1
Formal Language Theory and Finite Automata

Syllabus

Introduction to Formal language, introduction to language translation logic, Essentials of translation, Alphabets and languages, Finite representation of language, ***Finite Automata (FA):*** *An Informal Picture of FA, Finite State Machine (FSM), Language accepted by FA, Definition of Regular Language, Deterministic and Nondeterministic FA(DFA and NFA), epsilon-NFA, FA with output: Moore and Mealy machines -Definition, models, inter-conversion.*

Contents

1.1 Introduction to Formal Language

- Formal language is normally defined by using alphabets and formation rules.
- The **alphabet** of formal language is a set of symbols using which the language is built.
- The **formation rules** specify which strings of symbols are used in well-formed manner. The well-formed strings of symbols are also called **words**. The formation rules can be recursive.
- In theory of computations the formal languages are denoted by **regular expressions,regular languages, non regular languages, context free grammar** and so on.

1.2 Introduction to Language Translation Logic

Logic is a study of reasoning. In the study of formal languages mathematical logic is represented by **propositions**.

The **propositions** or statements are declarative sentences, which can be either true or false but not both.

For example :

I am proud of my country.

I like cricket.

Sugar is hot to taste.

You are always welcome.

These are propositions with **true** or **false** values. We can join two propositions by 'and', 'but', 'if', 'or'. These are called **connectives.**

Various connectives that are used are

i) Negation

ii) Conjunction

iii) Disjunction

iv) Implication

v) if and only if

Let us discuss each one by one -

I) Negation

This proposition is also called **NOT** proposition. It is denoted by ¬ or ~ or having horizontal bar on letter.

If S is a proposition then ¬S defines NOT S. The truth table for this connective is

S	¬S
T	F
F	T

II) Conjunction (AND)

If A and B are two propositions then conjunction of A and B is denoted by **A^B**. The truth table for this connective is

A	B	A^B
T	T	T
T	F	F
F	T	F
F	F	F

III) Disjunction (OR)

If A and B are two propositions then disjunction of A and B is denoted by **A∨B**. The truth table for this connective is

A	B	A∨B
T	T	T
T	F	T
F	T	T
F	F	F

iv) Implication

If A and B are two propositions then **if A then B** is a proposition. It can be denoted by **A → B** or **A ⇒ B**.

The truth table for this connective is

A	B	A → B
T	T	T
T	F	F
F	T	T
F	F	T

For example : Find truth value for following proposition.

"If an apple is not green then it must be red".

Let, A denotes "An apple is not green" and B denotes "It must be red". The statement A is false and B is true. Hence the proposition is **true**.

v) If and only if

If A and B are two propositions then A if and only if B. It can be denoted as A ↔ B. The truth table will be

A	B	A ↔ B
T	T	T
T	F	F
F	T	F
F	F	T

For example : "I will go to the school if and only if it is not raining". Here statement A is "I will go to the school". The statement B is "It is not raining". Thus the resultant proposition is false.

1.3 Alphabets and Languages

SPPU : Dec.-10, Marks 4

1.3.1 Defining Languages

The language is a **collection of appropriate strings.**

The language is defined using an input set. The input set is typically denoted by letter $\sum$.

For example : $\sum = \{\varepsilon, 0, 00, 000, ...\}$

Here the language L is defined as 'any number of Zero's.

Note that language is formed by appropriate strings and strings are formed by alphabets !

1.3.2 Symbols/Alphabets

An alphabet is a **finite collection of symbols**. We cannot define symbol formally.

The example of alphabet is,

$$S = \{a, b, c \ldots\ldots, z\}$$

The elements a, b,.....,z are the alphabets.

$$W = \{0, 1\}$$

Here 0 and 1 are alphabets, denoted by W.

Symbols : Members of an alphabet are called symbols.

For example, in string "0011" the prefixes can be 0, 00, 001. Similarly suffixes can be 1, 11, 011.

In string "Mango" the prefix can be 'Man' and the suffix can be 'go'.

Review Question

1. *Define the following terms with example - 1) Symbol 2) Alphabet.*

SPPU : Dec.-10, Marks 4

1.3.3 String or Word

String is finite collection of symbols from alphabet.

For example, if $\sum$ = {a, b} then various strings that can be formed from Σ are {ab, ab, aa, aaa, bb, bbb, ba, aba ...}. An infinite number of strings can be generated from this set.

- The **empty string** can be denoted by ε.
- The **prefix** of any string is any number of leading symbols of that string and **suffix** is any number of trailing.

1.4 Finite Automata(FA)

1.4.1 An Informal Picture of FA

Finite automata is useful to solve real-world problems. To simulate any real-world problem, the events are captured by the states of finite automata. The interactions among the participants is then represented by transitions.

For example : Suppose a customer wants to purchase some books on-line. Then there are three participants involved in this scenario. Those are customer, book-store and bank. The events that would take place in such a system are -

1. The customer may decide to **pay.**

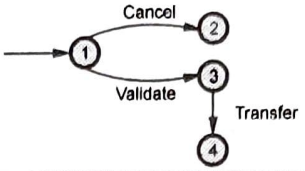
   The customer purchases the required book and sends the money to the store, by credit-card.

2. The customer may decide to **cancel** the purchase order. The money is sent to the bank with a message that the value of the money is to be added to the customer's bank account.

3. The book-store may **ship** books to the customer.

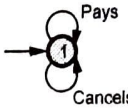
4. The store may get is **validated.** That means the credit card information will be validated by the bank. And it checked whether the customer's bank balance is greater than the purchase amount.

Then the finite automata for **bank** can be drawn as follows -



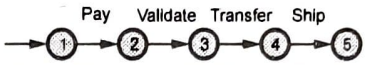
If customer cancels the purchase order then the money will not be transferred to the store. If customer purchases the book, then bank validates his credit card and transfers the money to the book-store.

The finite automata for **customer** can be



Customer performs only two actions either pays the money or cancel the order.

The finite automata for **book-store** can be -



The customer pays the money by credit card, the book store gets it validated from bank then the required amount of money gets transferred from customer's account to book-store's account. When the book-store gets the purchase amount then the books will be shipped to the customer.

1.4.2 Finite Representation of Language

The strings and languages can be accepted by a finite automata, when it reaches to a final state. There are two preferred notations for describing automata :

1. Transition diagram -

A transition diagram or transition graph can be defined as collection of -

1) Finite set of states K.

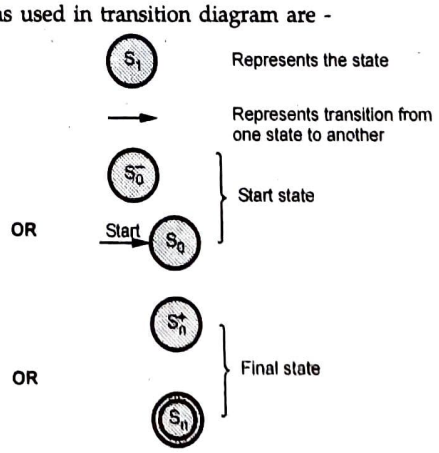
2) Finite set of symbols Σ .

3) A non empty set S of K. It is called **start state.**

4) A set $F \subseteq K$ of **final states.**

5) A transition function $K \times A \rightarrow K$ with K as state and A as input from Σ^*.

The notations used in transition diagram are -



For example :

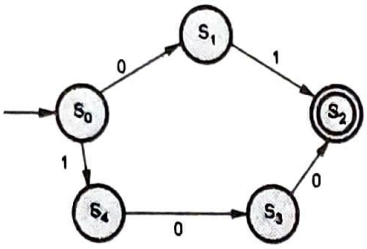
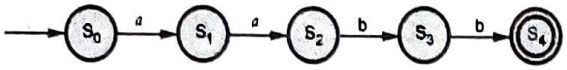


Fig. 1.4.1 Transition diagram

The FA can be represented using transition graph. The machine initially is in start state S_0 then on receiving input 0 it changes to state S_1. From S_0 on receiving input 1 the machine changes its state to S_4. The state S_2 is a final state or accept state. When we trace the input for transition diagram and reach to a final state at end of input string then it is said that the given input is **accepted by** transition diagram.

Another example :



We have drawn a transition diagram for the input aabb.

Note that the start state is S_0 and final state is S_4. The input set is $\Sigma = \{a, b\}$. The states S_1, S_2, S_3 are all intermediate states.

2. Transition table

This is a tabular representation of finite automata. For transition table the transition function is used.

For example :

States \ Input	a	b
q_0	q_1	-
q_1	-	q_2
q_2	q_2	-

The rows of the table corresponds to states and columns of the table corresponds to inputs.

1.4.3 Finite State Machine (FSM)

A finite state machine is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

Q is a finite **set of states**, which is non empty.

Σ is **input** alphabet, indicates input set.

q_0 is an **initial state** and q_0 is in Q i.e. $q_0 \in Q$.

F is a set of **final states.**

δ is a **transition function** or a mapping function. Using this function the next state can be determined.

1.4.4 Language Accepted by FA

The finite automata can be represented as :

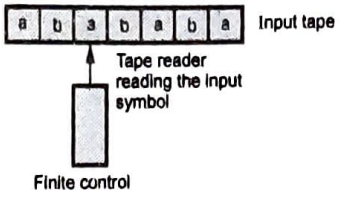


Fig. 1.4.2 Model for finite automata

The finite automata can be represented using :

i) **Input tape** - It is a linear tape having some number of cells. Each input symbol is placed in each cell.

ii) **Finite control** - The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right and at a time only one input symbol is read.

For example, suppose current state is q, and suppose reader is reading the symbol 'a' then it is finite control which decides what will be the next state at input 'a'. The transition from current state q with input w to next state q' producing w' will be represented as,

$$(q, w) \vdash (q', w')$$

If w is a string and M is a finite automata, then w is accepted by the FA.

iff $(w, s) \vdash^* (q, \varepsilon)$

with q as final state.

The set of strings accepted by a FA given by M then M is accepted by language L. The acceptance of M by some language L is denoted by L(M).

A machine M accepts a language L iff,

$$L = L(M).$$

1.5 Definition of Regular Language

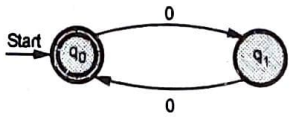
Regular languages are the sets which are accepted by finite automata.

For example : L = {ε, 00, 0000, 000000, ...}

i.e. the set of even number of zeros.

We can represent this set by a finite automata as shown in figure.

$$M = (\{q_0, q_1\}, \{0\}, \delta, q_0, q_0)$$



The given set L is said to be a regular languages because it is represented by finite automata.

1.6 Types of Finite Automata

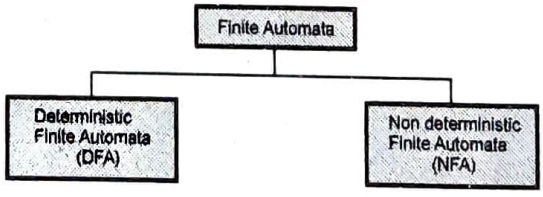


Fig. 1.6.1 DFA and NFA

There are two types of finite automata -

i) Deterministic Finite Automata.

ii) Non deterministic Finite Automata.

The deterministic finite automata is deterministic in nature, in the sense, each move in this automata is uniquely determined on current input and current state. On the other hand, it is non deterministic in nature, in NFA i.e. non deterministic finite automata.

1.7 Deterministic Finite Automata (DFA)

SPPU : May-10,12,13, Dec.-14,16, Aug.-15, Oct.-16, Marks 10

Definition of DFA

A deterministic finite automation is a collection of following things -

1) The finite set of states which can be denoted by Q.
2) The finite set of input symbols Σ.
3) The start state q_0 such that $q_0 \in Q$.
4) A set of final states F such that $F \in Q$.
5) The mapping function or transition function denoted by δ. Two parameters are passed to this transition function : One is current state and other is input symbol. The transition function returns a state which can be called as next state.

Example

For example, $q_0 = \delta (q_0, a)$ means from current state q_0, with input a the next state transition is q_2.

In short, the DFA is a five tuple notation denoted as :

$$A = (Q, \Sigma, \delta, q_0, F)$$

The name of DFA is A which is a collection of above described five elements.

It can be represented by following Fig. 1.7.1.

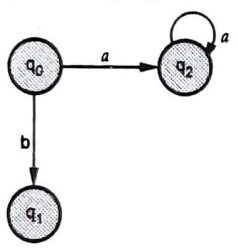


Fig. 1.7.1 Deterministic finite automata

Problems Based on DFA

Example 1.7.1 *Design a FA which checks whether the given binary number is even.*

Solution :

• Logic Used

The binary number is made up of 0's and 1's. When any binary number ends with 0 it is always even and when a binary number ends with 1 it is always odd. For example,

0100 is a even number, it is equal to 4.

0011 is a odd number, it is equal to 3.

• Design

While designing FA we will assume one start state, one state ending in 0 and other state for ending with 1. Since we want to check whether given binary number is even or not, we will make the state for 0, as the final state.

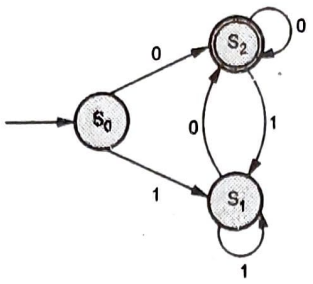


Fig. 1.7.2 DFA

• Simulation

The FA indicates clearly S_1 is a state which handles all the 1's and S_2 is a state which handles all the 0's. Let us take some input.

$$01000 \Rightarrow 0\underline{S_2}1000$$

$$01\underline{S_1}000$$

$$010\underline{S_2}00$$

$$0100\underline{S_2}0$$

$$01000S_2$$

Now at the end of input we are in final or in accept state so it is a even number. Similarly, let us take another input.

$$1011 \Rightarrow S_0 \vdash 1011$$

$$1S_1 \vdash 011$$

$$10S_2 \vdash 11$$

$$101S_1 \vdash 1$$

$$1011S_1 \vdash$$

Now we are at the state S_1 which is a non final state. Here the symbol "$\vdash$" denotes the symbols to be scanned. On LHS of $\vdash$ the already read symbols are given and at the R.H.S. of "$\vdash$" the symbols to be scanned are given.

• Representation of DFA using Transition Table

Another idea to represent FA with the help of transition table.

States \ Input	0	1
→ S_0	S_2	S_1
S_1	S_2	S_1
(S_2)	S_2	S_1

Table 1.7.1 Transition table

(1) Thus the table indicates in the first column all the current states. And under the column 0 and 1 the next states are shown.

(2) The first row of transition table can be read as : when current state is S_0, on input 0 the next state will be S_2 and on input 1 the next state will be S_1. The **arrow** marked to S_0 indicates that it is a **start state** and **circle** marked to S_2 indicates that it is a **final state**. Sometimes * is also used to indicate final state.

Example 1.7.2 *Design FA which accepts odd number of 1's and any number of 0's.*

Solution :

• Logic

In the problem statement, it is indicated that there will be a state which is meant for odd number of 1's and that will be the final state. There is no condition on number of 0's.

• Design

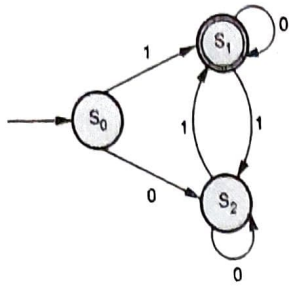


Fig. 1.7.3

Example 1.7.3 *Design FA which checks whether the given unary number is divisible by 3.*

Solution :

• Logic

The unary number is made up of ones. The number 3 can be written in unary form as 111, number 5 can be written as 11111 and so on. The unary number which is divisible by 3 can be 111 or 111111 or 111111111 and so on.

• Design

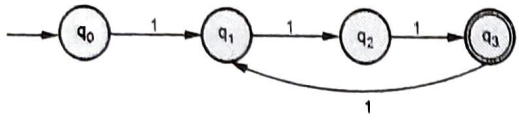


Fig. 1.7.4

Input \ State	1
→ q_0	q_1
q_1	q_2
q_2	q_3
(q_3)	q_1

• Simulation

Consider a number 111111 which is equal to 6 i.e. divisible by 3. So after complete scan of this number we reach to final state q_3.

Start 111111

State q_0

$1\underline{q_1}\,11111$

$11\underline{q_2}\,1111$

$111q_3\,111$

$1111\underline{q_1}\,11$

$11111\underline{q_2}\,1$

$111111q_3$ → Now we are in final state.

Example 1.7.4 *Design FA to check whether given decimal number is divisible by three.*

Solution :

• Logic

To determine whether the given decimal number is divisible by three, we need to take the input number digit by digit. Also, while considering its divisibility by three, we have to consider that the possible remainders could be 1, 2 or 0. The remainder 0 means, it is divisible by 3. Hence from input set {0, 1, 2, ..., 9} (since decimal number is a input), we will get either remainder 0 or 1 or 2 while testing its divisibility by 3. So we need to group these digits according to their remainders. The groups are as given below -

remainder 0 group : S_0 : (0, 3, 6, 9)

remainder 1 group : S_1 : (1, 4, 7)

remainder 2 group : S_2 : (2, 5, 8)

• Design

We will out these states as S_0,S_1 and S_2. The state S_0 will be the final state as it is remainder 0 state.

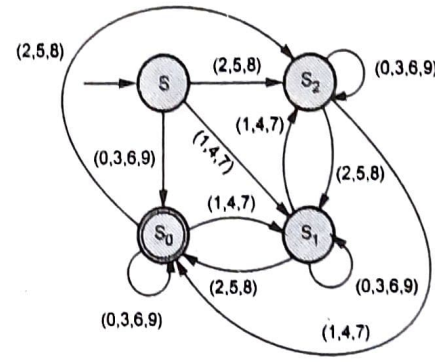


Fig. 1.7.5

• Simulation

Let us test the above FA, if the number is 36 then it will proceed by reading the number digit by digit.

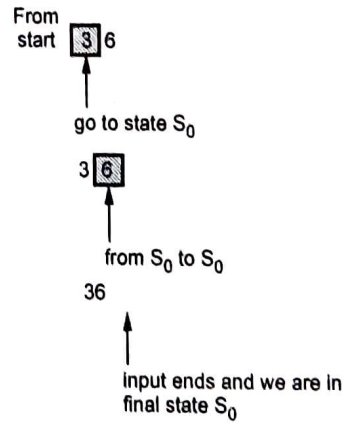


Fig. 1.7.6

Hence the number is divisible 3, isn't it ? Similarly if number is 121 which is not divisible by 3, it will be processed as

S 121

$1S_1\,21$

$12S_0\,1$

$121S_1$ which is remainder 1 state.

Example 1.7.5 *Design FA which checks whether a given binary number is divisible by three.*

Solution :

• Design

The input number is a binary number. Hence the input set will be $\Sigma = \{0, 1\}$. The start state is denoted by S, the remainder 0 is by S_0, remainder 1 by S_1 and remainder 2 by S_2.

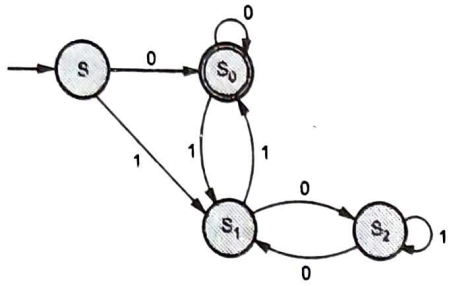


Fig. 1.7.7

• Simulation

Let us take some number 010 which is equivalent to 2. We will scan this number from MSB to LSB.

0 1 0
↑ ↑ ↑
S_0 S_1 S_2

Then we will reach to state S_2 which is remainder 2 state. Similarly, for input 1001 which is equivalent to 9 we will be in final state after scanning the complete input.

1 0 0 1
↑ ↑ ↑ ↑
S_1 S_2 S_1 S_0

Thus the number is really divisible by 3.

Example 1.7.6 *Design FA which accepts even number of 0's and even number of 1's.*

Solution :

• Logic

This FA will consider four different stages for input 0 and 1. The stages could be :

even number of 0 and even number of 1,

even number of 0 and odd number of 1,

odd number of 0 and even number of 1,

odd number of 0 and odd number of 1.

• Design

Let us try to design the machine :

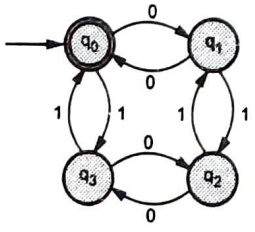


Fig. 1.7.8

Here q_0 is a start state as well as final state. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as :

q_0 : State of even number of 0's and even number of 1's.

q_1 : State of odd number of 0's and even number of 1's.

q_2 : State of odd number of 0's and odd number of 1's.

q_3 : State of even number of 0's and odd number of 1's.

The transition table can be as follows -

	0	1
→ q_0	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

Example 1.7.7 *Design FA to accept the string that always ends with 00.*

Solution :

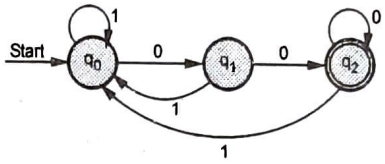


Fig. 1.7.9

If the input is 01001100 it will be processed as :

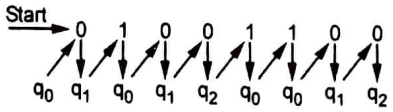


Fig. 1.7.10

The q_2 is a final state, hence the input is accepted.

Example 1.7.8 *Construct the transition graph for a FA which accepts a language L over $\Sigma\{0, 1\}$ in which every string start with 0 and ends with 1.*

Solution :

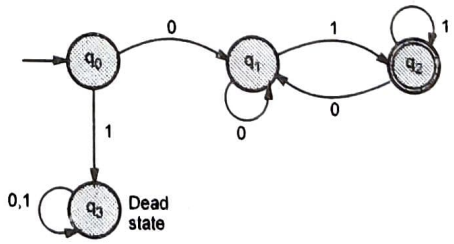


Fig. 1.7.11

In the above transition graph we have handled the case as if the input starts with 1 then it will be in q_3 state which is a dead state and never lead to final state. Thus this machine strictly handles the strings starting with 0 and ending with 1.

Example 1.7.9 *Design FA to accept L, where L = {Strings in which a always appears trippled} over the set Σ = {a, b}.*

Solution :

- **Logic**

For this particular language the valid strings are *aaa*b, b*aaaaaa*, bb*aaa*b and so on. The *a* always appears in a clump of 3. The TG (transition graph) will look like this -

- **Design**

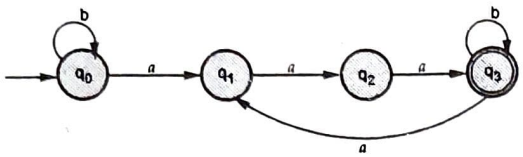


Fig. 1.7.12

Note that the sequence of tripple *a* is maintained to reach to the final state.

Example 1.7.10 *Design FA to accept L where all the strings in L are such that total number of a's in them are divisible by 3.*

Solution :

- **Logic**

As we have seen earlier, while testing divisibility by 3, we group the input as remainder 0, remainder 1 and remainder 2.

Hence,

S_0 : State of remainder 0.

S_1 : State of remainder 1.

S_2 : State of remainder 2.

- **Design**

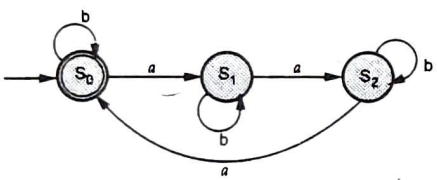


Fig. 1.7.13

Note the difference between previous example and this, here there is no condition as *a* should be in clump but total number of *a*'s in a string are divisible by 3. Hence b's are allowed in between.

Example 1.7.11 *Design a FA that reads strings made up of letters in the word. CHARIOT and recognize those strings that contain the word 'CAT' as a substring.*

Solution :

- **Logic**

To design this FA we will first consider the input set which will be Σ = {C, H, A, R, I, O, T}. From this input set we have to recognize the word 'CAT'. We can do that as follows -

Step 1 :

Fig. 1.7.14

Step 2 :

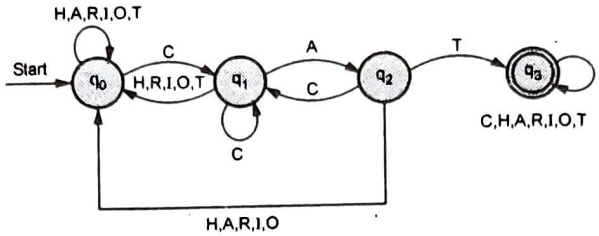


Fig. 1.7.15 Required finite automata

From Fig. 1.7.15 it is clear that on recognizing C initially from Σ = {C, H, A, R, I, O, T} we are moving to state q_1, other than C all the characters are remaining in state q_0 only. From q_1 the character A will be identified and from q_2 character T will be identifier. The transition table can be drawn as follows :

- **Design**

State \ Input	C	H	A	R	I	O	T
→ q_0	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
(q_3)	q_3	q_3	q_3	q_3	q_3	q_3	q_3

The substring may appear anywhere in the input string for any number of times. For example, 'HACCATHA'. From this 'CAT' can be recognized and we should reach to q_3 state finally.

Example 1.7.12 *Design DFA to accept odd and even numbers represented using binary notation.*

Solution :

- **Logic**

The binary number that ends with 0 is an even number and binary number that ends with 1 is an odd number. Hence the DFA is -

- **Design**

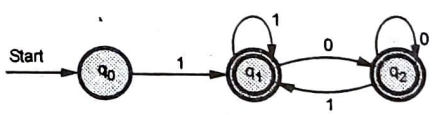


Fig. 1.7.16

Example 1.7.13 *Write a DFA to accept the language L = {L : |W| mod 5 ≠ 0}.*

Solution :

- **Logic**

The string which we obtain should not be divisible by 5. Hence the, DFA will be,

- **Design**

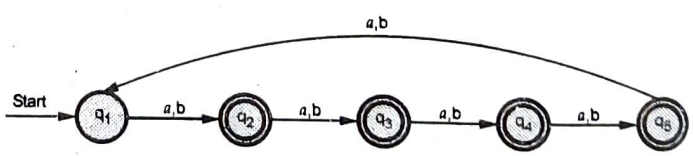


Fig. 1.7.17

The transition table can be drawn as follows -

State \ Input	a	b
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_4	q_4
q_4	q_5	q_5
q_5	q_1	q_1

We can consider the string "*abbb*". This string is a valid string as it is not divisible by 5 i.e *abbb* mod 5 ≠ 0. That also means that we should reach to final state on acceptance of this string. The simulation of this string for given DFA is :

- **Simulation**

$$\begin{aligned}\delta(q_1,abbb) &\vdash \delta(q_2,bbb)\\ &\vdash \delta(q_3,bb)\\ &\vdash \delta(q_4,b)\\ &\vdash \delta(q_5,\varepsilon)\end{aligned}$$

where q_5 is a final state.

Now consider string "*ababa*" which is invalid string as it is divisible by 5. That means ababa mod 5 = 0. That means we should reach to non final state on acceptance of this string. The simulation for this string is as given below.

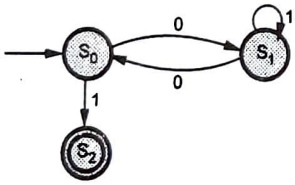
As q_1 is a start we will start from state q_1.

$$\begin{aligned}\delta(q_1, ababa) &\vdash \delta(q_2,baba)\\ &\vdash \delta(q_3,aba)\\ &\vdash \delta(q_4,ba)\\ &\vdash \delta(q_5,a)\\ &\vdash \delta(q_1,\varepsilon)\end{aligned}$$

Here q_1 is a non-final state. That means "*ababa*" is invalid string for the given DFA.

Example 1.7.14 *Design a DFA which accepts strings with even number of 0's followed by single 1 over* $\Sigma = \{0, 1\}$.

Solution : The DFA can be shown by a transition diagram as :



Here state S_1 will accept all the odd number of 0's and S_0 will accept all even number of 0's. It should then be followed by single 1. Hence S_2 is a final state.

- Consider the input :

	0	0	0	1	0	1
0	S_1	0	0	1	0	1
0	0	S_0	0	1	0	1
0	0	0	S_1	1	0	1
0	0	0	1	S_1	0	1
0	0	0	1	0	S_0	1
0	0	0	1	0	1	S_2

We now will reach to final state S_2, and the input ends here. Hence 000101 is a valid input string. The transition table for above drawn DFA can be represented as

States \ Input	0	1
→ S_0	S_1	S_2
S_1	S_0	S_1
(S_2)	-	-

Table 1.7.2 Transition table

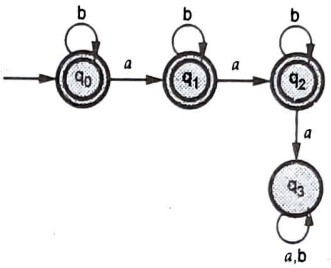
Example 1.7.15 *Design DFA which accepts all the strings not having more than two a's over* $\Sigma = \{a, b\}$.

Solution :

- **Logic**

In this DFA maximum two a's are accepted. If we try to accept third a then it should not lead us to final state.

- **Design**



The transition table can be as shown -

States \ Input	a	b
→ (q_0)	q_1	q_0
(q_1)	q_2	q_1
(q_2)	q_3	q_2
q_3	q_3	q_3

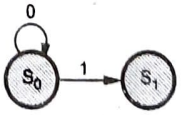
Table 1.7.3 Transition table

Example 1.7.16 *Design a DFA for accepting all the strings of* $\{L = 0^m 1^n | m \geq 0 \text{ and } n \geq 1\}$.

Solution :

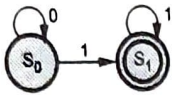
- **Logic**

This is a language in which all the 0's followed by all 1's. But number of zero's and number of 1's are different. The language explicitly mentions that there should be at least one 1. Any number of 0's followed by only one 1 is -



• Design

We have 1^n in the language. Hence the DFA can be



The transition table can be

State \ Input	0	1
S_0	S_0	S_1
S_1	-	S_1

• Simulation

Consider a string 00111, which is a valid string and is accepted by above drawn DFA.

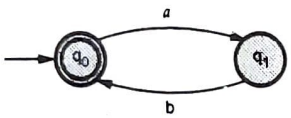
$$\begin{aligned}\delta(S_0, 00111) &\vdash (S_0, 0111)\\ &\vdash (S_0, 111)\\ &\vdash (S_1, 11)\\ &\vdash (S_1, 1)\\ &\vdash (S_1, \varepsilon)\end{aligned}$$

Thus we reach to final state S_1 on acceptance of 00111.

Example 1.7.17 *Design DFA over $\sum = \{a, b\}$ for i) $(ab)^n$ with $n \geq 0$ ii) $(ab)^n$ with $n \geq 1$.*

Solution : This is a language of ab in a pair in which i) Condition ε is accepted and in ii) Condition ε is not accepted.

The DFA for i) can be



ii) The DFA will be -

The string accepted by

i) $\delta(q_0, ab) \vdash (q_1, b)$
$\vdash (q_0, \varepsilon)$

Here q_0 is a final state.

ii) $\delta(q_0, ab) \vdash (q_1, b)$
$\vdash (q_2, \varepsilon)$

Here q_2 is final state.

Example 1.7.18 *Design DFA for accepting the set of integers.*

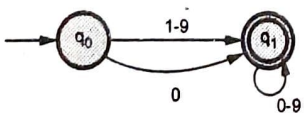
Solution :

• Logic

For representing any integer the set {0, 1, ... 9} is used. We can represent any integer value by taking appropriate combinations of symbols from {0, ... 9}.

• Design

DFA will be



Example 1.7.19 *Recognize the language given by following DFA.*

Solution : In the given DFA, at q_0 state a self loop for symbol *a* is given. That means any number of *a*'s are allowed. Then if single 'b' comes then it leads to final state. But after this b if anything i.e. 'a' or 'b' comes then we reach to a non final state q_2 and in this state whatever may come, we will be in q_2 state only. Basically state q_2 is called a dead state. Thus it is a language in which any number of *a*'s should be followed by single b only. Hence this language is denoted as

$\{L = a^n b \mid n \geq 0\}$.

Example 1.7.20 *Design DFA for a language of string 0 and 1 that I) Ending with 10 II) Ending with 11 III) Ending with 1* **PU : May-10, Marks 10**

Solution :

I) DFA which ends with 10

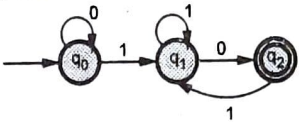


Fig. 1.7.18

II) DFA which ends with 11

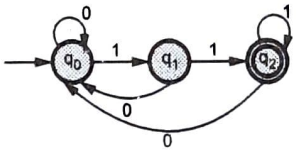


Fig. 1.7.19

III) DFA which ends with 1

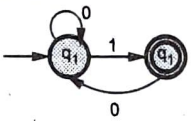


Fig. 1.7.20

Example 1.7.21 *Construct DFA for checking "whether a string over alphabet {a, b} contains a substring abb".* **SPPU : May-12, Marks 8**

Solution : The DFA will be -

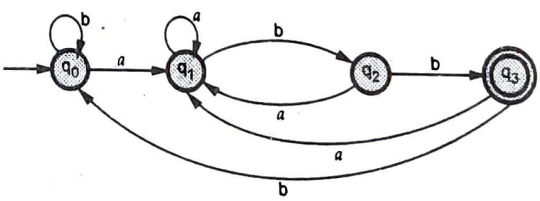


Fig. 1.7.21

Example 1.7.22 *Design a finite automata that reads strings made up of letters in the word 'CHARIOT' and recognize those strings that contain the word 'CAT' as a substring.* **SPPU : May-13, Marks 8**

Solution :

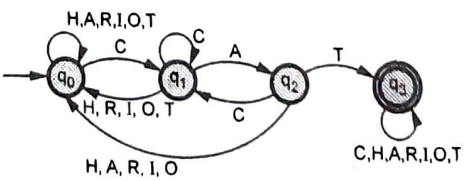


Fig. 1.7.22

State \ Input	C	H	A	R	I	O	T
→ q_0	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

$M = (\{q_0, q_1, q_2, q_3\}, \{C, H, A, R, I, O, T\}, \delta, q_0, \{q_3\})$

Example 1.7.23 *Design FSM over (a, b) accepting the string where number of a's are divisible by 2 and number of b's are divisible by 3.*

SPPU : Dec.-14, End Sem., Marks 6

Solution :

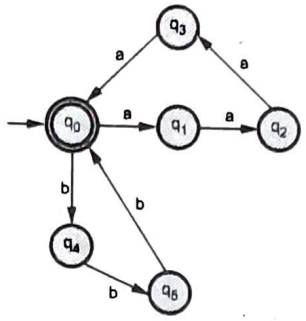
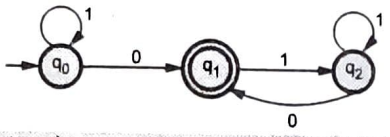


Fig. 1.7.23

Example 1.7.24 *Draw FA for strings which do not contain '00' as substring in it over alphabet* $\sum = \{0,1\}^*$

SPPU : Aug.-15, In Sem, Marks 2

Solution :



Example 1.7.25 *Construct a DFA over the alphabets {0, 1} for accepting the strings having number of 1's as multiple of 3.*

SPPU : Dec.-16, End sem, Marks 6

Solution :

- **Logic**

Here the condition is on number of 1's and not on number of 0's.

- **Design**

Hence DFA is as follows :

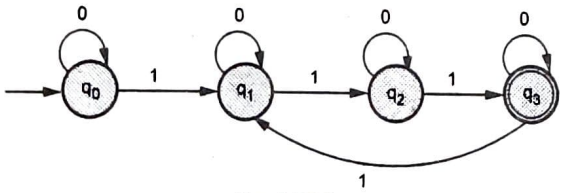


Fig. 1.7.24

- **Simulation :**

Consider string 0100110

q_0 ⊢ 0100110

$0q_0$ ⊢ 100110

$01q_1$ ⊢ 00110

$010q_1$ ⊢ 0110

$0100q_0$ ⊢ 110

$01001q_2$ ⊢ 10

$010011q_3$ ⊢ 0

$0100110q_3$

ACCEPT

Example 1.7.26 *Design Finite Automata (FA) for accepting strings, over* $\Sigma = \{0, 1\}$*, with even number of 0's and odd number of 1's.*

SPPU : Oct. 2016, In sem, Marks 4

Solution : The finite automata will be as follows -

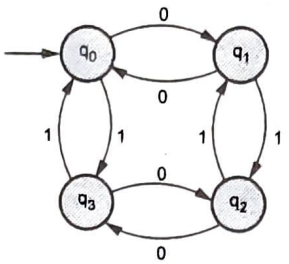


Fig. 1.7.25

1.8 Nondeterministic Finite Automata (NFA)

SPPU : Dec.-12,13, 14, Aug.-15, May-12, 14, Marks 4

Definition of NFA

The NFA can be formally defined as a collection of 5-tuples.

1) Q is a finite set of states.
2) Σ is a finite set of inputs.
3) δ is called next state or transition function.
4) q_0 is initial state.
5) F is a final state where $F \subseteq Q$.

There can be multiple final states. Thus the next question might be what is the use of NFA. The NFA is basically used in theory of computations because they are more flexible and easier to use than the DFAs.

Example

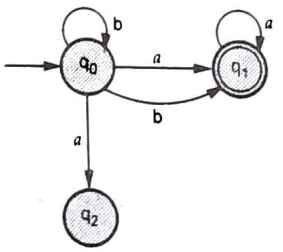


Fig. 1.8.1 Non deterministic finite automata

Difference in NFA and DFA

Sr. No.	NFA	DFA
1.	NFA stands for Non deterministic Finite Automata.	DFA stands for Determinnistic Finite Automata

2.	For every input symbol there can be more than one state transition.	For every input symbol there can be only one state transition.
3.	NFA can use empty string transition.	DFA can not use empty string transition.
4.	Construction of NFA is simple	Construction of DFA is difficult

Problems Based on NFA

Example 1.8.1 *An NFA with states 1-5 and input alphabet {a, b} has following transition table :*

q	δ(q, a)	δ(q, b)
1	{1, 2}	{1}
2	{3}	{3}
3	{4}	{4}
4	{5}	ϕ
5	ϕ	{5}

i) Draw transition diagram ii) Calculate δ (1, ab) iii) Claculate δ* (1, abaab)* **SPPU : May-14, Marks 6**

Solution :

i) Transition diagram

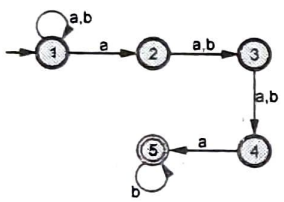


Fig. 1.8.2

ii) δ* (1, ab) = 1 or 3

iii) δ* (1, abaab) = 1 or 3 or 4 or 5

Example 1.8.2 *Give non-deterministic finite automata to accept the following language over {0, 1}*. The set of all strings containing either 101 or 110 as a substring.* **SPPU : Dec.-13, Marks 4**

Solution :

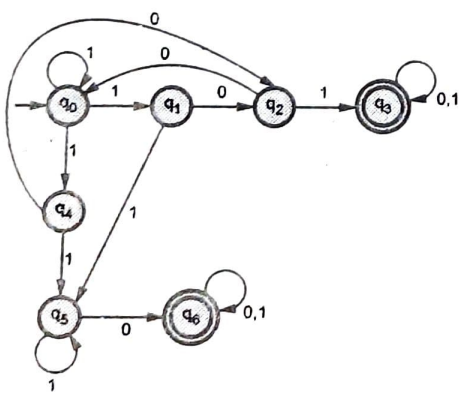


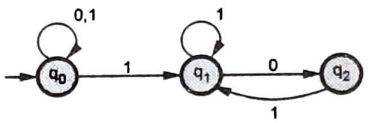
Fig. 1.8.3

Example 1.8.3 *Is NFA more powerful than DFA ? Justify.* **SPPU : Dec.-14, In Sem, Marks 2**

Solution : The NFA is not more powerful than DFA. In fact NFA can be converted to equivalent DFA. Thus any regular language can be represented by both NFA and DFA.

Example 1.8.4 *Draw NFA for strings ending with '10' over alphabet* $\sum = \{0,1\}^*$. **SPPU : Aug.-15, In Sem, Marks 2**

Solution :



Review Questions

1. *Differentiate between NFA and DFA.* **SPPU : May-12, Marks 2, May-14, Marks 4**
2. *Define formally with example - Non deterministic Finite Automata.* **SPPU : Dec.-12, May-14, Marks 2**

1.9 Epsilon - NFA **SPPU : May-11, Dec.-13, Marks 5**

The ε-transitions in NFA are given in order to move from one state to another without having any symbol from input set Σ .

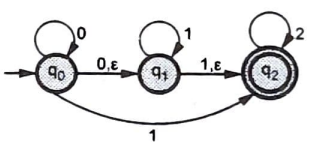


Fig. 1.9.1

- In this NFA with ε, q_0 is a **start state** with input 0.
- Initially with **input 0**, we can go to q_0 or q_1.
- Similarly with **input 1**, we can go to state q_1 or q_2.
- Thus it is not definite that on input 1 whether we will be in state q_1 or q_2. Hence it is called **non-deterministic Finite Automata** (NFA) and since there are some ε moves to simply change the states, it is called **NFA with ε**.

Definition

The language L accepted by NFA with ε, denoted by $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as :

Let, $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA with ε.

where

Q is a finite set of states.

Σ is input set.

δ is a transition or a mapping function for transitions from $Q \times \{\Sigma \cup \varepsilon\}$ to 2^Q.

q_0 is a start state.

F is a set of final states such that $F \in Q$.

The string w in L accepted by NFA can be represented as

$L(M) = \{w \mid w \in \Sigma^*$ and δ transition for w from q_0 reaches to F$\}$.

Example 1.9.1 *Construct NFA with ε which accepts a language consisting the strings of any number of a's followed by any number of b's. Followed by any number of c's.*

Solution : Here any number of a's or b's or c's means zero or more in number. That means there can be zero or more a's followed by zero or more b's followed by zero or more c's. Hence NFA with ε can be -

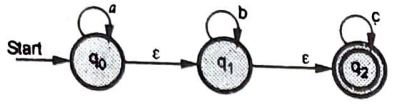


Fig. 1.9.2

Normally ε's are not shown in the input string. The transition table can be -

δ	Input: a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

(Σ)

We can parse the string *aa*bbcc as follows -

$\delta(q_0, aabbcc) \vdash \delta(q_0, abbcc)$
$\vdash \delta(q_0, bbcc)$
$\vdash \delta(q_0, \varepsilon bbcc)$
$\vdash \delta(q_1, bbcc)$
$\vdash \delta(q_1, bcc)$
$\vdash \delta(q_1, cc)$
$\vdash \delta(q_1, \varepsilon cc)$
$\vdash \delta(q_2, cc)$
$\vdash \delta(q_2, c)$
$\vdash \delta(q_2, \varepsilon)$

Thus we reach to accept state, after scanning the complete input string.

