

**Subject Name: Theory of Computation** 

Subject Code: IT-5001

Semester: 5<sup>th</sup>





## **Unit-II: Regular Grammar**

#### **Grammar:**

A grammar G can be formally written as a 4-tuple (N, T, S, P) where

- N or VN is a set of variables or non-terminal symbols
- T or ∑ is a set of Terminal symbols
- S is a special variable called the Start symbol, S ∈ N
- P is Production rules for Terminals and Non-terminals. A production rule has theform  $\alpha \to \beta$ , where  $\alpha$  and  $\beta$  are strings on  $V_N \cup \Sigma$  and least one symbol of  $\alpha$  belongsto  $V_N$ .

#### **Derivations from a Grammar:**

Strings may be derived from other strings using the productions in a grammar. If agrammar G has a production  $\alpha \rightarrow \beta$ , we can say that x  $\alpha$  y derives x  $\beta$  y in G. Thisderivation is written as:

$$x\alpha y \Rightarrow x\beta y$$

### Example:

Let us consider the grammar:

G2 = ( $\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \epsilon\}$ )

Some of the strings that can be derived are:

 $S \rightarrow aAb$  using production  $S \rightarrow aAb$ 

- → aaAbb using production aA → aAb
- → aaaAbbb using production aA → aAb
- $\rightarrow$  aaabbb using production A  $\rightarrow$   $\epsilon$



### Language generated by a Grammar:

The set of all strings that can be derived from a grammar is said to be the languagegenerated from that grammar. A language generated by a grammar G is a subset formally defined by

$$L(G) = \{ W \mid W \in \Sigma^*, S \Rightarrow W \}$$

If L(G1) = L(G2), the Grammar G1 is equivalent to the Grammar G2.

### Example:

If there is a grammar

G: N = {S, A, B} T = {a, b} P = {S 
$$\rightarrow$$
AB, A  $\rightarrow$ a, B  $\rightarrow$ b}

Here S produces AB, and we can replace A by a, and B by b. Here, the only accepted string is ab, i.e.,  $L(G) = {ab}$ 

## Regular Expression(RE):

Regular expressions are useful for representing certain sets of strings in an algebraic fashion. These describe the languages accepted by finite state automata.

### A Regular Expression can be recursively defined as follows:

- 1.  $\varepsilon$  is a Regular Expression indicates the language containing an empty string.(L ( $\varepsilon$ )= { $\varepsilon$ })
- 2.  $\phi$  is a Regular Expression denoting an empty language. (L ( $\phi$ ) = { })
- 3. x is a Regular Expression where L={x}
- 4. 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) U L(Y)where L(X+Y) = L(X) U 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\*) whereL(R\*) = (L(R))\*
- 5. If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

## **Regular Set:**

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

## **Properties of Regular Set:**

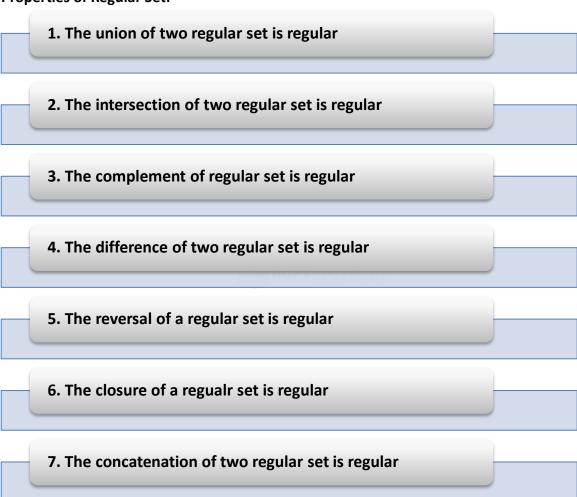


Figure 2.1: Properties of Regular Set

## **Identities related to Regular Expression:**

- 1.  $Ø^* = \varepsilon$
- 2.  $\varepsilon^* = \varepsilon$
- 3.  $RR^* = R^*R$
- 4. R\*R\* = R\*
- 5.  $(R^*)^* = R^*$
- 6.  $RR^* = R^*R$
- 7. (PQ)\*P = P(QP)\*



