**Point to be remembered based on your current knowledge**

**Pumping Lemma**

Logic of pumping lemma is based on the concept of Pigeon-hole principle which states that if there are n pigeons fly into m hole and n > m then atleast one hole must contain more than one pigeon. And logic of pumping lemma states that- finite state automaton can assume only a finite number of states and because there are infinitely many inputs sequence, therefore by the pigeonhole principle, there must be atleast one state to which the automata return over and over again.

**Steps to prove that a language is not regular by using PL are as follows−**

step 1 − We have to assume that L is regular

step 2 − So, the pumping lemma should hold for L.

step 3 − It has to have a pumping length (say P).

step 4 − All strings longer that P can be pumped |z|>=p.

step 5 − Now find a string 'z' in L such that |z|>=P

step 6 − Divide z into xyz.

step 7 − Show that xyiz ∉ L for some i.

step 8 − Then consider all ways that z can be divided into xyz.

step 9 − Show that none of these can satisfy all the 3 pumping conditions at same time.

step 10 − z cannot be pumped = CONTRADICTION.

**Closure Properties of Regular Language**

The different closure properties for regular languages are as follows −

* Union
* Intersection
* concatenation
* Kleene closure
* Complement

## **Union**

If L1 and If L2 are two regular languages, their union L1 U L2 will also be regular.

## Example, Let

L1 = {an | n > O} and L2 = {bn | n > O}

L3 = L1 U L2 = {an U bn | n > O} is also regular.

## **Intersection**

If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular.

Example

L1= {am bn | n > 0 and m > O} and

L2= {am bn U bn am | n > 0 and m > O}

L3 = L1 ∩ L2 = {am bn | n > 0 and m > O} are also regular.

## **Concatenation**

If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular.

Example, Let

L1 = {am | m > 0} and L2 = {bn | n > O}

L3 = L1. L2 = {am .bn | m > 0 and n > O} is also regular.

## **Kleene Closure**

If L1 is a regular language, its Kleene closure L1\* will also be regular.

Example, Let

L1 = a

L1\* = (a)\*

## **Complement**

If L(G) is a 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.

Example, Let

L(G) = {an | n > 3} L'(G) = {an | n <= 3}

**Note** − Two regular expressions are equivalent, if languages generated by them are the same.

**Ambiguous Grammar**

A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation. If the grammar is not ambiguous, then it is called unambiguous.

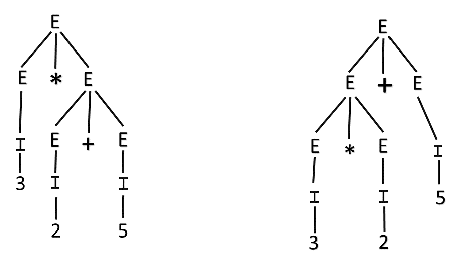
### Example 1:

Let us consider a grammar G with the production rule

1. E → I
2. E → E + E
3. E → E \* E
4. E → (E)
5. I → ε | 0 | 1 | 2 | ... | 9

**Solution:**

For the string "3 \* 2 + 5", the above grammar can generate two parse trees by leftmost derivation:



Since there are two parse trees for a single string "3 \* 2 + 5", the grammar G is ambiguous.

### Example 2:

Check whether the given grammar G is ambiguous or not.

1. E → E + E
2. E → E - E
3. E → id

**Solution:**

From the above grammar String "id + id - id" can be derived in 2 ways:

**First Leftmost derivation**

1. E → E + E
2. E→ id + E
3. E   → id + E - E
4. E → id + id - E
5. E → id + id- id

**Second Leftmost derivation**

1. E → E - E
2. → E + E - E
3. → id + E - E
4. → id + id - E
5. → id + id - id

Since there are two leftmost derivations for a single string "id + id - id", the grammar G is ambiguous.

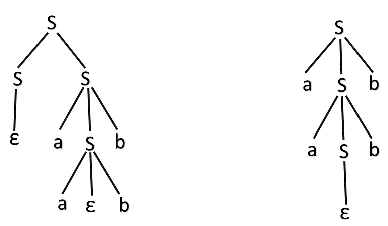
### Example 3:

Check whether the given grammar G is ambiguous or not.

1. S → aSb | SS
2. S → ε

**Solution:**

For the string "aabb" the above grammar can generate two parse trees



Since there are two parse trees for a single string "aabb", the grammar G is ambiguous.