1.9.1 ε - Closure

The ε - closure (p) is a set of all states which are reachable from state p on ε-transitions such that :

i) ε - closure (p) = p where $p \in Q$.

ii) If there exists ε - closure (p) = {q} and $\delta(q, \varepsilon) = r$ then ε - closure (p) = {q, r}.

Example 1.9.2 *Find ε - closure for the following NFA with ε.*

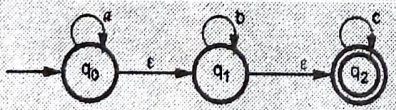


Fig. 1.9.3

Solution :

ε - closure $(q_0) = \{q_0, q_1, q_2\}$ means self state + ε - reachable states.

ε - closure $(q_1) = \{q_1, q_2\}$ means q_1 is a self state and q_2 is a state obtained from q_1 with ε input.

ε - closure $(q_2) = \{q_2\}$

Review Questions

1. *Describe - NFA with null moves.* **SPPU : May-11, Marks 2**
2. *Give the formal definitions of following with suitable examples: Epsilon closure, Non-deterministic finite automata and deterministic finite automata.* **SPPU : Dec.-13, Marks 5**

1.10 FA with Output : Moore and Mealy Machines

SPPU : May-06, 07, 10, 11, 12, 13, 14, Dec.-05, 06, 11, 12, 13, Marks 8

The finite automata is a collection of $(Q, \Sigma, \delta, q_0, F)$ where Q is a set of states including q_0 as a start state. In FA after reading the input string if we get final state then the string is said to be "acceptable". If we do not get final state then it is said that string is "rejected". That means there is no need of output for the finite Automata. The 'accept' or 'reject' acts like 'yes' or 'no' output for the machine. But if there is a need for specifying the output other than yes or no, then in such a case we require finite automata along with output. There are two types of FA with output and those are :

1) Moore machine

2) Mealy machine

Moore machine

Moore machine is a finite state machine in which the next state is decided by current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. The formal definition of Moore machine is,

Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where
- Q is finite set of states.
- Σ is finite set of input symbols.
- Δ is an output alphabet.
- δ is an transition function such that $Q \times \Sigma \rightarrow Q$. This is also known as state function.
- λ is output function $Q \rightarrow \Delta$. This function is also known as machine function.
- q_0 is the initial state of machine.

For example

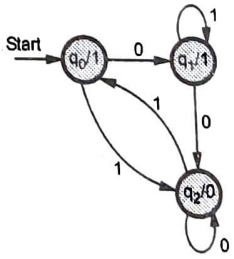


Fig. 1.10.1

Consider the Moore machine given below -

The transition table will be -

Current state	Next state (δ) 0	1	Output (λ)
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

In Moore machine output is associated with every state. In the above given Moore machine when machine is in q_0 state the output will be 1. For the Moore machine if the length of input string is n then output string has length n+1.

For the string 0110 then the output will be 11110.

Mealy machine

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. The Mealy machine can be defined as -

Mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q	is finite set of states.
Σ	is finite set of input symbols.
Δ	is an output alphabet.
δ	is state transition function such that $Q \times \Sigma \rightarrow Q$.
λ	is machine function such that $Q \times \Sigma \rightarrow \Delta$.
q_0	is initial state of machine.

For example,

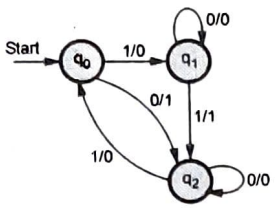


Fig. 1.10.2

For the input string 1001 the output will be 0001. In mealy machine the length of input string is equal to length of output string.

Difference between Moore and Mealy Machine

1) Length of Moore machine is one longer than Mealy machine for given input.
2) Secondly output of the input will be along the edges in case of Mealy machine but it should be associated with the state in case of Moore machine.

Problems Based on Moore and Melay Machine Modeling

Example 1.10.1 *Design a Moore machine to generate 1's complement of given binary number.*

SPPU : May-10, May-15, End Sem, Marks 6

Solution :

• Logic

To generate 1's complement of given binary number the simple logic which we will apply is that if input is 0 then output will be 1 and if input is 1 then output will be 0. That means there are three state one-start state, second state is for taking 0's as input and produces output as 1. Then third state is for taking 1's as input and producing output as 0.

• Design

Hence the Moore machine will be as shown in Fig. 1.10.3.

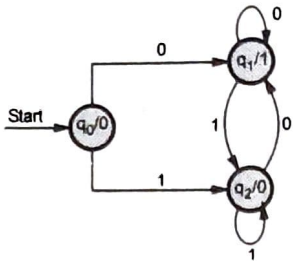


Fig. 1.10.3

• Simulation

For instance, take one binary number 1011 then

Input		1	0	1	1
State	q_0	q_2	q_1	q_2	q_2
Output	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transition table can be drawn as below -

Current state	Next state δ: 0	Next state δ: 1	Output λ
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

Thus Moore machine M = $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$; where Q = $\{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $\Delta = \{0,1\}$.

The transition table shows the δ and λ functions.

Example 1.10.2 *Design a Moore and Mealy machine for a binary input sequence such that if it has a substring 101 the machine outputs A if input has substring 110 it outputs B otherwise it outputs C.*

SPPU : Dec.-06, May-07, Marks 10

Solution :

• Logic

For designing such a machine we need to take care of two conditions and those are checking 101 and checking 110. If we get 101 the output will be A. If we recognize 110 the output will be B. For other strings the output will be C. We can make a partial design of it as follows.

• Design

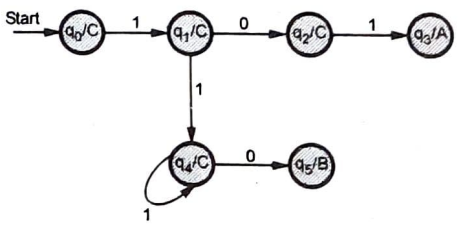


Fig. 1.10.4

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes.

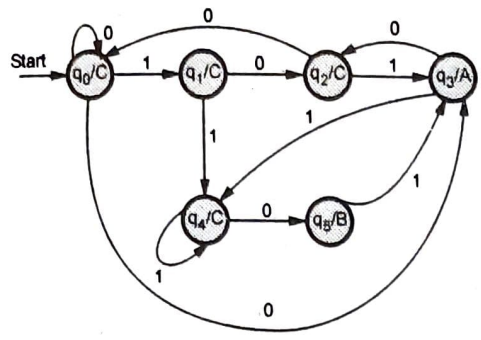


Fig. 1.10.5

Now the Mealy machine can be

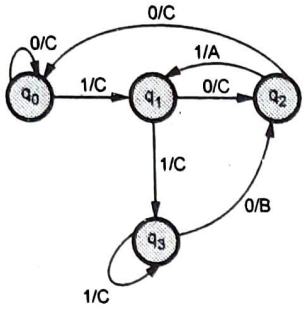


Fig. 1.10.6

Example 1.10.3 *Design a Mealy machine to find 2's complement of a given binary number.* **SPPU : Dec.-11, Marks 6**

Solution :

• Logic

For designing 2's complement of a binary number we assume that input is read from LSB to MSB. We will keep the binary number as it is until we read first 1. Keep that 1 as it is then change remaining 1's by 0's and 0's by 1's.

For example,

Let the binary number be

1011

←

read from LSB

Keep the first 1 from LSB as it is and toggle the remaining bits we will get

0101

Thus 2's complement of 1011 is 0101. The required Mealy machine will be -

• Design

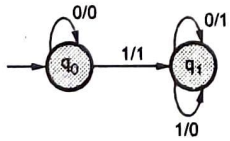


Fig. 1.10.7

Example 1.10.4 *Design a Moore machine which will increment the given binary number by 1.*

Solution :

• Logic

We will read the binary number form LSB one bit at a time. We will replace each 1 by 0 until we get first 0. Once we get first 0 we will replace it by 1 and then keep remaining bits as it is.

For example -

1	0	1	1	← Read from LSB bit by bit
1	0	1	0	
			↑	
1	0	0	0	
		↑		
1	1	0	0	
	↑			
1	1	0	0	
↑				

Using this logic we can built a mealy machine

• Design

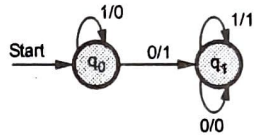


Fig. 1.10.8

Example 1.10.5 *Give Moore and Mealy machine for $\Sigma = \{0, 1, 2\}$, print the residue modulo 5 of input treated as a ternary number.*

Solution :

• Logic

The ternary number is made up of 0,1 and 2. The interpretation of ternary number n can be -

i) If we write 0 after n then number becomes 3n.

ii) If we write 1 after n then number becomes 3n+1.

iii) If we write 2 after n then number becomes 3n+2.

For example,

If n = 4 then its value is $4 \times 3^0 = 4$.

If we write 0 after 4 i.e.

$$40 = 4 \times 3^1 + 0 \times 3^0 = 12 \quad \text{i.e. } (3 \times 4)$$

If we write 1 after 4 then

$$41 = 4 \times 3^1 + 1 \times 3^0 = 13 \quad \text{i.e. } 3n+1$$

If we write 2 after 4 we get

$$42 = 4 \times 3^1 + 2 \times 3^0 = 14 \quad \text{i.e. } 3n+2$$

For residue modulo 5 we will get remainder 0, remainder 1, remainder 2, remainder 3 and remainder 4 values. Then we assume various states for these remainders as -

q_0 - remainder 0 state

q_1 - remainder 1 state

q_2 - remainder 2 state

q_3 - remainder 3 state

q_4 - remainder 4 state

Now consider n = 4,

For 4.0 we get decimal value 12 that means 12 % 5= 2 it gives remainder 2.

For 4.1 we get decimal value 13 that means 13 % 5= 3 it gives remainder 3.

Similarly 4.2 given remainder 4.

n = 4 itself is remainder 4. Hence we can design,

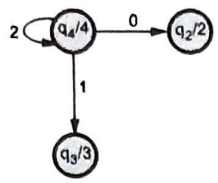


Fig. 1.10.9

• **Design**

Considering all possible cases we can design Moore machine as,

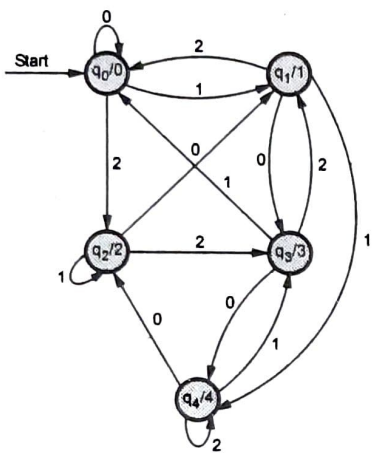


Fig. 1.10.10

Similarly the Mealy machine can be -

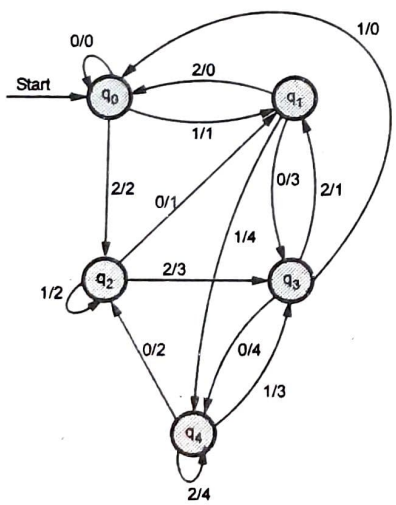


Fig. 1.10.11

Example 1.10.6 *Design a Moore machine that will read sequences made up of letters A, E, I, O, U and will give as output the same sequences except that in case where an I directly follows an E, it will be changed to U. Design the Mealy machine for the same.*

SPPU : May-13, Marks 8

Solution :

• **Logic**

We will assume a separate state for each alphabet. The output of that state will be corresponding letter. For instance

For alphabet A the state will be q_0 and output of q_0 will be A. For alphabet E the state will be q_1 and output of q_1 will be E. Continuing in this fashion we will have q_0, q_1, q_2, q_3 and q_4 states. The start state will be q_0 we will take care of one thing and that is when I comes immediately after E it will lead to the state q_4 because output of state q_4 is u. With this logic the corresponding Moore machine will be -

• **Design**

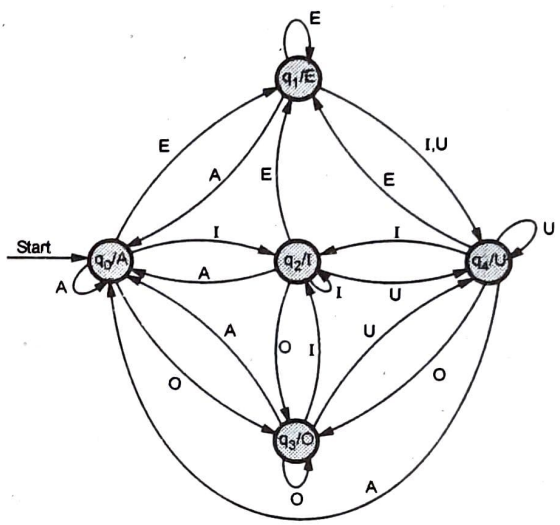


Fig. 1.10.12

Now we will draw the Mealy machine for the same problem.

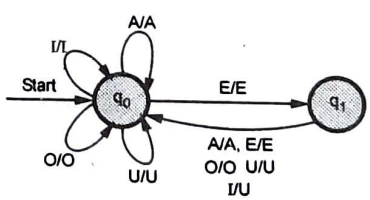


Fig. 1.10.13

Here the Mealy machine has the output along the edge itself we have got only two state. The transition changes from q_0 to q_1 when we get E. The I which is immediately following E will have the output U. Consider the string AIEOAEI will be give an output as AIEOAEU.

Example 1.10.7 *Construct a Moore machine to determine residue mod 3 for binary number.*

Solution :

• Logic

This Moore machine is also called remainder 3 tester. In this machine we will get remainder 0, remainder 1 and remainder 2. To interpret the given binary number in its decimal value we consider n as a number if 0 is written after n then its value becomes 2n. If 1 is written after n then it's value becomes 2n + 1. For instance, if n = 0 then its decimal value is 0 then,

$$01 = 2n+1 = 1\times0 + 1 = 1$$

$$011 = 2n+1 = (1\times2)+ 1 = 3$$

consider n = 1 its decimal value is 1. After 1 if 0 comes then its value will be

$$10 = 2n = 2 \times 1 = 2$$

If 1 comes after 10 then its value becomes,

$$101 = 2n+1 = (2 \times 2) + 1 = 5$$

• Design

With this logic we can construct a Moore machine with 3 states. q_0 is the start state and is considered as remainder 0 state. q_1 is considered to be remainder 1 state and q_2 is considered as remainder 2 state.

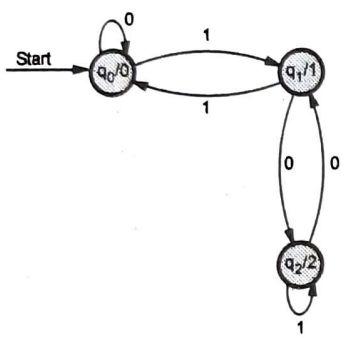


Fig. 1.10.14

Example 1.10.8 *Define Mealy machine. Construct a Mealy machine which can output EVEN/ODD if the total number of 1's in the input is even or odd. The input symbols are – 0 and 1.*

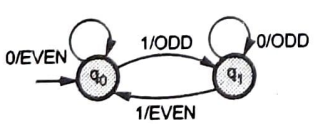
SPPU : Dec-13, Marks 5

Solution :

• Logic

The restriction is on number of 1's. There is no restriction on number of 0's. The Mealy machine will then be -

• Design



Example 1.10.9 *Give Mealy machine for the following processes "For input from (0+1)*, if input ends in 101, output X; if input ends in 110, output Y, otherwise output Z".* **SPPU : May-11, 14, Marks 8**

Solution :

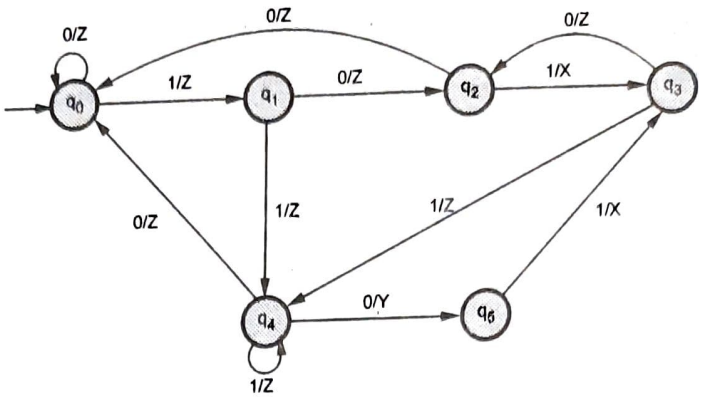


Fig. 1.10.15 Mealy machine

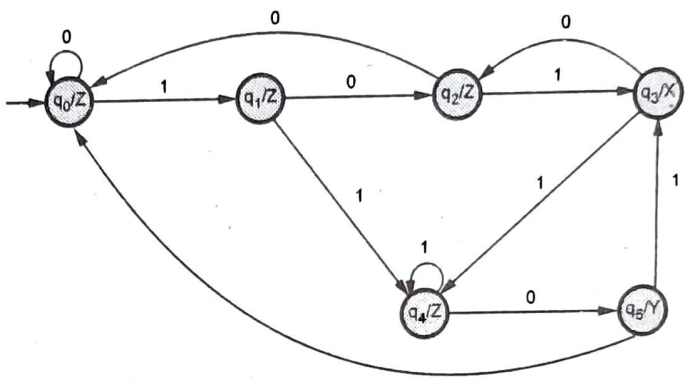
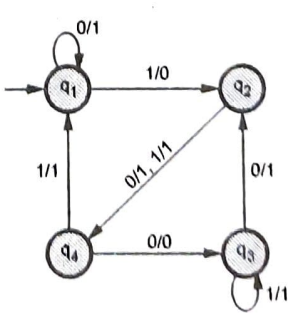


Fig. 1.10.16 Moore machine

Example 1.10.10 *Construct a Moore machine equivalent to the Melay machine M defined in the following table with input alphabet,* $\Sigma = \{0, 1\}$. **SPPU : Dec.-14, In Sem, Marks 6**

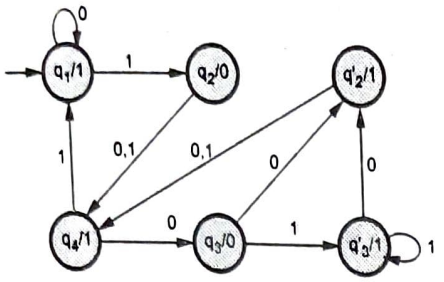
Present state	Next state			
	$\Sigma = 0$		$\Sigma = 1$	
	State	Output	State	Output
-> q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Solution : The Melay machine can be drawn as follows -



From this machine we can observe that the edges leading to q_1 and q_4 state have the output 1.

But the state q_3 has output 0 and 1. Similarly edges leading to state q_2 have output 0 and 1. Hence we will split q_2 as q_2 and q'_2. Similarly we will split q_3 as q_3 and q'_3. The Moore machine is as follows -



Example 1.10.11 *Design a Moore machine for computing the 2's complement of binary number. Convert it into its equivalent Mealy machine.* **SPPU : Oct. 2016, In sem. Marks 6**

Solution :

- **Logic Applied :** For designing 2's complement of a binary number, we assume that input is read from LSB to MSB. We will keep the binary number as it is until we read first 1. Keep that 1 as it is and then change remaining 1's by 0's and 0's by 1's.
- **Example :** Let the binary number be

1 0 1 1
←
Read from LSB

keep first 1 as it is and then toggle the bits. Hence we get
0101 ← This 2's complemented number.
The Moore machine will be –

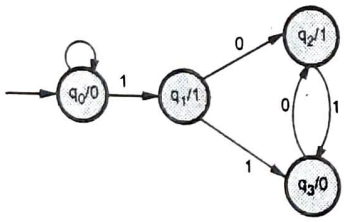


Fig. 1.10.17

- **Equivalent Mealy machine** – Refer ex. 1.10.3.

1.10.1 Inter-Conversions

1) **Method For conversion of Moore Machine to Mealy Machine**

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as -

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

2) **Method For conversion of Mealy Machine to Moore Machine**

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine then there exists a Moore machine M' equivalent to M.

The M' can be given as $M' = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where b_0 is an output symbol selected from Δ. We will calculate δ' and λ' as follows -

$$\delta'([q,b], a) = [\delta(q,a), \lambda(q,a)]$$

$$\lambda'([q,b]) = b$$

δ' defines the move made by M' on input a.

M' on input a.

λ' defines the output made for the corresponding state q.

Thus we have to calculate δ' and λ' for $q_0, q_1, q_2, \ldots\ldots q_n$ on input $a_0, a_1, \ldots\ldots a_n$ and should emit the outputs $b_1, b_2, b_3, \ldots\ldots b_n$.

Problems Based on Inter-Conversion

Example 1.10.12 *The Moore machine to determine residue mod 3 for binary number is given below. Convert it to Mealy equivalent machine.*

Q \ Σ	0	1	Output (λ)
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Solution : The transition diagram for the given problem can be drawn as -

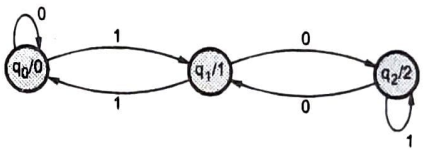


Fig. 1.10.18

The output function λ' can be obtained using following rule,

$$\lambda'(q,a) = \lambda(\delta(q,a))$$

Hence we will obtain output for every transition corresponding to input symbol.

$$\lambda'(q_0,0) = \lambda(\delta(q_0,0))$$
$$= \lambda(q_0) \quad \text{i.e. output of } q_0$$
$$\boxed{\lambda'(q_0,0) = 0}$$

$$\lambda'(q_0,1) = \lambda(\delta(q_0,1))$$
$$= \lambda(q_1) \quad \text{i.e. output of } q_1$$
$$\boxed{\lambda'(q_0,1) = 1}$$

$$\lambda'(q_1,0) = \lambda(\delta(q_1,0))$$
$$= \lambda(q_2)$$
$$\boxed{\lambda'(q_1,1) = 2}$$

$$\lambda(q_1,1) = \lambda(\delta(q_1,1))$$
$$= \lambda(q_0)$$
$$\boxed{\lambda'(q_1,1) = 0}$$

$$\lambda'(q_2,0) = \lambda(\delta(q_2,0))$$
$$= \lambda(q_1)$$

$$\boxed{\lambda'(q_2,0) = 1}$$

$$\lambda'(q_2,1) = \lambda(\delta(q_2,1))$$
$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_2,1) = 2}$$

Hence the transition table can be drawn as follows -

Q \ Σ	Input 0		Input 1	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

The transition diagram of Mealy machine is,

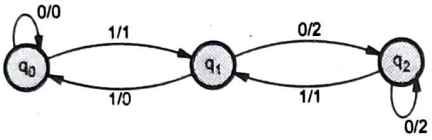


Fig. 1.10.19

The input string 10011 then the output for Mealy machine will be

next state : $q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0$

output : 1 2 2 1 0

The output sequence for Moore machine will be

Input : 1 0 0 1 1

next state : $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0$

output : 0 1 2 2 1 0

The length of output sequence is n+1 in Moore machine and is n in Mealy machine which is desired.

Example 1.10.13 *Convert the following Moore machine into equivalent Mealy machine* $M = (\{q_0,q_1\},\{a,b\},\{0,1\},\delta,\lambda,q_0)$

Solution :

δ	a	b	Output (λ)
q_0	q_0	q_1	0
q_1	q_0	q_1	1

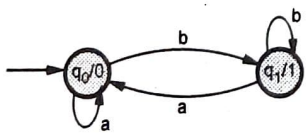


Fig. 1.10.20

The equivalent Mealy machine can be obtained as follows -

$$\lambda'(q_0,a) = \lambda(\delta(q_0,a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_0,b) = \lambda(\delta(q_0,b))$$
$$= \lambda(q_1)$$
$$= 1$$
$$\lambda'(q_1,a) = \lambda(\delta(q_1,a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_1,b) = \lambda(\delta(q_1,b))$$
$$= \lambda(q_1)$$
$$= 1$$

Hence the transition table can be drawn as follows -

Q \ Σ	Input a		Input b	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1

The equivalent Mealy machine will be,

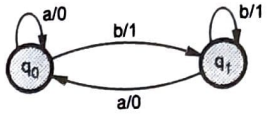


Fig. 1.10.21

Consider the output sequence for Moore machine for input sequence,

a b b a

Then

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0$$

0 0 1 1 0

Similarly output sequence for Mealy machine will be

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \longrightarrow q_0$$

0 1 1 1 0

We can note that length of Moore machine is n+1 and that of Mealy machine is n.

Example 1.10.14 *Consider the Moore machine described by the transition table given below. Construct corresponding Mealy machine.*

SPPU : Dec. 05, Marks 4, May-12, Marks 8

Present state	Next state		Output
	a = 0	a = 1	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Solution : The transition diagram for given Moore machine will be -

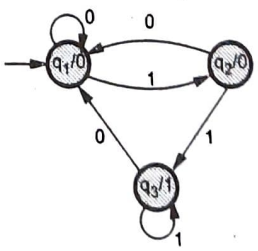


Fig. 1.10.22

The equivalent Mealy machine will be -

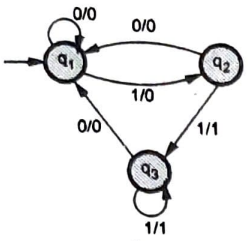


Fig. 1.10.23

Method for conversion of Mealy machine to Moore machine

May-06, 10.

In Moore machine the output is associated with every state and in Mealy machine the output is given along the edge with input symbol. To convert Moore machine to Mealy machine state output symbols are distributed to input symbol paths. But while converting Mealy machine to Moore machine we will create a separate state for every new output symbol and according incoming and outgoing edges are distributed.

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine then there exists a Moore machine M' equivalent to M.

The M' can be given as $M' = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where b_0 is an output symbol selected from Δ. We will calculate δ' and λ' as follows -

$$\delta'([q,b], a) = [\delta(q,a), \lambda(q,a)]$$

$$\lambda'([q,b]) = b$$

δ' defines the move made by M' on input a.

M' on input a.

λ' defines the output made for the corresponding state q.

Thus we have to calculate δ' and λ' for $q_0, q_1, q_2, \ldots\ldots q_n$ on input $a_0, a_1, \ldots\ldots a_n$ and should emit the outputs $b_1, b_2, b_3, \ldots\ldots b_n$.

Let us understand this method of conversion with the help of some examples.

Example 1.10.15 *Convert the following Mealy machine into equivalent Moore machine.*

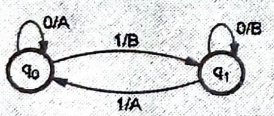


Fig. 1.10.24

Solution : The states for Moore machine will be $[q_0,A]$, $[q_0,B]$, $[q_1,A]$, $[q_1,B]$. Then we will calculate δ' and λ' as follows -

$$\delta'((q_0, A), 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

i.e [next state of q_0 on input 0, output of q_0 on input 0]

q_0 A

$$= [q_0, A]$$

$$\lambda'([q_0,A]) = A \rightarrow \text{i.e. output of state } q_0.$$

$$\delta'([q_1,A], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$

$$= [q_1,B]$$

$$\lambda'([q_1,A]) = A$$

$$\delta'([q_1,A], 1) = [\delta(q_1,1), \lambda(q_1,1)]$$

$$= [q_0,A]$$

$$\lambda'([q_1,A]) = A$$

Hence,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A

$$\delta'([q_1,B], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$

$$= [q_1,B]$$

$$\lambda'([q_1,B]) = B$$

$$\delta'((q_1,B), 1) = [\delta(q_1,1), \lambda(q_1,1)]$$

$$= [q_0,B]$$

$$\lambda'([q_1,B]) = B$$

Finally the transition table will be ,

$$\delta'((q_0,A), 1) = [\delta(q_0,1), \lambda(q_0,1)]$$

$$= [q_1,B]$$

$$\lambda'([q_0,A]) = A$$

Hence we can show a partial transition table as,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A

Continuing in this fashion we can calculate δ' and λ' as follows -

$$\delta'([q_0,B], 0) = [\delta(q_0,0), \lambda(q_0,0)]$$

$$= [q_0,A]$$

$$\lambda'[q_0,B]) = B$$

$$\delta'((q_1,B), 1) = [\delta(q_0,1), \lambda(q_0,1)]$$

$$= [q_1,B]$$

$$\lambda'([q_0,B]) = B$$

Hence

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B

State \ I/P	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A
$[q_1, B]$	$[q_1, B]$	$[q_0, A]$	B

The transition diagram will be,

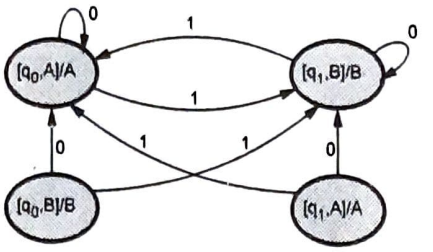


Fig. 1.10.25

Example 1.10.16 *Convert the following Mealy machine into equivalent Moore machine.*

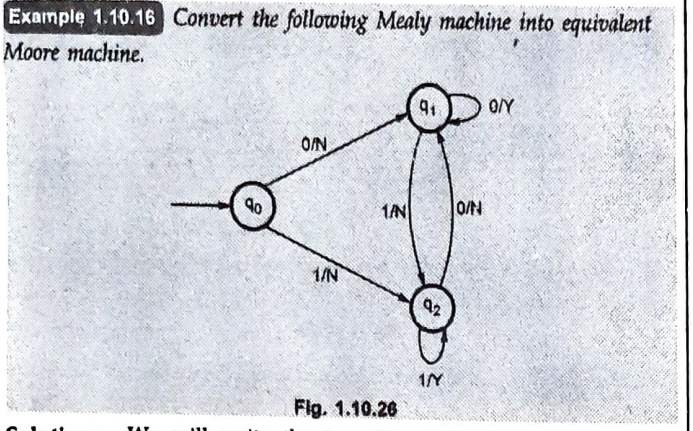


Fig. 1.10.26

Solution : We will write the transition table for given transition graph -

	Input 0	Output	Input 1	Output
q_0	q_1	N	q_2	N
q_1	q_1	Y	q_2	N
q_2	q_1	N	q_2	Y

Now we will find out the states and corresponding outputs for Moore machine. The states for Moore machine will be -

$[q_0,N]$, $[q_0,Y]$, $[q_1,N]$, $[q_1,Y]$, $[q_2,N]$, $[q_2,Y]$

Now let us calculate δ' and λ' for all the above given states -

$$\delta'([q_0,N)], 0) = [\delta(q_0,0), \lambda(q_0,0)]$$
$$= [q_1, N]$$
$$\lambda'([q_0,N]) = N$$

Similarly,

$$\delta'([q_0,N], 1) = [\delta(q_0,1), \lambda(q_0,1)]$$
$$= [q_2, N]$$
$$\lambda'([q_0,N]) = N$$

The partial transition table is -

State \ I/P	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N

Now, for remaining states the corresponding transitions and outputs can be obtained as follows -

$$\delta'([q_0,Y], 0) = [\delta(q_0,0), \lambda(q_0,0)]$$
$$= [q_1, N]N$$
$$\lambda([q_0,Y]) = Y$$
$$\delta'([q_0,Y], 1) = [\delta(q_0,1), \lambda(q_0,1)]$$
$$= [q_2 N]$$
$$\lambda'([q_0,Y]) = Y$$
$$\delta'([q_1N], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$
$$= [q_1 Y]$$
$$\lambda'([q_1,N]) = N$$
$$\delta'([q_1,N], 1) = [\delta(q_1,1), \lambda(q_1,1)]$$
$$= [q_2, N]$$
$$\lambda'([q_1,N]) = N$$
$$\delta'([q_1,Y], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$
$$= [q_1, Y]$$
$$\lambda'([q_1,Y]) = Y$$
$$\delta'([q_2,N], 0) = [\delta(q_2,0), \lambda(q_2,0)]$$
$$= [q_1, N]$$
$$\lambda'([q_2,N]) = N$$
$$\delta'([q_2,N], 1) = [\delta(q_2,1), \lambda(q_2,1)]$$
$$= [q_2, Y]$$
$$\lambda'([q_2,N]) = N$$
$$\delta'([q_1,Y], 1) = [\delta(q_1,1), \lambda(q_1,1)]$$
$$= [q_2, N]$$
$$\lambda'([q_1,Y]) = Y$$
$$\delta'([q_2,Y], 0) = [\delta(q_2,0), \lambda(q_2,0)]$$
$$= [q_1, N]$$
$$\lambda'([q_2,Y]) = Y$$
$$\delta'([q_2,Y], 1) = [\delta(q_2,1), \lambda(q_2,1)]$$
$$= [q_2, Y]$$
$$\lambda'([q_2,Y]) = Y$$

The transition table can be drawn as -

State \ Σ	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N
$[q_0, Y]$	$[q_1, N]$	$[q_2, N]$	Y
$[q_1, N]$	$[q_1, Y]$	$[q_2, N]$	N
$[q_1, Y]$	$[q_1, Y]$	$[q_2, N]$	Y
$[q_2, N]$	$[q_1, N]$	$[q_2, Y]$	N
$[q_2, Y]$	$[q_1, N]$	$[q_2, Y]$	Y

Example 1.10.17 *Construct a Moore machine equivalent to the Mealy machine M given below. δ is* **SPPU : Dec.-11, Marks 6**

	$a = 0$		$a = 1$	
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Solution : We will first design the transition diagram from given transition table.

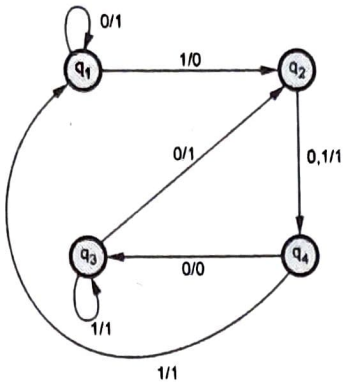


Fig. 1.10.27

The transition table for Moore machine will be

	0	1	Output
$[q_1]$	$[q_1]$	$[q_2]$	1
$[q_2]$	$[q_4]$	$[q_4]$	0 and 1
$[q_3]$	$[q_2]$	$[q_3]$	0 and 1
$[q_4]$	$[q_3]$	$[q_1]$	1

The Moore machine will be

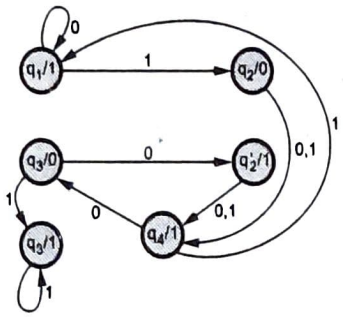


Fig. 1.10.28

Example 1.10.18 *Construct Mealy machine equivalent to Moore machine given as :* **SPPU : Dec.-12, Marks 6**

Present state	Next state		Output
	a = 0	a = 1	
$\rightarrow q_0$	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Solution : Equivalent Mealy machine can be obtained as follows :

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_1) = 0$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 1$$

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 1$$

$$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 1$$

$$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_1) = 0$$

$$\lambda'(q_3, 0) = \lambda(\delta(q_3, 0)) = \lambda(q_0) = 1$$

$$\lambda'(q_3, 1) = \lambda(\delta(q_3, 1)) = \lambda(q_3) = 1$$

Present state	Next state			
	a = 0	Output	a = 1	Output
$\rightarrow q_0$	q_1	0	q_2	1
q_1	q_3	1	q_2	1
q_2	q_2	1	q_1	0
q_3	q_0	1	q_3	1

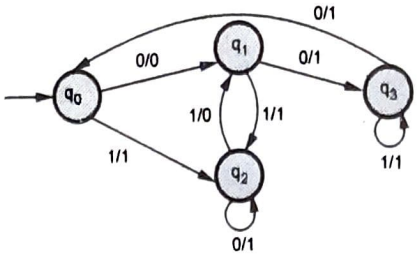


Fig. 1.10.29

Example 1.10.19 *Convert the following Mealy machine to Moore machine.* **SPPU : May-07, Marks 4**

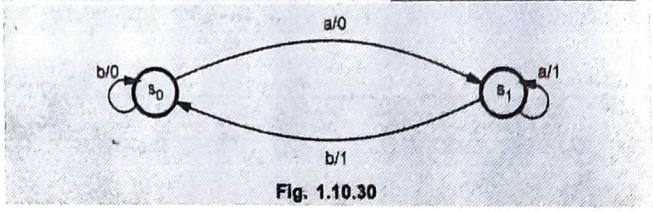


Fig. 1.10.30

Solution : If we observe state so then we notice that state so gets two incoming edges one for input b and output 0 and the other with input b and output 1. Similarly state S_1 gets two incoming edges with output 0 and output 1. Hence we split up the states as.

$S_0 \Rightarrow S_0$ and S_0' with output 0 and 1 respectively.

$S_1 \Rightarrow S_1$ and S_1' with output 0 and 1 respectively.

The Moore machine will then be -

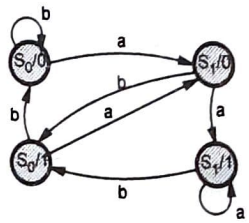


Fig. 1.10.31

Review Questions

1. *Define Moore and Mealy machine and how the equivalence of Moore and Mealy machine can be established with example.* **SPPU : May-14, Marks 6**
2. *Give formal definition for a Moore machine with a suitable example* **SPPU : Aug.-15, In-sem, Marks 2**

1.11 Case Study

FSM for Vending Machine

The states of coffee vending machine can be represented using finite State machine. Here is an illustrative vending machine with following assumptions

- When Rs. 15 are deposited then only release coffee.
- There will be single coin slot for Rs. 10 and Rs. 5
- No Change

The abstract representation of the machine will be

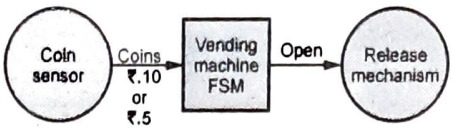


Fig. 1.11.1 Abstract model for vending machine

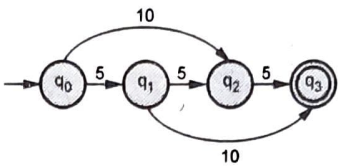
There are various cases that will lead to the state of release mechanism.

Case 1 : 3 coins of Rs. 5

Case 2 : 2 coins - firstly Rs. 5 coin and then Rs. 10 coin.

Case 3 : 2 coins - firstly Rs. 10 coin and then Rs. 5 coin.

The FSM can be as follows



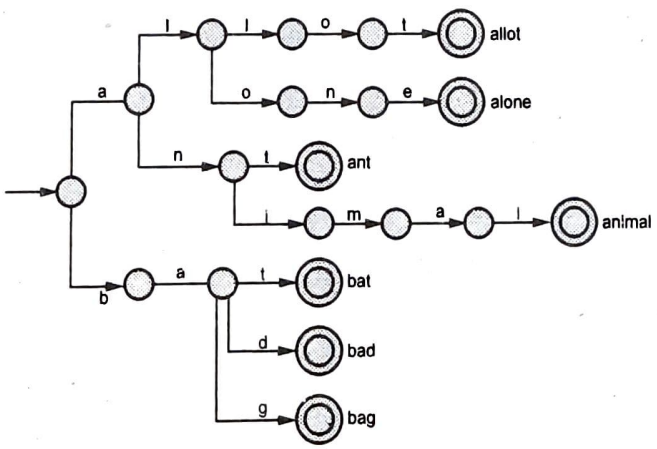
Her q_0 is the initial state with no coin inserted. The value on each of edge of FSM represents the value of coin. The state q_3 is a final state. When machine reaches to this state, the coffee will be released from the vending machine.

FSM in Spell Check

- Finite State machines can be used for simple spell checking.
- In this implementation, one or more words are provided by the user and those words would either be confirmed or rejected as being valid in that language.
- At the basic level,the finite state automata is generated for each of the valid string in the language.
- The common prefixes, or common suffixes are used to recognize the words. For example

The use of finite state machine improves the performance of pattern matching procedure.

All the final states in above FSM indicate the correct spellings and all the intermediate states represent incomplete or incorrect spelling of a particular word.



2
Regular Expressions (RE)

Syllabus

Operators of RE, Building RE, Precedence of operators, Algebraic laws for RE, *Conversions : NFA to DFA, RE to DFA Conversions : RE to DFA, DFA to RE Conversions : State/loop elimination, Arden's theorem Properties of Regular Languages : Pumping Lemma for Regular languages, Closure and Decision properties. Case Study: RE in text search and replace*

Contents

2.1 Operators of RE

SPPU : Dec.-16, Marks 6

Let Σ be an alphabet which is used to denote the input set. The regular expression over Σ can be defined as follows.

Definition

1. ϕ is a regular expression which denotes the **empty set.**
2. ε is a regular expression and denotes the set {ε} and it is a null string.
3. For each 'a' in Σ 'a' is a regular expression and denotes the set {a}.
4. If r and s are regular expressions denoting the languages L_1 and L_2 respectively, then

   r+s is equivalent to $L_1 \cup L_2$ i.e. union.

   rs is equivalent to $L_1 L_2$ i.e. concatenation.

   r^* is equivalent to L_1^* i.e. closure.

   The r^* is known as **kleen closure** or closure which indicates occurrence of r for ∞ number of times.

For example if Σ = {a} and we have regular expression R = a^*, then R is a set denoted by R = {ε, a, aa, aaa, aaaa, ...}

That is R includes any number of a's as well as empty string which indicates zero number of a's appearing, denoted by ε character.

