



RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Theory of Computation**

Subject Code: **IT-503**

Semester: **5th**



LIKE & FOLLOW US ON FACEBOOK

facebook.com/rgpvnotes.in

Department of Information Technology
Subject Notes
IT503 (A) - Theory of Computation
B.Tech, IT-5th Semester

Unit II

Syllabus : Regular grammars, regular expressions, regular sets, closure properties of regular grammars, Arden's theorem, Myhill-Nerode theorem, pumping lemma for regular languages, Application of pumping lemma, applications of finite automata, minimization of FSA.

Unit Objective: Obtain minimized DFA and Application of regular expression and conversion from RE to Finite Automata and Finite Automata to Regular Expression and Proving language are not regular.

Regular Grammar

Grammar:

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

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

Derivations from a Grammar:

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

$$\begin{array}{c} G \\ x\alpha y \Rightarrow x\beta y \end{array}$$

Example:

Let us consider the grammar:

$G_2 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \epsilon\})$

Some of the strings that can be derived are:

- $S \rightarrow aAb$ using production $S \rightarrow aAb$
- $\rightarrow aaAbb$ using production $aA \rightarrow aAb$
- $\rightarrow aaaAbbb$ using production $aA \rightarrow aAb$
- $\rightarrow aaabbb$ 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 language generated 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 $L(G_1) = L(G_2)$, the Grammar G_1 is equivalent to the Grammar G_2 .

Example:

If there is a grammar

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

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. ϵ is a Regular Expression indicates the language containing an empty string. ($L(\epsilon) = \{\epsilon\}$)
2. ϕ 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 $L(X) \cup L(Y)$ where $L(X+Y) = L(X) \cup L(Y)$.
 - $X.Y$ 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 $L(R^*) = (L(R))^*$
5. If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

Example:

Regular Expressions	Regular Set
$(0 + 10^*)$	$L = \{ 0, 1, 10, 100, 1000, 10000, \dots \}$
(0^*10^*)	$L = \{1, 01, 10, 010, 0010, \dots\}$
$(0 + \epsilon)(1 + \epsilon)$	$L = \{\epsilon, 0, 1, 01\}$
$(a+b)^*$	Set of strings of a's and b's of any length including the null string. So $L = \{ \epsilon, a, b, aa, ab, bb, ba, aaa, \dots \}$
$(a+b)^*abb$	Set of strings of a's and b's ending with the string abb. So $L = \{abb, aabb, babb, aaabb, ababb, \dots\}$

Regular Set:

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

Properties of Regular Set:

1. The union of two regular set is regular

2. The intersection of two regular set is regular

3. The complement of regular set is regular

4. The difference of two regular set is regular

5. The reversal of a regular set is regular

6. The closure of a regular set is regular

7. The concatenation of two regular set is regular

Figure 2.1: Properties of Regular Set

Identities related to Regular Expression:

1. $\emptyset^* = \epsilon$
2. $\epsilon^* = \epsilon$
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. $R\epsilon = \epsilon R = R$ (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. $(M + N)L = ML + NL$ (Right distributive law)
15. $\epsilon + RR^* = \epsilon + R^*R = R^*$

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 A_1 and A_2 are regular languages over the same alphabet Σ , then $A_1 \cup A_2$ is also a regular language.
- The set of regular languages is closed under the concatenation operation, i.e., if A_1 and A_2 are regular languages over the same alphabet Σ , then $A_1 A_2$ 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:

In order to find out a regular expression of a Finite Automaton, we use Arden's Theorem along with the properties of regular expressions.

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:

$$R = Q + (Q + RP)P \text{ [After putting the value } R = Q + RP \text{]}$$

$$R = Q + QP + RPP$$

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

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

$$R = Q (\epsilon + P + P^2 + P^3 + \dots)$$

$$R = QP^* \text{ [As } P^* \text{ represents } (\epsilon + P + P^2 + P^3 + \dots)]$$

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 in 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.
3. Let equivalence relation R_L be defined by :
 $xR_L y$ if for all z in Σ^* xz is in L exactly when yz is in L .
 Then R_L is of finite index.

