisible by three is **(0|1(01*0)*1)***. The examples of binary number divisible by 3 are 0, 011, 110, 1001, 1100, 1111, 10010 etc. The DFA corresponding to binary number divisible by 3 can be shown in Figure 4.

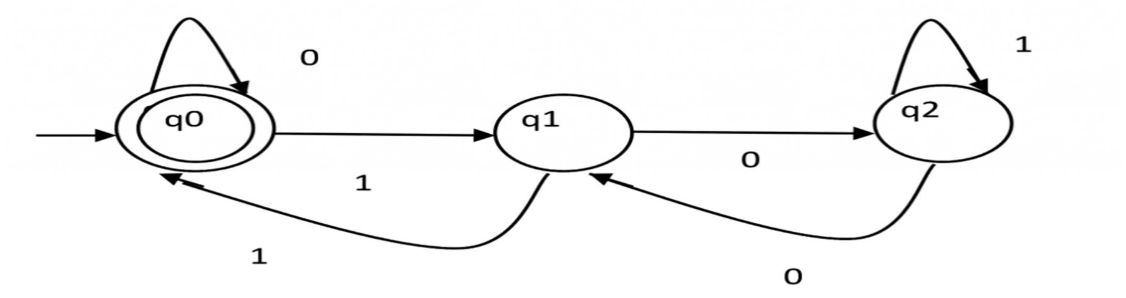


Figure 4

The above automata will accept all binary numbers divisible by 3. For 1001, the automata will go from q0 to q1, then q1 to q2, then q2 to q1 and finally q2 to q0, hence accepted. For 0111, the automata will go from q0 to q0, then q0 to q1, then q1 to q0 and finally q0 to q1, hence rejected.

- **String with regular expression (111 + 11111)* :** The string accepted using this regular expression will have 3, 5, 6(111 twice), 8 (11111 once and 111 once), 9 (111 thrice), 10 (11111 twice) and all other counts of 1 afterwards. The DFA corresponding to given regular expression is given in Figure 5.

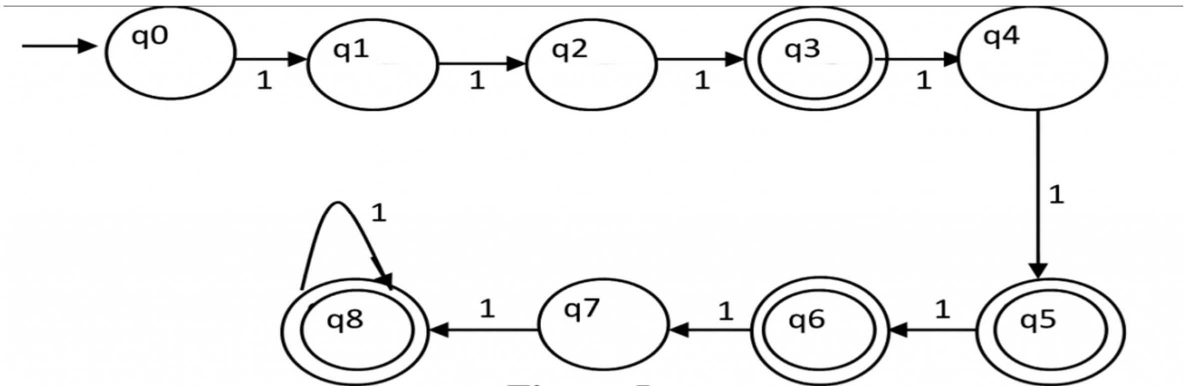


Figure 5

Question : What will be the minimum number of states for strings with odd number of a's?
Solution : The regular expression for odd number of a is b*ab*(ab*ab*)* and corresponding automata is given in Figure 6 and minimum number of states are 2.