Similarly there is a **positive closure** of L which can be shown as L^+. The L^+ denotes set of all the strings except the ε or null string. The null string can be denoted by ε or ^

If Σ = {a} and if we have regular expression R = a^+ then R is a set denoted by

$$R = \{a, aa, aaa, aaaa, ...\}$$

We can construct L* as

$$L^* = \varepsilon \cdot L^+$$

Example 2.1.1 *What is kleen closure ? What is positive closure ? For a given language L under what circumstances will L^+ and L^* be equal ?*

SPPU : Dec.-16, (End Sem), Marks 6

Solution :

- The Kleen closure is a set of strings of any length (including null string ε).
- The positive closure Σ^+ can be defined as $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3$... That means positive closure consists of all strings of any length except null string.
- Basically $L^+ \subseteq L^*$ But $L^+ = L^*$ if and only if $\varepsilon \in L^+$. The $\varepsilon \in L^+$ if and only if $\varepsilon \leftarrow L$. That mean ε is the shortest word in L.

2.2 Building RE

SPPU : Dec.-07,11,12,13,14,15 May-08,09, 10,11,13,14,16, Aug.-15,Oct.-16, Marks 6

Let us try to use regular expressions with the help of some examples.

Example 2.2.1 *Write the regular expression for the language accepting all combinations of a's over the set Σ = {a}.*

Solution : All combinations of a's means a may be single, double, tripple and so on. There may be the case that a is appearing for zero times, which means a null string. That is we expect the set of {ε, a, aa, aaa, ...}. So we can give regular expression for this as

$$R = a^*$$

That is kleen closure of a.

Example 2.2.2 *Design the regular expression (r.e.) for the language accepting all combinations of a's except the null string over Σ = {a}.*

Solution : The regular expression has to be built for the language

$$L = \{a, aa, aaa, ...\}$$

This set indicates that there is no null string. So we can denote r.e. as

$$R = a^+$$

As we know, positive closure indicates the set of strings without a null string.

Example 2.2.3 *Design regular expression for the language containing all the strings containing any number of a's and b's.*

Solution : The regular expression will be

$$\text{r.e.} = (a + b)^*$$

This will give the set as L = {ε, a, aa, ab, b, ba, bab, abab, ... any combination of a and b}.

The (a + b)* means any combination with a and b even a null string.

Example 2.2.4 *Construct the regular expression for the language containing all strings having any number of a's and b's, except the null string.*

Solution : r.e. = $(a + b)^+$

This regular expression will give the set of strings of any combination of a's and b's except a null string.

Example 2.2.5 *Construct the r.e. for the language accepting all the strings which are ending with 00 over the set Σ = {0, 1}.*

Solution : The r.e. has to be formed in which at the end, there should be 00. That means

$$\text{r.e.} = \text{(any combination of 0's and 1's) } 00$$

i.e. $$\text{r.e.} = (0+1)^* 00$$

Thus the valid string are 100, 0100, 1000, 10100 .... strings ending with 00.

Example 2.2.6 *Write r.e. for the language accepting the strings which are starting with 1 and ending with 0, over the set Σ = {0,1}.*

Solution : The first symbol in r.e. should be 1 and the last symbol should be 0.

So, $$R = 1(0 + 1)^* 0$$

Note that the condition is strictly followed by keeping starting and ending symbols correctly. In between them there can be any combination of 0 and 1 including a null string.

Example 2.2.7 *If L = {The language starting and ending with a and having any combination of b's in between }, then what is r ?*

Solution : The regular expression

$$r = a\, b^*\, a$$

Example 2.2.8 *Describe in simple English the language represented by the following regular expression r = (a + ab)**

Solution : We will first try to find out the set of strings, which can be possible by this r,

$$L(r) = \{a, aba, abab, aab, aaa, ...\}$$

The language is beginning with a but not having consecutive (in a clump) b's.

Example 2.2.9 *Write regular expression to denote the language L over Σ^*, where $\Sigma = \{a,b,c\}$ in which every string will be such that any number of a's is followed by any number of b's is followed by any number of c's.*

Solution : As we have seen any number of *a*'s means a^* any number of b's means b^* any number of c's means c^*. Since as given in problem statement, b's appear after *a*'s and c's appear after b's. So the regular expression could be -

$$r = a^*b^*c$$

Example 2.2.10 *Write a regular expression to denote a language L over Σ^*, where $\Sigma = \{a,b,c\}$ such that every string will have atleast one a followed by atleast one b followed by atleast one c.*

Solution : Now, in this problem the condition is slightly changed to "atleast". That means the null string is not allowed at all. So, we can write

$$R = a^+b^+c^+$$

Example 2.1.11 *Write r.e. to denote a language L which accepts all the strings which begin or end with either 00 or 11.*

Solution : The r.e. can be categorized into two subparts.

$$R = L_1 + L_2$$

L_1 = The strings which begin with 00 or 11.

L_2 = The strings which end with 00 or 11.

Let us find out L_1 and L_2.

L_1 = (00 + 11) (any number of 0's and 1's)

L_1 = (00 + 11) (0+1)*

Similarly,

L_2 = (any number of 0's and 1's) (00 + 11)

= (0 + 1)* (00 + 11)

Hence

$$R = [(00 + 11)(0 + 1)^*] + [(0 + 1)^*(00 + 11)]$$

Example 2.2.12 *Write a r.e. to denote a language L over Σ^*, where $\Sigma = \{a, b\}$ such that the 3rd character from right end of the string is always a.*

Solution : The r.e. contains the third symbol from right end as *a*

r = [any number of character of a's and b's] [a] [either a or b] [either a or b]

3rd 2nd 1st

$$r = (a+b)^* a (a+b) (a+b)$$

Thus the valid strings are b*a*bb or b*aaa* or b*aba* or *aa*b and so on.

Example 2.2.13 *Construct r.e. for the language L which accepts all the strings with atleast two b's over the $\Sigma = \{a,b\}$*

Solution : $R = (a + b)^* b (a + b)^* b (a + b)^*$

Atleast two b's are maintained but the two b's are surrounded by any combination of *a*'s and b's.

Example 2.2.14 *Construct r.e. for the language which consists of exactly two b's over the set $\Sigma = \{a,b\}$.*

Solution : Now, this problem is similar to the previous problem. But only difference is that there should be exactly two b's.

$$R = a^* b a^* b a^*$$

a^* indicates either the string contains any number of *a*'s or a null string. Thus we can derive any string having exactly two b's and any number of *a*'s.

Example 2.2.15 *Write r.e. which contains L having strings which should have atleast one 0 and atleast one 1.*

Solution : The required expression will be

$$R = [(0 + 1)^* 0 (0 + 1)^* 1(0 + 1)^*] + [(0 + 1)^* 1(0 + 1)^* 0(0 + 1)^*]$$

Example 2.2.16 *Construct r.e. which denotes a language L over the set $\Sigma = \{0\}$ having even length of string.*

Solution : Since $\Sigma = \{0\}$ there are strings in L of even length i.e.

L = {ε, 00, 0000, 000000, ...}.

So we can give the regular expression as

$$R = (00)^*$$

Thus kleen closure indicates the recursion of two zeros which will ultimately give the even length of the string.

Example 2.2.17 *Write r.e. which denotes a language L over the set $\Sigma = \{1\}$ having odd length of strings.*

Solution : R = 1 (11)*

The odd length strings are L = {1, 111, 11111, ...}. Note the difference between previous example and this one.

Example 2.2.18 *Describe the language denoted by following regular expression*

r.e. = $(b^*(aaa)^*b^*)^*$

Solution : The language can be predicted from the r.e. by finding out the meaning of it. We can split the r.e. as

r.e. = (any combination of b's) (*aaa*)* (any combination of b's).

L = {The language consists of the strings in which *a*'s appear trippled, there is no restriction on number of b's}

Example 2.2.19 *Construct regular expression for the language L over the set $\Sigma = \{a,b\}$ in which the total number of a's are divisible by 3.*

Solution : The total number of *a*'s are divisible by 3. So the valid strings can be

L = {b*aaa*, b*ab*a*ba*, *ab*a*bab*, ...}

The regular expression can be

$$r.e. = (b^* a b^* a b^* a b^*)^*$$

Example 2.2.20 *Write the r.e. to denote the language L over $\Sigma = \{a, b\}$ such that all the strings do not contain the substring "ab".*

Solution : The L = {ε, *a*, b, bb, *aa*, b*a*, b*aa* ...}

The r.e. = $(b^* a^*)$

In this regular expression if we substitute $a^* = \varepsilon$ we get all combination of b's and similarly if we substitute $b^* = \varepsilon$ then we will get all combinations of *a*'s. We have strictly maintained a condition for not to have *ab* as substring by giving the regular expression as $b^* a^*$.

Example 2.2.21 *Find all possible regular expression L ⊆ {a, b}* for*
a) The set of all strings ending in b
b) The set of all strings ending in ba
c) The set of all strings ending neither b nor ba.
SPPU : Dec.-07, Marks 6

Solution :

a) The regular expression for language containing strings ending with b is

r.e. = (a + b)* b.

b) The regular expression for language containing strings ending with ba is

r.e. = (a + b) * ba

c) The regular expression for language containing strings ending with neither b nor a is

r.e. = $(a + b)^* a^+ b^*$

Example 2.2.22 *Let L be a language. It is clear from the definition that $L^+ \subseteq L^*$. Under what circumstances are they equal ?*
SPPU : Dec.-07, May-08, Marks 4

Solution : The language L^+ denotes any combination of input symbols without ε.
The language L^* denotes any combination of input symbols along with ε.

Hence $L^+ \subseteq L^*$

But if $L_1 = \varepsilon + L^+$ then L_1 becomes equal to L^*. That means if ε is added as one symbol in the set generated by L^+ then L^* become equal.

Example 2.1.23 *Find all possible regular expression L ⊆ {a, b}* for*
a) The set of all strings ending in ab
b) The set of all strings ending in ba
c) The set of all stings ending neither ab nor ba.
SPPU : May-08, Marks 6

Solution :

a) The regular expression is

r.e. = (a + b)* ab

b) The regular expression is

r.e. = (a + b)* ba

c) The regular expression is

(a + b)* aa + (a + b)* bb

Example 2.1.24 *Consider the two regular expressions*
r = 0 + 1*s = 01* + 10* + 1*0 + (0*1)*.*
a) Find the string corresponding to r but not to s
b) Find the string corresponding to s but not to r
c) Find the string corresponding to both r and f.
SPPU : May-10, Marks 8

Solution :

a) The string corresponding to r but not to s : {00,000,0000, ...}

b) The string corresponding to s but not to r : {01 or 10}

c) The string corresponding to both r and f : {0, 1, 11, 111, 1111, 11111, ...}

Example 2.2.25 *Find a regular expression corresponding to each of the following subsets of {0, 1}**
1) The language of all strings containing exactly two 0's.
2) The language of all strings containing at least two 0's.
3) The expression of all strings that do not end with 01.
SPPU : May-09, Marks 6

Solution :

1) If a language contains all strings containing exactly two 0's then, it can be at any position, may or may not be surrounded by 1's. Hence r.e. Will be

r.e. = 1* 0 1* 01*

2) If a language contains all strings containing at least two 0's, then it may be at any position, and may or may not be surrounded by (0+1)*. Hence r.e. will be
r.e. = (0+1)* 0 (0+1)* 0 (0+1)*

3) The language contains all the strings that do not end with 01 means it may end with either 0's or end with either 1's or end with 10. Hence r.e. will be - r.e. = [(0+1)* 0*] + [(0+1)* (10)*]

Example 2.2.26 *Obtain the regular expressions for the following sets :*
1) $L1 = \{b^2, b^5, b^8, b^{11}, b^{14} ...\}$ 2) $L2 = \{a^{2n+1} | n > 0\}$
SPPU : May-11, Marks 4

Solution :

1) $L1 = \{b^2, b^5, b^8, b^{11}, b^{14} ...\}$

The regular expression for L1 = $\{b^2, b^5, b^8, b^{11}, ...\}$ is

Regular expression = bb (bbb)*

2) $L2 = \{a^{2n+1} | n > 0\}$

The regular expression for L2 = $\{a^{2n+1} | n > 0\}$ is

Regular expression = $a (aa)^+$

Example 2.2.27 *Obtain in plain english language represented by following regular expression.*
a) 0(10*10*)*1(0*10*1)*0* b) 0*(0*10*1)*0**
SPPU : May-11, Marks 4

Solution :

a) 0 * (10 * 10*) * 1 (0 * 10 * 1) * 0 * :

L = {The language containing group of even number of 1's separated by single 1}

b) 0 * (0 * 10 * 1) * 0 * :

L = {The language containing even number of 1's and any number of 0's}

Example 2.2.28 *Represent following formal languages using regular expressions.*
1) All string's of a's and b's without any combination of double letters. 2) All string's of 0's and 1's with even number of 0's. 3) All string's of a's and b's containing at least two a's. **SPPU : May-11, Marks 6**

Solution :

1) All string's of a's and b's without any combination of double letters : (01) * + (10) *

2) **All string's of 0's and 1's with even number of 0's :** $(1 * 01 * 01^*)^*$

3) **All string's of a's and b's containing at least two a's :** $(b^* a b^* a b^*)^+$

Example 2.2.29 *Find the regular expressions representing the following sets :*

i) The set of all strings over {a, b} with three consecutive b's.

ii) The set of all strings over {0, 1} beginning with 00.

iii) The set of all strings over {0, 1} ending with 00 and beginning with 1.

SPPU : Dec.-11, Marks 6

Solution :

i) r.e. = $a^*(bbb)^* a^*$

ii) r.e. = $00 (0 + 1)^*$

iii) r.e. = $1 (0 + 1)^* 00$

Example 2.2.30 *Find regular expressions representing the following sets :*

i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's. ii) The set of all strings over {a, b} in which the number of occurrences of a is divisible by iii) The set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a.

SPPU : Dec.-12, Marks 6

Solution :

i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.

$((ab) + b)^* a a ((ba) + b)^* / ((ba) + a)^* bb ((ab) + a)^*$

ii) The set of all strings over {a, b} in which the number of occurance of a is divisible by 3.

$\Rightarrow b^* (aaa)^* b^*$

iii) The set of all strings over {a, b}, in which these are at least two occurances of b between any two occurances of a.

$\Rightarrow (a + b)^* (aa) (bb)^+ (aa) (a + b)^*$

Example 2.2.31 *Write R.E. for the following*

i) Σ = (0, 1) odd number of 1's in strings

ii) Σ = (0, 1) triple 0 must never appear in string.

SPPU : May-14, Marks 6

Solution :

i) r.e. = $(0 * 10 *) (10 * 10 *)^*$

ii) r.e. = $(1 + 01)^* 00 (1 + 10)^*$

Example 2.2.32 *Give english description of the language of the following regular expression*

$(1 + \varepsilon) (00^*1)^* 0^*$

SPPU : Dec.-13, Marks 6

Solution : This expression generates the strings as -

{ ε, 1, 101, 1001, 10, 100, 1010, ...}

This regular expression gnerates all strings in which every 1 is separated by one or more 0. It also accepts the string which has only one 1 or the empty string.

Example 2.2.33 *Write regular expressions for the following languages over {0, 1}**

i) The set of strings that begin with 110

ii) The set of all strings not containing 101 as a substring.

SPPU : Dec.-13, Marks 6

Solution :

i) r.e. = $110 (0 + 1)^*$

ii) r.e. = $(0 + 11 * 00)^* (\varepsilon + 11^* (\varepsilon + 0))$

Example 2.2.34 *Write regular expressions for each of the following languages :*

i) For Σ = {a, b}, set of all strings with no consecutive 'a's and 'b's.

ii) For Σ = {0, 1}, set of all strings in which every 0 is immediately followed by at least two 1's.

iii) $L = \{1^{(2n+1)} \mid n \geq 1\}$

SPPU : Dec.-14, In Sem, Marks 6

Solution : i) $[(ab)^* + (ba)^*]$ ii) $[(011 + 1)^*]$ iii) $[1(11)^*]$

Example 2.2.35 *Describe the languages accepted by the following Regular Expressions and justify with suitable examples :*

i) $a^* \cdot (a + b)^* \cdot ab$ ii) $(1^*.0.1^*.0.1^*)^*$ iii) $a^* b + b^* a$

SPPU : Aug.-15, In Sem, Marks 6

Solution :

i) The language containing the strings that begins with single a and end with 'ab'.

ii) The language containing the strings having even number of 0's.

iii) The language containing strings that either begin with any number of a's and end with single b or begin with any number of b's and end with single a.

Example 2.2.36 *Write regular expressions for the following languages over the alphabet $\sum$ = {a, b} i) All strings that do not end with 'aa'.*

ii) All strings that contain an even number of 'b's.

iii) All strings which do not contain the substring 'ba'.

SPPU : Dec.-15, End Sem, Marks 6

Solution :

i) The language contains all the strings that do not end with aa means it may end with bb, ba or ab. Hence r. e. will be

$(a + b)^* ((ab)^* + (ba)^* + b^*)$

ii) r. e. = $(a^* b a^* b)^*$

iii) The L = { ε, a, b, bb, aa, ab, abb, ... }

r. e. = $(a^* b^*)$

Example 2.2.37 *Determine a regular expression over the alphabet {x, y} For the following :*

i) All strings containing exactly two x's.

ii) All strings that do not end with xy.

iii) All strings starting with yy.

SPPU : May-16, End Sem, Marks 6

Solution :

i) The regular expression is

$$r.e = y^* xy^* xy^*$$

ii) The strings do not end with xy means string can end with x or y or with yx.

The r.e. will then be –

$(x+y)^* yx + (x+y)^* x + y^*$

iii) The regular expression will be

$yy(x+y)^*$

Example 2.2.38 *Give the Regular Expression for the following languages :*

i) $L = \{x | x \in \{a,b\}^*$ and x contains exactly two a's$\}$

ii) L = {a, c, ab, cb, abb, cbb, abbb, cbbb, ...}

iii) $L = \{x |\ x \in (a, b)^*$ and x is any string that begins in "abb" or a$\}$

SPPU : Oct.-16, In Sem, Marks 6

Solution :

i) $b^* ab^* ab^*$

ii) The set contains all the string that either begin with a or c but always end with any number of b's.

$\therefore$ $r.e. = (a+c)b^*$

iii) $r.e. = (abb + a)(a+b)^*$

Review Question

1. *Define with example - Regular expression.*

SPPU : May-13, Marks 2

2.3 Precedence of Operators

There are three main operators that can be used in RE - **Union, concatenation** and **kleen closure**. For regular expressions, the following is the order of precedence for the operators :

i) The **star operator (kleen closure)** is having highest precedence. That means the r.e. with star operator must be grouped first.

ii) The next precedence is for **dot operator** i.e. for concatenation. The concatenation is an **associative** operator, hence the way we group the elements of regular expression does not matter much. For example 0110 can be grouped as (01)(10), or 0(110) or (011)0

iii) Final precedence is given to + **(i.e. Union) operation.** The union or + operator is also associative operator.

2.4 Algebraic Laws for RE

The two regular expressions P and Q are equivalent (denoted as P = Q) if and only if P represents the same set of strings as Q does. For showing this equivalence of regular expressions we need to show some identities of regular expressions.

Let P, Q and R are regular expressions then the identity rules are as given below.

1.	$\varepsilon R = R\varepsilon = R$	
2.	$\varepsilon^* = \varepsilon$	ε is null string
3.	$(\phi)^* = \varepsilon$	ϕ is empty string.
4.	$\phi R = R\phi = \phi$	
5.	$\phi + R = R$	
6.	$R + R = R$	
7.	$RR^* = R^*R = R^+$	
8.	$(R^*)^* = R^*$	
9.	$\varepsilon + RR^* = R^*$	
10.	$(P + Q)R = PR + QR$	
11.	$(P + Q)^* = (P^* Q^*) = (P^* + Q^*)^*$	
12.	$R^*(\varepsilon + R) = (\varepsilon + R)R^* = R^*$	
13.	$(R + \varepsilon)^* = R^*$	
14.	$\varepsilon + R^* = R^*$	
15.	$(PQ)^* P = P(QP)^*$	
16.	$R^* R + R = R^* R$	

2.5 Equivalence of Regular Expressions

SPPU : Dec.-08,09,11,Oct.-16, May-06,07,09,11, Oct.-16, Marks 10

Let us see one important theorem named **Arden's Theorem** which helps in checking the equivalence of two regular expressions.

Arden's Theorem : Let, P and Q be the two regular expressions over the input set Σ. The regular expression R is given as

$$R = Q + RP$$

Which has a unique solution as $R = QP^*$.

Proof : Let, P and Q are two regular expressions over the input string Σ.

If P does not contain ε then there exists R such that

$$R = Q + RP \quad \ldots (1)$$

We will replace R by QP^* in equation (1)

Consider R.H.S. of equation (1)

$$= Q + QP^* P$$

$$= Q(\varepsilon + P^* P)$$

$$= QP^*$$

$\because$ $\varepsilon + R^* R = R^*$

Thus $\boxed{R = QP^*}$

is proved. To prove that $R = QP^*$ is a unique solution, we will now replace L.H.S. of equation 1 by Q + RP. Then it becomes

$$Q + RP$$

But again R can be replaced by Q + RP.

$\therefore$ $Q + RP = Q + (Q + RP)\,P$

$= Q + QP + RP^2$

Again replace R by Q + RP.

$= Q + QP + (Q + RP)\,P^2$

$= Q + QP + QP^2 + RP^3$

Thus if we go on replacing R by Q + RP then we get,

$Q + RP = Q + QP + QP^2 + ... + QP^i + RP^{i+1}$

$= Q(\varepsilon + P + P^2 + ... P^i) + RP^{i+1}$

From equation (1),

$R = Q(\varepsilon + P + P^2 + ... + P^i) + RP^{i+1}$ ... (2)

Where $i \geq 0$

Consider equation (2),

$$R = Q\underbrace{(\varepsilon + P + P^2 + ... + P^i}_{P^*} + RP^{i+1}$$

$\therefore$ $R = QP^* + RP^{i+1}$

Let w be a string of length i.

In RP^{i+1} has no string of less than i + 1 length. Hence w is not in set RP^{i+1}. Hence R and QP^* represent the same set. Hence it is proved that

$R = Q + RP$ has a unique solution.

$R = QP^*$.

Problems for Equivalence of Two

Example 2.5.1 *Prove*

$(1 + 00*1) + (1 + 00*1)(0 + 10*1)*(0 + 10*1) = 0*1(0 + 10*1)*$

SPPU : Dec.-11, Marks 4

Solution : Let us solve L.H.S. first,

$(1 + 00*1) + (1 + 00*1)(0 + 10*1)*(0 + 10*1)$

We will take $(1 + 00*1)$ as a common factor

$$1 + 00*1 \quad \underbrace{(\varepsilon + (0 + 10*1)*(0 + 10*1))}_{\downarrow}$$

$(\varepsilon + R*R)$ where $R = (0 + 10*1)$

As we know, $(\varepsilon + R*R) = (\varepsilon + RR*) = R*$

$\therefore$ $(1 + 00*1)((0 + 10*1)*)$ out of this consider

$$\underbrace{(1 + 00*1)}_{\downarrow}(0 + 10*1)*$$

Taking 1 as a common factor

$(\varepsilon + 00*)\,1\,(0 + 10*1)*$

Applying $\varepsilon + 00* = 0*$

$0*1(0 + 10*1)* =$ R.H.S.

Hence the two regular expressions are equivalent.

Example 2.5.2 *Show that* $(0*1*)* = (0 + 1)*$.

Solution :

Consider L.H.S. $= (0*1*)*$

$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10, ...\}$

$=$ {any combination of 0's, any combination of 1's, any combination of 0 and 1, ε}

Similarly,

R.H.S. $= (0 + 1)*$

$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10, ...\}$

$=$ {ε, any combination of 0's, any combination of 1's, any combination of 0 and 1}

Hence, L.H.S. = R.H.S. is proved.

Example 2.5.3 *Show that* $(ab)* \neq (a*b*)$

SPPU : Oct.-16, In Sem, Marks 2

Solution : Consider

L.H.S. $= (ab)* = \{\varepsilon, ab, abab, ababab, ...\}$

and R.H.S. $= (a*\,b*) = \{\varepsilon, a, aa, aa, b, bb, bbb, ab ...\}$

Note that in the L.H.S, the r.e. will give the strings of '*ab*' combination i.e. *ab*, *abab*, *ababab* and so on. But in R.H.S. there is possibility of *ab* but not of *abab* and so on.

Thus L.H.S. $\neq$ R.H.S. is proved.

Example 2.5.4 *Show that* $(r + s)* \neq r* + s*$

Solution : Let us consider the L.H.S. first,

L.H.S. $= (r + s)*$

$= \{\varepsilon, r, rr, s, ss, rs, sr, rsrs, ...\}$

$=$ {ε , any combination of r and s}

R.H.S. $= \{\varepsilon, r, rr, rrr, s, ss, ...\}$

{ε, any combination of only r or any combination of only s}

Note that in R.H.S. there is no combination of r and s together. Hence

L.H.S. $\neq$ R.H.S. is proved.

Example 2.5.5 *Prove that* $r(s + t) = rs + rt$

Solution : In case of L.H.S.

L.H.S. $= r\,(s + t)$ solving this further

$= r \cdot s + r \cdot t$

i.e. r is concatenated with s or with t

$=$ R.H.S.

L.H.S. $=$ R.H.S. is proved.

Example 2.5.6 *Check for the following regular expressions for equivalence and justify.*

i) $R_1 = [(a + bb)^*\,(b + aa)^*]^*$

$R_2 = (a + b)^*$

Solution : Here R_1 and R_2 are equivalent because the strings which can be generated by R_1 can be generated by R_2 or vice versa.

For instance, consider,

R_1 = *abab* – this string can be generated by selecting **a** from $(a + bb)^*$ then **b** from $(b + aa)^*$ again **a** from $(a + bb)^*$ and **b** from $(b + aa)^*$.

R_2 = *abab* – this sting can be selecting **a** then **b** repetitively from $(a + b)^*$

Thus $R_1 = R_2$ is proved

Example 2.5.7 $R_1 = (a + b)^*\,abab^*$

$R_2 = b*a\,(a + b)^*\,ab^*$

Solution : Here strings generated by R_1 are same as R_2
For instance R_1 can generate **ababab**. Similarly R_2 can generate abababb.
Thus $R_1 = R_2$ is proved.

Example 2.5.8 *Prove* $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$

Solution : Let, L.H.S. is
$\varepsilon + \underbrace{1^*(011)^*}\ \underbrace{(1^*(011)^*)^*}$

If we consider $1^*(011)^*$ as P_1 then,

$= \varepsilon + P_1 P_1^*$

$= P_1^*$

$\because \varepsilon + P_1P_1^* = P_1^*$

We can put $P_1 = 1^*(011)^*$ then,

$= (1^*(011)^*)^*$

Now consider $P_2 = 1$ and $P_3 = (011)$ then it becomes

$= (P_2^* P_3^*)^*$

$= (P_2 + P_3)^*$

$\because (P^* Q^*)^* = (P + Q)^*$

$= (1 + 011)^*$

$=$ R.H.S.

Thus L.H.S. = R.H.S. Hence
$\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$ is proved.

Example 2.5.9 *Prove or disprove the following for regular expression r, s and t.*
a) $(rs+r)^*r = r(sr+r)^*$ *b)* $s(rs+s)^*r = rr^*s(rr^*s)^*$
c) $(r+s)^* = r^* + s^*$ **SPPU : May-07, Marks 10**

Solution :

a) $(rs + r)^*r = r(sr + r)^*$

Let $r_1 = (rs + r)^*r$

$= \{\varepsilon, rs, r, rsrs, rsr, rsr, rr...\}\ r$

$= \{\varepsilon, rsr, rr, rsrsr, rsrr, rsrr, rrr ....\}$

$r_2 = r(sr + r)^*$

$= r\{\varepsilon, sr, r, sr\ sr, srr, rsr, rr ...\}$

$= \{\varepsilon, rsr, rr, srsrr, r ...\}$

Here L.H.S. = R.H.S.
Hence $(rs + r)^*r = r(sr + r)^*$

b) $s(rs + s)^*r = rr^*s(rr^*s)^*$

Let $r_1 = s(rs + s)^*r$

$= s\{\varepsilon, rs, s, rss, srs, rsrs, ss ...\}\ r$

$= \{\varepsilon, rss, ss, rsss, srss, rsrss, sss ...\}\ r$

$= \{\varepsilon, rrss, ssr, rrsss, srssr, rsrssr, sssr\}$

Let $r_2 = rr^*s(rr^*s)^*$

$r\{\varepsilon, r, rr, rrr, rrrr .....\}\ s$

$(r\{\varepsilon, r, rr, rrr, rrrr .....\}\ s)^*$

$rs\{\varepsilon, r, rr, rrr, rrrr .....\}$

$\{rs\} \cup \{\varepsilon, r, rr, rrr, rrrr .....\}$

$\{\varepsilon, rrs, rrrs, rrrrs, rrrrrs .....\}$

$\{\varepsilon, rrs, rrrs, rrrs, rrrrrs\}$

$= \{\varepsilon, rrss, ssr, rrsss .....\}$

Here L.H.S. = R.H.S.
Hence $s(rs + s)^*r = rr^*s(rr^*s)^*$

c) $(r + s)^* = r^* + s^*$

Let $r_1 = (r + s)^*$

$= (\varepsilon, r, s, rs, sr, rsr, srs .....)$

Let $r_2 = r^* + s^*$

$= (\varepsilon, r, rr, rrr, rrr....) \cup (\varepsilon, s, ss, sss, ssss)$

$= (\varepsilon, r, s, s, rs, sr, rsr, srs ....)$

Here L.H.S. = R.H.S.
Hence $(r + s)^* = r^* + s^*$

Example 2.5.10 *Prove*
a) $\phi^* = \varepsilon$ *b)* $(r^*s^*)^* = (r + s)^*$
SPPU : Dec.-08, Marks 4. Oct.-16, (In Sem), Marks 2

Solution :

a) Let L.H.S. $= \phi^*$

$= \{\varepsilon, \phi, \phi\phi, \phi\phi\phi, ...\}$

$\because R^* = \{\varepsilon, R, RR, RRR, ...\}$

$= \{\varepsilon\}$ or empty set

$\because \phi R = R\phi = \phi$ i.e. empty set.

$= \varepsilon$

$=$ R.H.S.

As, L.H.S. = R.H.S. , $\phi^* = \varepsilon$ is proved.

b) Let, $(r^*s^*)^*$

L.H.S. $= \{\varepsilon, r, rr, rs, sr, rrs, srr, srs, ...\}$

$=$ ε + any combination of r and s

R.H.S. $= \{\varepsilon, r, rr, rs, sr, rsr, rrs, srr, ...\}$

$=$ ε + any combination of r and s

As, L.H.S. = R.H.S.
$(r^*s^*)^* = (r + s)^*$ is true.

Example 2.5.11 *Explain your answer in each of the following :*
1) Every subset of a regular language is regular
2) Every regular language has a regular proper subset.
SPPU : May-06, Dec.-08, Marks 6

Solution :

1) Every subset of regular language need not be regular. To support if consider a language.

$L_1 = 0^*$ is regular but

$L_2 = \{0^{2n} | n \geq 1\}$ is not a regular language.

where $L_2 \subseteq L_1$ Hence given statement is **false.**

2) Every regular language has a regular proper subset. To support this statement, consider a language.

$L_1 = \phi$ it has no proper subset.

L_1 = {ε} then $L_2 = \phi$ is a proper subset of L_1

L_1 = Language of any substring, has any substring which forms its proper subset.

Hence given statement is **true.**

Example 2.5.12 *If S = {aa, b} write all the string in S* which are having length 4 or less, also say the following is True or False.*
i) $(S^+)^* = (S^*)^*$ ii) $(S^+)^+ = S^+$ iii) $(S^*)^+ = (S^+)^*$

SPPU : May-09, Marks 4

Solution : S = {aa, b}

1) String having length 0 = { ε }

2) Strings having length 1 = { b }

3) Strings having length 2 = { aa, bb }

4) Strings having length 3 = { aab, bbb, baa }

5) Strings having length 4 = { aaaa, aabb, bbbb, bbaa, baab }

Thus we obtain the string with length 4 or less by combining all above obtained sets, { ε, b, aa, bb, aab, bbb, baa, aaaa, aabb, bbbb, bbaa, baab }

i) $(S^+)^* = (S^*)^*$

L.H.S. = {aa, b, aab, bb, bbb, ...}*

= { ε + any combination of aa and b }

R.H.S. = { ε, aa, b, aab, bb, bbb, ...}*

= { ε + any combination of aa and b}

As L.H.S. = R.H.S. given statement is **true.**

ii) $(S^+)^+ = S^+$

L.H.S. = {aa, b, aab, bb, bbb, ...}+

= any combination of aa and b without ε.

R.H.S. = {aa, b, aab, bb, bbb, ...}

= any combination of aa and b without ε.

As L.H.S. = R.H.S. given statement is **true.**

iii) $(S^*)^+ = (S^+)^*$

L.H.S. = { ε + any combination of aa and b }+

R.H.S. = { any combination of aa and b without ε } + ε

L.H.S. = R.H.S.

Hence given statement is **true.**

Example 2.5.13 *Give the examples of sets that demonstrate the following inequality. Here r_1, r_2, r_3 are regular expressions :*
1) $r_1 + \varepsilon \neq r_1$ 2) $r_1 \cdot r_2 \neq r_2 \cdot r_1$
3) $r_1 \cdot r_1 \neq r_1$ 4) $r_1 + (r_2 \cdot r_3) \neq (r_1 + r_2) \cdot (r_1 r_3)$.

SPPU : Dec.-09, Marks 10

Solution : Assume $r_1 = a$, $r_2 = b$, $r_3 = ab$

1) L.H.S. = $r_1 + \varepsilon$ = { a, ε }

R.H.S. = r_1 = {a}

∴ L.H.S. ≠ R.H.S.

2) L.H.S. = $r_1 r_2$ = ab

R.H.S. = $r_2 r_1$ = ba

∴ L.H.S. ≠ R.H.S.

3) L.H.S. = $r_1 r_1$ = aa

R.H.S. = r_1 = a

∴ L.H.S. ≠ R.H.S.

4) L.H.S. = $r_1 + (r_2 r_3)$

= a + (b · ab)

= a + bab = {a, bab}

R.H.S. = $(r_1 + r_2) \cdot (r_1 r_3)$

= (a + b) (a · ab)

= {aaab, baab}

∴ L.H.S. ≠ R.H.S.

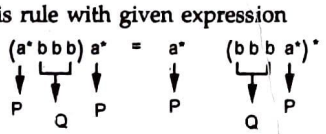
Example 2.5.14 *Check the equivalence of the regular expressions.*
1) $(a^* bbb)^* a^*$ AND $a^*(bbb\,a^*)^*$
2) $((a + bb)^* aa)^*$ AND $\varepsilon + (a + bb)^*.aa$ SPPU : May-11, Marks 4

Solution :

1) $(a^* bbb)^* a^*$ AND $a^*(bbb\,a^*)^*$

By identity rule $(PQ)^* P = P(QP)^*$

Let us map this rule with given expression



Hence given expressions are equivalent

2) $((a + bb)^* aa)^*$ AND $\varepsilon + (a + bb)^*.aa$

L.H.S. = ((a + bb) * aa) *

= {ε, aaa, bbaa, aaaa, bbaaa, abbaa, ...}

R.H.S. = ε + (a + bb) * aa

= {ε, aaa, bbaa, aaaa, bbaaa, abbaa, ...}

L.H.S. = R.H.S.

Hence given expressions are equivalent.

2.6 NFA with ε to NFA without ε

SPPU : Dec.-14, 15, Oct-16, Marks 6

Steps for Conversion

1. Find out all the ε transitions from each state from Q. That will be called as ε-**closure** $\{q_1\}$ where $q_1 \in Q$.
2. Then δ′ transitions can be obtained. The δ′ transitions means an ε-closure on δ moves.
3. Step - 2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ε can be built.

$$\delta'(q, a) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q,\varepsilon), a\right)\right)$$

where $\hat{\delta}(q,\varepsilon) = \varepsilon\text{ - closure}(q)$

Example 2.6.1 *Convert the given NFA with ε to NFA without ε.*

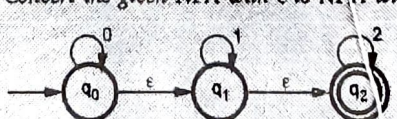


Fig. 2.6.1

SPPU : Dec.-14, Oct-16, In Sem, Marks 4
Dec.-15, End Sem, Marks 6

Solution : We will first obtain ε - closure of each state i.e. we will find out ε - reachable states from current state.
Hence

$$\varepsilon\text{ - closure }(q_0) = \{q_0, q_1, q_2\}$$
$$\varepsilon\text{ - closure }(q_1) = \{q_1, q_2\}$$
$$\varepsilon\text{ - closure }(q_2) = \{q_2\}$$

As ε - closure (q_0) means with null input (no input symbol) we can reach to q_0, q_1 or q_2. In a similar manner for q_1 and q_2 ε - closures are obtained. Now we will obtain δ′ transitions for each state on each input symbol.

$$\delta'(q_0, 0) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 0\right)\right)$$
$$= \varepsilon\text{ - closure }\left(\delta(\varepsilon\text{ - closure}(q_0), 0)\right)$$
$$= \varepsilon\text{ - closure }(\delta(q_0, q_1, q_2), 0)$$
$$= \varepsilon\text{ - closure }(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$$
$$= \varepsilon\text{ - closure }(q_0 \cup \phi \cup \phi)$$
$$= \varepsilon\text{ - closure }(q_0) = \{q_0, q_1, q_2\}$$
$$\delta'(q_0, 1) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 1\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(q_0, q_1, q_2), 1)$$
$$= \varepsilon\text{ - closure }(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$$
$$= \varepsilon\text{ - closure }(\phi \cup q_1 \cup \phi)$$
$$= \varepsilon\text{ - closure }(q_1)$$
$$\delta'(q_0, 1) = \{q_1, q_2\}$$
$$\delta'(q_1, 0) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 0\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_1), 0))$$
$$= \varepsilon\text{ - closure }(\delta(q_1, q_2), 0)$$
$$= \varepsilon\text{ - closure }(\delta(q_1, 0) \cup \delta(q_2, 0))$$
$$= \varepsilon\text{ - closure }(\phi \cup \phi)$$
$$= \varepsilon\text{ - closure }(\phi)$$
$$= \phi$$
$$\delta'(q_1, 1) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 1\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_1), 1))$$
$$= \varepsilon\text{ - closure }(\delta(q_1, q_2), 1)$$
$$= \varepsilon\text{ - closure }(\delta(q_1, 1) \cup \delta(q_2, 1))$$
$$= \varepsilon\text{ - closure }(q_1 \cup \phi)$$
$$= \varepsilon\text{ - closure }(q_1)$$
$$= \{q_1, q_2\}$$
$$\delta'(q_2, 0) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 0\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_2), 0))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 0))$$
$$= \varepsilon\text{ - closure }(\phi)$$
$$= \phi$$
$$\delta'(q_2, 1) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 1\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_2), 1))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 1))$$
$$= \varepsilon\text{ - closure }(\phi)$$
$$\delta'(q_2, 1) = \phi$$
$$\delta'(q_0, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_0), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_0, q_1, q_2), 2)$$
$$= \varepsilon\text{ - closure } (\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(\phi \cup \phi \cup q_2)$$
$$= \varepsilon\text{ - closure }(q_2)$$
$$= \{q_2\}$$
$$\delta'(q_1, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_1), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_1, q_2), 2)$$
$$= \varepsilon\text{ - closure }(\delta(q_1, 2) \cup \delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(\phi \cup q_2)$$
$$= \{q_2\}$$
$$\delta'(q_2, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_2), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(q_2)$$
$$= \{q_2\}$$

Now we will summarize all the computed δ′ transitions -

$\delta'(q_0, 0) = \{q_0, q_1, q_2\}$,
$\delta'(q_0, 1) = \{q_1, q_2\}$,
$\delta'(q_0, 2) = \{q_2\}$
$\delta'(q_1, 0) = \phi$,
$\delta'(q_1, 1) = \{q_1, q_2\}$,
$\delta'(q_1, 2) = \{q_2\}$
$\delta'(q_2, 0) = \phi$,
$\delta'(q_2, 1) = \phi$,
$\delta'(q_2, 2) = \{q_2\}$

From this we can write the transition table as -

State \ Input	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

The NFA will be as shown in Fig. 2.6.2.

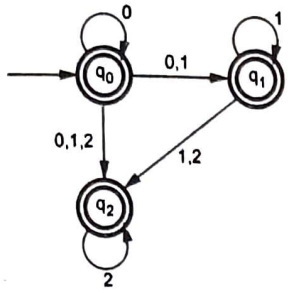


Fig. 2.6.2

Here q_0, q_1 and q_2 is a final state because ε - closure (q_0), ε - closure (q_1) and ε - closure (q_2) contains final state q_2.

Example 2.6.2 *Convert the following NFA with ε to NFA without ε.*

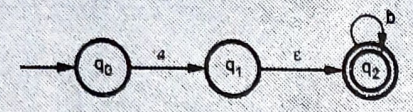


Fig. 2.6.3

Solution : We will first obtain ε - closures of q_0, q_1 and q_2 as follows.

ε - closure $(q_0) = \{q_0\}$

ε - closure $(q_1) = \{q_1, q_2\}$

ε - closure $(q_2) = \{q_2\}$

Now the δ' transitions on each input symbol is obtained as

$$\delta'(q_0,a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_0,\varepsilon),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_0),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_0,a)\right)$$
$$= \varepsilon\text{-closure}(q_1)$$
$$= \{q_1, q_2\}$$

$$\delta'(q_0,b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_0,\varepsilon),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_0),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_0,b)\right)$$
$$= \phi$$

$$\delta'(q_1,a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_1,\varepsilon),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_1),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_1,q_2),a\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_1,a)\cup\delta(q_2,a)\right)$$
$$= \varepsilon\text{-closure}(\phi\cup\phi)$$
$$= \phi$$

$$\delta'(q_1,b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_1,\varepsilon),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_1),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_1,q_2),b\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_1,b)\cup\delta(q_2,b)\right)$$
$$= \varepsilon\text{-closure}(\phi\cup q_2)$$
$$= \varepsilon\text{-closure}(q_2)$$
$$= \{q_2\}$$

$$\delta'(q_2,a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_2,\varepsilon),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_2),a\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_2,a)\right)$$
$$= \varepsilon\text{-closure}(\phi)$$
$$= \phi$$

$$\delta'(q_2,b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_2,\varepsilon),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_2),b\right)\right)$$
$$= \varepsilon\text{-closure}\left(\delta(q_2,b)\right)$$
$$= \varepsilon\text{-closure}(q_2)$$
$$= \{q_2\}$$

The transition table can be -

State	Input: a	Input: b
→ q_0	$\{q_1, q_2\}$	ϕ
q_1	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_2\}$

States q_1 and q_2 becomes the final as ε - closures of q_1 and q_2 contains the final state q_2. The NFA can be shown by following transition diagram -

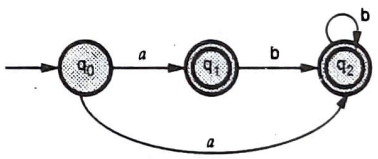


Fig. 2.6.4

2.7 NFA to DFA Conversion

SPPU : Dec.-05, 09.11.13.May-09.10.12.Aug.-15.Marks 10

2.7.1 NFA without ε to DFA Conversion

We will discuss the method of converting NFA to its equivalent DFA.