Example:-To show $L = \{a^n b^n \mid n \geq 1\}$ 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.

$$a^n \sim a^m \quad \text{for } m \neq n$$

- By Right invariance

$$a^n b^n \sim a^m b^n \quad \text{for } m \neq n$$

- 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

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

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

$$\{0^n : n \text{ is a prime number}\}$$

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 regular language 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 string w in L:

$$|w| \geq c$$

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

1. $|y| > 0$
2. $|xy| \leq c$
3. For all $k \geq 0$, the string xy^kz 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 w such that $|w| \geq c$
 - b) Select y such that $|y| \geq 1$
 - c) Select x such that $|xy| \leq c$
 - 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 $L = \{a^i b^i \mid i \geq 0\}$ 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 $|w| = 2n \geq n$.
- By pumping lemma, let $w = xyz$, where $|xy| \leq n$.
- Let $x = a^p$, $y = a^q$, and $z = a^r b^n$, where $p + q + r = n$, $p \neq 0$, $q \neq 0$, $r \neq 0$. Thus $|y| \neq 0$.
- Let $k = 2$. Then $xy^2z = a^p a^{2q} a^r b^n$.
- Number of a 's = $(p + 2q + r) = (p + q + r) + q = n + q$
- Hence, $xy^2z = a^{n+q} b^n$. Since $q \neq 0$, xy^2z is not of the form $a^n b^n$.
- Thus, xy^2z 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 they are 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, a DFA is minimal if and only if all the states are distinguishable.

Suppose there is a DFA $D = \langle Q, \Sigma, q_0, \delta, F \rangle$ which recognizes a language L . Then the minimized DFA $D' = \langle Q', \Sigma, q_0, \delta', F' \rangle$ can be constructed for language L as:

Step 1: We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P_0 .

Step 2: Initialize $k = 1$

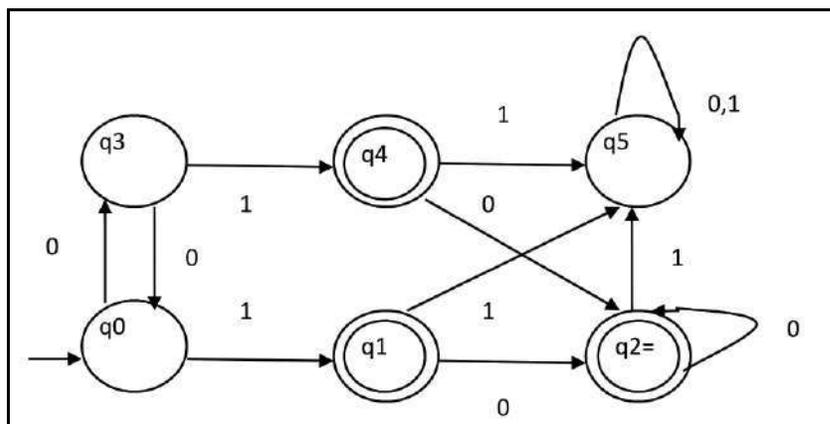
Step 3: Find P_k by partitioning the different sets of P_{k-1} . In each set of P_{k-1} , we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in P_k .

Step 4: Stop when $P_k = P_{k-1}$ (No change in partition)

Step 5: All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in P_k .

Example:

Consider the following DFA



Step 1. P_0 will have two sets of states. One set will contain q_1, q_2, q_4 which are final states of DFA and another set will contain remaining states. So $P_0 = \{\{q_1, q_2, q_4\}, \{q_0, q_3, q_5\}\}$.

Step 2. To calculate P_1 , we will check whether sets of partition P_0 can be partitioned or not:

i) For set $\{q_1, q_2, q_4\}$:

$\delta(q_1, 0) = \delta(q_2, 0) = q_2$ and $\delta(q_1, 1) = \delta(q_2, 1) = q_5$, So q_1 and q_2 are not distinguishable.

Similarly, $\delta(q_1, 0) = \delta(q_4, 0) = q_2$ and $\delta(q_1, 1) = \delta(q_4, 1) = q_5$, So q_1 and q_4 are not distinguishable.

Since, q_1 and q_2 are not distinguishable and q_1 and q_4 are also not distinguishable, So q_2 and q_4 are not distinguishable. So, $\{q_1, q_2, q_4\}$ set will not be partitioned in P_1 .

ii) For set $\{q_0, q_3, q_5\}$:

$\delta(q_0, 0) = q_3$ and $\delta(q_3, 0) = q_0$

$\delta(q_0, 1) = q_1$ and $\delta(q_3, 1) = q_4$

Moves of q_0 and q_3 on input symbol 0 are q_3 and q_0 respectively which are in same set in partition P_0 .

Similarly, Moves of q_0 and q_3 on input symbol 1 are q_1 and q_4 which are in same set in partition P_0 .

So, q_0 and q_3 are not distinguishable.