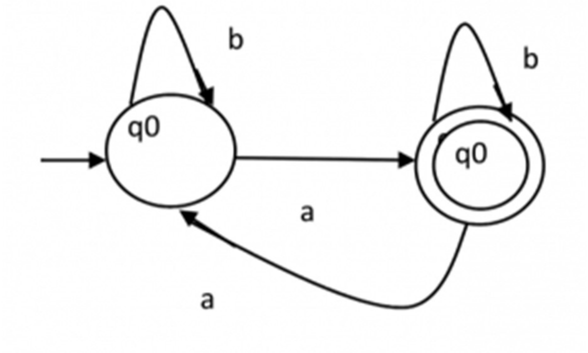


Figure 6

Day-9

Regular Expressions and Regular Grammars:

Regular Expressions, Regular Grammar and Regular Languages

As discussed in Chomsky Hierarchy, Regular Languages are the most restricted types of languages and are accepted by finite automata.

Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language {ε}.
- If a ∈ Σ (Σ represents the input alphabet), a is regular expression with language {a}.
- If a and b are regular expression, a + b is also a regular expression with language {a,b}.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a* (0 or more times a) is also regular.

	Regular Expression	Regular Languages
a followed by 0 or more b	$(a.b^*)$	{a,ab,abb,abbb.....}
set of vowels	(a ∪ e ∪ i ∪ o ∪ u)	{a,e,i,o,u}
Any number of vowels followed by any number of consonants	$v^*.\ c^*$ where v=(a ∪ e ∪ i ∪ o ∪ u) and c=(b ∪ c ∪.... ∪ z)	{ε,a,ai,aiu,bc,abc....} where ε represents empty string (in case of 0 vowels and 0 consonants)

Regular Grammar : A grammar is regular if it has rules of form A -> a or A -> aB or A ->ε where ε is a special symbol called NULL.

Regular Languages : A language is regular if it can be expressed in terms of regular expression.

Closure Properties of Regular Languages

Union : If L1 and If L2 are two regular languages, their union L1 ∪ L2 will also be regular. For example, L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$
L3 = L1 ∪ L2 = $\{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Intersection : If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,
L1= $\{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and L2= $\{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$
L3 = L1 ∩ L2 = $\{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

Concatenation : If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,
L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$
L3 = L1.L2 = $\{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

Kleene Closure : If L1 is a regular language, its Kleene closure L1* will also be regular. For example,
L1 = (a ∪ b)
L1* = (a ∪ b)*
Complement : If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example,
L(G) = $\{a^n \mid n > 3\}$
L'(G) = $\{a^n \mid n <= 3\}$

Note : Two regular expressions are equivalent if languages generated by them are same. For example, (a+b*)* and (a+b)* generate same language. Every string which is generated by (a+b*)* is also generated by (a+b)* and vice versa.

How to solve problems on regular expression and regular languages?

Question 1 : Which one of the following languages over the alphabet {0,1} is described by the regular expression?
(0+1)*0(0+1)*0(0+1)*
(A) The set of all strings containing the substring 00.
(B) The set of all strings containing at most two 0's.
(C) The set of all strings containing at least two 0's.
(D) The set of all strings that begin and end with either 0 or 1.

Solution : Option A says that it must have substring 00. But 10101 is also a part of language but it does not contain 00 as substring. So it is not correct option.
Option B says that it can have maximum two 0's but 00000 is also a part of language. So it is not correct option.
Option C says that it must contain atleast two 0. In regular expression, two 0 are present. So this is correct option.
Option D says that it contains all strings that begin and end with either 0 or 1. But it can generate strings which start with 0 and end with 1 or vice versa as well. So it is not correct.

Question 2 : Which of the following languages is generated by given grammar?
S->aS|bS|ϵ
(A) $\{a^n b^m \mid n,m \geq 0\}$
(B) {w∈{a,b}*|w has equal number of a's and b's}
(C) $\{a^n \mid n \geq 0\} \cup \{b^n \mid n \geq 0\} \cup \{a^n b^n \mid n \geq 0\}$
(D) {a,b}*
Solution : Option (A) says that it will have 0 or more a followed by 0 or more b. But S -> bS => baS => ba is also a part of language. So (A) is not correct.
Option (B) says that it will have equal no. of a's and b's. But But S -> bS => b is also