Let, $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA which accepts the language L(M). There should be equivalent DFA denoted by $M' = (Q', \Sigma', \delta', q_0', F')$ such that L(M) = L(M').

The conversion method will follow following steps -

1) The start state of NFA M will be the start for DFA M'. Hence add q_0 of NFA (start state) to Q'. Then find the transitions from this start state.

2) For each state $[q_1, q_2, q_3, \ldots\ldots q_i]$ in Q' the transitions for each input symbol Σ can be obtained as,

   i) $\delta'([q_1, q_2, \ldots\ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots\ldots \delta(q_i, a)$ $= [q_1, q_2, \ldots\ldots q_k]$ may be some state.

   ii) Add the state $[q_1, q_2, \ldots\ldots q_k]$ to DFA if it is not already added in Q'.

iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots\ldots q_k]$. If we get some state $[q_1, q_2, \ldots\ldots q_n]$ which is not in Q' of DFA then add this state to Q'.

iv) If there is no new state generating then stop the process after finding all the transitions.

3) For the state $[q_1, q_2, \ldots\ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots\ldots q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Example 2.7.1 *Convert the given NFA to DFA.*

State \ Input	0	1
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_1
q_2	q_3	q_3
((q_3))	ϕ	q_2

Solution : As we know, first we will compute δ' function.

$$\delta(\{q_0\}, 0) = \{q_0, q_1\}$$

Hence $\delta'([q_0], 0) = [q_0, q_1]$

Similarly,

$$\delta(\{q_0\}, 1) = \{q_0\}$$

Hence $\delta'([q_0], 1) = [q_0]$

Thus we have got a new state $[q_0, q_1]$.

Let us check how it behaves on input 0 and 1.

So,

$$\begin{aligned}\delta'([q_0, q_1], 0) &= \delta([q_0], 0) \cup \delta([q_1], 0)\\ &= \{q_0, q_1\} \cup \{q_2\}\\ &= \{q_0, q_1, q_2\}\end{aligned}$$

Hence a new state is generated i.e. $[q_0, q_1, q_2]$

Similarly,

$$\begin{aligned}\delta'([q_0, q_1], 1) &= \delta([q_0], 1) \cup \delta([q_1], 1)\\ &= \{q_0\} \cup \{q_1\}\\ &= \{q_0, q_1\}\end{aligned}$$

No new state is generated here.

Again δ' function will be computed for $[q_0, q_1, q_2]$, the new state being generated.

State \ Input	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$

As, you have observed in the above table for a new state $[q_0, q_1, q_2]$ the input 0 will give a new state $[q_0, q_1, q_2, q_3]$ and input 1 will give a new state $[q_0, q_1, q_3]$, because

$$\begin{aligned}\delta'([q_0, q_1, q_2], 0) &= \delta'([q_0], 0) \cup \delta'([q_1], 0) \cup \delta'([q_2], 0)\\ &= \{q_0, q_1\} \cup \{q_2\} \cup \{q_3\}\\ &= \{q_0, q_1, q_2, q_3\}\\ &= [q_0, q_1, q_2, q_3]\end{aligned}$$

Same procedure for input 1. Thus the final DFA is as given below.

State	0	1
→ $[q_0]$	$[q_0, q_1]$	$[q_0]$
(($[q_1]$))	$[q_2]$	$[q_1]$
$[q_2]$	$[q_3]$	$[q_3]$
$[q_3]$	ϕ	$[q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
(($[q_0, q_1, q_3]$))	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
(($[q_0, q_1, q_2, q_3]$))	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

DFA for example 2.7.1

Example 2.7.2 *Construct DFA equivalent to the given NFA.*

SPPU : Dec.-05, Marks 10

State \ Input	0	1
→ p	{p, q}	p
q	r	r
r	s	–
s	s	s

Solution : The NFA M = {{p, q, r, s}, {0, 1}, δ {p}, {s}}

The equivalent DFA will be constructed.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	–
([s])	[s]	[s]
[p, q]	[p, q, r]	[p, r]

Continuing with the generated new states.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	-
(*) [s]	[s]	[s]
[p, q]	[p, q, r]	[p, r]
[p, q, r]	[p, q, r, s]	[p, r]
[p, r]	[p, q, s]	[p]
(*) [p, q, r, s]	[p, q, r, s]	[p, r, s]
(*) [p, q, s]	[p, q, r, s]	[p, r, s]
(*) [p, r, s]	[p, q, s]	[p, s]
(*) [p, s]	[p, q, s]	[p, s]

The final state F = { [s], [p, q, r, s], [p, q, s], [p, r, s], [p, s] }

The transition graph shows two disconnected parts. But part I will be accepted as final DFA because it consists of start state and final states, in part II there is no start state.

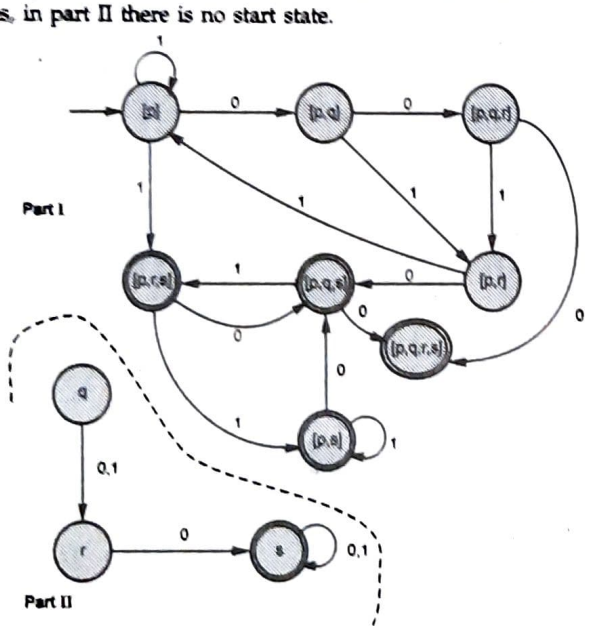


Fig. 2.7.1

Example 2.7.3 *Convert the given NFA to DFA.*

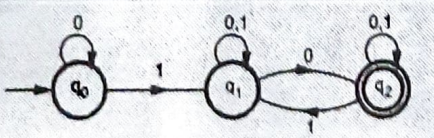


Fig. 2.7.2

Solution : For the given transition diagram in Fig. 3.6.10 we will first construct the transition table.

→	State / Σ	0	1
	q_0	q_0	q_1
	q_1	$\{q_1, q_2\}$	q_1
	(*) q_2	$\{q_2\}$	$\{q_1, q_2\}$

Now we will obtain δ' transitions for each state.

$\delta'([q_0], 0) = \{q_0\}$

$= [q_0]$

$\delta'([q_0], 1) = \{q_1\}$

$= [q_1]$

$\delta'([q_1], 0) = \{q_1, q_2\}$

$= [q_1, q_2]$ → new state generated.

$\delta'([q_1], 1) = \{q_1\} = [q_1]$

$\delta'([q_2], 0) = \{q_2\}$

$= [q_2]$

$\delta'([q_2], 1) = \{q_1, q_2\}$

$= [q_1, q_2]$

Now we will obtain δ' transitions on $[q_1, q_2]$.

$\delta'([q_1, q_2], 0) = \delta([q_1], 0) \cup \delta([q_2], 0)$

$= \{q_1, q_2\} \cup \{q_2\}$

$= \{q_1, q_2\}$

$= [q_1, q_2]$

$\delta'([q_1, q_2], 1) = \delta([q_1], 1) \cup \delta([q_2], 1)$

$= \{q_1\} \cup \{q_1, q_2\}$

$= [q_1, q_2]$

The state $[q_1, q_2]$ is final state as well because it contains a final state q_2. The transition table for the constructed DFA will be -

→	State / Σ	0	1
	$[q_0]$	$[q_0]$	$[q_1]$
	$[q_1]$	$[q_1, q_2]$	$[q_1]$
	(*) $[q_2]$	$[q_2]$	$[q_1, q_2]$
	(*) $[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$

The transition diagram will be -

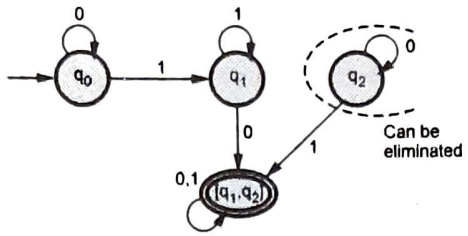


Fig. 2.7.3

Consider a string 0010 we will get -

Using NFA

q_0 0010 $\vdash$ $0q_00110$

$\vdash$ $00q_0110$

$\vdash$ $001q_110$

$\vdash$ $0011q_10$

$\vdash$ $00110q_2$ or $\vdash$ $00110q_1$

Accept non-accept.

Using DFA

q_000110 $\vdash$ $0q_00110$

$\vdash$ $00q_0110$

$\vdash$ $001q_110$

$\vdash$ $0011q_10$

$\vdash$ $00110\ [q_1, q_2]$

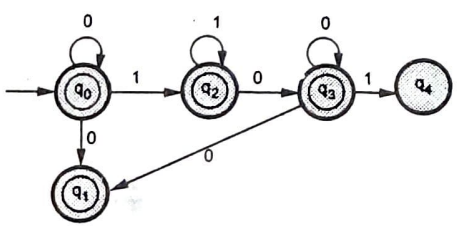
Accept state.

Thus both the NFA and DFA accept the same string 00110.

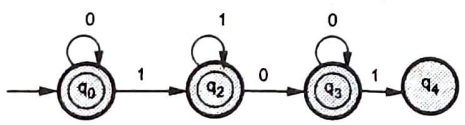
Example 2.7.4 *Construct a NFA and then equivalent DFA accepting strings over {0,1}, for accepting all possible strings of zeroes and ones not containing 101 as a substring.*

SPPU : May-09, Dec.-09, Marks 10

Solution : The NFA representing the strings that do not contain 101 as a substring is



The equivalent DFA will be

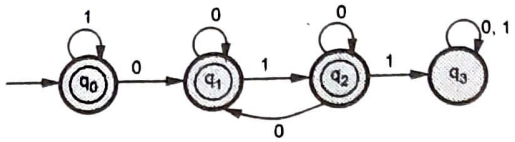


Example 2.7.5 *Construct a NFA and then equivalent DFA accepting string over {0, 1}, for accepting all possible strings of zeroes and ones which does not contain 011 a a substring.*

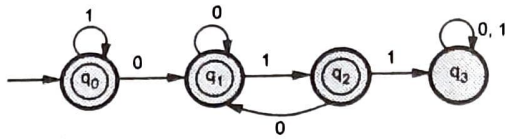
SPPU : May-09, 10, 12, Dec.-09, Marks 10

Solution :

NFA will be



DFA will be



Example 2.7.6 *Convert given non-deterministic finite automata (NFA) to deterministic finite automata (DFA). Informally describe the language it accepts.*

δ	0	1
p	{p, q}	{p}
q	ϕ	{r}
*r	{p, r}	{q}

SPPU : Dec.-13, Marks 6

Solution :

$\delta(p, 0)$ = {p, q} → **[p, q]** new state

$\delta(p, 1)$ = {p} = [p] state

$\delta(q, 0)$ = ϕ

$\delta(q, 1)$ = {r} = [r] state

$\delta(r, 0)$ = {p, r} = **[p, r] new state**

$\delta(r, 1)$ = {q} = [q] state

Now, we will find input transitions on newly generated states.

$\delta([p, q], 0)$ = [p, q]

$\delta([p, q], 1)$ = [p, r]

$\delta([p, r], 0)$ = {p, q, r} = **[p, q, r] new state**

$\delta([p, r], 1)$ = {p, q} = **[p, q]** new state

$\delta([p, q, r], 0)$ = [p, q, r]

$\delta([p, q, r], 1)$ = [p, q, r]

The transition table for NFA is -

		0	1
→	[p]	[p, q]	[p]
	[q]	ϕ	[r]
*	[r]	[p, r]	[q]
	[p, q]	[p, q]	[p, r]
*	[p, r]	[p, q, r]	[p, q]
*	[p, q, r]	[p, q, r]	[p, q, r]

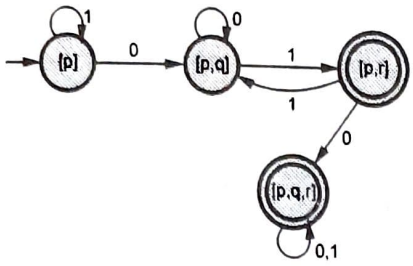


Fig. 2.7.4

Example 2.7.7 *Convert following NFA into its equivalent DFA.*

SPPU : Aug.-15, In Sem, Marks 4

Q \ Σ ->	0	1
->P	P, Q	R
Q	R	R
R	S	Q
* S	S	S

Solution : $\delta(P,0) = \{P, Q\} = [P, Q]$ New state

$\delta(P,1) = R$

$\delta(Q,0) = R$

$\delta(Q,1) = R$

$\delta(R,0) = S$

$\delta(R,1) = Q$

$\delta(S,0) = S$

$\delta(S,1) = S$

The δ transitions can be applied on new state [P, Q]

$\therefore \delta([P,Q],0) = \{P, Q, R\} = [P, Q, R]$ New state

$\delta([P,Q],1) = R$

$\delta([P,Q,R],0) = \{P, Q, R, S\} = [P, Q, R, S]$ New state

$\delta([P,Q,R),1) = \{R, Q\} = [R, Q]$ New state

$\delta([P,Q,R,S],0) = [P, Q, R, S]$

$\delta([P,Q,R,S),1) = \{R, Q, S\} = [R, Q, S]$ New state

$\delta([R,Q],0) = \{R, S\} = [R, S]$ New state

$\delta([R,Q),1) = [R, Q]$

$\delta([R,Q,S],0) = [R, S]$

$\delta([R,Q,S),1) = [R, Q, S]$

$\delta([R,S],0) = S$

$\delta([R,S),1) = [Q, S]$ New state

$\delta([Q,S],0) = [R, S]$

$\delta([Q,S),1) = [R, S]$

As no new state is getting generated we will stop applying δ transitions. The transition table for DFA is as follows.

State \ Input	0	1
→ [P]	[P, Q]	[R]
[Q]	[R]	[R]
[R]	[S]	[Q]
* [S]	[S]	[S]
[P, Q]	[P, Q, R]	[R]
[P, Q, R]	[P, Q, R, S]	[R, Q]
* [P, Q, R, S]	[P, Q, R, S]	[R, Q, S]
[R, Q]	[R, S]	[R, Q]
* [R, Q, S]	[R, S]	[R, Q, S]
* [R, S]	[S]	[Q, S]
* [Q, S]	[R, S]	[R, S]

2.7.2 NFA with ε to DFA Conversion

Subset Construction Method Algorithm

Step 1 : Initialization step

Let, ε - closure (s) be the states in Dstates of DFA.

Step 2 : Repeatation step

```
While (T are the unmarked states in Dstates)
  {
    mark T
    for (each symbol a)
    {
      U = ε - closure (move (T, a))
      if (U is not in Dstates)
        Add U as unmarked state in Dstates
      Dtran [T, a] = U
    } // end of for
  } // end of while.
```

Example 2.7.8 *Convert the following NFA with ε to equivalent DFA.*

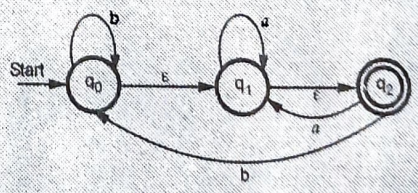


Fig. 2.7.5

Solution : To convert this NFA we will first find ε-closures.

ε - closures $\{q_0\} = \{q_0, q_1, q_2\}$

ε - closures $\{q_1\} = \{q_1, q_2\}$

ε - closures $\{q_2\} = \{q_2\}$

Let us start from ε-closure of start state

ε - closure $\{q_0\}$ = $\{q_0, q_1, q_2\}$ we will call this **state as A.**

Now let us find transitions on A with every input symbol.

$\delta'(A, a)$ = ε - closure $(\delta(A, a))$

= ε - closure $(\delta(q_0, q_1, q_2), a)$

= ε - closure $\{\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\}$

= ε - closure $\{q_1\}$

= $\{q_1, q_2\}$. Let us call it as **state B.**

$\delta(A, b)$ = ε - closure $(\delta(A, b))$

= ε - closure $(\delta(q_0, q_1, q_2), b)$

= ε - closure $\{\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\}$

= ε - closure $\{q_0\}$

= $\{q_0, q_1, q_2\}$ i.e. A.

Hence we can state that

$\delta(A, a)$ = B

$\delta(A, b)$ = A

Now let us find transitions for state B = $\{q_1, q_2\}$.

$\delta(B, a)$ = ε - closure $(\delta(q_1, q_2), a)$

= ε - closure $\{q_1\}$

= $\{q_1, q_2\}$ i.e. B

$\delta(B, b)$ = ε - closure $(\delta(q_1, q_2), b)$

= ε - closure $\{\delta(q_1, b) \cup \delta(q_2, b)\}$

= ε - closure $\{q_0\}$

= $\{q_0, q_1, q_2\}$ i.e. A.

Hence the generated DFA is

	a	b
→ A	B	A
B	B	A

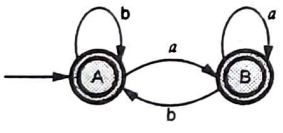


Fig. 2.7.6

Example 2.7.9 *Convert the given NFA into its equivalent DFA -*

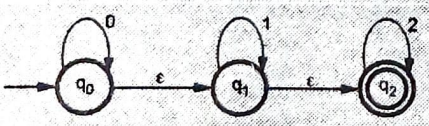


Fig. 2.7.7

Solution : Let us obtain ε - closure of each state.

ε - closure (q_0) = $\{q_0, q_1, q_2\}$

ε - closure (q_1) = $\{q_1, q_2\}$

ε - closure (q_2) = $\{q_2\}$

Now we will obtain δ' transition. Let ε - closure (q_0) = $\{q_0, q_1, q_2\}$ call it as **state A.**

$\delta(A, 0)$ = ε - closure $\{\delta((q_0, q_1, q_2), 0)\}$

= ε - closure $\{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\}$

= ε - closure $\{q_0\}$

= $\{q_0, q_1, q_2\}$ i.e. **state A**

$\delta(A, 1)$ = ε - closure $\{\delta((q_0, q_1, q_2), 1)\}$

= ε - closure $\{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\}$

= ε - closure $\{q_1\}$

= $\{q_1, q_2\}$. Call it as **state B**

$\delta(A, 2)$ = ε - closure $\{\delta((q_0, q_1, q_2), 2)\}$

= ε - closure $\{\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\}$

= ε - closure $\{q_2\}$

= $\{q_2\}$. Call it as **state C.**

Thus we have obtained

$\delta(A, 0)$ = A

$\delta(A, 1)$ = B

$\delta(A, 2)$ = C

i.e.

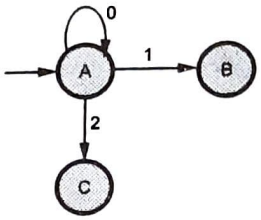


Fig. 2.7.8

Now we will find transitions on states B and C for each input.

Hence

$\delta(B, 0)$ = ε - closure $\{\delta(q_1, q_2), 0\}$

= ε - closure $\{\delta(q_1, 0) \cup \delta(q_2, 0)\}$

= ε - closure $\{\phi\}$

= ϕ

$\delta(B, 1)$ = ε - closure $\{\delta(q_1, q_2), 1\}$

= ε - closure $\{\delta(q_1, 1) \cup \delta(q_2, 1)\}$

= ε - closure $\{q_1\}$

= $\{q_1, q_2\}$ i.e. **state B itself.**

$\delta(B, 2)$ = ε - closure $\{\delta(q_1, q_2), 2\}$

= ε - closure $\{\delta(q_1, 2) \cup \delta(q_2, 2)\}$

= ε - closure $\{q_2\}$

= $\{q_2\}$ i.e **state C.**

Hence

$\delta(B, 0)$ = ϕ

$\delta(B, 1)$ = B

$\delta(B, 2)$ = C

The partial transition diagram will be

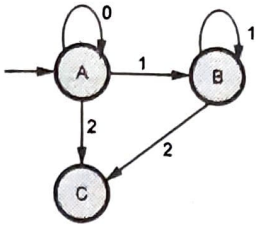


Fig. 2.7.9

Now we will obtain transitions for C :

$$\delta(C,0) = \varepsilon\text{ - closure }\{\delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure }\{\phi\}$$
$$= \phi$$
$$\delta(C,1) = \varepsilon\text{ - closure }\{\delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{\phi\}$$
$$= \phi$$
$$\delta'(C,2) = \varepsilon\text{ - closure }\{\delta(q_2,2)\}$$
$$= q_2$$

Hence the DFA is

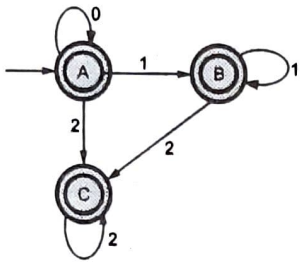


Fig. 2.7.10

As A = $\{q_0,q_1,q_2\}$ in which final state q_2 lies hence A is final state in B = $\{q_1,q_2\}$ the state q_2 lies hence B is also final state in C = $\{q_2\}$, the state q_2 lies hence C is also a final state.

Example 2.7.10 *Convert the given NFA with ε to its equivalent DFA.*

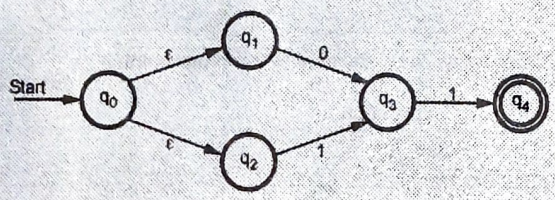


Fig. 2.7.11

Solution :

ε - closure $\{q_0\}$ = $\{q_0,q_1,q_2\}$

ε - closure $\{q_1\}$ = $\{q_1\}$

ε - closure $\{q_2\}$ = $\{q_2\}$

ε - closure $\{q_3\}$ = $\{q_3\}$

ε - closure $\{q_4\}$ = $\{q_4\}$

Now, Let ε - closure $\{q_0\}$ = $\{q_0,q_1,q_2\}$ be state A.

Hence
$$\delta'(A,0) = \varepsilon\text{ - closure }\{\delta((q_0,q_1,q_2),0)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,0)\cup\delta(q_1,0)\cup\delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure }\{q_3\}$$
$$= \{q_3\}\text{ call it as state B.}$$
$$\delta(A,1) = \varepsilon\text{ - closure }\{\delta((q_0,q_1,q_2),1)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,1)\cup\delta(q_1,1)\cup\delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{q_3\}$$
$$= q_3 = B.$$

The partial DFA will be

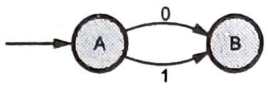


Fig. 2.7.12

Now,

$$\delta(B,0) = \varepsilon\text{ - closure }\{\delta(q_3,0)\}$$
$$= \phi$$
$$\delta(B,1) = \varepsilon\text{ - closure }\{\delta(q_3,1)\}$$
$$= \varepsilon\text{ - closure }\{q_4\}$$
$$= \{q_4\}\text{ i.e. state C}$$
$$\delta(C,0) = \varepsilon\text{ - closure }\{\delta(q_4,0)\}$$
$$= \phi$$
$$\delta(C,1) = \varepsilon\text{ - closure }\{\delta(q_4,1)\}$$
$$= \phi$$

The DFA will be,

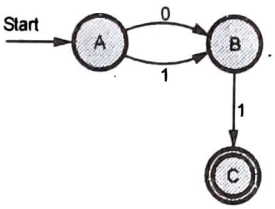


Fig. 2.7.13

Example 2.7.11 *Convert the given NFA– ∧ to an NFA.*

SPPU : Dec.-11, Marks 6

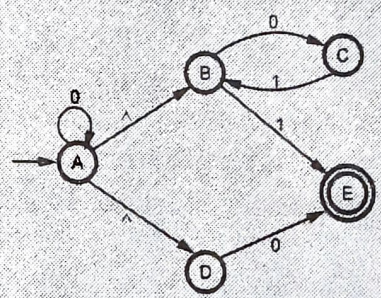


Fig. 2.7.14

Solution : We will ε-closures of A, B, C, D and E.

ε-closure (A) = {A, B, D}

ε-closure (B) = {B}

ε-closure (C) = {C}

ε-closure (D) = {D}

ε-closure (E) = {E}

The δ′ transitions on each input symbol is -

$$\delta'(A,0) = \varepsilon\text{-closure }(\delta(\varepsilon\text{-closure }(A),0))$$

$= \varepsilon\text{-closure}(\delta((A, B, D), 0))$

$= \varepsilon\text{-closure}(A, C, E)$

$= \varepsilon\text{-closure}(A) \cup \varepsilon\text{-closure}(C) \cup \varepsilon\text{-closure}(E)$

$= \{A, B, D\} \cup \{C\} \cup \{E\}$

$\delta'(A, 0) = \{A, B, C, D, E\}$

$\delta'(A, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A), 1))$

$= \varepsilon\text{-closure}(\delta((A, B, D), 1))$

$= \varepsilon\text{-closure}((E))$

$\delta'(A, 1) = \{E\}$

$\delta'(B, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(B), 0))$

$= \varepsilon\text{-closure}(\delta(B, 0))$

$= \varepsilon\text{-closure}(C)$

$\delta'(B, 0) = \{C\}$

$\delta'(B, 1) = \{E\}$

$\delta'(C, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C), 0))$

$= \varepsilon\text{-closure}(\delta(C, 0))$

$\delta'(C, 0) = \phi$

$\delta'(C, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C), 1))$

$= \varepsilon\text{-closure}(\delta(C, 1))$

$= \varepsilon\text{-closure}(B)$

$\delta'(C, 1) = \{B\}$

$\delta'(D, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 0))$

$= \varepsilon\text{-closure}(\delta(D, 0))$

$= \varepsilon\text{-closure}(E)$

$= \{E\}$

$\delta'(D, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 1))$

$= \varepsilon\text{-closure}(\delta(D, 1))$

$= \varepsilon\text{-closure}(\phi)$

$\delta'(D, 1) = \phi$

$\delta'(E, 0) = \varepsilon\text{-closure}(\delta(E, 0))$

$= \varepsilon\text{-closure}(\phi)$

$\delta'(E, 0) = \phi$

Similarly, $\delta'(E, 1) = \phi$

The NFA can be represented by following transition table.

State \ i/p	0	1
A	{A, B, C, D, E}	E
→ B	C	E
C	ϕ	B
D	E	ϕ
(E)	ϕ	ϕ

2.8 RE to DFA Conversion

SPPU : May-10,12,14,15, Aug.-15, Dec.-11,13,14,16, Marks 10

Regular expression can be converted to FA using two methods.

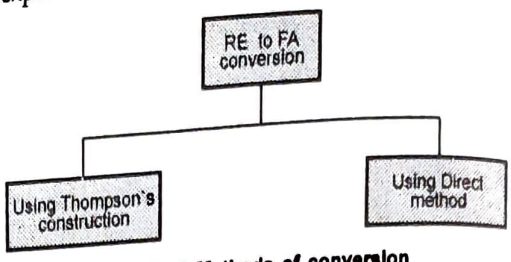


Fig. 2.8.1 Methods of conversion

1. Thomson's Construction

- There are some patterns of regular expressions for which the finite automata is constructed.
- These patterms are -

Case 1 : Zero operators

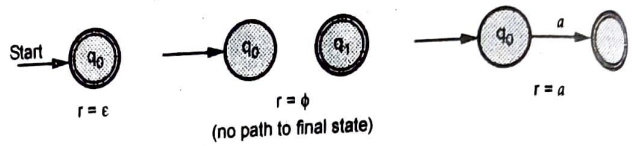


Fig. 2.8.2 Finite automata for given regular expression

Case 2 : Union case

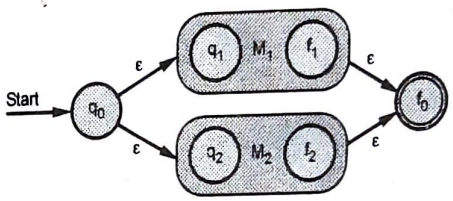


Fig. 2.8.3 The machine M for union

Let $r = r_1 + r_2$ where r_1 and r_2 be the regular expressions.

Case 3 : Concatenation case

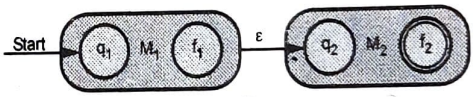


Fig. 2.8.4 Machine M for concatenation

$r = r_1 r_2$ be regular expression

Case 4 : Closure case

$r = r_1^*$ be regular expression

The machine M will be

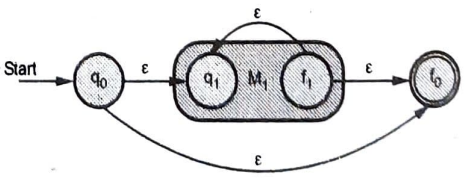


Fig. 2.8.5

- **Direct Method**
- This method is a direct method for obtaining FA from given regular expression. This is called a **subset** method. The method is given as below -

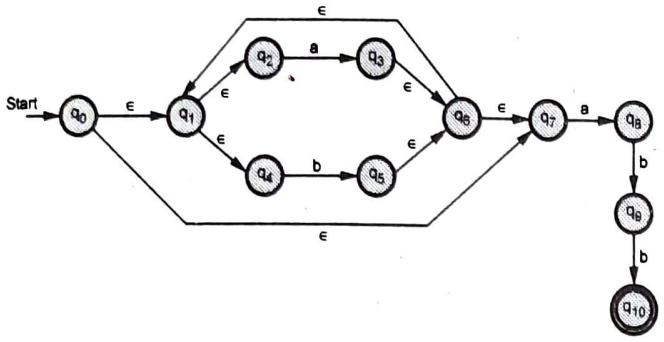
Step 1 : Design a transition diagram for given regular expression, using NFA with ε moves.

Step 2 : Convert this NFA with ε to NFA without ε.

Step 3 : Convert the obtained NFA to equivalent DFA.

Example 2.8.1 *Construct DFA for the Regular Expression (a + b)* abb.* **SPPU : May-12, Marks 6, Aug.-15, In Sem, Marks 8**

Solution : We will first construct NFA - ε for given r.e.



ε - closure (q_0) = $\{q_1, q_2, q_4, q_7\}$ = Call it as state A.

$\delta(A,a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_7), a)$

= ε - closure (q_3, q_8)

= $\{q_1, q_2, q_3, q_4, q_6, q_7, q_8\}$ = Call it as state B.

∴ $\delta(A, a)$ = **B**

$\delta(A,b)$ = ε - closure $(\delta(q_1, q_2, q_4, q_7), b)$

= ε - closure $(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\}$

$\delta(A, a)$ = **C**

$\delta(B, a)$ = ε - closure $(\delta(q_1, q_2, q_3, q_4, q_6, q_7, q_8), a)$

= ε - closure (q_3, q_8)

$\delta(B, a)$ = **B**

(B,b) = ε - closure $(\delta(q_1, q_2, q_3, q_4, q_6, q_7, q_8), b)$

= ε - closure (q_5, q_9)

= $\{q_1, q_2, q_4, q_5, q_6, q_7, q_9\}$

$\delta(B, b)$ = **D**

$\delta(C, a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7,), a)$

= ε - closure (q_3, q_8)

$\delta(C, a)$ = **B**

$\delta(c, b)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7), b)$

= ε - closure (q_5)

$\delta(C, b)$ = **C**

$\delta(D, a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_9), a)$

= ε - closure (q_3, q_8)

$\delta(D, a)$ = **B**

$\delta(D, b)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_9), b)$

= ε - closure (q_5, q_{10})

= $\{q_1, q_2, q_4, q_5, q_6, q_7, q_{10}\}$

$\delta(D, b)$ = **E**

$\delta(E, a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_{10}), a)$

= ε - closure $(\{q_3, q_8\})$

$\delta(E, a)$ = **B**

$\delta(E, b)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_{10})b)$

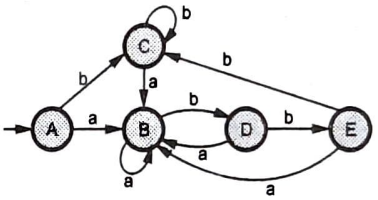
= ε - closure (q_5)

$\delta(E, b)$ = **C**

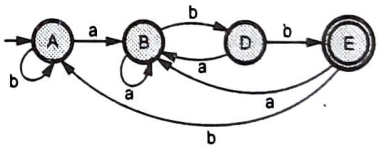
The transition table will be

State \ I/P	a	b
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

The transition diagram will be



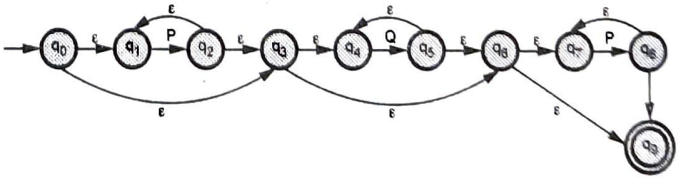
The states A and C are equivalent. Hence we can have minimized DFA as



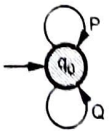
Example 2.8.2 *Show that P*(QP*)* = (P + Q)** **SPPU : May 2014, Marks 4**

Solution : We will show this equality using FA from R.E.

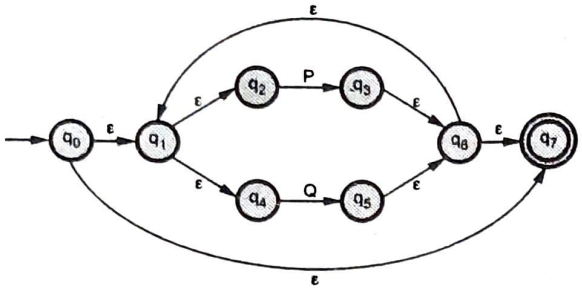
Step 1 : L.H.S. = P* (QP*)*. The NFA with ε for this regular expression will be



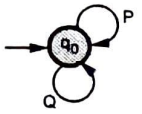
On eliminating ε moves we get -



Step 2 : R.H.S. = (P + Q)*



On elimiating ε moves we get -



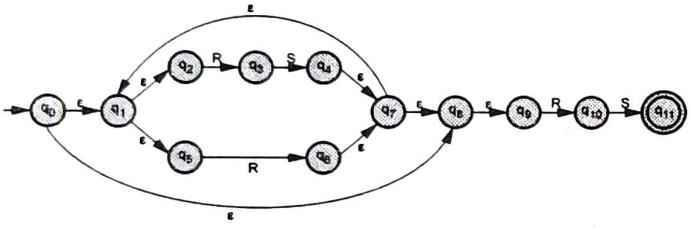
From Step 1 and Step 2 P*(QP*)* = (P + Q)* true.

Example 2.8.3 *Prove or disprove each of the following about regular expression : 1) (RS + R)*RS = (RR*S)* 2) (R + S)* = R* + S**

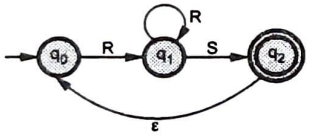
SPPU : Dec. 2013, Marks 4

Solution :

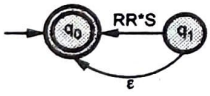
Step 1 : L.H.S. = (RS + R)* RS



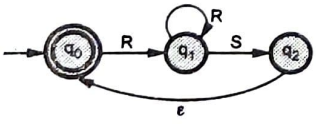
On eliminating ε transitions we get -



Step 2 : R.H.S. = (RR*S)*

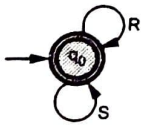


As L.H.S. and R.H.S. both does not yield same states (change in final state)

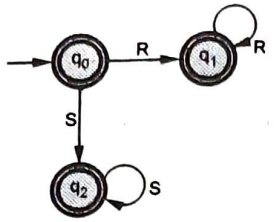


(RS + R)* RS ≠ (RR*S)*

2) Step 1 : (R + S)*



Step 2 : R* + S*

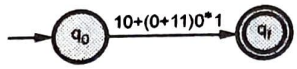


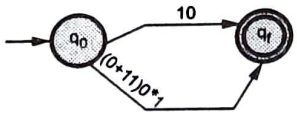
This proves that (R + S)* ≠ (RR*S)*

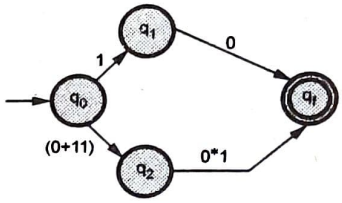
Example 2.8.4 *Design a FA from given regular expression* **10 + (0 + 11) 0* 1.**

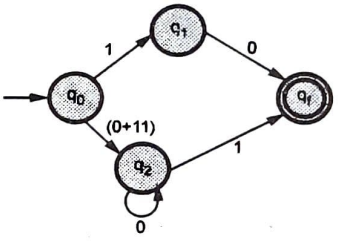
SPPU : Dec. 2011, Marks 10

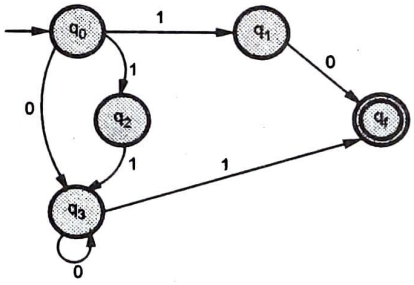
Solution : First we will construct the transition diagram for given regular expression.











Now we have got NFA without ε. Now we will convert it to required DFA for that, we will first write a transition table for this NFA.

State \ Input	0	1
q_0	q_3	$\{q_1, q_2\}$
q_1	q_f	ϕ
q_2	ϕ	q_3
q_3	q_3	q_f
q_f	ϕ	ϕ

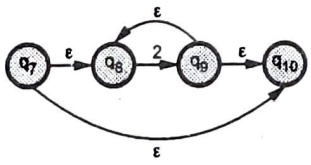
The equivalent DFA will be

State \ Input	0	1
$[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1]$	$[q_f]$	ϕ
$[q_2]$	ϕ	$[q_3]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
$[q_f]$	ϕ	ϕ

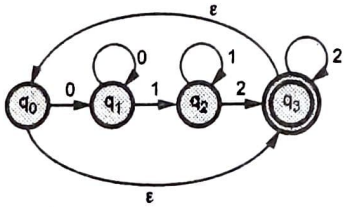
Example 2.8.5 *Construct minimized DFA accepting language represented by regular expression 0*1*2*. Convert given regular expression to NFA with ε moves and then convert NFA with ε moves to DFA using direct method.* **SPPU : Dec. 2013, Marks 8**

Solution :

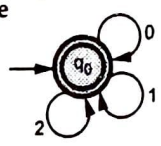
Step 1 :



Step 2 : Byeliminating ε moves



Step 3 : The DFA will be



Example 2.8.6 *Find FA for a given RE : (0 + 1)*·(010 + 101)·(0 + 1)*.* **SPPU : Dec.-14, End Sem, Marks 4**

Solution : The language L contains substring 010 or 101. Hence FA will be,

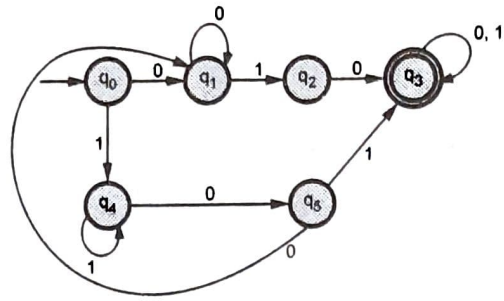
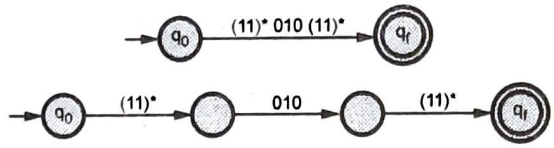


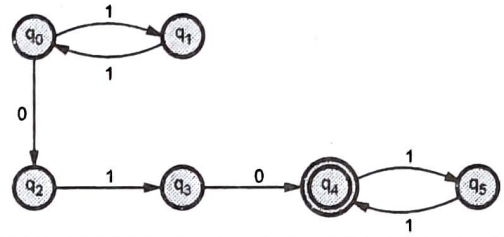
Fig. 2.8.6

Example 2.8.7 *Construct FA for the regular expression : (11)* · 010.(11)*.* **SPPU : May-15, End Sem, Marks 4**

Solution :



The FA will be :



Example 2.8.8 *Design a Moore machine and Mealy to generate 1's complement of given binary number.* **SPPU : May-10,15, End Sem, Marks 6**

Solution : To generate 1's complement of given binary number the simple logic which we will apply is that if input is 0 then output will be 1 and if input is 1 then output will be 0. That means there are three state one-start state, second state is for taking 0's as input and produces output as 1. Then third state is for taking 1's as input and producing output as 0.

Hence the Moore machine will be as shown in Fig. 2.8.7.

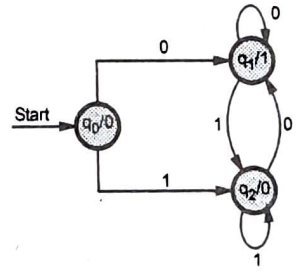


Fig. 2.8.7

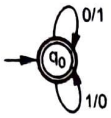
For instance, take one binary number 1011 then

Input		1	0	1	1
State	q_0	q_2	q_1	q_2	q_2
Output	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transition table can be drawn as below -

Current state	Next state 0	Next state 1	Output
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

Thus Moore machine M = $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$;
where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $\Delta = \{0,1\}$.
The transition table shows the δ and λ functions.