$\delta(q_0, 0) = q_3$ and $\delta(q_5, 0) = q_5$ and $\delta(q_0, 1) = q_1$ and $\delta(q_5, 1) = q_5$

Moves of q_0 and q_5 on input symbol 0 are q_3 and q_5 respectively which are in different set in partition P_0 . So, q_0 and q_5 are distinguishable. So, set $\{q_0, q_3, q_5\}$ will be partitioned into $\{q_0, q_3\}$ and $\{q_5\}$. So,

$P_1 = \{\{q_1, q_2, q_4\}, \{q_0, q_3\}, \{q_5\}\}$

To calculate P_2 , we will check whether sets of partition P_1 can be partitioned or not:

iii) For set $\{q_1, q_2, q_4\}$:

$\delta(q_1, 0) = \delta(q_2, 0) = q_2$ and $\delta(q_1, 1) = \delta(q_2, 1) = q_5$, So q_1 and q_2 are not distinguishable.

Similarly, $\delta(q_1, 0) = \delta(q_4, 0) = q_2$ and $\delta(q_1, 1) = \delta(q_4, 1) = q_5$, So q_1 and q_4 are not distinguishable.

Since, q_1 and q_2 are not distinguishable and q_1 and q_4 are also not distinguishable, So q_2 and q_4 are not distinguishable. So, $\{q_1, q_2, q_4\}$ set will not be partitioned in P_2 .

iv) For set $\{q_0, q_3\}$:

$\delta(q_0, 0) = q_3$ and $\delta(q_3, 0) = q_0$

$\delta(q_0, 1) = q_1$ and $\delta(q_3, 1) = q_4$

Moves of q_0 and q_3 on input symbol 0 are q_3 and q_0 respectively which are in same set in partition P_1 . Similarly, Moves of q_0 and q_3 on input symbol 1 are q_1 and q_4 which are in same set in partition P_1 . So, q_0 and q_3 are not distinguishable.

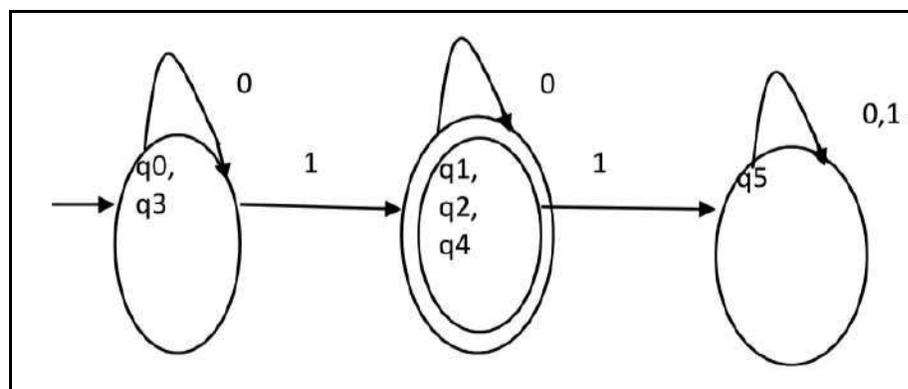
v) For set $\{q_5\}$:

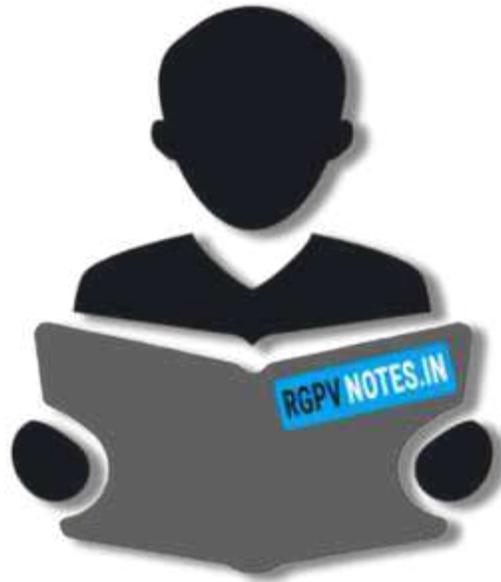
Since we have only one state in this set, it can't be further partitioned. So,

$P_2 = \{\{q_1, q_2, q_4\}, \{q_0, q_3\}, \{q_5\}\}$

Since, $P_1 = P_2$. So, this is the final partition. Partition P_2 means that q_1, q_2 and q_4 states are merged into one. Similarly, q_0 and q_3 are merged into one.

Minimized DFA corresponding to DFA is as below:





RGPVNOTES.IN

We hope you find these notes useful.

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

If you have any queries or you want to submit your
study notes please write us at
rgpvnotes.in@gmail.com



LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in