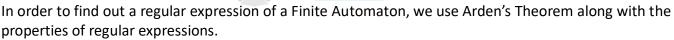
- 8.  $(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$
- 9.  $R + \emptyset = \emptyset + R = R$  (The identity for union)
- 10.  $R\varepsilon = \varepsilon R = R$  (The identity for concatenation)
- 11.  $\emptyset$ L = L $\emptyset$  =  $\emptyset$  (The annihilator for concatenation)
- 12. R + R = R (Idempotent law)
- 13. L(M + N) = LM + LN (Left distributive law)
- 14. (M + N) L = LM + LN (Right distributive law)
- 15.  $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

## Closure properties of Regular Language(RL):

If certain languages are regular then language formed by certain operations is also regular. These is called Closure properties of Regular Language(RL).

- The set of regular languages is closed under the union operation, i.e., if A1 and A2 are regular languages over the same alphabet  $\Sigma$ , then A1 U A2 is also a regular language.
- The set of regular languages is closed under the concatenation operation, i.e., if A1 and A2 are regular languages over the same alphabet  $\Sigma$ , then A1 A2 is also a regular language.
- The set of regular languages is closed under the star operation, i.e., if A is a regular language, then A\* is also a regular language.
- The set of regular languages is closed under the complement operation. i.e., Complement of RL is regular.
- The set of regular languages is closed under the difference operation i.e., Difference of two RL is regular.
- The set of regular languages is closed under the reversal operation i.e., Reversal of a RL is regular.

### **Arden's Theorem:**



#### Statement:

Let P and Q be two regular expressions.

If P does not contain null string, then R = Q + RP has a unique solution that is R = QP\*

### **Proof:**

R = Q + (Q + RP)P [After putting the value R = Q + RP]

R = Q + QP + RPP

When we put the value of R recursively again and again, we get the following equation:

R = Q + QP + QP2 + QP3....

 $R = Q (\varepsilon + P + P2 + P3 + ....)$ 

 $R = QP^* [As P^* represents (\varepsilon + P + P2 + P3 + ....)]$ 

Hence, proved.

Assumptions for Applying Arden's Theorem:

- 1. The transition diagram must not have NULL transitions
- 2. It must have only one initial state

### Myhill-Nerode Theorem:

A language L is regular if and only if RL has a finite number of equivalence classes. Moreover, the number of states is the smallest DFA recognizing L is equal to the number of equivalence classes of RL.



The following three statements are equivalent

- 1. The set L  $\in \Sigma^*$  is accepted by a FSA
- 2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
- Let equivalence relation R<sub>L</sub> be defined by :

 $xR_Ly$  if for all z in  $\Sigma^*$  xz is in L exactly when yz is in L.

Then R<sub>I</sub> is of finite index.

Example:-To show  $L = \{a^nb^n \mid n \ge 1\}$  is not regular

- Assume that L is Regular
- Then by Myhill Nerode theorem we can say that L is the union of sum of the Equivalence classes and etc
  - a, aa,aaa,aaaa,.......
- Each of this cannot be in different equivalence classes.

$$a^n \sim a^m$$
 for  $m \neq n$ 

By Right invariance

$$a^nb^n \sim a^m b^n$$
 for  $m \neq n$ 

Hence contradiction: The L cannot be regular.

# **Pumping lemma:**



Pumping lemma is tool that can be used to prove that certain languages are not regular. Observe that for a regular language,

- 1. The amount of memory that is needed to determine whether or not a given string is the language is finite and independent of the length of the string, and
- 2. If the language consists of an infinite number of strings, then this language should contain infinite subsets having a fairly repetitive structure.

Intuitively, languages that do not follow both point should be non-regular.

### **Example:**Consider the language

$$\{0^n \ 1^n : n \ge 0\}.$$

This language should be non-regular, because it seems unlikely that a DFA can remember how many 0s it has seen when it has reached the border between the 0s and the 1s. Similarly the language

should be non-regular, because the prime numbers do not seem to have any repetitive structure that can be used by a DFA.