Mealy machine

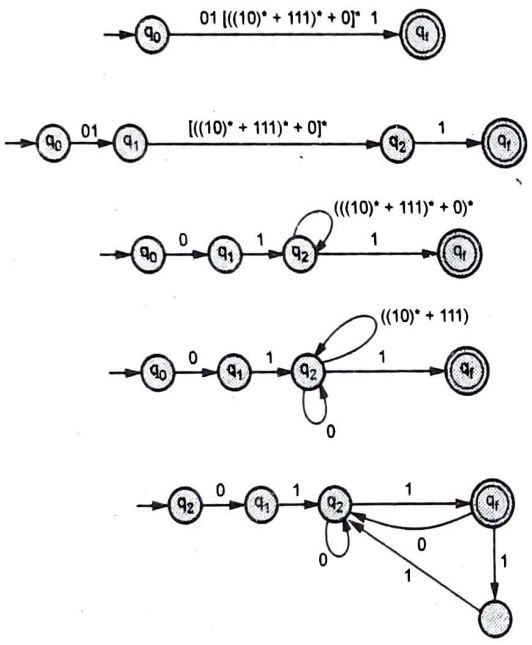
Example 2.8.9 *Construct finite automata for the following regular expressions :*

i) 01 [((10)* + 111)* + 0]* 1

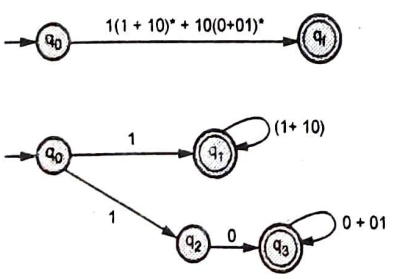
ii) 1(1+ 10)* + 10(0+ 01)* **SPPU : Dec.-16, End Sem. Marks 6**

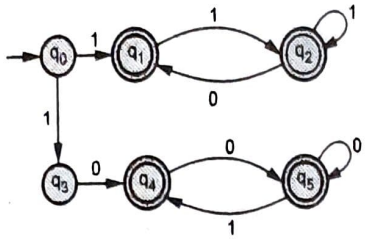
Solution :

i)



ii)





2.9 DFA to RE Conversions

SPPU : May-09,10,14,15,16, Dec.-08,09,10,13,14,15, Aug.-15, Marks 10

There are two methods for conversion of DFA to Regular Expression.

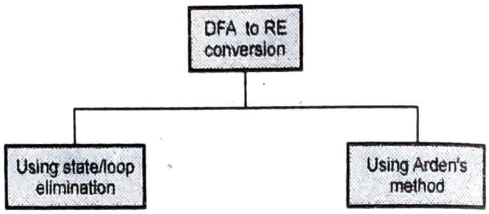


Fig. 2.9.1 DFA to RE conversion methods

2.9.1 State/Loop Elimination

1) The state elimination is the simplest method in which the states of DFA are removed one by one. Until we left with one start and one final state, for each removed state regular expression is generated.
2) In this method, each newly generated regular expression acts as input for a state which is next to removed state.
3) The advantage of this method is it becomes easier to visualize regular expression from the states of FA.

Example 2.9.1 *Convert the following FA to RE using state elimination method.*

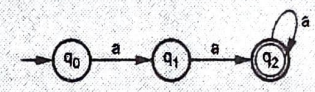


Fig. 2.9.2

Solution :

Step 1 :

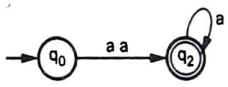


Fig. 2.9.3

Step 2 :

Fig. 2.9.4

Step 3 : R.E. = aaa^*

Example 2.9.2 *Obtain RE from given FA using loop /state elimination method.*

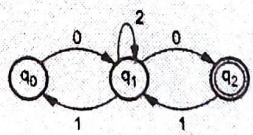


Fig. 2.9.5

Solution :

Step 1 :

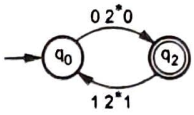
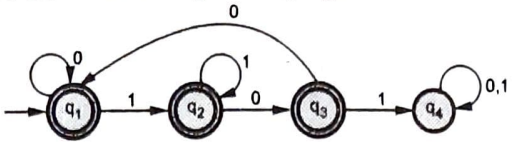


Fig. 2.9.6

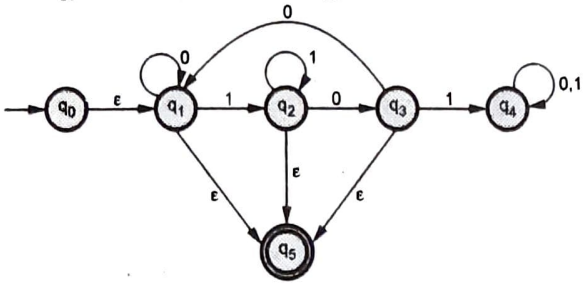
Step 2 : R.E. = $[(02^*0)(1\,2^*1)]^*$

Example 2.9.3 *Construct NFA and DFA for accepting all possible string of 0's amd 1's not containing 101 as substring. Find regular expression for it.* **SPPU : May-14, Marks 8**

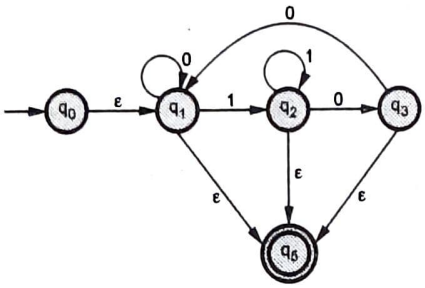
Solution : The DFA for the given language is



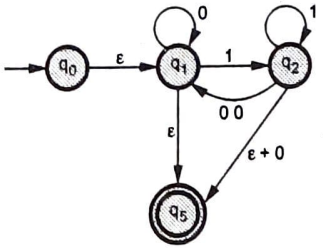
Now we will convert this DFA to NFA with ε. We will add one start state q_0 and new final state i.e. q_5.



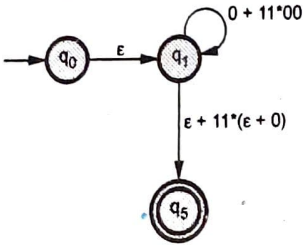
Now we will eliminate state q_4. The NFA will be -



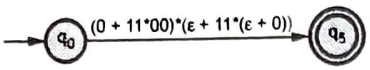
We will eliminate state q_3, without changing meaning of it.



We will eliminate state q_2.



Now we will eliminate state q_1. The NFA will be -



The regular expression will be **(0 + 11*00)* (ε + 11* (ε + 0))**

Example 2.9.4 *Find the regular expression corresponding to each of the following subset of* $[0, 1]^*$

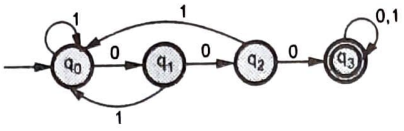
a) The language of all strings not containing the substring 000

b) The language of all strings that do not contain the substring 110

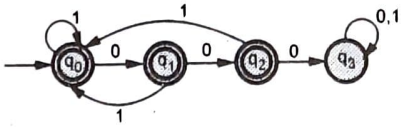
c) The language of all strings containing both 101 and 010 as substring. **SPPU : May-10, Marks 10**

Solution :

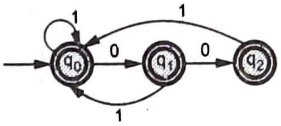
a) To obtain r.e. for this L we will first design FA for strings containing 000 as substring.



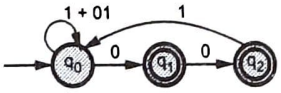
Now to obtain FA for strings not containing 000 as substring just make final state of above FA as non-final and non-final states as final state. The FA will then be -



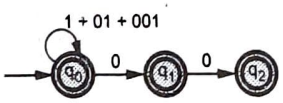
Clearly in above FA q_3 is a dead state. So eliminate it.



Now we will eliminate the edge between q_0 and q_1 which is giving $(01)^*$

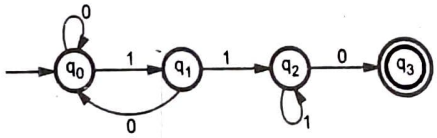


Similarly, we will eliminate edge from q_2 to q_0 giving $(001)^*$

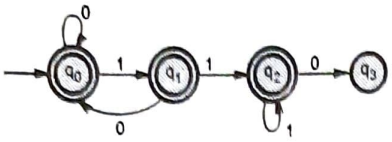


This generates **r.e. = (1+01+001)* [ε + 0 + 00]**

b) Let us draw the transition graph for the language that contains 110.



Now for the language that do not contain 110, we will change final states to non-final states and non-final states to final state.



From this diagram, the regular expression will be.

$$\text{r.e.} = [0^*(10)^* + 1^*]$$

c) Regular expression = $[(0+1)^*101\ (0+1)^*110\ (0+1)^*] +$

$$(0+1)^*110(0+1)^*\ 101(0+1)^*]$$

2.9.2 Arden's Theorem

Conversion Method

Following algorithm is used to build the r.e. from given DFA.

1. Let q_1 be the initial state.
2. There are $q_2, q_3, q_4, \ldots q_n$ number of states. The final state may be some q_j where $j \leq n$.
3. Let α_{ji} represents the transition from q_j to q_i.
4. Calculate q_i such that

   $$q_i = \alpha_{ji} \cdot q_j$$

   If q_i is a start state

   $$q_i = \alpha_{ji}\, q_j + \varepsilon$$
5. Similarly compute the final state which ultimately gives the regular expression r.

Example 2.9.5 *Construct r.e. from given DFA.*

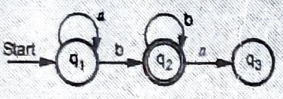


Fig. 2.9.7

Solution : Let us write down the equations

$$q_1 = q_1\, a + \varepsilon$$

Since q_1 is a start state ε will be added and the input a is coming to q_1 from q_1 hence we write

State = Source state of input × Input coming to it

Similarly $q_2 = q_1\, b + q_2\, b$

Let us simplify q_1 first

$$q_1 = q_1\, a + \varepsilon$$

We can re-write it as

$$q_1 = \varepsilon + q_1\, a$$

which is similar to R = Q + RP which further gets reduced to R = QP*.

Assuming $R = q_1$, $Q = \varepsilon$, $P = a$

We get $q_1 = \varepsilon \cdot a^*$

$$q_1 = a^* \quad \because\ \varepsilon \cdot R^* = R^*$$

Substituting value of q_1 in q_2 we get

$$q_2 = q_1\, b + q_2\, b$$

$$q_2 = a^*\, b + q_2\, b$$

We can compare this equation with R = Q + R P assuming $R = q_2$, $Q = a^*b$, $P = b$ which gets reduced to R = QP*.

$$\therefore \quad q_2 = a^*b \cdot b^*$$

As $R\, R^* = R^+$

$$q_2 = a^* \cdot b^+$$

From the given DFA, if we want to find out the regular expression, we normally calculate the equation for final state. Since in the given DFA q_2 is a final state and $q_2 = a^* b^+$. We can conclude as the DFA represents $a^* b + b^+$ as regular expression.

Example 2.9.6 *Find out the regular expression from given DFA.*

SPPU : Dec.-15, End Sem, Marks 6

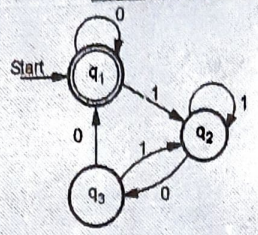


Fig. 2.9.8

Solution : Let us solve the DFA by writing the regular expression for each state.

$$q_1 = q_1\, 0 + q_3\, 0 + \varepsilon \quad \because \text{ Initial state}$$

$$q_2 = q_2 1 + q_3\, 1 + q_1\, 1$$

$$q_3 = q_2 0$$

For getting the r.e. we have to solve q_1 the final state.

$$q_2 = q_2 1 + q_2\, 01 + q_1\, 1$$

$$q_2 = q_2 (1 + 01) + q_1\, 1$$

We will compare R = Q + R P with above equation, so $R = q_2$, $Q = q_1 1$, $P = (1 + 01)$ which ultimately gets reduced to QP*.

$$q_2 = q_1\, 1(1 + 01)^*$$

Substituting this value to q_1

$$q_1 = q_1\, 0 + q_3 0 + \varepsilon$$

$$= q_1\, 0 + q_2\, 00 + \varepsilon$$

$$= q_1\, 0 + q_1\, (1(1 + 01)^*)\, 00 + \varepsilon$$

$$q_1 = q_1\, (0 + 1(1 + 01)^* 00) + \varepsilon$$

Again $R = Q + R\, P$

where $R = q_1$

$Q = \varepsilon$

$P = 0 + 1(1 + 01)^* 00$

Hence $q_1 = \varepsilon \cdot [0 + 1(1 + 01)^* \cdot 00]^*$

$q_1 = [0 + 1(1 + 01)^* \cdot 00]^*$ $\because$ $\varepsilon \cdot R = R$ is required r.e.

Example 2.9.7 *Represent the language given by following DFA.*

SPPU : Dec.-14, End Sem, Marks 4

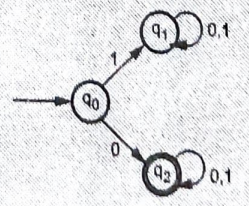


Fig. 2.9.9

Solution : To find the language, we will solve for regular expression,

$$q_0 = \varepsilon$$

$$q_2 = q_2 0 + q_2 1 + q_0 0$$

Substituting the value of q_0 in q_2

$$q_2 = q_2 (0 + 1) \varepsilon \cdot 0$$

$$q_2 = q_2 (0 + 1) + 0 \quad \because R = Q + R P \Rightarrow QP^*$$

$$\boxed{q_2 = 0 (0 + 1)^*} \quad \text{where } Q = 0, P = (0 + 1), R = q_2$$

This is a final regular expression, which represents the language containing all the strings which start with letter 0.

Example 2.9.8 *Describe in english the language indicated by following regular expression.* **SPPU : Dec. 2010, Marks 6**

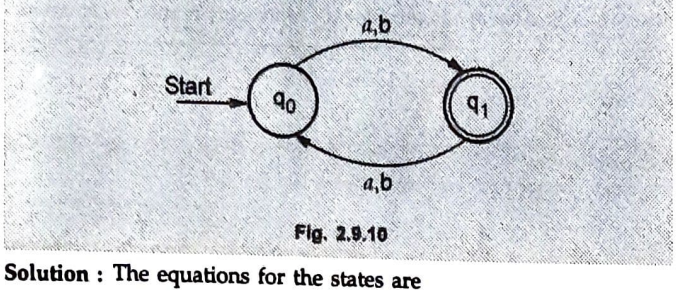


Fig. 2.9.10

Solution : The equations for the states are

$$q_0 = q_1 a + q_1 b + \varepsilon \quad \ldots(1)$$

$$q_1 = q_0 (a + b) \quad \ldots(2)$$

We will simplify q_0 first

$$q_0 = (a + b) q_1 + \varepsilon$$

Let us put this value of q_0 into equation (2).

$$q_1 = ((a + b) q_1 + \varepsilon) (a + b)$$

$$q_1 = (a + b) (a + b) q_1 + (a + b)$$

This is equivalent to R = Q + R P where R = q_1, Q = (a + b), P = (a + b) (a + b) which gets reduced to QP*.

$$\therefore \quad \boxed{q_1 = (a + b) \cdot [(a + b) (a + b)]^*}$$

This is a regular expression indicated by DFA.

The language given by this DFA is the language in which it accepts all the strings of odd length.

Example 2.9.9 *Consider the FA and construct regular expressions that is accepted by it. Fig. 2.9.11.* **SPPU : Dec.-13, Marks 6**

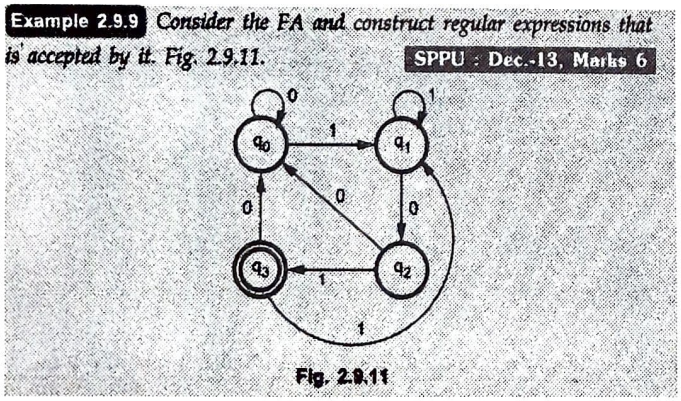


Fig. 2.9.11

Solution : We will first write the equations for each state, assuming q_0 as initial state

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \varepsilon \quad \ldots(1)$$

$$q_1 = q_0 1 + q_1 1 + q_3 1 \quad \ldots(2)$$

$$q_2 = q_1 0 \quad \ldots(3)$$

$$q_3 = q_2 1 \quad \ldots(4)$$

Now we will place value of q_2 in equation (4)

$$q_3 = q_1 01 \quad \ldots(5)$$

Put value of equation (5) in equation (2)

$$q_1 = q_0 1 + q_1 1 + (q_1 01) 1$$

$$q_1 = q_0 1 + q_1 1 + q_1 011$$

$$\underbrace{q_1}_{R} = \underbrace{q_1}_{R} \underbrace{(1 + 011)}_{P} + \underbrace{q_0 1}_{Q}$$

gives us

$$q_1 = q_0 1 (1 + 011)^* \quad \because R = Q + RP \text{ gives } R = QP^*$$

Now solving q_0

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \varepsilon$$

$$= q_0 0 + (q_1 0) 0 + (q_1 01) 0 + \varepsilon$$

$$q_0 = q_0 0 + q_1 (00 + 010) + \varepsilon$$

$$= q_0 0 + q_0 1 (1 + 011)^* (00 + 010) + \varepsilon$$

$$q_0 = q_0 (0 + 1 (1 + 011)^* (00 + 010)) + \varepsilon$$

$$\therefore \quad q_0 = \varepsilon (0 + 1(1 + 011)^* (00 + 010))^*$$

Now we will solve

$$q_3 = q_1 01$$

$$= q_0 1 (1 + 011)^* 01$$

$$q_3 = (0 + 1(1 + 011)^* (00 + 010))^* 1 (1 + 011)^* 01.$$

As q_3 is a final state, we get the r.e. as equation for q_3.

$$\therefore \quad \boxed{\text{r.e.} = (0 + 1(1 + 011)^* (00 + 010))^* 1 (1 + 011)^* 01)}$$

Example 2.9.10 *Design the finite automata and then equivalent regular expression using Arden's theorem that accepts the set of all strings over the alphabet {a, b} with an equal number of a's and b's, such that each prefix has atmost one more a than b's and atmost one more b than a's.* **SPPU : Dec.-08, 09, May-09, Marks 10**

Solution : The FA will be as shown in Fig. 2.9.12.

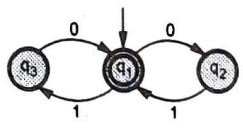


Fig. 2.9.12

The valid strings 0101 contains prefix of 1 as 010 in which number of 0's are more than 1. Similarly in a valid string 1010 the prefix of 0 contains more number of 1's than 0's .

Now in order to obtain the regular expression for this FA, we will write state equations as follows -

$$q_1 = q_2 1 + q_3 0 + \varepsilon \quad \ldots(1)$$

$$q_1 = q_1 0 \quad \ldots(2)$$

$$q_3 = q_1 1 \quad \ldots(3)$$

There is only one final state and that is q_1. Hence equation of q_1 state is the regular expression.

$q_1 = q_1\,(01 + 10) + \varepsilon$ (R R P Q) ∵ Applying Arden's theorem.

Let, $q_1 = q_2 1 + q_3 0 + \varepsilon$

$= q_1 01 + q_1 10 + \varepsilon$ ∵ Put equations 2 and 3 in equation 1

$q_1 = \varepsilon(01 + 10)^*$ ∵ R = Q + RP is R = QP*

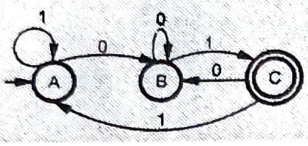
$q_1 = (01 + 10)^*$ ∵ $\varepsilon R^* = R^*$

Hence regular expression for given language is

r.e. = $(01 + 10)^*$

Example 2.9.11 *Convert the following finite automation into its equivalent regular expression using Arden's Theorem.*

SPPU : May-15, End Sem. Marks 6



Solution :

$A = A1 + C1 + \varepsilon$ ... (1)

$B = B0 + A0 + C0$ ... (2)

$C = B1$ ... (3)

Putting equation (3) in equation (1), we get -

$A = A1 + B11 + \varepsilon$

$A = A\,1 + B11 + \varepsilon$ (R P Q)

$\therefore \quad A = (B11 + \varepsilon)1^*$ ... (4)

Now, put equation (3) in equation (2),

$B = B0 + A0 + B10$ ...(5)

put equation (4) in equation (5),

$B = B0 + (B11 + \varepsilon)\,1^*\,0 + B\,10$

$= B0 + B\,111^*\,0 + 1^*0 + B\,10$

$B = B\,(0 + 111^*\,0 + 10) + 1^*\,0$ (R P Q)

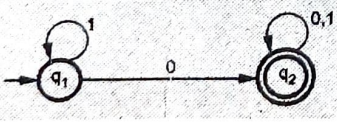
$\therefore \quad B = 1^*0\,(0 + 111^*\,0 + 10)^*$ ... (6)

Put equation (6) in equation (3),

$C = 1^*0\,(0 + 111^*\,0 + 10)^*\,1$ is R.E.

Example 2.9.12 *Obtain the regular expression that denotes the language acepted by the following DFA, using Arden's Theorem :*

SPPU : Aug.-15, In Sem. Marks 4



Solution : Let us write the equations for each state

$q_1 = q_1 1 + \varepsilon$ ... (1)

$q_2 = q_1 0 + q_2(0 + 1)$ ... (2)

Let us solve equation (1) using Arden's theorem

$q_1 = q_1\,1 + \varepsilon$ (R R P Q)

$\therefore \quad q_1 = \varepsilon 1^* = 1^*$ ∵ R = RP + Q gives R = QP*

$\therefore \quad q_1 = 1^*$ ... (3)

Putting equation (3) in (2) we get

$q_2 = 1^*0 + q_2\,(0+1)$ (R Q R P)

$\therefore \quad q_2 = 1^*0(0 + 1)^*$ ... (4)

As q_2 is a final state equation for q_2 represents regular expression.

$\therefore$ **r.e. = $1^*0(0 + 1)^*$**

Example 2.9.13 *Make use of Arden's theorem to determine the regular expression for the finite automata shown in Fig. 2.9.13.*

SPPU : May-16 Marks 6

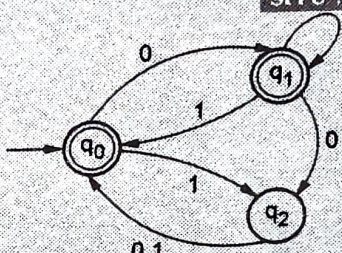


Fig. 2.9.13

Solution :

$q_0 = \varepsilon + q_1 1 + q_2(0 + 1)$ ...(1)

$q_1 = q_0 0 + q_1 2$ ...(2)

$q_2 = q_0 1 + q_0 0$ ...(3)

Now we will put equation (3) in (1)

$q_0 = \varepsilon + q_1 1 + (q_0 1 + q_1 0)(0 + 1)$

i.e. $q_0 = \varepsilon + q_1 1 + q_0 10 + q_0 11 + q_1 00 + q_1 01$

i.e. $= \varepsilon + q_1(1 + 00 + 01) + q_0 1(0 + 1)$

$q_0 = \varepsilon + q_1(1 + 00 + 01) + q_0 1(0 + 1)$ (R Q R P)

$\therefore \quad q_0 = [\varepsilon + q_1(1 + 00 + 01)](1(0 + 1))^*$ ... (4)

Now solving equation (2) with the help of equation (4)

$q_1 = q_0 0 + q_1 2$

$q_1 = [\varepsilon + q_1(1 + 00 + 01)](1(0 + 1))^* 0 + q_1 2$

$= (1(0+1))^* 0 + q_1(1 + 00 + 01)(1(0 + 1))^* 0 + q_1 2$

$q_1 = (1\,(0 + 1))^* 0 + q_1\,[(1 + 00 + 01)(1(0 + 1))^* 0 + 2]$ (R Q R P)

$\therefore \quad q_1 = (1(0+1))^* 0[(1 + 00 + 01)(1(0 + 1))^* 0 + 2]^*$

From equation (4) the q_0 will be

$$q_0 = (\varepsilon + q_1 + (1 + 00 + 01))(1(0 + 1))^*$$

$$= (1(0 + 1))^* + q_1(1 + 00 + 01)(1(0 + 1))^*$$

$$q_0 = (1(0 + 1))^* + ((1(0 + 1)^*0)(1 + 00 + 01)(1(0 + 1)^*0 + 2)$$
$$(1 + 00 + 01)(1(0 + 1))^*$$

Now for final regular expression,

$$\text{R.E.} = q_0 + q_1$$

$$\therefore \text{R.E.} = [(1 + (0 + 1))^* + ((1(0 + 1))^*0)(1 + 00 + 01)$$
$$((1(0 + 1)^*0 + 2)(1 + 00 + 01)(1(0 + 1))^*] +$$
$$[1(0 + 1)^*0[(1 + 00 + 01)(1(0 + 1))^*0 + 2]$$

2.10 Minimization of DFA

SPPU : Dec.-12, May-10, Oct.-16, Marks 8

- The minimization of FSM means reducing the number of states from given FA.
- While minimizing FSM we first find out which two states are equivalent we can represent those two states by one representative state.
- **Definition of equivalent states** - The two states q_1 and q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non final states for all $x \in \Sigma^*$ (Σ^* indicate any string of any length) we can minimize the given FSM by finding equivalent states.

Method for Construction of Minimum State Automata :

Step 1 : We will create a set π_0 as $\pi_0 = \{ Q_1^0, Q_2^0 \}$ where Q_1^0 is set of all final states and $Q_2^0 = Q - Q_1^0$ where Q is a set of all the states in DFA.

Step 2 : Now we will construct π_{K+1} from π_K. Let Q_i^K be any subset in π_K. If q_1 and q_2 are in Q_i^K they are (K + 1) equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are K equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are residing in the same equivalence class π_K. Then it is said that q_1 and q_2 are (K + 1) equivalent. Thus from Q_i^K we create (K + 1) equivalence classes. Repeat step 2 for every Q_i^K in π_K and obtain all the elements of π_{K+1}.

Step 3 : Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Step 4 : Then replace all the equivalent states in one equivalence class by representative state. This helps in minimizing the given DFA.

Let us understand this method with the help of some examples.

Example 2.10.1 *Construct the minimum state automaton for the following transition diagram.* SPPU : Dec.-12, Marks 8

Solution : We will first construct a transition table for the given DFA.

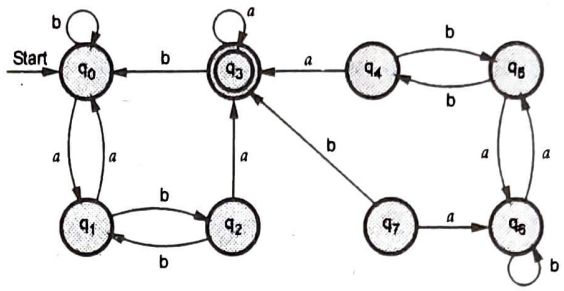


Fig. 2.10.1

We will start constructing equivalence classes.

	State \ Input	a	b
→	q_0	q_1	q_0
	q_1	q_0	q_2
	q_2	q_3	q_1
	q_3	q_3	q_0
	q_4	q_3	q_5
	q_5	q_6	q_4
	q_6	q_5	q_6
	q_7	q_8	q_3

There is only one final state i.e. q_3. Hence we can partition Q as Q_1^0 and Q_2^0.

$$Q_1^0 = F = \{q_3\} \text{ As } \{q_3\} \text{ cannot be partitioned further.}$$

$$Q_2^0 = Q - Q_1^0$$

$$= \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$$

$$\therefore \quad \pi_0 = \{\{q_3\}, \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}\}$$

Creating π_1

Now, we will compare q_0 with q_2.

	a	b
q_0	q_1	q_0
q_2	q_3	q_1

Under a - column of q_0 and q_2 we get q_1 and q_3 respectively. But q_1 and q_3 lie in different states. Hence they are not 1 - equivalent. Similarly consider q_0 and q_4.

	a	b
q_0	q_1	q_0
q_4	q_3	q_5

Under a - column of q_0 and q_4 we get q_1 and q_3 which lie in different sets. Hence q_0 is not 1 - equivalent to q_4. Similarly q_0 is not 1 - equivalent to q_7 (observe b-column of q_0 and q_7). But q_0 is 1 - equivalent to q_1, q_5 and q_6.

$$Q_1' = \{q_3\}$$

$$Q_2' = \{q_0, q_1, q_5, q_6\}$$

q_2 is 1 - equivalent to q_4. Hence

$$Q_3' = \{q_2, q_4\}$$

Since under a-column we get q_3 only for q_2 and q_4. And under b - column of q_2 and q_4 we get q_1 and q_5. But q_1 and q_5 lie in one set. Thus q_2 and q_4 are 1 - equivalent.

The only element left over in Q_2^0 is q_7.

$$\therefore \quad Q_4' = \{q_7\}$$

The equivalence class is -

$$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$$

Creating π_2

$$Q_1^2 = \{q_3\}$$

Now q_0 is 2 - equivalent to q_6 because

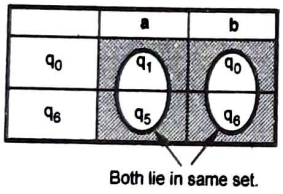


Fig. 2.10.2 (a)

But q_0 is not 2 - equivalent to q_1 and q_5.

Hence $Q_2^2 = \{q_0, q_6\}$

But q_1 is 2 - equivalent to q_5.

Hence $Q_3^2 = \{q_1, q_5\}$

Similarly q_2 is 2 - equivalent to q_4.

$$Q_4^2 = \{q_2, q_4\}$$

$$Q_5^2 = \{q_7\}$$

Thus, $\pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$

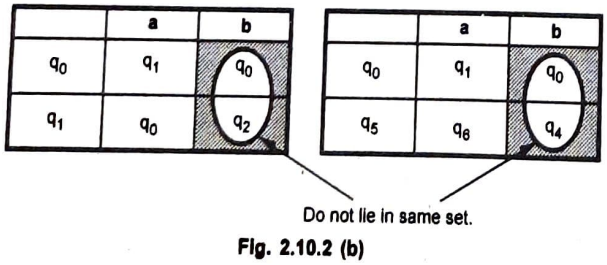


Fig. 2.10.2 (b)

Creating π_3

$$Q_1^3 = \{q_3\}$$

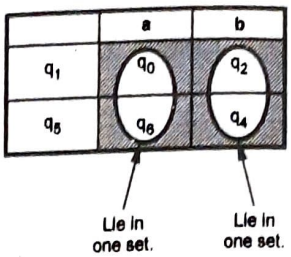


Fig. 2.10.2 (c)

As q_0 is 3 - equivalent to q_6,

$$Q_2^3 = \{q_0, q_6\}$$

As q_1 is 3 - equivalent to q_5,

$$Q_3^3 = \{q_1, q_5\}$$

As q_2 is 3 - equivalent to q_4,

$$Q_4^3 = \{q_2, q_4\}$$

and $Q_5^3 = \{q_7\}$

$$\therefore \quad \pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

As $\pi_3 = \pi_2$ we have got equivalence class from π_2. Hence we can write a transition table as

	State \ Input	a	b
→	$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
	$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
	$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
*	$[q_3]$	$[q_3]$	$[q_0, q_6]$
	$[q_7]$	$[q_0, q_6]$	$[q_3]$

The transition diagram with minimized states is -

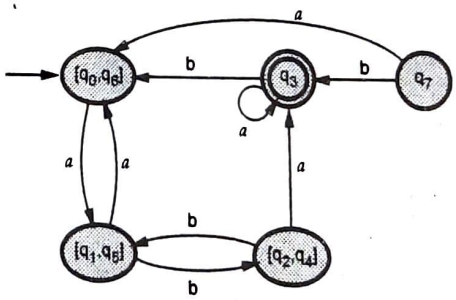


Fig. 2.10.3

2.10.1 Table Driven Method

This method helps in finding equivalent states in one stroke. Hence it is an efficient method of minimization of given DFA. Let us understand this method with the help of some example.

Example 2.10.2 *Minimize the DFA as given below.*

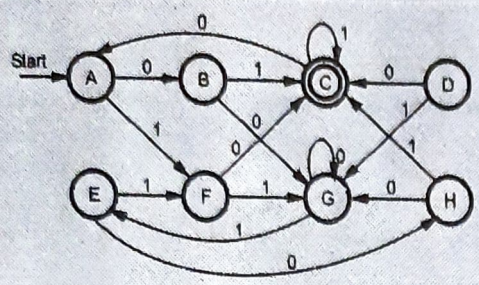


Fig. 2.10.4 SPPU : May-10, Marks 8

Solution :

Step 1 : We will build the transitions table for given DFA as follows.

	0	1
→ A	B	F
B	G	C
C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Step 2 : We will now construct a table for each pair of states.

B							
C							
D							
E							
F							
G							
H							
	A	B	C	D	E	F	G

We will mark X for all the final and non final states. Hence

B							
C	X	X					
D			X				
E			X				
F			X				
G			X				
H			X				
	A	B	C	D	E	F	G

Thus we have marked X for (A, C), (B, C), (C, D), (C, E), (C, F), (C, G), (C, H).

Step 3 : Now we will consider every pair form above table. Consider pair (G, H).

δ (G, 0) = G, δ (G, 1) = E or C ← Consider δ transition for this pair

δ (H, 0) = G, δ (H, 1) = E or C

As for δ (G, 1) = E and δ (H, 1) = C. We can see X in (C, E) pair. Hence pair (G, H) is not equivalent.

Hence we will mark **X in pair (G, H)**. Thus we will find the equivalent pairs.

Consider pair (B, H)

δ (B, 0) = G, δ (B, 1) = C

δ (H, 0) = G, δ (H, 1) = C

Thus the pair **(B, H) is equivalent**. Thus we obtain,

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
	A	B	C	D	E	F	G

Step 4 : Thus we get equivalent pairs as (A, E), (B, H), (D, F). Hence the minimized DFA will be,

	0	1
→ A	B	D
B	G	C
(C)	A	C
D	C	G
G	G	A

Example 2.10.3 *Minimize the DFA given below :*

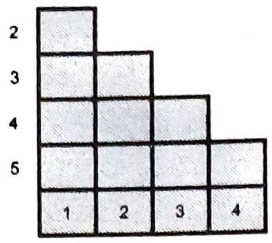
Input symbols and states	Next state	
	a	b
→*1	3	2
2	4	1
3	5	4
4	4	4
5	3	2

Initial state : 1 and final states : states 1 and 5

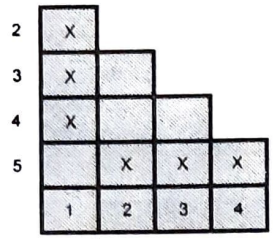
SPPU : Oct.-16, In Sem, Marks 5

Solution :

Step 1 : We will now construct table for each pair of states.



Step 2 : We will mark × for all the final and non-final states.



Step 3 : Now we will consider every remaining pair from above table

i) Consider pair (3, 4).

$\delta(3, a) = 5$, $\delta(3, b) = 4$
$\delta(4, a) = 4$, $\delta(4, b) = 4$

Consider pair (5,4) or (4,5)

As there is × in (5, 4). We declare that pair (3, 4) is not equivalent. Hence mark × **for pair (3, 4).**

ii) Consider pair (2, 4)

$\delta(2, a) = 4$, $\delta(2,b) = 1$
$\delta(4, a) = 4$, $\delta(4,b) = 4$

Consider pair (1,4)

As there is × in (1, 4). The pair (2, 4) is not equivalent. Hence mark × **for pair (2, 4).**

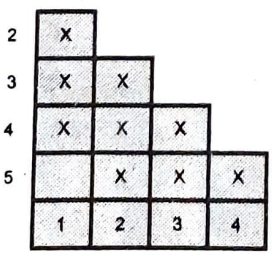
iii) Consider pair (2, 3)

$\delta(2, a) = 4$, $\delta(2, b) = 1$
$\delta(3, a) = 5$, $\delta(3, b) = 4$

We find cross X in these pairs

∴ Pair (2, 3) are not equivalent and put × in it.

Thus the table will be



Step 4 : Now consider pair (5, 1)

$\delta(5, a) = 3$ $\delta(5, b) = 2$

$\delta(1, a) = 3$ $\delta(1, b) = 2$

As both the states generate same next states for input symbols a and b.

The states 5 and 1 are equivalent. we can minimize DFA by assigning 1 for 5 and eliminating state 5.

The transition table for minimized DFA is –

State \ Input	a	b
1	3	2
2	4	1
3	1	4
4	4	4

2.11 Pumping Lemma for Regular Languages

SPPU : Dec.-06,08,12, May-06,07,09,12,13,14,16, Marks 8

This is a basic and important theorem used for checking whether given string is accepted by regular expression or not. In short, this lemma tells us whether given language is regular or not.

One key theme is that any language for which it is possible to design the finite automata is definitely the regular language.

Theorem : Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$ we can write z = u v w such that $|u\ v| \leq n$, $|v| \geq 1$ for all $i \geq 0$, $u\ v^i\ w$ is in L. The n should not be greater than the number of states.

Proof : If the language L is regular it is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$. With some particular number of states say, n. Consider the input can be $a_1, a_2, a_3, \ldots a_m$, $m \geq n$. The mapping function δ could be written as $\delta(q_0, q_1, q_2, q_3, \ldots q_i) = q_i$.

The transition diagram is as shown in Fig. 2.11.1.

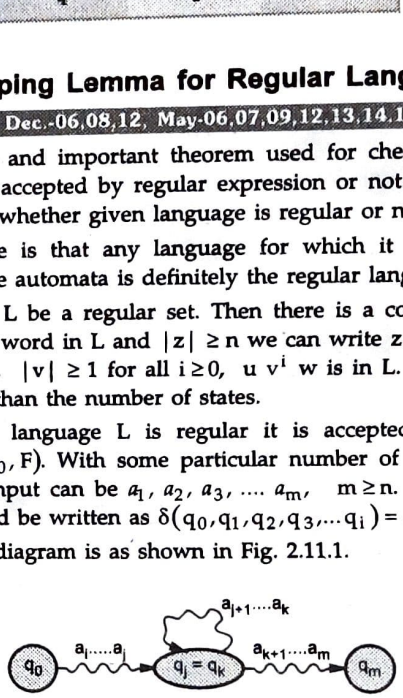


Fig. 2.11.1 Pumping lemma

If q_m is in F i.e. $q_1, q_2, q_3, \ldots q_m$ is in L(M) then $a_1, a_2, \ldots a_j\ a_{k+1}\ a_{k+2} \ldots\ldots a_m$ is also in L(M). Since there is path from q_0 to q_m that goes through q_j but not around the loop labelled $a_{j+1} \ldots a_k$. Thus

$$\delta(q_0, a_1, a_j\ a_{k+1} \ldots a_m) = \delta(\delta(q_0, q_1, \ldots q_j), a_{k+1} \ldots a_m)$$
$$= \delta(q_j, q_{k+1} \ldots q_m)$$
$$= \delta(q_k, q_{k+1} \ldots q_m)$$
$$= q_m$$

That is what we have proved i.e. given any long string can be accepted by FA, we should be able to find a substring near the beginning of the string that may be pumped i.e. repeated as many times as we like and resulting string may be accepted by FA.

The pumping lemma is used to check whether given language is **regular or not.**

2.11.1 Applications of Pumping Lemma

The pumping lemma is used to check whether given language is regular or not. We will discuss it with the help of some examples.

Example 2.11.1 *Show that the set* $L = \{b^{i^2} \mid i > 1\}$ *is not regular.*

SPPU : Dec.-12, Marks 6

Solution : We have to prove that the language $L = b^{i^2}$ is not regular. This language is such that number of b's is always a perfect square.

For example, if we take i = 1

$L = b^{1^2} = b$ the length $= 1^2$

$= b^{2^2} = bbbb$ length $= 2^2$

and so on.

Now let us consider

$L = b^{n^2}$ where length $= n^2$

it is denoted by z.

$|z| = n^2$

By pumping lemma $z = u\ v\ w$

where $1 \leq |v| \leq n$

As $z = u\ v^i\ w$ where i = 1

Now we will pump v i.e. make i = 2.

As we made i = 2 we have added one n^2.

$1 \leq |v| \leq n.$

$n^2 + 1 \leq |u\ v\ w| \leq n + n^2$

i.e. $n^2 + 1 \leq |u\ v\ w| \leq n^2 + n + n + 1$

i.e. $n^2 + 1 \leq |u\ v\ w| \leq (n + 1)^2$

$= n^2 \leq |u\ v\ w| \leq (n + 1)^2$

Thus the string lies between two consecutive perfect squares. But the string is not a perfect square. Hence we can say the given language is not regular.

For example,

$L = b^{i^2}$

Let $i = 2$

$L = bbbb$

$L = uvw$

Assume $uvw = bbbb$

Take $u = b$

$v = bb$

$w = b$

By pumping lemma, even if we pump v i.e. increase v then language should show the length as perfect square.

uvw

$= uv.vw$

$= bbbbbb$

$=$ length of b is not a perfect square

Thus the behaviour of the language is not regular, as after pumping something onto it does not show the same property (being square for this example.)

Example 2.11.2 *Is the following language regular ? Justify your answer* $L = \{0^{2n} \mid n \geq 1\}$.

Solution : This is a language in which length of string is always even.

i.e. $n = 1$; $L = 00$

$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$

$L = 0^{2n}$

$|z| = 2n = u\ v^i\ w$

If we add 2n to this string length.

$|z| = 4n = uv.vw$

$=$ even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Example 2.11.3 *Prove* $L = \{a^p \mid p$ *is a prime*$\}$ *is not regular.*

SPPU : Dec.-06, May-07, Marks 6, May-12, Marks 8

Solution : Let us assume L is a regular and p is a prime number.

$L = a^P$

$|z| = uvw$ $i = 1$

Now consider $L = u\ v^i\ w$ where $i = 2$

$= uv.vw$

Adding 1 to p we get,

$p < |uvvw|$

$p < p+1$

But p + 1 is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Example 2.11.4 *Show that* $L = \{0^n\ 1^{n+1} \mid n > 0\}$ *is not regular.*

Solution : Let us assume that L is a regular language.