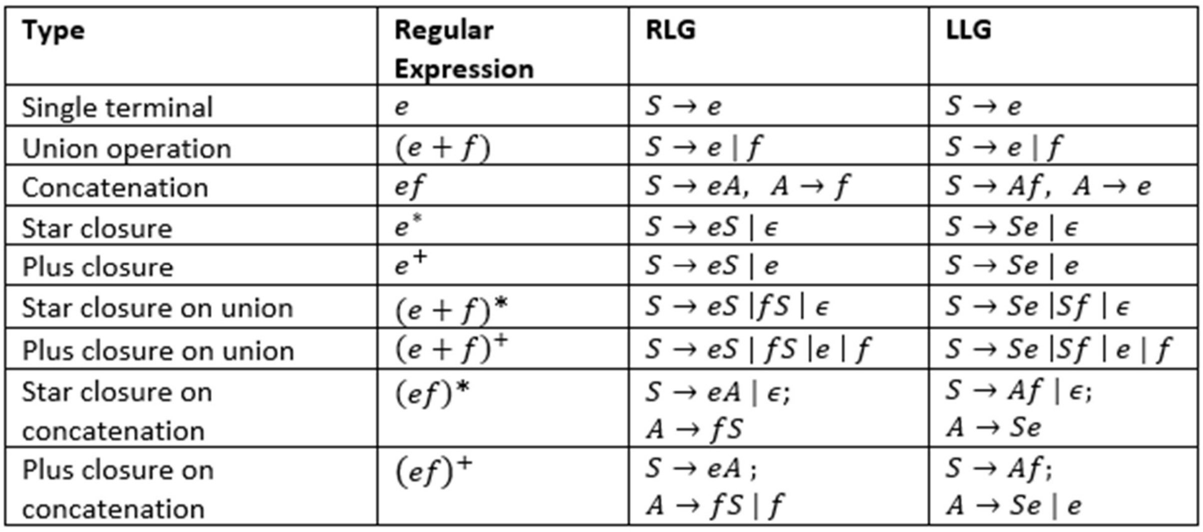
a part of language. So (B) is not correct.
Option (C) says either it will have 0 or more a’s or 0 or more b’s or a’s followed by b’s. But as shown in option (A), ba is also part of language. So (C) is not correct.
Option (D) says it can have any number of a’s and any numbers of b’s in any order. So (D) is correct.

Question 3 : The regular expression 0*(10*)* denotes the same set as
(A) (1*0)*1*
(B) 0+(0+10)*
(C) (0+1)*10(0+1)*
(D) none of these

Solution : Two regular expressions are equivalent if languages generated by them are same.
Option (A) can generate 101 but 0*(10*)* cannot. So they are not equivalent.
Option (B) can generate 0100 but 0*(10*)* cannot. So they are not equivalent.
Option (C) will have 10 as substring but 0*(10*)* may or may not. So they are not equivalent.

Steps to convert regular expressions directly to regular grammars and vice versa

It came across following intuitive rules to convert basic/minimal regular expressions directly to regular grammar (RLG for Right Linear Grammars, LLG for Left Linear Grammars):



Type	**Regular Expression**	**RLG**	**LLG**
Single terminal	e	$S \to e$	$S \to e$
Union operation	$(e+f)$	$S \to e \mid f$	$S \to e \mid f$
Concatenation	ef	$S \to eA,\ \ A \to f$	$S \to Af,\ \ A \to e$
Star closure	e^*	$S \to eS \mid \epsilon$	$S \to Se \mid \epsilon$
Plus closure	e^+	$S \to eS \mid e$	$S \to Se \mid e$
Star closure on union	$(e+f)^*$	$S \to eS \mid fS \mid \epsilon$	$S \to Se \mid Sf \mid \epsilon$
Plus closure on union	$(e+f)^+$	$S \to eS \mid fS \mid e \mid f$	$S \to Se \mid Sf \mid e \mid f$
Star closure on concatenation	$(ef)^*$	$S \to eA \mid \epsilon;$ $A \to fS$	$S \to Af \mid \epsilon;$ $A \to Se$
Plus closure on concatenation	$(ef)^+$	$S \to eA\,;$ $A \to fS \mid f$	$S \to Af;$ $A \to Se \mid e$