This property is called the **pumping lemma**. If a language does not have this property, then it must be non-regular. The pumping lemma states that any sufficiently long string in a regularlanguage can be pumped, i.e., there is a section in that string that can be repeated any number of times, so that the resulting strings are all in the language.

### Theorem:

Let L be a regular language. Then there exists a constant 'c' such that for every stringw in L:  $|w| \ge c$ 

RGPV NOTES.IN

We can break w into three strings, w = xyz, such that:

- 1. |y| > 0
- 2.  $|xy| \le c$
- 3. For all  $k \ge 0$ , the string xykz is also in L.

## Applications of Pumping Lemma:

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

- If L is regular, it satisfies Pumping Lemma.
- If L does not satisfy Pumping Lemma, it is non-regular.

### Method to prove that a language L is not regular

- At first, we have to assume that L is regular.
- So, the pumping lemma should hold for L.
- Use the pumping lemma to obtain a contradiction
  - a) Select  $\mathbf{w}$  such that  $|\mathbf{w}| \ge \mathbf{c}$
  - b) Select **y** such that  $|y| \ge 1$
  - c) Select **x** such that  $|xy| \le c$
  - d) Assign the remaining string to z.
  - e) Select k such that the resulting string is not in L.

## Hence L is not regular.

**Example:** Prove that  $L = \{a^ib^i \mid i \ge 0\}$  is not regular.

- At first, we assume that **L** is regular and n is the number of states.
- Let  $w = a^n b^n$ . Thus  $|w| = 2n \ge n$ .
- By pumping lemma, let w = xyz, where  $|xy| \le n$ .
- Let  $x = a^p$ ,  $y = a^q$ , and  $z = a^r b^n$ , where p + q + r = n,  $p \ne 0$ ,  $q \ne 0$ ,  $r \ne 0$ . Thus  $|y| \ne 0$ .

KUPVINU I ED. IIV

- Let k = 2. Then  $xy^2z = a^pa^{2q}a^rb^n$ .
- Number of as = (p + 2q + r) = (p + q + r) + q = n + q
- Hence,  $xy^2z = a^{n+q}b^n$ . Since  $q \ne 0$ ,  $xy^2z$  is not of the form  $a^nb^n$ .
- Thus, xy<sup>2</sup>z is not in L. Hence L is not regular.

### **Application of Finite Automata:**

Some of the major applications of finite automata are:

Compiler Design: Lexical Analysis

Special purpose hardware design

Protocol specification



String matching algorithm

#### Minimization of DFA:

DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states.

If X and Y are two states in a DFA, we can combine these two states into  $\{X, Y\}$  if theyare not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of  $\delta$  (X, S) and  $\delta$  (Y, S) is accepting and another is not accepting. Hence, aDFA is minimal if and only if all the states are distinguishable.

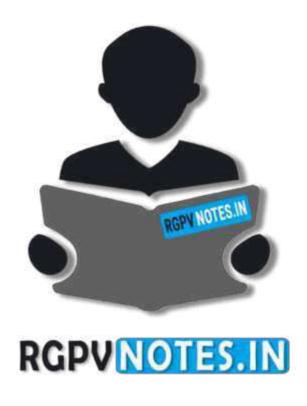
**Step 1:** All the states Q are divided in two partitions: final states and non-final states and are denoted by P0. All the states in a partition are 0th equivalent. Take a counter k and initialize it with 0.

**Step 2:** Increment k by 1. For each partition in Pk, divide the states in Pk into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input S such that  $\delta(X, S)$  and  $\delta(Y, S)$  are (k-1)-distinguishable.

**Step 3:** If Pk ≠ Pk-1, repeat Step 2, otherwise go to Step 4.

**Step 4:** Combine kth equivalent sets and make them the new states of the reducedDFA.





We hope you find these notes useful.

You can get previous year question papers at <a href="https://qp.rgpvnotes.in">https://qp.rgpvnotes.in</a>.

If you have any queries or you want to submit your study notes please write us at <a href="mailto:rgpvnotes.in@gmail.com">rgpvnotes.in@gmail.com</a>