$|z| = |uvw|$

$= 0^n\ 1^{n+1}$

Length of string $|z| = n + n + 1 = 2n + 1.$

That means length is always odd.

By pumping lemma

$= |uv.vw|$

That is if we add 2n + 1

$2n + 1 < (2n + 1) + 2n + 1$

$2n + 1 < 4n + 2$

But if n = 1 then we obtain 4n + 2 = 6 which is no way odd. Hence the language becomes irregular.

Even if we add 1 to the length of |z|, then

$|z| = 2n + 1 + 1 = 2n + 2$

$=$ even length of the string.

So this is not a regular language.

The simple way to know whether given language is regular or not is that try to draw **finite automata** for it, if you can easily draw the FA for the given L then that language is surely the regular otherwise not.

Example 2.11.5 *Show that set L = {0^i 1^i | i ≥ 1} is not regular.*

Solution : Assume that L = {0^i 1^i | i ≥ 1} is regular.

Let, w = 0^n 1^n such that |w| = 2n. By pumping lemma we can write

w = xyz such that |xy| ≤ n and |y| ≠ 0.

Now if xy^i z ∈ L then the language L is said to be regular.

There are many cases – i) y has only 0's ii) y has only 1's iii) y has both 0's and 1's.

i) If y has only 0's then the string

w = 0^{n-k} 1^n = xz since y = 0^k and i = 0

Surely n – k ≠ n. Hence xz ∉ L.

Hence our assumption of being L regular is wrong.

ii) If y has only 1's then, for i = 0 and y = 1^k

$$\therefore \quad w = xz = 0^n 1^{n-k}$$

As n ≠ n – k, xz = w ∉ L

Again L is not regular.

iii) If y has 0's and 1's then

$$w = 0^{n-k} 0^k 1^j 1^{n-j} = xy^i z$$

If i = 2 then

$$w = 0^{n-k} 0^{2k} 1^{2j} 1^{n-j} \notin L$$

Hence from all these 3 cases it is clear that language L is not regular.

Example 2.11.6 *Use pumping lemma to show whether or not the following languages are regular :*

i) $A_1 = \{www \mid w \in (a, b)^*\}$ ii) $A_2 = \{a^{2n} \mid n \geq 0\}$.

Solution : i) Let,

$A_1 = \{www \mid w \in (a, b)^*\}$ is assumed to be a regular language. Then by pumping lemma the string w = xyz ∈ A_1 such that

$$w = xy^i z$$

Here $|w| = |www| = 3n$ where n > 0

By pumping lemma if

$$w = xy^i z$$

$$= ww^i w$$

$$|w| = > 3n$$

That means w ∉ L. This shows that given language is not regular. Hence our assumption is wrong.

ii) $A_2 = \{a^{2n} \mid n \geq 0\}$ is a language in which length of string is always even.

i.e. If n = 1 then A_2 = aa

If n = 2 then A_2 = aaaa and so on

If w = xyz then

$$|w| = a^{2n} = xy^i z$$

Then |w| > 2n. That means we may get a string of odd length. Hence given language is not regular.

Example 2.11.7 *Let L be any subset of 0*. Prove that L* is regular.*

SPPU : May-07, 14, Marks 4

Solution : Let L ⊆ 0* i.e. L can be $\{\varepsilon, 0, 00, 000 \ldots\}$

Now we will apply pumping lemma to determine whether L is subset of 0*.

Assume w = xyz such that $|xy| \leq n |y| \geq 0$

If we consider string w = 0000

We will map w = xyz to 0000

By pumping lemma, if w = xy^i z ∈ L the L is regular. If i = 2 then

i.e. w ∈ L

This proves that string w is regular. Hence if L is subset of 0* then L* is regular is proved.

Example 2.11.8 *Using pumping lemma for regular sets prove that the language L = {a^m b^n | m > n is not regular}* **SPPU : Dec.-08, Marks 6**

Solution : Let us assume

$$L = \{a^m b^n \mid m > n\} \text{ is regular.}$$

As m > n, if we assume m = n + 1 then

$$L = a^{n+1} b^n = a^n a b^n$$

By pumping lemma L ∈ W such that

$$W = xyz$$

If we map $L = a^n a b^n$ to W then

$$W = a^n \; a \; b^n \text{. Assume } x = a^n \quad y = ab, \quad z = b^{n-}$$

If $W = xy^i z$ with i = 2 then

$$W = a^n a b a b b^{n-1} \notin L.$$

$$W = a^n b^n \notin L$$

This proves that our assumption of L being regular is wrong. Given L is not regular.

Example 2.11.9 *Prove that the following is not regular languages the set of strings of the form 0^i 1^j such that the greatest common divisor of i and j is 1.*

Solution : Let L = 0^i 1^j such that gcd (i, j) = 1.

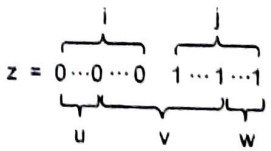
We assume that this is a regular language. Now consider three cases -

Case 1 : Let

$$z = \overbrace{0 \ldots 0}^{i} \overbrace{1 \ldots 1}^{j} \text{ be a string} \in L.$$

According to pumping lemma if z = uvw is a string ∈ L. then $z = uv^i w \in L$. That means even if we pump some substring the string z ∈ L.

We map z = uvw as



This is a case in which v consists of substring $0^n 1^n$.

By pumping lemma z = $uv^i w$ if i = 2, z = uvvw

Then we get

$$z = 0^{i-m}(01)^m 1^{j-m} \notin L = \{0^i 1^j\}$$

Case 2 : Let,

$$z = \underbrace{\overbrace{0 \ldots 0}^{i}}_{uv} \underbrace{\overbrace{1 \ldots 1}^{j}}_{w} \in L$$

By pumping lemma z = $uv^i w$. If i = 2 then,

$$z = uvvw$$

i.e.

$$z = 0^{i+m} 1^j$$

But gcd (i+m, j) ≠ 1.

Hence $z \notin [L = \{0^i 1^j\}]$

Case 3 : Let,

$$z = \overbrace{0 \ldots 0}^{i}\overbrace{1 \ldots 1}^{j}$$

$$u \quad vw$$

By pumping lemma, $z = uv^i w$. If i = 2 then,

$$z = uvvw$$

i.e. $z = 0^i 1^{j+m}$

But gcd (i, j+m) ≠ 1.

Hence $z \notin [L = \{0^i 1^j\}]$

From above 3 cases we can prove that $L = \{0^i 1^j \mid \gcd(i, j) = 1$ is not a regular language.

Example 2.11.10 *Prove or disprove that the language L given by L = { $a^m b^n$ | m ≠ n, m and n are positive integers} is regular.*

Solution : Let L = { $a^m b^n$ | m ≠ n} be a language. Assume that L is regular language. Now consider following case -

Case 1 : Assume z = *aaa*bbbb ∈ L

By pumping lemma if z = uvw then if we pump some strings to make z = $uv^i w$ then if z ∈ L then such a language is called regular.

Let,

$$z = \underset{\downarrow}{a}\ \underbrace{a\ a\ b\ b}_{\downarrow}\ \underbrace{b\ \ b}_{\downarrow}$$

$$u \quad v \quad w$$

if z = $uv^i w$ and if i = 2 then z = uvvw

$$\therefore \quad z = a\ \underbrace{(a\ a\ b\ b)}_{\downarrow}\ \underbrace{(a\ a\ b\ b)}_{\downarrow}\ \underbrace{b\ \ b}_{\downarrow}$$

$$u \quad v \quad v \quad w$$

$$z = a^3 b^2 a^2 b^4 \notin a^m b^n \notin L$$

Case 2 : Assume L ∈ z such that

$$z = \underset{\downarrow}{a}\ \underbrace{a}_{\downarrow}\ \underbrace{a}_{\downarrow}\ \underbrace{a\,b\,b\,b\,b\,b\,b}_{\downarrow}$$

$$u \quad v \quad w$$

By pumping lemma, $z = uv^i w \in L$ for a regular language. If i = 2 then,

$$z = uvvw$$

$$\therefore \quad z = \underset{\downarrow}{a}\ \underbrace{a}_{\downarrow}\ \underbrace{a}_{\downarrow}\ \underbrace{a}_{\downarrow}\ \underbrace{a\,b\,b\,b\,b\,b}_{\downarrow}$$

$$u \quad v \quad v \quad w$$

$$z = a^6 b^6 \in a^m b^n \text{ because } m = n.$$

From these case we get $z \notin L$. Thus our assumption of L being regular is wrong. Hence given language { $L = a^m b^n \mid m \neq n$ } is not a regular.

Example 2.11.11 *Show that L = {ww / w ∈ {a,b}} is not regular.*

SPPU : Dec.-12, Marks 6

Solution : Assume that the language L is regular. This also means that there exists some DFA with n states.

If we consider $w = a^n b$ i.e. string of {a, b}

then $ww = a^n b a^n b$ is in L

i.e. $|ww| = 2n + 2 > n$

by pumping lemma, we can write

$ww = xyz$ with $|y| = 0$ $|xy| \leq n$

iii) For checking regularity we need to find whether $xy^i z \in L$ or not.

Case 1 : $w = xy^i z$

we assume i = 0. This is a case with y having no b's.

Then $w = xy$ i.e. $y = a^k$

Y containing no b's at all

$$w = a^m b a^n b$$

where $m = n - k$

Then we can not write xz in the form of ww. Because w belongs to set {a, b} *.

Case 2 : This is a case with y having only one b.

Then assume

$$w = xy^i z$$

These w = xz with only one b.

But then $xz \notin L$ as in language L there should be even number of a's and even number of b's. [we have assumed |ww| = 2n + 2].

Thus from case 1 and case 2 we get the contradictions. So L is not regular.

Example 2.11.12 *Define Pumping Lemma and apply it to prove the following :*

$L = \{0^m 1^n 0^{m+n} \mid m >= 1$ and $n >= 1\}$ is not regular.

SPPU : May-09, 16, Marks 6

Solution : Assume that L is regular by pumping lemma, we can write w = xyz with $|y| = 0$ $|xy| \leq n$

For checking the regularity we need to find whether $xy^i z \in L$ or not.

$$w = xyz$$

$$= 0^m 1^n 0^{m+n}$$

Case 1 : If we assume y has only 1's i.e.

$$w = \underset{\downarrow}{0^m}\ \underset{\downarrow}{1^n}\ \underset{\downarrow}{0^{m+n}} \text{ then with i = 2,}$$

$$x \quad y \quad z$$

$w = xy^i z$ becomes xyyz

i.e. $= 0^m 1^n 1^n 0^{m+n}$

$$w = 0^m 1^{2n} 0^{m+n} \notin L$$

Case 2 : If we assume y is set of x y 01 i.e.

$$w = \underset{\downarrow}{0^{m-k}}\ \underbrace{0^k\ \ 1^{n-p}}_{\downarrow}\ \underbrace{1^p\ \ 0^{m+n}}_{\downarrow}$$

$$x \quad y \quad z$$

then with i = 2 $w = xyyz$

i.e. $w = 0^{m-k} (0^k 1^{n-p}) (0^k 1^{n-p}) 1^p 0^{m+n}$

$$w = 0^{m-k} 0^{2k} 1^{2n-2p} 1^p 0^{m+n}$$

$$= 0^{m-k} 1^{2n-p} 0^{m+n} \notin L$$

From above two cases by applying pumping lemma i.e. by using

$$w = xy^i z \notin L$$

Hence our assumption of L being regular is wrong.

∴ Given L is not regular.

Example 2.11.13 *Find whether the language is regular or not*

$L = \{WW^R \in \{a,b\}\}$

Solution : Consider, $L = \{w \in (a, b) \mid w = w^R\}$

Consider $w = abab$

$w^R = baba$

Hence $L = ww^R$

The string z can be denoted by

$z = uvw$. where $|z| \geq n$ and $|uv| \leq n$ $|v| \geq 1$.

We assume $z = \underbrace{ab}_{n}\underbrace{ab}_{n}\underbrace{ba}_{n}\underbrace{ba}_{n}$

We can split $z = uvw$ as

$z = \underbrace{a}_{u}\underbrace{b}_{1}\underbrace{ab\,ba\,ba}_{w}$

$|u| = n-1$

$|v| = 1$

i.e. $|uv| = |u| + |v|$

$= n - 1 + 1$

$= n$

Now according to pumping lemma,

when $z = uv^iw \in L$ then language is said to be regular. We will assume that $L = ww^R$ is regular. And we will find uv^iw.

$\therefore$ $z = uv^iw = ababbaba$

Assume $i = 0$ i.e. there will be

$z = uv^0w = uw$

Then the string becomes

$z = aabbaba$

$\neq ww^R$

i.e. $z = aabbaba \notin L$ ...(1)

Now consider $i = 2$ then

$z = uv^2w$

$= abbabbaba$

$\neq ww^R$

i.e. $z = abbabbaba \notin L$ ...(2)

From equations (1) and (2) we can state that

$z = uv^iw \notin L$

Hence our assumption that $L = ww^R$ being regular is wrong. Hence we can prove that

$z = ww^R$ is not regular.

Review Questions

1. *Explain the property pumping lemma of regular set. Where do we apply this property ? Give the suitable example.* **SPPU : May-06, Marks 6**
2. *State pumping lemma for regular sets.* **SPPU : May-13, Marks 2**

2.12 Closure Properties of Regular Languages

If certain languages are regular and language L is formed from them by certain operations (such as union or concatenation) then L is also regular. These properties are called closure properties of regular languages. Such languages represent the class of regular laguages which is closed under the certain specific operations.

The closure properties express the idea that when one or many languages are regular then certain related languages are also regular. The closure properties of regular languages are as given below.

1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular languages is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular languages is regular.
6. The closure operation on a regular language is regular.
7. The concatenation of regular language is regular.
8. A homomorphism of regular languages is regular.
9. The inverse homomorphism of regular language is regular.

Theorem 1 : If L_1 and L_2 are two languages then $L_1 \cup L_2$ is regular.

Proof : If L_1 and L_2 are regular then they have regular expression $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cup L_2 = L(R_1 + R_2)$ thus we get $L_1 \cup L_2$ as regular language. (any language given by some regular expression is regular).

Theorem 2 : The complement of regular language is regular.

Proof : Consider L_1 be regular language which is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$. The complement of regular language is $\overline{L}_1$ which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$. That means M is a DFA with final states $\in$ F and M' is a DFA in which all the non-final states of M become final. In other words, we can say that the strings that are accepted by M are rejected by M' similarly, the strings rejected by M are accepted by M'.

Thus as $\overline{L}_1$ is accepted by DFA M', it is regular.

Theorem 3 : If L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular.

Proof : Consider that languages L_1 is regular. That means there exists some DFA M_1 that accepts L_1. We can write $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Similarly being L_2 regular there is another DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Let, L be the language obtained from $L_1 \cap L_2$. We can the simulate $M = (Q, \Sigma, \delta, q, F)$.

where $Q = Q_1 \cap Q_2$

$\delta = \delta_1 \cap \delta_2$ a mapping function derived from both the DFAS.

$q \in Q$ which is initial state of machine M.

$F = F_1 \cap F_2$, the set of final states, which is common for M_1 and M_2 both.

It is clear that there exists some DFA which accepts $L_1 \cap L_2$ i.e. L Hence L is a regular language. This proves that if L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular. In other words the regular language is closed under intersection.

Theorem 4 : If L_1 and L_2 are two regular languages then $L_1 - L_2$ is regular.

Proof : The $L_1 - L_2$ can also be denoted as $L_1 \cup \overline{L}_2$

Consider L_1 be regular language which is accepted by DFA $M = (Q, \Sigma, \delta, q_0, F)$. The complement of regular language L_1 is $\overline{L}_1$ which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$. That means M is a DFA with final state's set F and M' is a DFA in which all the non

final states of M become final states and all the final states of M become non-final states. Thus L_1 and $\overline{L}_2$ are two regular languages. That also means : these languages are accepted by regular expressions. If $L_1 = L(R_1)$ and $\overline{L}_2 = L(R_2)$. Then $L_1 \cup \overline{L}_2 = L(R_1 + R_2)$. This ultimately shows that $L_1 \cup \overline{L}_2$ is regular. In other words $L_1 - L_2$ is regular. Thus regular languages are closed under difference.

Theorem 5 : The reversal of a regular languages is regular.

Proof : Reversal of a string means obtaining a string which is written from backward that is w^R is denoted as reversal of string w. That means $L(w^R) = (L(w))^R$. This proof can be done with basis of induction.

Basis : If $w = \varepsilon$ or ϕ then w^R is also ε or ϕ

i.e. $(\varepsilon)^R = \varepsilon$ and $(\phi)^R = \phi$

Hence $L(w^R)$ is also regular.

Induction :

Case 1 : If $w = w_1 + w_2$ then

$$w = (w_1)^R + (w)^R$$

As the regular language is closed under union. Then w is also regular.

Case 2 : If $w = w_1 w_2$

Consider $w_1 =$ (ab, bb)

and $w_2 =$ (bbba, aa)

The $w_1^R =$ (ba, aa)

$w_2^R =$ (aaab, bb) then

$w = w_1^R w_2^R$

(ba, aa, aaab, bb) is also regular. Thus given language is regular one.

Theorem 6 : The closure operation on a regular language is regular.

Proof : If language L_1 is regular then it can be expressed as $L_1 = L(R_1^*)$. Thus for a closure operation a language can be expressed as a language of regular expressions. Hence L_1 is said to be a regular language.

Theorem 7 : If L_1 and L_2 are two languages then $L_1 \cdot L_2$ is regular. In other words regular languages are closed under concatenation.

Proof : If L_1 and L_2 are regular then they can be expressed as $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cdot L_2 = L(R_1 \cdot R_2)$ thus we get a regular language. Hence it is proved that regular languages are closed under concatenation.

Theorem 8 : A homomorphism of regular languages is regular.

Proof : The term homomorphism means substitution of string by some other symbols.

For instance the string " aabb" can be written as 0011 under homomorphism. Clearly here, a is replaced by 0 and b is replaced by 1. Let $\sum$ is the set of input alphabets and Γ be the set of substitution symbols then $\sum^* \rightarrow \Gamma^*$ is homomorphism. The definition of homomorphism can be extended as

Let, $w = a_1 a_2 \ldots a_n$

$$h(w) = h(a_1)h(a_2)\ldots h(a_n)$$

If L is a language that belongs to set $\sum$, then the **homomorphic image** of L can be defined as :

$$h(L) = \{h(w) : w \in L\}$$

To prove that if L is regular h(L) is also regular consider following example -

Let, $\sum = \{a, b\}$ and w = abab

Let $h(a) = 00$

and $h(b) = 11$

Then we can write h(w) = h(a) h(b) h(a) h(b)

= 00110011

The homomorphism to language is applied by applying homomorphism on each string of language.

$\therefore$ If L = ab*b then,

L = {ab, abb, abbb, abbbb, ...}

Now h(L) = {0011, 001111, 00111111, 0011111111, ...}

$\therefore$ $h(L) = 00\,(11)^+$ As it can be represented by a regular expression, it is a regular language. Hence it is proved that if L is regular then h(L) is also regular. In other words, family of regular languages is closed under homomorphism.

Theorem 9 : The inverse homomorphism of regular language is regular.

Proof : Let $\sum^* \rightarrow \Gamma^*$ is homomorphism.

The $\sum$ is the input set and Γ be the substitution symbols used by homomorphic function.

Let, L be the regular language where $L \in \sum$, then h(L) be homomorphic language.

The inverse homomorphic language can be represented as $h^{-1}(L)$

Let,

$$h^{-1}(L) = \{w \mid w \in L\}$$

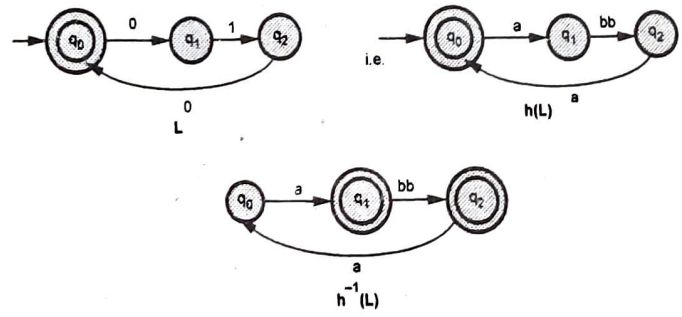
If L is regular then h(L) is also regular because regular language is closed under homomorphism. That if there exist a FA M = $(Q, \in, \delta, q_0, F)$ which accepts L then h(L) must also be accepted by FA M. For complement of L i.e. language L' the inverse homomorphic language is $h^{-1}(L)$. Let M' be the FA in which all the final states of M become non-final states and all the non-final states of M become the final states. Clearly the language L' can be accepted by M' Hence $h^{-1}(L)$ must also be accepted by FA M'.

For example - Let L = (010)* be a regular language. And L = {010, 010010,...}

Let h(0) = a and h(1) = bb be homomorphic function. Then

h(L) = (abba)*

This L can be represented by following DFA.



Thus there exists a FA which accepts $h^{-1}(1)$. This shows that, inverse homomorphism of regular language is regular.

2.13 Decision Properties of Regular Languages

Following is the set of questions for determining the decision properties of regular languages.

1. Is a particular string w belongs to some language L ? **[Membership Property]**

   Let L be a language which can be represented by DFA. If a string w is chosen for simulating this DFA processing and then if w reaches to an accept state starting from the start state then it is said that.

2. Is the given language empty ? **[Emptiness property]**

   The regular language is modelled by a finite automata to decide whether it is empty or not. If there exists some path from start state to accept state for deriving such language then the given language is non empty. But if accepting states are separated from start state or in other words if there is no path reaching to an accept state from a start state while deriving a given language then such a language is called empty language.

3. Do the two descriptions of a language represent the same language ? or whether two languages are equivalent ? **[Equivalence property]**

   Any regular language can be represented by NFA, NFA with DFA or by regular expression. These representations can be interconverted. This property is called equivalence of corresponding two models. For instance : NFA to DFA, DFA to r.e. and so on. If two different descriptions can be modelled by two equivalent but different models, the one can decide whether they are describing the same language or not.

2.14 Limitation of FA

SPPU : Aug.-15, Marks 2

The main limitation of FA is that it can not remember arbitrarily long input sequence because it does not contain any memory. Hence it can not solve following types of problems -

1. Checking well formedness of parenthesis
2. Checking palindrome condition of given language.

Thus FA cannot accept non regular and context free languages. It can accept only regular languages.

Review Question

1. *What are the limitations of finite automata? Justify with suitable examples* SPPU : Aug.-15, In Sem, Marks 2

2.15 Case Study : RE in Text Search and Replace

SPPU : Dec.-05, 06, 10,13,14, May-13,14, Marks 6

1. Regular expressions in UNIX

There are various UNIX notations used for regular expressions. These notations have many additional capabilities and features. These notations have ability to name and refer previous strings that have a matched pattern. This ability helps in recognizing nonregular languages. UNIX regular expressions allow to write **character classes.** There are some rules for these character classes which are as given below -

1) The symbol · stands for any single character.
2) The sequence [1, 2, 3, ... 10] means 1+2+3+..+10.
3) The range of characters can be specified using square brackets.

   For example : [a-z] denotes set of lower case letters.
4) For matching with some special character a backslash is used.

   For example : \- means match with character –.
5) Special notations can be used to represent some common class of characters.

   For example : [:digit:] represents the class [0-9]. The [:alpha:] is same as [A-Za-Z].
6) The operator | is used to denote union.
7) ? is used to denote preceding zero or single character.

   For example : - ? [0-9] means the number can be positive or negative number.
8) + is used to denote **one or more** occurrences.
9) * is used to denote **zero or more** occurrences.
10) {n} means **n copies of.**

    For example : a {3} means aaa.

2. Lexical analyzers

Compiler uses this program of lexical analyzer in the process of compilation. The task of lexical analyzer is to scan the input program and separate out the 'tokens'.

For example : Identifier is a category of token in the source language and it can be identified by regular expression as

(letter) (letter + digit)*

If anything in the source language matches with this regular expression then it is recognized as identifier. The letter is nothing bit a set {A, B, ... Z, a, b, ... z} * and digit is {0, 1, ... 9} *. The regular expression is effective way for identifying token from a language.

We can represent the above regular expression using a finite automata as follows -

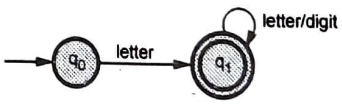


Fig. 2.15.1

3. Finding patterns in text

Regular expressions are useful notations used for finding the patterns from given text. A large class of pattern can be identified by regular expression.

For example : The strings ending with digits is a pattern in the given text which can be recognized by following regular expression -

[a-zA-Z] + [0-9] +

Thus use of regular expressions for identifying patterns from given text makes the job of compiler more simplified.

In web applications also the pattern matching is done using regular expressions.

4. GREP utility in Unix

In Unix there is a GREP utility for searching the desired pattern in the file. For searching the pattern from the Grep makes use of regular expressions. When particular string is present in the file, then using grep we get the line containing that string from the file.

For example : If we type the grep command on Unix prompt then we will get the output as follows $ grep hello test.txt.
hello friends

```
$ grep  b.g   test.txt.
        big brother  ←  these are the lines
                bug  ←  present in test.txt.
        bag of books ←  file containing
              bigger ←  b[any single character] g
```

Here dot stands for matching with one character.

Review Questions

1. *Explain the following applications of regular expression.*
   i) GREP utilities in unix. ii) Finding pattern in text.
   SPPU : Dec.-05, 06, May-14, Marks 6
2. *Explain the use of regular expressions in unix with any one example.*
   SPPU : Dec.-10, Marks 4
3. *Explain the applications of regular expressions in lexical analysis phase of compiler.*
   SPPU : Dec.-13, May-13, Marks 6
4. *State applications of Regular Expression*
   SPPU : Dec.-14 ,End-Sem, Marks 4

3
Context Free Grammars (CFG) and Languages

Syllabus

Introduction, Regular Grammar, ***Context Free Grammar*** *- Definition, Derivation, Language of grammar, sentential form, parse tree, inference, derivation, parse trees, ambiguity in grammar and Language- ambiguous Grammar,* ***Simplification of CFG****: Eliminating unit productions, useless production, useless symbols, and ε-productions, Normal forms- Chomsky normal form, Greibach normal form, Closure properties of CFL, Decision properties of CFL, Chomsky Hierarchy,* ***Application of CFG:*** *Parser, Markup languages, XML and Document Type Definitions.*
Case Study- CFG for Palindromes, Parenthesis Match,

Contents

3.1 Introduction

A grammar is a 4 tuple G = (V,T,P,S) where V is a finite set of **non terminal symbols**, T is a finite set of **terminal symbols**, P is a set of **production rules** and S is a **start symbol**.

The P is of the form A->B where A and B ∈ (VUT)*.

The grammar is normally represented using derivation tree.

For example

The production rules for defining natural language are as follows -

Sentence → Noun verb

Noun → Rama | Sita | Gopal

Verb → sings | eats | dances

A derivation tree can also be built using the production rules.

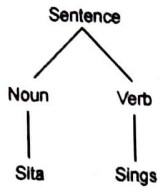


Fig. 3.1.1 Phrase tree for the string Sita Sings

3.2 Regular Grammar

SPPU : Dec.-14, Aug.-15, May-16, Marks 8

A regular grammar is defined as

G = (V, T, P, S) where

V is set of symbols called **non terminals** which are used to define the rules.

T is a set of symbols called **terminals**.

P is a set of **production rules**.

S is a **start symbol** which ∈ v.

The production rules P are of the form.

$$A \rightarrow aB$$

$$A \rightarrow a$$

Where A and B are non terminal symbols, and a is terminal symbol.

There are two types of Regular Grammar -

i) **Left Linear Grammar**

ii) **Right Linear Grammar**

3.2.1 Left Linear and Right Linear Grammar

If the non terminal symbol appears as a rightmost symbol in each production of regular grammar then it is called **right linear grammar.** The right linear grammar is of following form

$$A \rightarrow aB$$

$$A \rightarrow a$$

$$A \rightarrow \varepsilon$$

Where A and B are non terminal symbols and 'a' is a terminal symbol.

If the non terminal symbol appears as a leftmost symbol in each production of regular grammar then it is called **left linear grammar.** The left linear grammar is of following form.

$$A \rightarrow Ba$$

$$A \rightarrow a$$

$$A \rightarrow \varepsilon$$

Where A and B are non terminal symbols and 'a' is a terminal symbol.

3.2.2 Conversion from Right Linear Grammar to Left Linear Grammar

The given right linear grammar can be converted to left linear using following steps.

1) Construct transition graph for given grammar.

2) Interchange initial and final states.

3) Reverse the directions of all the edges in the graph.

4) Write left linear grammar for this newly formed transition graph.

Example 3.2.1 *Convert following right linear grammar to its equivalent left linear grammar.*

$S \rightarrow 0A \mid 1B$

$A \rightarrow 0C \mid 1A \mid 0$

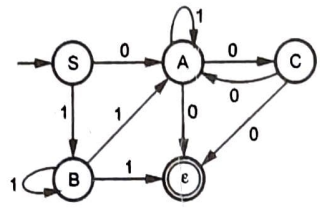
$B \rightarrow 1B \mid 1A \mid 1$

$C \rightarrow 0 \mid 0A$

SPPU : Dec.-14, In sem, Marks 6

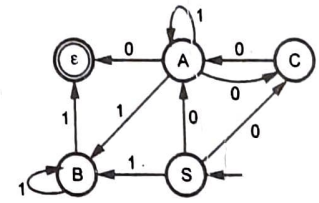
Solution :

Step 1 : We will construct transition graph for given grammar.



Step 2 : Interchange initial and final states.

Step 3 : Reverse the directions of all the edges in the graph.



Step 4 : The left linear grammar will be

$S \rightarrow A0 \mid C0 \mid B1$

$A \rightarrow A1 \mid B1 \mid C0 \mid 0$

$B \rightarrow B1 \mid 1$

$C \rightarrow \mid A0$

Example 3.2.2 *Convert the given right-linear grammar to its equivalent left-linear form :*

$S -> aA|dB$

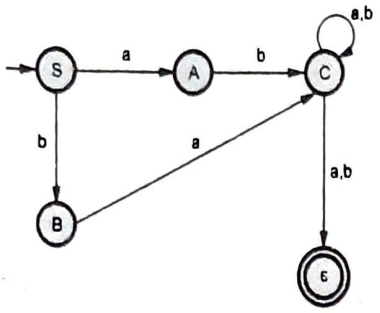
$A -> bC$

$B -> aC$

$C -> aC|bC|a|b$

SPPU : Aug.-15, In Sem, Marks 6

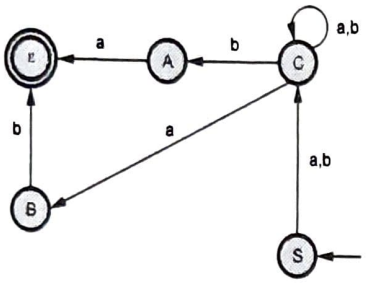
Solution : We will first design DFA from given grammar



Now i) Reverse initial and final states

ii) Reverse the directions of all edges.

The DFA will be



The left linear grammar will be

$S \rightarrow CA|Cb$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow Ca|Cb|Ba|Ab$

3.2.3 Conversion from Left Linear Grammar to Right Linear Grammar

The steps to convert left linear grammar to right linear grammar are.

1) Construct transition graph for given grammar.
2) Interchange initial and final states.
3) Reverse the directions of all the edges in the graph.
4) Write right linear grammar for newly formed transition graph.

Example 3.2.3 *Convert the given left linear grammar to right linear grammar.*

S→A0 | C0 | B1

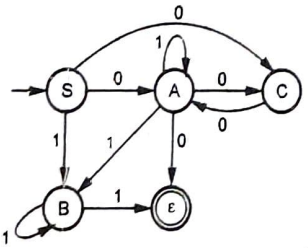
A→A1 | B1 | C0 | 0

B→B1 | 1

C→ A0

Solution :

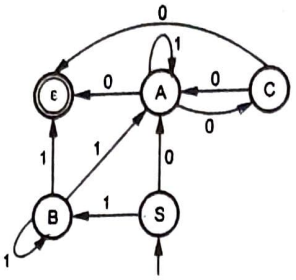
Step 1 :



Step 2 : Now interchange initial and final states.

Step 3 : Reverse the directions of all the edges in the graph.

The transition graph will be



Step 4 : The right linear grammar will be

$S \rightarrow 1B \mid 0A$

$A \rightarrow 0C \mid 1A \mid 0$

$B \rightarrow 1B \mid 1A \mid 1$

$C \rightarrow 0A \mid 0$

Example 3.2.4 *Give the Right and Left linear grammar for the following DFA shown in Fig. 3.2.1.* **SPPU : May 2016, Marks 8**

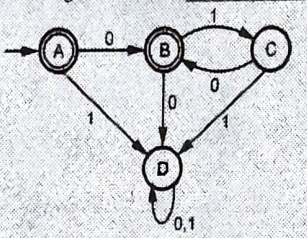


Fig. 3.2.1

Solution :

I) Right Linear Grammar

Let, G = (V, T, P, S) be the context free grammar

V = (A, B, C, D) A, is a start state.

T = (0, 1)

$A \rightarrow 0B \mid 1D$		$A \rightarrow 0B$
$A \rightarrow 0 \mid \varepsilon$		$A \rightarrow 0 \mid \varepsilon$
$B \rightarrow 1C$		$B \rightarrow 1C$
$B \rightarrow 0D$	By eliminating Useless Production	$C \rightarrow 0C$
$C \rightarrow 0C$		$C \rightarrow 0$
$C \rightarrow 0$		
$C \rightarrow 1D$		
$D \rightarrow 0D \mid 1D$		

II) Left Linear grammar

Step 1 : We will interchange initial and final state.

Step 2 : We will reverse the directions of all the edges in the graph. The resultant transition graph will as shown in Fig. 3.2.2.

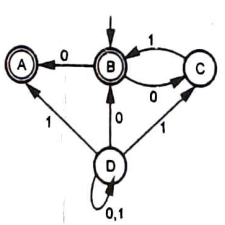


Fig. 3.2.2

Step 3 : The left linear grammar will be

$$B \rightarrow 0A \mid C0 \mid 0$$
$$A \rightarrow \varepsilon$$
$$C \rightarrow B1 \mid 1$$
$$D \rightarrow A1 \mid B0 \mid C1 \mid D0 \mid D1$$

3.3 Interconversion between FA and Regular Grammar

SPPU : May-16, Marks 8

3.3.1 FA to Regular Grammar

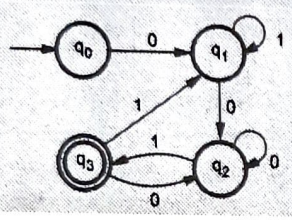
Method for conversion from FA to Regular Grammar

- Conversion from FA to RG is as follows -

$$G = (\{A_0, A_1, \ldots A_n\}, \Sigma, P, A_0)$$

- Where P the set of production rules can be defined by following rules.
  1. $A_i \rightarrow a A_j$ is a production rule if $\delta(q_i, a) = q_j$, where $q_j \notin F$
  2. $A_i \rightarrow a A_j$ and $A_i \rightarrow a$ are production rules if $\delta(q_i, a) = a_j$ where $q_j \in F$.

Example 3.3.1 *Construct a regular grammar for*



Solution : The equivalent regular grammar can be denoted by G = (V, T, P, S) where

$$V = \{A_0, A_1, A_2, A_3\}$$

We assume nonterminals A_0, A_1, A_2, A_3 for the states q_0, q_1, q_2, q_3 respectively.

The production rules can be

$$A_0 \rightarrow 0A_1$$
$$A_1 \rightarrow 1A_1$$
$$A_1 \rightarrow 0A_2$$
$$A_2 \rightarrow 0A_2$$
$$A_2 \rightarrow 1A_3$$
$$A_2 \rightarrow 1$$
$$A_3 \rightarrow 1A_1$$
$$A_3 \rightarrow 0A_2$$

3.3.2 Regular Grammar to FA

- If there is a production of the form $A_i \rightarrow a$, the corresponding transition terminates at a new state. This is the unique final state. Thus the DFA M can be

$$M = (\{q_0, q_1, \ldots, q_n, q_f\}, \Sigma, \delta, q_0, \{q_f\})$$

- The δ is defined as
  i) Each production $A_i \rightarrow a A_j$ induces a transition from q_i to q_j with label a , on the edge.
  ii) Each production $A_k \rightarrow a$ induces a transition from q_k to q_f with label a , on the edge.

Example 3.3.2 *Let, $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$ Where the production rules are $P = \{ A_0 \rightarrow a A_1$*

$$A_1 \rightarrow b A_1$$
$$A_1 \rightarrow a$$
$$A_1 \rightarrow b A_0$$
$$\}$$

Construct FA equivalent to G.

Solution :

Step 1 : We will construct FA M as

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

Where q_0 and q_2 are initial and final states.

Step 2 :

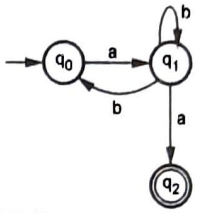
$A_0 \rightarrow a A_1$ means transition from q_0 to q_1 on input a.

$A_1 \rightarrow bA_1$ means transition from q_1 to q_1 on input b.

$A_1 \rightarrow a$ means transition from q_1 to q_2 on input a.

$A_1 \rightarrow bA_0$ means transition from A_1 to A_0 on input b.

The FA can then be drawn as follows.



Example 3.3.3 *Construct DFA for the following left linear grammar*

$S \rightarrow B1 \mid A0 \mid C0$

$B \rightarrow B1 \mid 1$

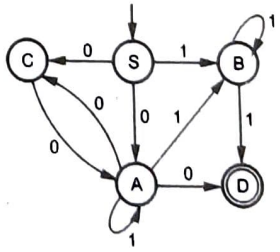
$A \rightarrow A1 \mid B1 \mid C0 \mid 0$

$C \rightarrow A0$

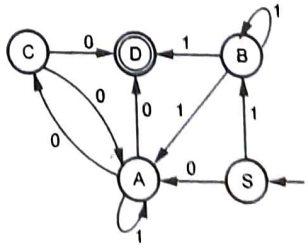
SPPU : May-16, Marks 8

Solution :

Step 1 : We will create transition diagram for given set of production rules. For handing the productions $B \rightarrow 1$ and $A \rightarrow 0$ we will also add one more state D.



Step 2 : Interchange the start and final state and interchange all the directions.



Step 3 : The transition table for above transition graph will be

State \ Input	0	1
→ S	A	B
A	C,D	A
B	ϕ	A,B,D
C	A,D	ϕ
* D	ϕ	ϕ

$\delta(S,0)$ = {A} = [A] state

$\delta(S,1)$ = {B} = [B] state

$\delta(A,0)$ = {C, D} = **[C, D] new state**

$\delta(A,1)$ = {A} = [A]

$\delta(B,0)$ = ϕ

$\delta(B,1)$ = ϕ {A,B,D} = **[A,B,D] new state**

$\delta(C,0)$ = {A,D} = **[A,D] new state**

$\delta(C,1)$ = ϕ

$\delta(D,0)$ = $\delta(D,1)$ = ϕ

The δ transitions on new states can be applied

$\delta([C,D],0)$ = [A,D]

$\delta([C,D],1)$ = ϕ

$\delta([A, B, D],0)$ = [C,D]

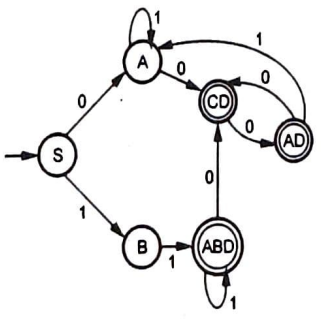
$\delta([A, B, D],1)$ = [A,B,D]

$\delta([A,D],0)$ = [C,D]

$\delta([A,D],1)$ = [A]

As no new state is generated we will stop applying δ transitions.

State \ i/P	0	1
[S]	[A]	[B]
[A]	[C,D]	[A]
[B]	ϕ	[A,B,D]
[C]	[A,D]	ϕ
[D]	ϕ	ϕ
[C,D]	[A,D]	ϕ
[A,B,D]	[C,D]	[A,B,D]
[A,D]	[C,D]	[A]



3.4 Context Free Grammar- Definition

The context free grammar can be formally defined as a set denoted by G = (V,T,P,S) where V and T are set of non terminals and terminals respectively. P is set of production rules, where each production rule is in the form of

Non terminal → Non terminals

or Non terminal → Terminals

S is a start symbol.

For example

P = { S → S + S
S → S * S
S → (S)
S → 4 }

If the language is 4 + 4 * 4 then we can use the production rules given by P. The start symbol is S. The number of non terminals in the rules P is one and the only non terminal i.e. S. The terminals are +, *, (,) and 4.

We are using following conventions.

1. The capital letters are used to denote the non terminals.

2. The lower case letters are used to denote the terminals.

Example : The formation of production rules for checking syntax of any English statement is

SENTENCE → NOUN VERB

NOUN → Rama | Sita | Gopal

VERB → goes | writes | sings

Thus if want to derive a string "Rama sings" then we can follow the above rules.

3.5 Sentential Form

Consider G = (V, T, P, S) be a context free grammar then, we can derive a string w from it. This w can obtained from (VUT)* where V denotes the set of non-terminal symbols and T denotes the set of terminal symbols. The derivation of w from start symbol S can be written as $S \Rightarrow w$ which is called as **sentential form.** If $S \underset{lm}{\Rightarrow} w$ then w is a **left-sentential form.**

If $S \underset{rm}{\Rightarrow} w$ then w is a **right-sentential form.**

For example

Consider the grammar,

S → S + S | S * S | 4

Then for deriving the string w = 4 + 4 * 4 we use above grammar as -

$$S \underset{lm}{\Rightarrow} S * S \underset{lm}{\Rightarrow} S + S * S \underset{lm}{\Rightarrow} 4 + S * S \underset{lm}{\Rightarrow} 4 + 4 * S \underset{lm}{\Rightarrow} 4 + 4 * 4$$

Here S + S * S is called left-sentential form.

$$S \underset{rm}{\Rightarrow} S + S \underset{rm}{\Rightarrow} S + S * S \underset{rm}{\Rightarrow} S + S * 4 \underset{rm}{\Rightarrow} S + 4 * 4 \underset{rm}{\Rightarrow} 4 + 4 * 4$$

S + S * 4 is called right sentential form. Thus we can obtain the desired language L by certain rules. Let us now solve some examples for derivation of CFG.

3.6 Language of Grammar

SPPU : Dec.-05,06,07,08,09,10,13,14, May-08,09,10,13,14, Oct.-16, Marks 12

The language of grammar can be defined by creating the production rules for the given condition. The language generated by context free grammar is called as Context Free Language(CFL).

Let us understand the CFL with the help of various examples

Example 3.6.1 *Construct the CFG for the language having any number of a's over the set* $\Sigma = \{a\}$.

Solution : As we know the regular expression for above mentioned language is

r.e. = a^*

Let us build the production rules for the same,

$S \rightarrow aS$ rule 1

$S \rightarrow \varepsilon$ rule 1

Now if want "*aaaa*" string to be derived we can start with start symbols.

S

a S

a a S ∵ rule 1

a a a S rule 1

a a a a S rule 1

a a a a a S rule 1

a a a a a ε rule 2

= *a a a a a*

The r.e. = a^* suggest a set of {ε, *a*, *aa*, *aaa*, ...}. We can have a null string simply because S is a start symbol, and rule 2 gives S → ε.

Example 3.6.2 *Try to recognize the language L for given CFG.*

$G = [\{S\}, \{a,b\}, P, \{S\}]$

where $P = \begin{Bmatrix} S \rightarrow aSb \\ S \rightarrow ab \end{Bmatrix}$

Solution : Since S → *a* S b | *a* b is a rule. | indicates the 'or' operator.

$S \rightarrow aSb$

If this rule can be recursively applied then,

S

a S b

↓

a a S b b

↓

a a a S b b b

and if finally we can put S → *ab* then it becomes *aaaa* bbbb. Thus we can have any number of *a*'s first then equal number of b's following it. Hence we can guess the language as $\{L = a^n b^n \text{ where } n \geq 1\}$. The only way to recognize the language is to try out various strings from the given production rules. Simply by observing the derived strings, one can find out the language getting generated from given CFG.

Example 3.6.3 *Construct the CFG for the regular expression* $(0 + 1)^*$.

SPPU : Dec.-05, Marks 8

Solution : The CFG can be given by,

P = {S → 0S | 1S

S → ε }

The rules are in combination of 0's and 1's with the start symbol. Since $(0 + 1)^*$ indicates {ε, 0, 1, 01, 10, 00, 11, ...} in this set ε is a string. So in the rules we can set the rule S → ε.

Example 3.6.4 *Construct a grammar for the language containing strings of at least two a's.*

Solution : Let G = (V, T, P, S)

where V = {S, A}

T = {*a*, b}

P = {S → A*a* A*a* A

A → *a*A | bA | ε }

The rule S → A*a*A*a*A is something in which the two *a*'s are maintained since at least two *a*'s should be there in the strings. And A → *a*A |bA|ε gives any combination of *a*'s and b's i.e. this rules gives the strings of $(a + b)^*$.

Thus the logic for this example will be

(any thing) *a* (any thing) *a* (any thing)

So before *a* or after *a* there could be any combination of *a*'s and b's.

Example 3.6.5 *Construct a grammar generating* $L = wcw^T$ *where* $w \in \{a,b\}^*$

Solution : The strings which can be generated for given L is {*aacaa*, bcb, *abcba*, b*aca*b ...}

The grammar could be

S → *a* S *a*

S → b S b

S → c

Since the language L = w c w^T where w ∈ $(a + b)^*$

Hence S → *a* S *a* or S → b S b. The string *abcba* can be generated from given production rules as

S

a S *a*

a b S b

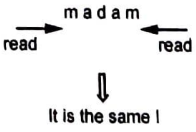
a b c b *a*

a	b	c	b	*a*
a	b	c	b	*a*
a	b	c	b	*a*

Thus any of this kind of string could be derived from the given production rules.

Example 3.6.6 *Construct CFG for the language L which has all the strings which are all palindrome over* $\Sigma = \{a,b\}$.

Solution : As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.



For example, the string "madam" is a palindrome because

Since the language L is over Σ = {*a*, b}. We want the production rules to be build *a*'s and b's. As ε can be the palindrome, *a* can be palindrome even b can be palindrome. So we can write the production rules as

G = ({S}, {*a*, b}, P, S)

P can be S → *a* S *a*

S → b S b

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \varepsilon$

The string abaaba can be derived as

S	
aSa	$abaaba$
$abSba$	$abaaba$
$abaSab$	$aabbaaba$
$abaεab$	$aabbaaba$
$abaaba$	$abaaba$

which is a palindrome.

Example 3.6.7 *Construct CFG which consists of all the strings having at least one occurrence of 000.*

Solution : The CFG for this language can be equivalent to r.e. (any thing) (000) (anything)

Thus r.e. = $(0 + 1)^* 000 (0 + 1)^*$

Let us build the production rules as

$S \rightarrow ATA$

$A \rightarrow 0A \mid 1A \mid \varepsilon$

$T \rightarrow 000$

Example 3.6.8 *Construct CFG for the language in which there are no consecutive b's, the strings may or may not have consecutive a's.*

Solution : $S \rightarrow aS \mid bA \mid a \mid b \mid \varepsilon$

$A \rightarrow aS \mid a \mid \varepsilon$

In the above rules, there is no condition on occurrence of *a*'s. But no consecutive b's are allowed. Note that in the rule $S \rightarrow bA$, and A gives all the strings which are starting with letter *a*.

Thus $G = (\{S, A\}, \{a, b\}, P, S)$

Let us derive the string

Derivation	Input	Productions
S		
aS	aab ab	S → aS
aaS	aab ab	S → aS
aabA	aabab	S → bA
aabaS	aabab	A → aS
aabab	aabab	S → b

Example 3.6.9 *Recognize the context free language for the given CFG.*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Solution : To find the language denoted by given CFG we try to derive the rules and get various strings. Then after observing those strings we come to know, which language it is denoting. Let us rewrite all those rules and number them

$S \rightarrow aB$	rule 1
$S \rightarrow bA$	rule 2
$A \rightarrow a$	rule 3
$A \rightarrow aS$	rule 4
$A \rightarrow bAA$	rule 5
$B \rightarrow b$	rule 6
$B \rightarrow bS$	rule 7
$B \rightarrow aBB$	rule 8

Now let us apply the rules randomly starting with start symbol.

S	
aB	rule 1
$aaBB$	rule 8
$aabSB$	rule 7
$aabbAB$	rule 2
$aabbaB$	rule 3
$aabbabS$	rule 7
$aabbabbA$	rule 2
$aabbabba$	rule 3

We have got some string as "*a*abb*a*bb*a*". Let us try out something else.

S	
bA	rule 2
bbAA	rule 5
bba S A	rule 4
bbaaBA	rule 1
bbaabA	rule 6
bbaaba	rule 3

Now we have got "bb*a*ab*a*" such a string !

Thus by obtaining more and more strings by applying more and more rules randomly will help us to find out what language it indicates. Observe the strings generated carefully, we can draw a conclusion as these strings contain equal number of *a*'s and equal number of b's. Even you can try out various rules to obtain some more strings. And see that it is a language L containing all the strings having equal number of *a*'s and b's.

Example 3.6.10 *Construct CFG for the language containing at least one occurrence of double a.*

Solution : The CFG can be built with a logic as : one rule we will built for double *a* i.e. $A \rightarrow aa$.

And the other rule we will built for any number of *a*'s and b's in any combination.

i.e. $B \rightarrow aB \mid bB \mid \varepsilon$ which is always equivalent to $(a + b)^*$ (you can note it as golden rule !)

If we combine both of these rules we can get the desired context free grammar.

$$\begin{pmatrix}\text{one occurrence}\\ \text{of double } a\end{pmatrix} \quad \begin{matrix} S \rightarrow BAB \Rightarrow (\text{anything}) \\ (\text{anything})\end{matrix}$$

$A \rightarrow aa$

$B \rightarrow aB \mid bB \mid \varepsilon$

Thus the CFG contains V = {A, B, S} T = {*a*, b}. The start symbol is S.

Let us derive "abaab"

S		
BAB		
aBAB	abaab	B → a B
abBAB	abaab	B → bB
abAB	abaab	B → ε
abaaB	abaab	A → aa
abaabB	abaab	B → bB
abaabε	abaab	B → ε
= abaab	abaab	

Example 3.6.11 *Construct CFG for the language containing all the strings of different first and last symbols over Σ = {0, 1}.*

Solution : Since the problem statement says as if the string starts with 0 it should end with 1 or if the string starts with 1 it should end with 0.

$$S \rightarrow 0\,A\,1 \mid 1\,A\,0$$

$$A \rightarrow 0\,A \mid 1\,A \mid \varepsilon$$

Thus clearly, in the above CFG different start and end symbols are maintained. The non terminal A → 0A|1A|ε indicates (0 + 1)*. Thus the given CFG is equivalent to the regular expression [0(0 + 1)* 1+ 1(0 + 1)* 0] .

Example 3.6.12 *Construct the CFG for the language $L = a^n b^{2n}$ where $n \geq 1$.*

Solution : As in some previous example we have seen the case of $a^n b^n$. Now the number of b's are doubled. So we can write

$$S \rightarrow a\,S\,b\,b \mid a\,b\,b$$

It is as simple as this !

Example 3.6.13 *Construct the production rules for defining a language $L = \{a^x b^y \mid x \neq y\}$.* **PU : Dec.-06, Marks 8**

Solution : This is the language in which all the a's must appear before all the b's. But total number of a's must not be equal to total number of b's. Either number of a's must be equal to number of b's or number of b's must be equal to number of a's. The rule

$$S \rightarrow aSb$$

gives us equal number of a's must be followed by equal number of b's. But according to problem statement we need more number of a's either or more number of b's. Hence the required production rules can be -

$$S \rightarrow aSb \mid R1 \mid R2$$

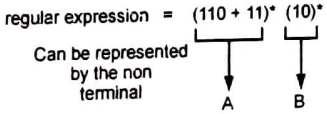
$$R1 \rightarrow aR1 \mid a$$

$$R2 \rightarrow bR2 \mid b$$

where S is a start symbol.

Example 3.6.14 *Find the CFG for the regular expression (110 + 11)*(10)*.*

Solution : Let



If we assume S as start symbol then,

$$S \rightarrow AB$$

Now for each non terminal A and B we can define production rules as

$$A \rightarrow 110A \mid 11\,A \mid \varepsilon$$

$$B \rightarrow 10\,B \mid \varepsilon$$

Hence, finally the CFG for given r.e. can be

$$S \rightarrow AB$$

$$A \rightarrow 110A \mid 11A \mid \varepsilon$$

$$B \rightarrow 10B \mid \varepsilon$$

Example 3.6.15 *Build a CFG for generating the integers.*

Solution : The integer is any numerical value without decimal place which can be either signed or unsigned. Hence the CFG can be -

$$S \rightarrow GI$$

$$G \rightarrow + \mid -$$

$$I \rightarrow DI \mid D$$

$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \ldots \mid 9$$

Here G is for denoting sign, I denotes integer and D denotes digits from 0 to 9. To understand the rules lets derive these rules for some integer.

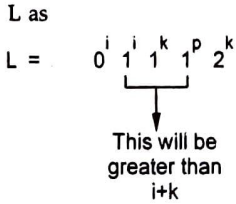
Let, – 21 can be derived as

S → GI	
– I	G → –
– DI	I → DI
– 2I	D → 2
– 2D	I → D
– 21	D → 1

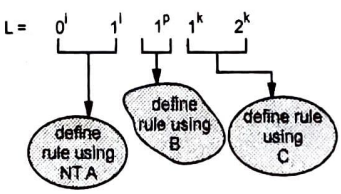
Example 3.6.16 *Build a CFG for the language $L = \{0^i\,1^j\,2^k \mid j > i + k\}$.*

Solution : As the language $L = 0^i\,1^j 2^k$ such that j > i + k.

Hence we can rewrite L as



We can again rewrite L for simplification as -



$$A \rightarrow 0\,A\,1 \mid \varepsilon$$

$$B \rightarrow 1\,B \mid 1$$

$$C \rightarrow 1\,C\,2 \mid \varepsilon$$

Hence the CFG can be

$$S \rightarrow A\,B\,C$$

$A \rightarrow 0\,A\,1 \mid \varepsilon$

$B \rightarrow 1\,B \mid 1$

$C \rightarrow 1\,C\,2 \mid \varepsilon$

Example 3.6.17 *Build a CFG for the language* $L = \{0^i\,1^j\,2^k \mid i = j\}$

Solution : As i = j and there is no condition on k. We can say that k is an arbitrary value. Then we rewrite L as

$\therefore$ $S \rightarrow A\,B$

$A \rightarrow 0\,A\,1 \mid \varepsilon$

$B \rightarrow 2\,B \mid \varepsilon$

Example 3.6.18 *Obtain CFG for the language* $L = \{0^i\,1^j\,2^k \mid j \leq k\}$

Solution : Here $j \leq k$. That means the language L has two variations.

$L_1 = 0^i\,1^j\,2^j$

$L_2 = 0^i\,1^j\,2^k$ where $k > j$.

Hence $L_2 = 0^i\,1^j\,2^j \cdot 2^k$

Hence the required CFG can be

$S \rightarrow A\,B$

$A \rightarrow 0\,A \mid \varepsilon$

$B \rightarrow 1\,B\,2\,C \mid 12$

$C \rightarrow 2\,C \mid \varepsilon$

Example 3.6.19 *Give CFG for set of odd length settings in {0, 1}* with middle symbol '1'.* **SPPU : May-14, Marks 4**

Solution : $S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 1$

Consider the string 01110 for derivation

S

0S0

01S10

01110

Example 3.6.20 *If the grammar G is given by production* $S \rightarrow aSa \mid bSb \mid aa \mid bb \mid \varepsilon$ *show that*

- *Any string in L(G) is of length 2n,* $n > = 0$
- *The number of strings of length 2n is* 2^n **SPPU : Dec.-13, Marks 2**

Solution : The CFG denotes the language L of even length in which the two middle symbols equal. Hence it will generate the strings such as

L(G) = { ε, aa, bb, aaaa, aaab, baab, baaa, abba, abbb, bbbb, bbba, ...}

- If n = 0, the string is ε.
- If n = 1, the string of length 2 (i.e. 2 × 1) are aa and bb (i.e. $2^n = 2$).

Example 3.6.21 *Describe the language generated by each of these grammars. Justify your answer with an example.*

1) $S \rightarrow a\,S\,a \mid b\,S\,b \mid \varepsilon$ *2)* $S \rightarrow a\,S\,a \mid b\,S\,b \mid a \mid b$

3) $S \rightarrow a\,S\,b \mid b\,S\,a \mid \varepsilon$ *4)* $S \rightarrow SS \mid bS \mid a$

SPPU : Dec.-10, Marks 12

Solution :

1) L = The language containing all strings of even length palindrome.

S	S → aSa
aSa	S → bSb
abSba	S → ε
abba	

2) L = The language containing all strings of odd length palindrome.

S	S → aSab
aSb	S → bSb
abSba	S → a
ababa	

3) L = The language containing all strings which contains equal number of a's and b's. But having different first and last symbol.

S	S → aSb
aSb	S → bSa
abSab	S → ε
abab	

4) L = The language containing all strings that end with lettter a.

S	S → SS
SS	S → bS
bSS	S → a
baS	S → a
baa	

Example 3.6.22 *Write a CFG for generating identifiers in higher-level languages such as 'C'. Identifiers can be defined by the regular expression (letter). (letter | digit)*.* **SPPU : May-13, Marks 4**

Solution :

$S \rightarrow LA$

$A \rightarrow LA \mid DA \mid \epsilon$

$L \rightarrow a \mid b \mid c \ldots z$

$D \rightarrow 0 \mid 1 \mid 2 \ldots 9$

Example 3.6.23 *In each case find context free grammar generating given language*

a) The set of odd length strings in (a, b) with middle symbol a.*

b) The set of even length strings (a, b) with the two middle symbols equal.* **SPPU : Dec.-07, May-08, Marks 8**

Solution :

a) Required CFG is -

$S \rightarrow aSa$

$S \rightarrow aSb$

$S \rightarrow bSb$

$S \rightarrow bSb$

$S \rightarrow a$

b) This language can be denoted by regular expression.

regular expression = $(a + b)^* (aa + bb) (a + b)^*$

Hence CFG can be given as

$S \rightarrow aSa \mid aSb \mid bSb \mid bSa$

$S \rightarrow aa$

$S \rightarrow bb$

Example 3.6.24 *Describe the language generated by each of these grammars*

a) The regular grammar with production :

$S \rightarrow aA \mid bC \mid b$

$A \rightarrow aS \mid bB$

$B \rightarrow aC \mid bA \mid a$

$C \rightarrow aB \mid bS$

b) The regular grammar with production :

$S \rightarrow bS \mid aA \mid \varepsilon$

$A \rightarrow aA \mid bB \mid b$

$B \rightarrow bS$

SPPU : Dec.-07, 09, May-08, 09, Marks 8

Solution :

a) For obtaining the language generated by given CFG we will first draw FA for it.

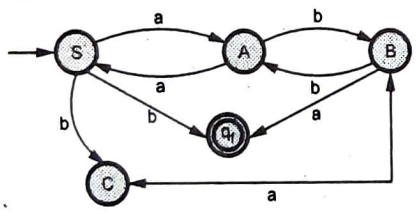


Fig. 3.6.1

We can merge q_f and state C. Hence the FA will be.

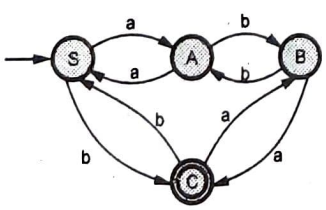


Fig. 3.6.2

We can derive some strings from above drawn FA. Those are {aab, bbb, b, aba, ...}

That means given language L accepts the strings containing even number of a's and odd number of b's.

b) For obtaining the language generated by given CFG we will first draw FA for it.

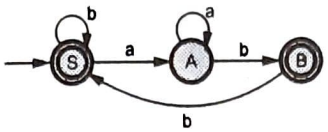


Fig. 3.6.3

The r.e. = $\left(a^* aa^* b (\varepsilon + b)\right)^* + b^*$

This regular expression denotes the strings made up of either any number of b's or the strings that contain the substring 'ab' but not contain the substrings 'aba' at all.

Example 3.6.25 *Give the context free grammars for the following languages*

a) $L = \{0^m 1^n 0^{m+n} \mid m, n >= 0\}$ *b)* $L = \{0^a 1^b 2^c \mid \mid a - c \mid = b\}$

SPPU : Dec.-08, Marks 12

Solution :

a) Let, $L = \{0^m 1^n 0^{m+n}\}$

We will rewrite L as

$= 0^m 1^n 0^n 0^m$ $\because n + m = m + n$

Hence P = {

$S \rightarrow 0S0 \mid 0100$

$S \rightarrow 01A00$

$A \rightarrow 1A0 \mid \varepsilon$

}

Consider the derivation for string 001000 as

S	
0S0	$S \rightarrow 0S0$
001000	$S \rightarrow 0100$

b) Let, $L = \{0^a 1^b 2^c \mid a - c \mid = b\}$

If we arrange $a = \mid b+c \mid$ then we can rewrite L as

$= 0^{b+c} 1^b 2^c$

$= 0^c 0^b 1^b 2^c$

Hence CFG can be

$S \rightarrow 0S2 \mid 0012$

$S \rightarrow 00A12$

$A \rightarrow 0A1 \mid \varepsilon$

Let us derive the string 000000112222 as follows -

S	$S \rightarrow 0S2$
0S2	$S \rightarrow 0S2$
00S22	$S \rightarrow 0S2$
000S222	$S \rightarrow 00A12$
00000A12222	$A \rightarrow 0A1$
000000A112222	$A \rightarrow \varepsilon$
000000112222	

i.e. $0^6 1^2 2^4 \in \{L = 0^a 1^b 2^c \mid \mid a - c \mid = b\}$

Example 3.6.26 *Give the context free grammar for the following languages a)* $L = \{a^n b^{2n} \mid n > 1\}$ *b)* $L = \{a^m b^n \mid n > m\}$

SPPU : Dec.-08, Marks 12

Solution :

a) Let, $L = \{a^n b^{2n} | n>1\}$

The production rules for the CFG of above language will be -

$S \rightarrow aAbb$

$A \rightarrow aAbb \mid abb$

b) Let, $L = \{a^m b^n \mid n > m\}$

The L can be written as $L = a^m b^m b^K$ where $K \geq 1$.

Hence required CFG will be

$S \rightarrow aSbB \mid \varepsilon$

$B \rightarrow bB \mid b$

Example 3.6.27 *Give the context free grammars for the following languages.*

a) (011 + 1) (01)**

b) $0^i 1^{i+k} 0^k$ where i, k > = 0.

SPPU : May-09, Marks 12, May-14, Marks 8

Solution : a) The CFG can be denoted by

$G = (\{S, A, B, C, D\}, \{0, 1\}, P, S)$

where $P = \{$

$S \rightarrow AB$

$A \rightarrow CA \mid \varepsilon$

$C \rightarrow 011 \mid 1$

$B \rightarrow DB \mid \varepsilon$

$D \rightarrow 01$

$\}$

Let us derive a valid string 01110101 $\in$ (011 + 1)* (01)*

S

AB

CAB

CCAB

CCADB

CCADDB

011CADDB

0111ADDB

0111DDB

011101DB

01110101B

0111101

b) Let us rewrite the language L as

$L = 0^i 1^{i+k} 0^k$

$L = 0^i 1^i 1^k 0^k$

Hence required CFG $G = (\{S, A, B\}, \{0, 1\}, P, S)$

Hence

$P = \{S \rightarrow AB$

$= A \rightarrow 0A1 \mid \varepsilon$

$= B \rightarrow 1B0 \mid \varepsilon$

$\}$

Example 3.6.28 *Describe the language generated by each of these grammar and justify your answer with the example string derive from the grammar of the productions given below.*

1) $S \rightarrow aSa \mid bSb \mid aAb \mid bAa$

$A \rightarrow aAa \mid bAb \mid a \mid b \mid \varepsilon$

2) $S \rightarrow bT \mid aT \mid \varepsilon$

$T \rightarrow aS \mid bS$

PU : May-10, Marks 12

Solution :

1) This is a language which is a combination of two language L_1 and L_2, where L_1 denotes all the strings that are palindrome.

L_2 denotes all the strings containing the substrings as palindrome but their first and last letter are different.

i.e. L_2 = [a (Palindrome string) b + b (palindrome string) a]

2) To derive the string for this CFG we will draw a FA

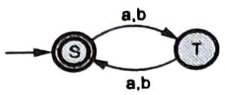


Fig. 3.6.4

Clearly, this FA suggest a language containing all the strings having even length.

Example 3.6.29 *Find the context free grammar for the following languages.*

i) $L = \{a^n b^m | n \neq m\}$

ii) $\{a^n b^m c^k | n = m$ or $m \leq k\}$

SPPU : Dec.-06, Marks 8

Solution : i) $L = \{a^n b^m | n \neq m\}$: Refer example 3.6.26 (b).

ii) $\{a^n b^m c^k | n = m$ or $m \leq k\}$:

Here L is a combination of L_1 and L_2 where

$L_1 = \{a^n b^m c^k | n = m\}$

$L_2 = \{a^n b^m c^k | m \leq k\}$

The CFG for L_1 can be

$S \rightarrow AB$

$A \rightarrow aA\,b \mid \varepsilon$

$B \rightarrow c\,B \mid \varepsilon$

The CFG for L_2 can be

$S \rightarrow PQ$

$P \rightarrow aP \mid \varepsilon$

$Q \rightarrow b\,Q\,c\,R \mid b\,c$

$R \rightarrow c\,R \mid \varepsilon$

Hence the CFG for L will be

$S \rightarrow XY$

$X \rightarrow AB$

$A \rightarrow aA\,b \mid \varepsilon$

$B \rightarrow c\,B \mid \varepsilon$

$Y \rightarrow PQ$

$P \rightarrow a\,P \mid \varepsilon$

$Q \rightarrow b\,Q\,c\,R \mid b\,c$

$R \rightarrow c\,R \mid \varepsilon$

Example 3.6.30 *Give context free grammars for the following languages.*

i) L = {S → aAb, A → aA | bA | ε, x | x ε {a, b} with strings of starting with 'a' and ending 'b'}*

ii) L = {x | x ε {a, b} with strings of even length palindrome}.*

SPPU : Dec.-19, In Sem, Marks 4

Solution : i) S → aAb, A → aA | bA | ε.

ii)

Step 1 : As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.

Step 2 : Let G = ({S}, {a, b}, P, S)

Step 3 : The string abaaba can be derived as

P can be

S → a S a
S → b S b
S → a
S → b
S → ε

S
a S a
a b S b a
a b a S a b
a b a ε a b
a b a a b a

a b a a b a
a b a a b a
a b a a b a
a b a a b a
a b a a b a

Example 3.6.31 *Give context free grammars for the following languages :*

i) L = {x | x ∈ (,)} with strings having well-formed parentheses (WFP)*

ii) L = {$a^m b^n c^{m+n}$} m,n >=0} SPPU : Oct. 2016, In sem, Marks 6

Solution :

i) The production rule are

P = {
S → ( ) S | ( ) S | (S) | ( )
}

ii) We will first rewrite the given L as

L = $a^m b^n c^n c^m$

Hence P = {
S → aSc | abcc
S → abAcc
A → bAc | ε
}

Consider the derivation for string

aabccc

S
aSc
aabccc

3.7 Derivation

SPPU : May-16, Marks 4

- The production rules are used to derive certain strings. If G = (V, T, P, S) be some context free grammar then generation of some language using specific rules is called **derivation.**
- As we know, derivation for any string means replacement of non terminal by its appropriate definition. There may be a situation, in which there are many non terminals. Then which non terminal should be replaced by its definition is sometimes confusing to decide.
- Hence we normally apply **two methods of deriving.**
- The **leftmost derivation** is a derivation in which the leftmost non terminal is replaced first from the sentential form.
- The **rightmost derivation** is a derivation in which rightmost non terminal is replaced first from the sentential form.

Let us see how it works.

For example

S → XYX	S → XYX
S → aYX	S → XYa
S → abX	S → Xba
S → aba	S → aba
Leftmost derivation	**Rightmost derivation**

Note that we have replaced first X from left to right in leftmost derivation and XYX the last X i.e. the rightmost symbol.

Actually, we may use leftmost derivation or rightmost derivation we get the same string. The type of derivation does not affect on getting of a string.

Example 3.7.1 *Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by,*

S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

Solution : Let us see the leftmost derivation first,

S	
aB	S → aB
aaBB	B → aBB
aabB	B → b
aabbS	B → bS
aabbaB	S → aB
aabbabS	B → bS
aabbabbA	S → bA
aabbabba	A → a

is derived. Now let us solve using rightmost derivation.

S	
aB	S → aB
aaBB	B → aBB
aaBbS	B → bS
aaBbbA	S → bA
aaBbba	A → a
aabSbba	B → bS
aabbAbba	S → bA
aabbabba	A → a

is a derivation.

Example 3.7.2 *Derive the string 1000111 for leftmost and rightmost derivation using CFG.*

G = (V, T, P, S) where
V = {S, T}
T = {0, 1}
P = { S → T00T
T → 0T | 1T | ε }

Solution : The leftmost derivation can be

S	
T00T	S → T00T
1T00T	T → 1T
10T00T	T → 0T
10 ε 00T	T → ε
1000T	
10001T	T → 1T
100011T	T → 1T
1000111T	T → 1T
1000111 ε	T → ε
1000111	

Rightmost derivation -

S	
T00T	
T001T	T → 1T
T0011T	T → 1T
T00111T	T → 1T
T00111 ε	T → ε
T00111	
1T00111	T → 1T
10T00111	T → 0T
10 ε 00111	T → ε
1000111	

Example 3.7.3 *Write the CFG for language* $L = \{0^i 1^j 0^k \mid j > i + k\}$. *Show the derivation of the string '0111100'.*

SPPU : May-15, End Sem. Marks 4

Solution : The CFG can be

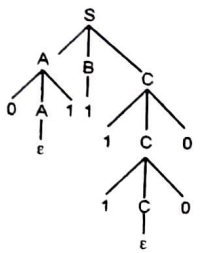
$S \rightarrow ABC$

$A \rightarrow 0A1 \mid \varepsilon$

$B \rightarrow 1B \mid 1$

$C \rightarrow 1C0 \mid \varepsilon$

Derivation of 0111100



3.8 Parse Trees

Parse trees is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from given set of production rules. The parse tree is also called derivation tree.

Properties of Parse Tree

Following are properties of any parse tree -

1. The root node is always a node indicating start symbol.
2. The derivation is read from left to right.
3. The leaf nodes are always terminal nodes.
4. The interior nodes are always the non terminal nodes.

For example,

$S \rightarrow bSb \mid a \mid b$ is a production rule. The S is a start symbol.

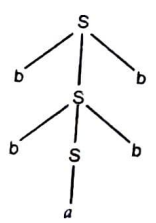
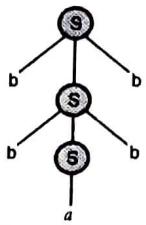


Fig. 3.8.1 Parse tree

The above tree is a derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes we can obtain the desired string. The same tree can also be denoted by,



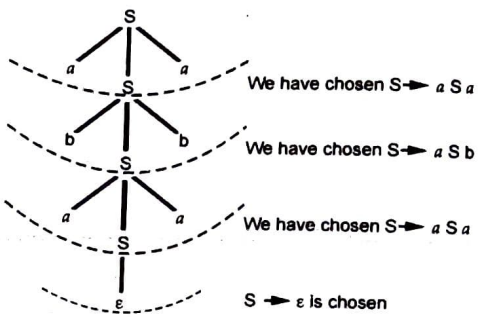
Problems on parse tree

Example 3.8.1 *Draw a parse tree for the string abaaba for the CFG given by,*

G where $P = \{S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a \mid b \mid \varepsilon\}$



Example 3.8.2 *Construct the parse tree for the string aabbabba from the CFG given by*

$S \rightarrow aB \mid bA$

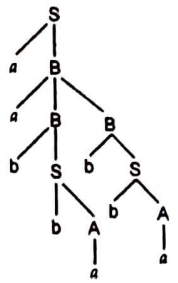
$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Solution : To draw a tree we will first try to obtain derivation for the string *aabbabba*

S	
*a*B	S → *a*B
a [*a*BB]	S → *a*BB
aa [bS] B	B → bS
aab [bA] B	S → bA
aabb a [B]	A → *a*
aabba [bS]	B → bS
aabbab [bA]	S → bA
aaBbabb [*a*]	A → *a*

Now let us draw a tree.



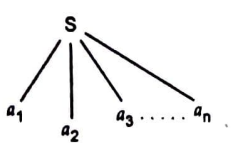
3.9 Relationship between Parse Tree, Inference and Derivation

Theorem : Let G = (V, T, P, S) be a context free grammar. Then $S \overset{*}{\Rightarrow} a$ if and only if there is a derivation tree in grammar G which gives the string a.

Proof : If there is a non terminal S in V then $S \overset{*}{\Rightarrow} a$ if and only if there is a S-tree which gives the string a. We can easily prove this using induction theorem. We can apply induction on number of interior nodes.

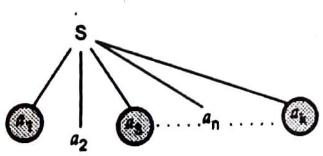
Basis of Induction :

Let us assume S is the only one interior which gives the string a which can be obtained by deriving $S \rightarrow a_1, a_2, a_3, \ldots a_n$. Then there exists a derivation tree which derives $a_1, a_2, a_3 \ldots a_n$ and gives the string a.



Induction step :

We assume, that the derivation will enable us to draw the tree with K–1 interior nodes. We have to prove, that it is possible to draw the derivation tree for K interior nodes which gives the string a. In the derivation tree not all the nodes are leaves or interior nodes. Some are leaves where as others are interior nodes considering for them S is a parent node.



Thus this derivation tree will ultimately give the string a.

3.10 Ambiguity In Grammar and Language

The grammar can be derived in either leftmost derivation or rightmost derivation. One can draw a derivation tree called as **parse tree** or **syntax tree** based on these derivations. The parse tree has to be unique even though the derivation is leftmost or rightmost.

• Definition

If there exists more than one parse trees for a given grammar, that means there could be more than one leftmost or rightmost derivation possible and then that grammar is said to be ambiguous grammar.

• Example

For example : The CFG given by G = (V, T, P, S)

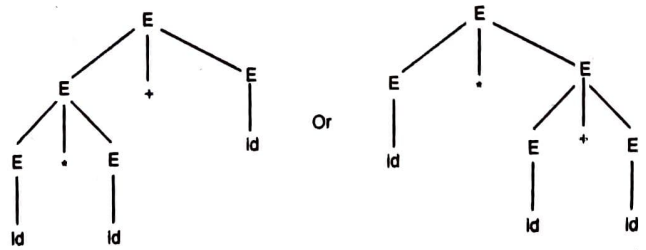
where V = {E}

T = {id}

P = {E → E + E
E → E * E
E → id }

S = {E}

Now if the string is id * id + id then we can draw the two different parse trees indicating our id * id + id.



Thus the above grammar is an ambiguous grammar. Let us solve some exercise on it.

Problems on ambiguity of grammar

Example 3.10.1 *Check whether the given grammar is ambiguous or not.*

$S \rightarrow iCtS$

$S \rightarrow iCtSeS$

$S \rightarrow a$

$C \rightarrow b$

Solution : To check whether given grammar is ambiguous or not we will have some string for derivation tree such as ibtibtibtaea.

Let us draw the derivation tree, for the string ibtibtibtaea

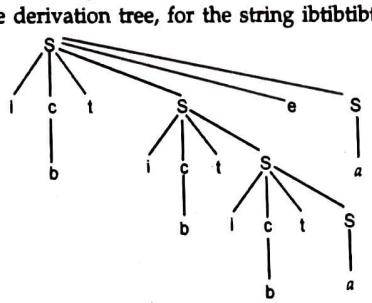


Fig. 3.10.1 (a) For example 3.10.1

or

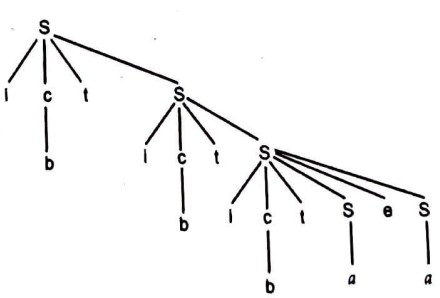


Fig. 3.10.1 (b) For example 3.10.1

Thus we have got more than two parse trees. Hence the given grammar is ambiguous.

Example 3.10.2 *Consider the grammar* $G = (V, \Sigma, R, S)$.

where $V = \{S, A\}$

$\Sigma = \{a, b\}$

$R = \{ S \rightarrow AA$

$A \rightarrow AAA$

$A \rightarrow a$

$A \rightarrow bA$

$A \rightarrow Ab \}$

Show that this is an ambiguous grammar

Solution : Consider a string babbab.

Refer Fig. 3.10.2.

Thus there are more than one parse tree getting generated. Hence the grammar is ambiguous.

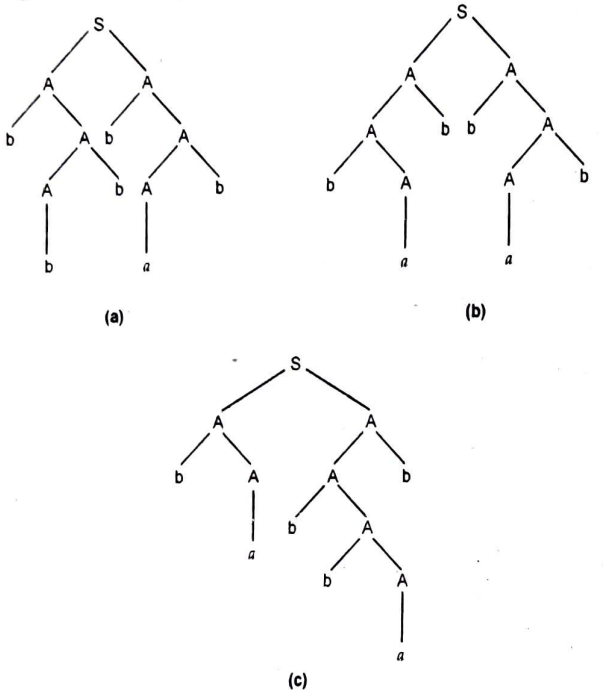


Fig. 3.10.2 For example 3.10.2

3.10.1 Inherent Ambiguity

- **Definition :**

A context free grammar is called inherently ambiguous if all the production rules in this grammar are ambiguous.

- **Example :**

For example -

$S \rightarrow XYZ \mid aaYbb$

$X \rightarrow aaY \mid aa$

$Y \rightarrow baZ \mid ba$

$Z \rightarrow bZb \mid bb$

Then the parse trees for the string "aababb" can be as shown below -

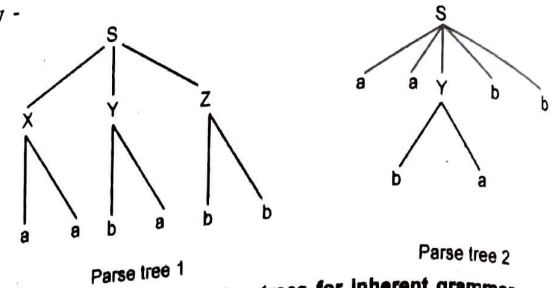


Fig. 3.10.3 Derivation trees for inherent grammar

Note that in above context free grammar there are two productions for each rule this makes every rule of the grammar as ambiguous.

3.10.2 Removal of Ambiguity

Consider the ambiguous grammar as -

$E \rightarrow E + E \mid E * E \mid id$

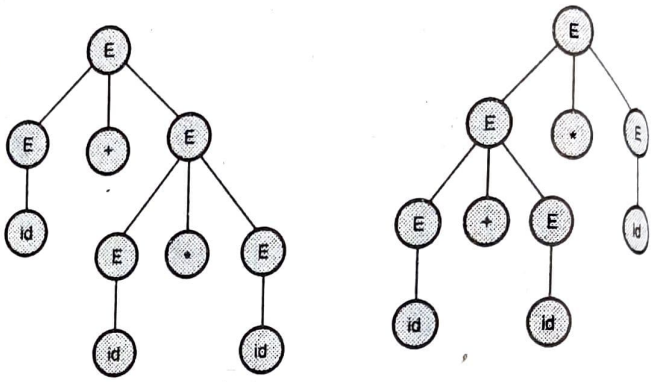


Fig. 3.10.4 Ambiguous grammar

is an ambiguous grammar. We will design the parse tree for id + id * id as follows.

For removing the ambiguity we will apply one rule : If the grammar has left associative operator (such as +, –, *, /) then induce the left recursion and if the grammar has right associative operator (exponential operator) then induce the right recursion.

The unambiguous grammar is

$E \rightarrow E+T$

$E \rightarrow T$

$T \rightarrow T*F$

$T \rightarrow F$

$F \rightarrow id$

Example 3.10.3 *Give an ambiguous grammar for if then else statement and then re-write an equivalent unambiguous grammar.*

SPPU : May-14, Marks 8

Solution : The ambiguous grammar for **if then else** statement is as follows -

$S \rightarrow iCtS$

$S \rightarrow iCtSeS$

$S \rightarrow a$

$S \rightarrow b$

Where i stands for **if**, C stands for **condition**, S stands for statement, e stands for **else**, and t stands for **then**. For making the

given grammar unambiguous we match **else** with closest previous **unmatched then**. Hence on removing ambiguity we will get following rules -

$$S \rightarrow M \mid U$$
$$M \rightarrow iCtMeM \mid a$$
$$U \rightarrow iCtS \mid iCtMeU$$
$$C \rightarrow b$$

Here M and U non-terminals for matched and unmatched statments.

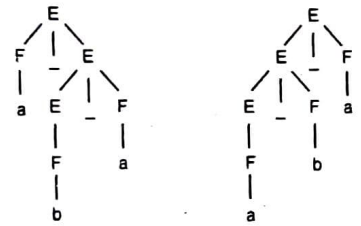
Example 3.10.4 *Consider the grammar*
$G = (V = \{E, F\}, T = \{a, b, -\}, E, P)$
Where P consist of rules :
$E \rightarrow F - E, F \rightarrow a, E \rightarrow E - F, F \rightarrow b, E \rightarrow F$
1) Show that G is ambiguous
2) Remove the ambiguity, if possible of G. **SPPU : May-11, Marks 6**

Solution :
Let,

$$E \rightarrow F - E$$
$$F \rightarrow a$$
$$E \rightarrow E - F$$
$$F \rightarrow b$$
$$E \rightarrow F$$

are the production rules.

1) Consider the string a-b-a



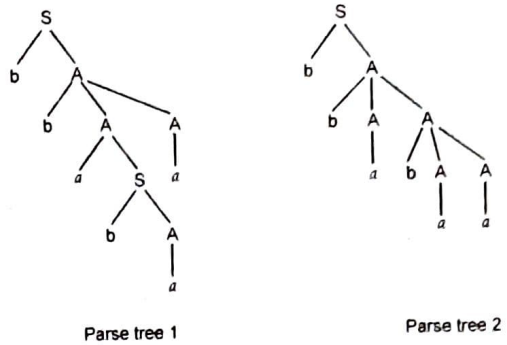
Parse tree 1 **Parse tree 2**

2) The unambiguous grammar will be

$$E \rightarrow E - F \mid F$$
$$F \rightarrow a \mid b$$

Example 3.10.5 *Given a CFG as :*
$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$
Is ab, baba, abbbaa in L (G) ? What is the language of this CFG ? Is the CFG ambiguous ? **SPPU : May-12, Marks 8**

Solution : The {ab, baba, abbbaa} $\in$ L (G). The language of this CFG is L which contains equal number of a's and b's. The given CFG is ambiguous because consider the string bbabaa.



Parse tree 1 **Parse tree 2**

Fig. 3.10.5

Example 3.10.6 *Show that following grammar is ambiguous :*
$S \rightarrow a \mid absb \mid aAb$
$A \rightarrow bs \mid aAAb$ **SPPU : Dec.-12, Marks 4**

Solution : Consider the string absb. Try to find left and right derivation tree.

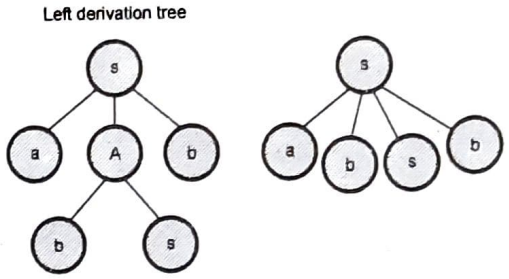
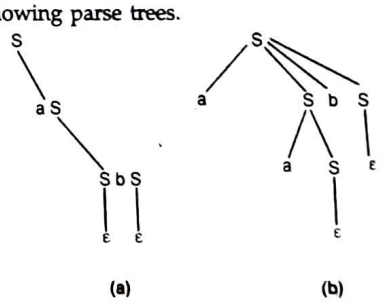


Fig. 3.10.6

For same string we have two different derivation tree so given grammar is ambiguous.

Example 3.10.7 *Consider the grammar having production.*
$S \rightarrow aS \mid \varepsilon$
$S \rightarrow aSbS$
This grammar is ambiguous.
i) Show in particular that the string aab has two parse trees.
ii) Find an unambiguous grammar for the same.
SPPU : Dec.-05, Marks 8

Solution : Given grammar with production
$S \rightarrow a\,S \mid \varepsilon$
$S \rightarrow a\,Sb\,S.$
i) a a b has following parse trees.



(a) **(b)**

Fig. 3.10.7

unambiguous grammar for above is
$S \rightarrow a\,S \mid XS \mid \varepsilon$
$X \rightarrow aXXb \mid \varepsilon$

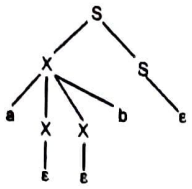


Fig. 3.10.7 (c)

Example 3.10.8 *Let G be the grammar*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

For the string aaabbabbba find

1) Leftmost derivation *2) Rightmost derivation*

3) Parse tree *4) Is the grammar unambiguous*

SPPU : May-07, Marks 8; Dec.-14, End Sem. Marks 4

Solution : 1) Leftmost derivation :

S → aB
→ aaBB
→ aaaBBB
→ aaabSBB
→ aaabbABB
→ aaabbaBB
→ aaabbababB
→ aaabbabbS
→ aaabbabbbA
→ aaabbabbba.

2) Rightmost derivation :

S → aB
→ aaBB
→ aaBbS
→ aaBbbA
→ aaBbba
→ aaaBBbba
→ aaaBbbba
→ aaabSbbba
→ aaabbAbbba
→ aaabbabbba.

3) Parse tree :

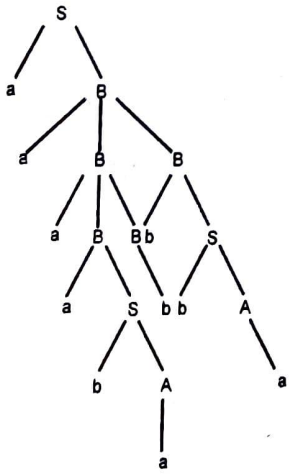
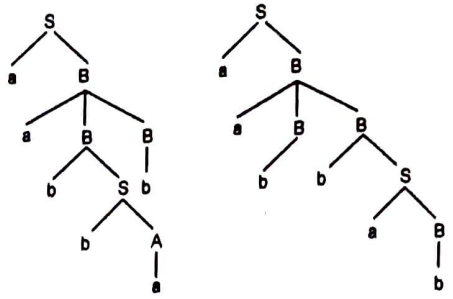


Fig. 3.10.8

4) No, the grammar is ambiguous. Following derivations for aabbab shows -



Example 3.10.9 *Derive (a101 + b1) * (a1 + b) using leftmost and rightmost derivation where grammar is given as*

$E \rightarrow I, E \rightarrow E + E, E \rightarrow E * E, E \rightarrow (E), I \rightarrow a,$

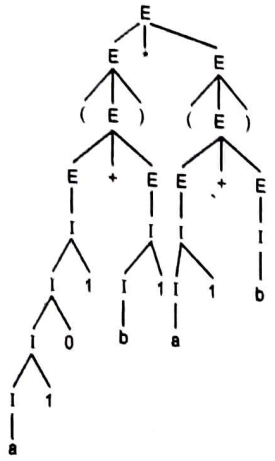
$I \rightarrow b, I \rightarrow Ia, I \rightarrow Ib, I \rightarrow I0, I \rightarrow I1.$

Test whether the grammar is ambiguous. **SPPU : Dec.-13, Marks 6**

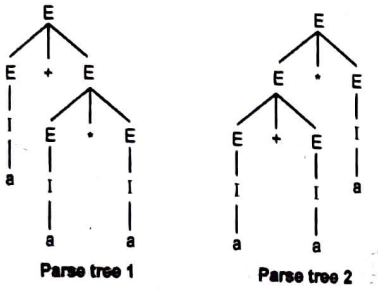
Solution :

Leftmost derivation	Rightmost derivation
E	E
E * E	E * E
(E) * E	E * (E)
(E + E) * E	E * (E + E)
(I + E) * E	E * (E + I)
(I1 + E) * E	E * (E + b)
(I01 + E) * E	E * (I + b)
(I101 + E) * E	E * (I1 + b)
(a101 + E) * E	E * (a1 + b)
(a101 + I) * E	(E) * (a1 + b)
(a101 + I1) * E	(E + E) * (a1 + b)
(a101 + b1) * E	(E + I) * (a1 + b)
(a101 + b1) * (E)	(E + I1) * (a1 + b)
(a101 + b1) * (E + E)	(E + b1) * (a1 + b)
(a101 + b1) * (I + E)	(I + b1) * (a1 + b)
(a101 + b1) * (I1 + E)	(I1 + b1) * (a1 + b)
(a101 + b1) * (a1 + E)	(I01 + b1) * (a1 + b)
(a101 + b1) * (a1 + I)	(I101 + b1) * (a1 + b)
(a101 + b1) * (a1 + b)	(a101 + b1) * (a1 + b)

The derivation is as follows -



The grammar is ambiguous. Consider the derivation for the string a + a * a.



Parse tree 1 Parse tree 2

As there are more than one parse trees for deriving the given string the grammar is said to be ambiguous.

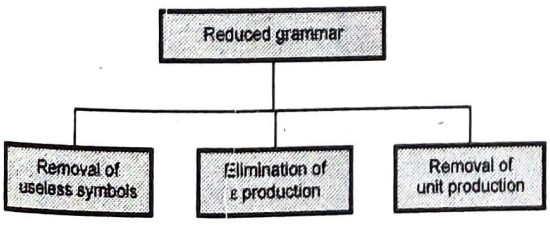
3.11 Simplification of CFG

As we have seen various languages can effectively be represented by context tree grammar. All the grammars are not always optimized. That means grammar may consists of some extra symbols (non terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below.

Properties of reduced grammar -

1. Each variable (i.e. non terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as $X \rightarrow Y$ where X and Y are non terminals.
3. If ε is not in the language L then there need not be the production $X \rightarrow \varepsilon$.

We see the reduction of grammar as below.



Let us study the reduction process in detail.

3.11.1 Removal of Useless Symbols

Any symbol is useful when it appears on the right hand side, in the production rule and generates some terminal string. If no such derivation exists then it is supposed to be an useless symbol.

A symbol p is useful if there exists some derivation in the following form.

$$S \overset{*}{\Rightarrow} \alpha p \beta$$

and $$\alpha p \beta \overset{*}{\Rightarrow} w$$

Then p is said to be useful symbol.

Where α and β may be some terminal or non terminal symbol and will help us to derive certain string w in combination with P. Let us see what exactly means the useless symbol with some example.

For example, G = (V, T, P, S) where V = {S, T, X}, T = {0, 1}

$S \rightarrow 0T|1T|X|0|1$ rule 1

$T \rightarrow 00$ rule 2

Now, in the above CFG, the non terminals are S, T and X.

To derive some string we have to start from start symbol S.

S

0T $S \rightarrow 0T$

000 $T \rightarrow 00$

Thus we can reach to certain string after following these rules.

But if $S \rightarrow X$ then there is no further rule as a definition to X. That means there is no point in the rule $S \rightarrow X$. Hence we can declare that X is a useless symbol. And we can remove this so after removal of this useless symbol CFG becomes

G = (V, T, P, S) where V = {S, T}

T = {0, 1} and P = { $S \rightarrow 0T|1T|0|1$

$T \rightarrow 00$ }

S is a start symbol.

Problems on removal of useless symbol

Example 3.11.1 *Consider the CFG G = (V, T, P, S) where V = {S, A, B} T = {0, 1}*

P = { $S \rightarrow A\,11\,B|\ 11A$

$S \rightarrow 1\,B|\ 11$

$A \rightarrow 0$

$B \rightarrow BB$ } For removing useless symbols

Solution : Now in the given CFG if we try to derive any string A gives some terminal symbol as 0 but B does not give any terminal string. By following the rules with B we simply get ample number of B's and no significant string. Hence we can declare **B as useless symbol** and can remove the rules associated with it. Hence after removal of useless symbols we get,

$S \rightarrow 11\,A \mid 11$

$A \rightarrow 0$

Example 3.11.2 *Find CFG with no useless symbols equivalent to*

$S \rightarrow AB \mid CA$ $B \rightarrow BC \mid AB$

$A \rightarrow a$ $C \rightarrow aB \mid b$

Solution : Consider, the rule

$S \rightarrow AB$ 1

$S \rightarrow CA$ 2

In the rule 2, C and A can be replaced by some terminating string but in rule 1, B cannot be replaced by terminating string. It tends to form a never ending loop.

E.g. : $S \rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaAB$ and so on.

Thus we come to know as B is an useless symbol. Removing B the rules are now,

$$S \rightarrow CA$$
$$A \rightarrow a$$
$$C \rightarrow b$$

Example 3.11.3 *Consider the following CFG*

$G = (V, \Sigma, R, S)$

where $V = \{S, X, Y\}$

$\Sigma = \{0, 1\}$

$R = \{S \rightarrow XY \mid 0$

$X \rightarrow 1\}$

Remove the useless symbols from it.

Solution : As

$$S \rightarrow XY$$
$$S \rightarrow 0$$
$$X \rightarrow 1$$

It is easy to recognize that there is no derivation for Y. Hence we can eliminate the production with Y, and the rules are

$$S \rightarrow 0$$
$$X \rightarrow 1$$

Example 3.11.4 *Remove useless symbols from the following grammar.*

$S \rightarrow aA \mid bB$

$A \rightarrow aA \mid a$

$B \rightarrow bB$

$D \rightarrow ab \mid Ea$

$E \rightarrow aC \mid d$

Solution : We will first find all the productions which are giving terminal symbols.

$$A \rightarrow a$$
$$D \rightarrow ab$$
$$E \rightarrow d$$

Now consider start symbol

$$S \rightarrow aA \mid bB$$

Now since $B \rightarrow bB$ we will be continuously in a loop with B productions. Thus B is an useless symbol. So we will eliminate it. Then the rules are

$$S \rightarrow aA$$
$$A \rightarrow a \mid aA$$
$$D \rightarrow ab \mid Ea$$
$$E \rightarrow aC \mid d$$

Again if we look at E productions there is a production $E \rightarrow aC$. But there is no rule for C. Hence we will remove the production $E \rightarrow aC$.

Now,

$$S \rightarrow aA$$
$$A \rightarrow aA \mid a$$
$$D \rightarrow ab \mid Ea$$
$$E \rightarrow a$$

Now the rules S and A indicates that there is no D and E in the derivation. Again we get D and E as useless symbols. So we will eliminate D and E. Finally after removal of all useless symbols the production rules are ,

$$S \rightarrow aA$$
$$A \rightarrow aA \mid a$$

Example 3.11.5 *Eliminate the useless symbols from the following grammar.*

$S \rightarrow aS \mid A \mid C$

$A \rightarrow a$

$B \rightarrow aa$

$C \rightarrow aCb$

Solution : We will first write all the productions that are giving terminal symbols.

$$A \rightarrow a$$
$$B \rightarrow aa$$

But if we look at start symbol S then,

$$S \rightarrow aS \mid A \mid C$$

There is no production for B from start symbol. Thus B becomes a useless symbol.

Similarly,

$$S \rightarrow C \rightarrow aCb \rightarrow aaCbb \rightarrow aaaCbbb \rightarrow \ldots$$

There is no terminating symbol for C. Therefore we will eliminate C. Then set of rules are

$$S \rightarrow aS \mid A$$
$$A \rightarrow a$$

Thus we obtain the grammar having these two rules after removal of useless symbols B, C.

Example 3.11.6 *Eliminate the useless symbols from following grammar.*

$S \rightarrow aA \mid a \mid Bb \mid cC$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

$C \rightarrow cCD$

$D \rightarrow ddd$

Solution : Consider all the productions that are giving terminal symbols.

$$S \rightarrow a$$
$$B \rightarrow a$$
$$D \rightarrow ddd$$

Now consider

$$S \rightarrow cC$$
$$C \rightarrow cCD$$
$$D \rightarrow ddd.$$

When we try to derive the string using production rule for C we get,

$S \rightarrow cC \rightarrow ccCD \rightarrow ccCddd \rightarrow cccCDddd \rightarrow cccCdddddd \rightarrow \ldots$

We will not get any terminal symbol for C. Thus we get a useless symbol C. To reach to D the only rule available is by using C. But as C gets eliminated, there is no point in keeping D. Hence D also will be removed.

$\therefore$ $S \rightarrow aA \mid a \mid Bb$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

is the reduced grammar.

3.11.2 Elimination of ε Productions from Grammar

As we have seen in finite automata and regular expression that ε or a null string indicates a string with no value. We have also seen that even though some NFA contains ε moves we can convert that NFA without ε moves. Even in context free grammar, if at all there is ε production we can remove it, without changing the meaning of the grammar. Thus ε productions are not necessary in a grammar.

For example,

If $S \rightarrow 0S \mid 1S \mid \varepsilon$

Then we can remove ε production. But we have to take a care of meaning of CFG. i.e. meaning of CFG should not get changed if we place $S \rightarrow \varepsilon$ in other rules we get $S \rightarrow 0$ when $S \rightarrow 0\,S$ and $S \rightarrow \varepsilon$ as well as $S \rightarrow 1$ when $S \rightarrow 1\,S$ and $S \rightarrow \varepsilon$

Hence we can rewrite the rules as

$S \rightarrow 0S \mid 1S \mid 0 \mid 1$

Thus ε production is removed.

Problems on removal of ε production

Example 3.11.7 *Remove the ε production from following CFG by preserving meaning of it.*

$S \rightarrow XYX$

$X \rightarrow 0X \mid \varepsilon$

$Y \rightarrow 1\,Y \mid \varepsilon$

Solution : Now, while removing ε production we are deleting the rules $X \rightarrow \varepsilon$ and $Y \rightarrow \varepsilon$. To preserve the meaning of CFG we are actually placing ε at right hand side wherever X and Y have appeared.

Let us take

$S \rightarrow XYX$

if first X at right hand side is ε.

Then $S \rightarrow YX$

Similarly if last X in R.H.S. = ε.

Then $S \rightarrow XY$

If $Y = \varepsilon$ then

$S \rightarrow XX$

If Y and X are ε then,

$S \rightarrow X$ also

$S \rightarrow Y$ when both X are replaced by ε

$\therefore$ $S \rightarrow XY \mid YX \mid XX \mid X \mid Y$

Now let us consider

$X \rightarrow 0X$

If we place ε at right hand side for X then,

$X \rightarrow 0$

$X \rightarrow 0X \mid 0$

Similarly $Y \rightarrow 1\,Y \mid 1$

Collectively we can rewrite the CFG with removed ε productions as

$S \rightarrow XY \mid YX \mid XX \mid X \mid Y$

$X \rightarrow 0X \mid 0$

$Y \rightarrow 1Y \mid 1$

Example 3.11.8 *For the CFG given below remove the ε production*

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \varepsilon$

Solution : According to the replacement procedure, we will place ε at S in the sentential form.

$S \rightarrow a\,S\,a$ if $S = \varepsilon$

$S \rightarrow a\,a$

Similarly if $S \rightarrow b\,S\,b$ if $S = \varepsilon$

$S \rightarrow b\,b$

Thus finally the rules are

$S \rightarrow a\,S\,a \mid b\,S\,b \mid a\,a \mid b\,b$

Example 3.11.9 *Eliminate the ε productions from the CFG given below.*

$A \rightarrow 0\,B\,1 \mid 1\,B\,1$

$B \rightarrow 0\,B \mid 1\,B \mid \varepsilon$

Solution : Now the ε production is $B \rightarrow \varepsilon$ we will delete this production. And then add the productions having B replaced by ε.

$A \rightarrow 0\,B\,1$

if $B = \varepsilon$

$A \rightarrow 0\,1$

Similarly $A \rightarrow 1\,B\,1 \rightarrow 11$

$\therefore$ $A \rightarrow 0\,B\,1 \mid 1\,B\,1 \mid 01 \mid 11$

Similarly, $B \rightarrow 0\,B$

if $B = \varepsilon$

$B \rightarrow 0$

as well as $B \rightarrow 1$

$B \rightarrow 0\,B \mid 1\,B \mid 0 \mid 1$

Collectively, we can write

$A \rightarrow 0\,B\,1 \mid 1\,B\,1 \mid 01 \mid 11$

$B \rightarrow 0\,B \mid 1\,B \mid 0 \mid 1$

Example 3.11.10 *Construct CFG without ε production from the one which is given below.*

$S \rightarrow a \mid Ab \mid aBa$

$A \rightarrow b \mid \varepsilon$

$B \rightarrow b \mid A$

Solution : If you observe carefully, then not only A have ε production but even B also indicates ε production i.e. $A \rightarrow \varepsilon$ straight forward but $B \rightarrow A \rightarrow \varepsilon$. Let us apply the method of replacement.

$S \rightarrow A\,b$

if $A = \varepsilon$ then

$S \rightarrow b$

if

$$B = \varepsilon$$
$$S \rightarrow aa$$
$$S \rightarrow a|A\ b|b|a\ a|a\ B\ a$$
$$A \rightarrow b$$
$$B \rightarrow b$$

Finally the rules are

$$S \rightarrow a|A\ b|b|a\ a|a\ B\ a$$
$$A \rightarrow b$$
$$B \rightarrow b$$

Example 3.11.11 *Consider the CFG given below.*

$S \rightarrow Y$
$X \rightarrow Zb$
$Y \rightarrow BW$
$Z \rightarrow AB$
$W \rightarrow Z$
$A \rightarrow aA \mid bA \mid \varepsilon$
$B \rightarrow Ba \mid Bb \mid \varepsilon$

Solution : In this example only A and B shows ε productions.

If we put A = ε in A production then

$$A \rightarrow a \mid b$$
$$B \rightarrow a \mid b$$

Also

$$Z \rightarrow A\ B \quad \text{Putting } A = \varepsilon,\ B = \varepsilon$$
$$Z \rightarrow \varepsilon\varepsilon \rightarrow \varepsilon$$

So wherever Z appears we place ε

$$X \rightarrow b$$

Since $W \rightarrow Z \rightarrow \varepsilon$ we can straightaway delete it.

$$\therefore \quad Y \rightarrow b$$

Collectively let us rewrite the rules

$$S \rightarrow XY$$
$$X \rightarrow b$$
$$Y \rightarrow b$$
$$A \rightarrow a\ A|b\ A|a|b$$
$$B \rightarrow B\ a|B\ b|a|b$$

Since S is a start symbol and it defines XY only whereas X and Y yields b as a terminal symbol. There is no chance for producing rules by A or B. Hence A and B are useless symbols and we can remove it. Finally rules are

$$S \rightarrow XY$$
$$X \rightarrow b$$
$$Y \rightarrow b$$

which ultimately

$$S \rightarrow bb$$

Example 3.11.12 *Consider the CFG given below,*

For eliminating ε productions

$S \rightarrow P\,0\,Q \mid Q\,Q \mid 0\,R \mid R\,Q\,P$
$P \rightarrow R\,0 \mid 1\,R \mid R\,R \mid R\,Q\,P$
$Q \rightarrow Q\,0 \mid P\,Q \mid \varepsilon$
$R \rightarrow 0\,P \mid Q\,Q\,Q$

Solution : As we can see that $Q \rightarrow \varepsilon$.

Similarly $R \rightarrow QQQ \rightarrow \varepsilon$.

Hence we will replace Q and R by ε and accordingly add the production rules.

Similarly $P \rightarrow R\,R \rightarrow \varepsilon$

Let us modify the rules if P or Q or R becomes ε.

$$S \rightarrow P\ 0\ Q|0|0\ R|Q\ Q|R\ Q\ P$$
$$P \rightarrow R\ 0|0|1|1\ R|RR|RQP$$
$$Q \rightarrow Q\ 0|0|PQ$$
$$R \rightarrow 0\ P|0|Q\ Q\ Q$$

Note that with P, Q, R other meanings are associated, they are not purely giving ε to us. Therefore we cannot remove P, Q and R symbols.

3.11.3 Removing Unit Productions

The unit productions are the productions in which one non terminal gives another non terminal.

For example if

$$X \rightarrow Y$$
$$Y \rightarrow Z \qquad Z \rightarrow X$$

Then X, Y and Z are unit productions. To optimize the grammar we need to remove the unit productions.

If $A \overset{*}{\Rightarrow} B$ is a unit production and $B \rightarrow X_1\ X_2\ X_3\ .....\ X_n$ then while removing $A \rightarrow B$ production we should add a rule $A \rightarrow X_1\ X_2\ X_3\ ....\ X_n$.

Let us solve something.

Example 3.11.13 *If the CFG is as below.*

$S \rightarrow 0A \mid 1B \mid C$
$A \rightarrow 0S \mid 00$
$B \rightarrow 1 \mid A$
$C \rightarrow 01$

then remove the unit productions.

Solution : Clearly $S \rightarrow C$ is a unit production. But while removing $S \rightarrow C$ we have to consider what C gives. So, we can add a rule to S.

$$S \rightarrow 0A|1B|01$$

Similarly $B \rightarrow A$ is also a unit production so we can modify it as

$$B \rightarrow 1|0S|00$$

Thus finally we can write CFG without unit production as

$$S \rightarrow 0A|1B|01$$
$$A \rightarrow 0S|00$$

B → 1|0S|00

C → 01

Example 3.11.14 *Optimize the CFG given below by reducing the grammar.*
S is a start symbol.

S → A|0CI

A → B|01|10

C → e|CD

Solution : S → A → B is a unit production

C → ε is a null production.

C → CD B and D are useless symbol.

Reducing a grammar we have to avoid all the above conditions.

Let S → A

i.e. A → B is a useless symbol because B is not defined further more.

S → 01|10

i.e. S → 01|10|0C1

But C → ε

Hence ultimately S → 01|10

A → B but we can remove this production since B is a useless symbol.

Hence A → 01|10

But the start symbol S → 01|10

There is no A in the derivation of A so by considering A also as a useless symbol we get final CFG as

S → 01|10

Example 3.11.15 *Eliminate the unit productions from following grammar*

S → AB
A → a
B → C|b
C → D
D → E|bC
E → d|Ab

Solution : As,

S → AB and

A → a

There is only one rule with A and that too giving terminal symbol. Hence there is no question of getting unit production with A. Now consider

B → C

C → D

D → E

It is clear that B, C and D are unit productions. As E → d|Ab, we will replace value of D. Then,

D → d | Ab | bC

Similarly as C → D we can write

C → d|Ab|bC

Hence B becomes

B → d|Ab|bC|b

Thus the grammar after removing unit productions

S → AB

A → a

B → d|Ab|bC|b

C → d|Ab|bC

D → d|Ab|bC

E → d|Ab.

Now there is no path for D and E. From start state we can remove them by considering useless symbols. The optimized grammar will be

S → AB

A → a

B → d|Ab|bC|b

C → d|Ab|bC

Example 3.11.16 *Simplify the following grammar :*
i) S → Ab, A → a, B → C | b, C → D, D → E, E → a
ii) S → 0A0 | 1B1 | BB, A → C, B → S | A, C → S | ε

SPPU : Dec.-16, End Sem, Marks 8

Solution :

i) Let,

S → Ab

A → a

B → C | b

C → D

D → E

E → a

Step 1 : Here B → C, C → D, D → E and E → a, are unit productions. These can be eliminated and we get simply B → a | b.

Step 2 : Now if we observe the start state S, we can clearly say that non terminal B is non-reachable. Hence eliminate it as useless symbol.

Step 3 : Thus we get the simplified grammar, on removing B, C, D and E as

S → Ab

A → a

ii) Let,

S → 0A0

S → 1B1

S → BB

A → C

B → S | A

C → S | ε

Step 1 : The A → C, C → S | ε, are unit productions. These can be eliminated A → S | ε.

Step 2 : B → A, A → S | ε are unit productions, these can be eliminated by B → ε

Step 3 : As we have

$$S \rightarrow 0A0 \mid 1B1$$

Also $A \rightarrow S$ and $B \rightarrow S$

∴ We can reduce it as

$$S \rightarrow 0S0 \mid 1S1$$

Step 4 : Similarly $S \rightarrow BB$ can be reduced.

Step 5 : Finally we get

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$$

as simplified productions.

3.12 Normal Forms

There are two important normal forms - **Chomsky's Normal Form** and **Greibach Normal Form**

Let us study these normal forms with the help of examples -

3.12.1 Chomsky Normal Form

The Chomsky's Normal Form can be defined as

Non terminal → Non terminal · Non terminal
Non terminal → Terminal

The given CFG should be converted in the above format then we can say that the grammar is in CNF. Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, ε productions and unit productions from it. Thus this reduced grammar can be then converted to CNF.

Let us solve some examples for Chomsky's Normal Form.

Problems on CNF

Example 3.12.1 *Convert the following CFG into CNF*

S → aaaaS

S → aaaa

Solution : As we know the rule for Chomsky's normal form is

Non terminal → Non terminal · Non terminal

Non terminal → Terminal

But the CFG given is

$S \rightarrow aaaaS$ rule 1

$S \rightarrow aaaa$ rule 2

If we add a rule

Then out rule 1 and 2 becomes,

$$A \rightarrow AAAAS$$

$$S \rightarrow AAAA$$

Let us take

$S \rightarrow A \boxed{AAAS}$ can be replaced by P_1

If we define

$P_1 \rightarrow AAAS$ then the rule becomes

$S \rightarrow A\ P_1$ which is in CNF

But the new rule P_1 is not CNF, so let us convert it

$P_1 \rightarrow A \boxed{AAS}$ can be replaced by P_2

then $P_1 \rightarrow A\ P_2$ which is in CNF

and $P_2 \rightarrow AAS$ which is not in CNF so let us convert it to CNF

$P_2 \rightarrow A \boxed{AS}$ can be replaced by P_3

$P_2 \rightarrow A\ P_3$ where $P_3 \rightarrow AS$

Now both P_2 and P_3 are in CNF.

Collectively rewrite these rules (these all indicate rule 1 of the given CFG).

$$S \rightarrow A\ P_1$$

$$P_1 \rightarrow A\ P_2$$

$$P_2 \rightarrow A\ P_3$$

$$P_3 \rightarrow A\ S$$

Now consider rule 2

$$S \rightarrow AAAA$$

If we break the rule 2 as

$S \rightarrow \boxed{AA}\boxed{AA}$

$\downarrow \quad \downarrow$ indicated by

$P_4 \quad P_5$

The rule becomes

$S \rightarrow P_4\ P_5$ which is in CNF.

But P_4 and P_5 indicate the same rule. So we can eliminate either of them.

Hence rule 2 becomes

$$S \rightarrow P_4\ P_4$$

Finally we can collectively show the CFG converted to CNF as

$$S \rightarrow A\ P_1$$

$$P_1 \rightarrow A\ P_2$$

$$P_2 \rightarrow A\ P_3$$

$$P_3 \rightarrow A\ S$$

$$S \rightarrow P_4\ P_4$$

$$P_4 \rightarrow AA$$

$$A \rightarrow a$$

Example 3.12.2 *Convert the given CFG to CNF S → aSa | bSb |a|b*

Solution : Let us start by adding new symbols for the terminals.

$$S \rightarrow ASA$$

$$A \rightarrow a$$

$$S \rightarrow BSB$$

$$B \rightarrow b$$

Note that although $s \rightarrow a$ and $A \rightarrow a$ are similar still we are not replacing any *a* by S. This is because, S is not simply giving *a*. It has other productions also. We want such a non terminal having only one production and that is *a*. Same is with B. Hence we have added the rules $A \rightarrow a$ and $B \rightarrow b$.

Let us take $S \rightarrow ASA$ for converting to CNF.

$S \rightarrow A \boxed{SA}$

replace it by A_1

$S \rightarrow A\ A_1$

$A_1 \rightarrow SA$

Take, $S \rightarrow B\boxed{SB}$

replace it by A_2

$S \rightarrow B A_2$ both are in CNF

$A_2 \rightarrow SB$

Hence we can write the rule in CNF as

$S \rightarrow A A_1$

$A_1 \rightarrow SA$

$A \rightarrow a$

$S \rightarrow B A_2$

$A_2 \rightarrow SB$

$B \rightarrow b$

$S \rightarrow a$

$S \rightarrow b$

Example 3.12.3 *Consider* $a = (\{S, A\}, \{a, b\}, P, S)$

Where P consists of

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$

Converi it to its equivalent CNF.

Solution : Let us consider,

$S \rightarrow aAS$

If we put $R_1 \rightarrow a$

then $S \rightarrow R_1AS$

and if $R_2 \rightarrow AS$

then S becomes

$S \rightarrow R_1 R_2$ is now in CNF

$S \rightarrow a$ is already in CNF

$A \rightarrow SbA$

Let $R_3 \rightarrow b$

Hence $A \rightarrow S R_3A$

Add $R_4 \rightarrow R_3A$

then $A \rightarrow S R_4$

$A \rightarrow SS$ is already in CNF

Now as we have defined $R_1 \rightarrow a$ and $R_3 \rightarrow b$.

$A \rightarrow R_1 R_3$ will be a conversion in CNF of $A \rightarrow ab$.

Finally, we can write

$S \rightarrow R_1 R_2$ $A \rightarrow S R_4$

$S \rightarrow a$ $R_4 \rightarrow R_3A$

$R_1 \rightarrow a$ $R_3 \rightarrow b$

$R_2 \rightarrow AS$ $A \rightarrow SS$

$A \rightarrow R_1 R_3$

Example 3.12.4 *Convert the given CFG to CNF.*

Consider $G = (V, T, P, S)$

Where $V = \{S, A, B\}$

$T = \{a, b\}$

P consists of

$S \rightarrow aB$ $A \rightarrow bAA$

$S \rightarrow bA$ $B \rightarrow b$

$A \rightarrow a$ $B \rightarrow bS$

$A \rightarrow aS$ $B \rightarrow aBB$

SPPU : Dec.-05, Marks 12, Dec.-14, In Sem, Marks 6; Dec.-16, End Sem, Marks 8

Solution : Let us start with first rule.

$R_1 \rightarrow a$

$S \rightarrow R_1B$ is in CNF for $S \rightarrow aB$

$R_2 \rightarrow b$

$S \rightarrow R_2A$ is in CNF for $S \rightarrow bA$

$A \rightarrow a$

$A \rightarrow aS$ can be written as

$A \rightarrow R_1S$

Now $A \rightarrow bAA$ can be written as

$A \rightarrow R_2\boxed{AA}$ replace it by R_3

i.e. $R_3 \rightarrow AA$

then $A \rightarrow R_2R_3$

$B \rightarrow a$

$B \rightarrow bS$ can be written as

$B \rightarrow R_2S$ is in CNF

Now, $B \rightarrow aBB$

i.e. $B \rightarrow R_1BB$ $\because$ $R_1 \rightarrow a$

Let $R_4 \rightarrow BB$

then $B \rightarrow R_1 R_4$

Finally we can write,

$S \rightarrow R_1B$

$S \rightarrow R_2A$

$A \rightarrow R_1S$

$A \rightarrow R_2 R_3$

$B \rightarrow b$

$B \rightarrow R_2S$

$B \rightarrow R_1 R_4$

$R_1 \rightarrow a$

$R_2 \rightarrow b$

$R_3 \rightarrow AA$

$R_4 \rightarrow BB$

is a Chomsky's normal form.

Example 3.12.5 *Convert the following CFG to CNF*

$S \rightarrow ABA$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

SPPU : May-13, 14, Marks 6

Solution : In the Chomsky's normal form the ε production is not allowed. So first we will eliminate ε productions.

$A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$

If put ε instead of A and B

We can get,

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$

Similarly, $A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Now S → A and S → B is unit production getting introduced in the grammar after removal of ε production. So we will remove unit production also,

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Now let us convert this grammar to Chomsky's normal form.

Let

$S \rightarrow ABA$

$\boxed{S \rightarrow ABA}$

$\boxed{R_1 \rightarrow BA}$

then $S \rightarrow AR_1$

Now $\boxed{S \rightarrow AB}$ $\boxed{S \rightarrow BA}$ $\boxed{S \rightarrow AA}$ already in CNF

Let $S \rightarrow aA$

We will define $R_2 \rightarrow a$

∴ $\boxed{S \rightarrow R_2A}$

$\boxed{R_2 \rightarrow a}$

$\boxed{S \rightarrow a}$ already in CNF

$S \rightarrow bB$

Let $\boxed{R_3 \rightarrow b}$ then

$\boxed{S \rightarrow R_3B}$

$S \rightarrow b$ already in CNF

Now consider the rules,

$\boxed{A \rightarrow aA}$

We can rewrite it as

$\boxed{A \rightarrow R_2A}$

$\boxed{A \rightarrow a}$

Also, $B \rightarrow bB$ can be rewritten as

$B \rightarrow R_3 B$

$B \rightarrow b$

Collect all the CNF s to integrate and we will get

$S \rightarrow A R_1 \mid AB \mid BA \mid AA \mid R_2A \mid R_3B \mid a \mid b$

$A \rightarrow R_2A \mid a$

$B \rightarrow R_3 B \mid b$

$R_1 \rightarrow BA$

$R_2 \rightarrow a$

$R_3 \rightarrow b$

is done.

Example 3.12.6 *Convert the following grammar to Chomsky's normal form*

$S \rightarrow a \mid b \mid a S S.$

Solution : The Chomsky's normal form is

$NT \rightarrow NT \cdot NT$

$NT \rightarrow$ Terminal

We will convert the given grammar in this form

$S \rightarrow a$ already in CNF

$S \rightarrow b$ already in CNF

$S \rightarrow a S S$

Now we will convert

$S \rightarrow a S S$ into equivalent CNF

$S \rightarrow R_1 R_2$

$R_1 \rightarrow a$

$R_2 \rightarrow S S$

∴ The equivalent CNF of given grammar is -

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow R_1 R_2$

$R_1 \rightarrow a$

$R_2 \rightarrow S S.$

Example 3.12.7 *Convert the following grammar to Chomsky's normal form*

$S \rightarrow \sim S \mid [S \supset S] \mid p \mid q.$

SPPU : Dec.-13, Marks 6

Solution : Chomsky's normal form is -

Non terminal → Non terminal · Non terminal

Non terminal → Terminal

Consider given grammar rule by rule and let us convert it into CNF

$S \rightarrow \sim S$

Can be written as

$S \rightarrow AS$

$A \rightarrow \sim$

Similarly $S \rightarrow [S \supset S]$

Can be written as

$S \rightarrow BC$

$B \rightarrow DE$

$E \rightarrow SF$

$D \rightarrow [$

$F \rightarrow \supset$

$C \rightarrow SG$

G →]

And, S → p

S → q

is already in CNF.

Hence the complete grammar written in CNF is -

S → AS

A → ~

S → BC

B → DE

E → SF

D → [

F → ⊃

C → SG

G →]

S → p

S → q

Example 3.12.8 *Convert the following grammar to CNF*

$S \rightarrow AACD$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

$A \rightarrow aDa \mid bDb \mid \varepsilon$

SPPU : May-10, Dec.-10, Marks 6

Solution : Before converting any grammar to its normal form we must first simplify it by removing ε production and by eliminating useless symbols.

Let S → A A A D

But there is no production rule defined for given non-terminal D. Hence S and D are useless symbols. But S is a start symbol and if we eliminate it then the grammar becomes **invalid**. Hence **conversion** of such grammar to CNF is **not possible**.

Example 3.12.9 *Reduce the following grammar to Chomsky Normal Form :*

$S \rightarrow 1A \mid 0B$

$A \rightarrow 1AA \mid 0S \mid 0$

$B \rightarrow 0BB \mid 1S \mid 1$

SPPU : Dec.-13, Marks 6

Solution : Refer example 3.12.4 by assuming a = 0 and b = 1.

Example 3.12.10 *Convert the following CFG into Chomsky Normal Form (CNF) :*

$S \text{ - > } AB$

$A \text{ - > } CA \mid \wedge$

$B \text{ - > } DB \mid \wedge$

$C \text{ - > } 011 \mid 1$

$D \text{ - > } 01$

SPPU : Dec.-15, End Sem, Marks 6

Solution : Before converting the given grammar to CNF we will eliminate ∧ production.

A → CA | ∧ becomes

A → CA | 011 | 1

Similarly

B → DB | ∧ becomes

B → DB | 01

The CNF conversion is as follows :

Production Rule	CNF
S → AB	already is in CNF
A → CA	is already in CNF
A → 011	A → XY X → 0 Y → ZZ Z → 1
B → DB	is already in CNF
B → XZ	

3.12.2 Greibach Normal Form

Now we will discuss one more interesting normal form called **Greibach Normal Form.**

The rule for GNF is

Non-terminal → One terminal · Any number of non-terminals

In short,

$$NT \rightarrow t \cdot NT$$

For example :

S → aA
S → a
} is in GNF

But S → AA or

S → Aa is not in GNF

To convert given CFG into GNF very interesting procedure is followed. We can use two important lemmas based on which it is easy to convert given CFG to GNF.

Lemma 1 : Let,

G = (V, T, P, S) be a given CFG and if there is a production A → Ba and $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Then we can convert A rule to GNF as

$$A \rightarrow \beta_1 a \mid \beta_2 a \mid \beta_3 a \mid \ldots \mid \beta_n a$$

For example :

S → Aa

A → aA | bA | aAS | b

Then we can convert S rule in GNF as

S → aAa | bAa | aASa | ba

A → aA | bA | aAS

Note that both the rules are in GNF.

Lemma 2 : Let,

G = (V, T, P, S) be a given CFG and if there is a production.

$$A \rightarrow Aa_1 \mid Aa_2 \mid Aa_3 \mid \ldots \mid Aa_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Such that β_i do not start with A then equivalent grammar in Greibach normal form can be,

$$A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

$$A \rightarrow \beta_1 Z \mid \beta_2 Z \mid \beta_3 Z \mid \ldots \mid \beta_n Z$$

$$Z \rightarrow a_1 \mid a_2 \mid a_3 \mid \ldots \mid a_n$$

$$Z \rightarrow a_1 Z \mid a_2 Z \mid \ldots \mid a_n Z$$

For example :

Consider,

$$A \rightarrow A1 \mid 0B \mid 2$$

Here $\beta_1 = 0B$ and $\beta_2 = 2$, $a_1 = 1$ then,

$$A \rightarrow 0B \mid 2$$

$$A \rightarrow 0BZ \mid 2Z$$

$$Z \rightarrow 1$$

$$Z \rightarrow 1Z$$

Let us solve some problems based on conversion to GNF.

Example 3.12.11 *Convert the following grammar to Greibach Normal Form.*

$S \rightarrow abSb$

$S \rightarrow aa$

Solution : As the GNF form is

$$NT \rightarrow t \cdot NT_i \qquad \text{where } i \geq 0.$$

Consider each production rule

$$S \rightarrow abSb$$

If we create another NT for defining terminal b we will get,

$$B \rightarrow b$$

Hence

$$S \rightarrow aBSB$$

Similarly for production.

$$S \rightarrow aa$$

We need to define terminal symbol a.

$$A \rightarrow a$$

Hence, $S \rightarrow aA$

Now, to summerize, equivalent GNF will be

$$S \rightarrow aBSB$$

$$S \rightarrow aA$$

Example 3.12.12 *Convert the given CFG to GNF.*

$S \rightarrow ABA$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

SPPU : Dec.-10, Marks 6

Solution : As before converting the CFG to its normal form we usually reduce the grammar i.e. we first eliminate ε production, unit production and useless symbols.

As $A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$

Let us simplify the given CFG by removing ε productions and the CFG will look like this,

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

[We can even remove the unit productions]

Let us replace A by aA or a in S

$$S \rightarrow aABA \mid aAB \mid aBA \mid aB \mid aAA \mid aA \mid a$$

which is GNF

Similarly $S \rightarrow BA \mid B$ can be written as

$$S \rightarrow bBA \mid bA \mid bB \mid b$$

is also in GNF.

Finally equivalent GNF will be -

$$S \rightarrow aABA \mid aAB \mid aBA \mid aB \mid aAA \mid aA \mid a$$

$$S \rightarrow bBA \mid bA \mid bB \mid b$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Example 3.12.13 *Convert given CFG to GNF where $V = \{S, A\}$, $T = \{0, 1\}$ and P is*

$S \rightarrow AA \mid 0$

$A \rightarrow SS \mid 1$

SPPU : Dec.-12, Marks 6

Solution : Let us rename S as A_1 and A as A_2 then given CFG becomes

$$A_1 \rightarrow A_2A_2 \mid 0$$

$$A_2 \rightarrow A_1A_1 \mid 1$$

Let us start with A_2. The rule for A_2 is

$$A_2 \rightarrow A_1A_1 \mid 1$$

Now replace first A_1 on R.H.S. by rule of A_1

$\therefore$ $A_2 \rightarrow A_2A_2A_1 \mid 0A_1 \mid 1$

According to lemma 1 if

$$A \rightarrow Aa_1 \mid Aa_2 \mid \ldots \mid Aa_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Then,

$$A \rightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_n$$

$$A \rightarrow \beta_1 Z \mid \beta_2 Z \ldots \mid \beta_n Z$$

$$Z \rightarrow a_1 \mid a_2 \mid \ldots \mid a_n$$

$$Z \rightarrow a_1 Z \mid a_2 Z \mid \ldots \mid a_n Z$$

We can map this lemma to our A_2 rule as $A = A_2$, $a_1 = A_2A_1$, $\beta_1 = 0A_1$ and $\beta_2 = 1$.

Then we get,

$$A_2 \rightarrow 0A_1 \mid 1$$

$$A_2 \rightarrow 0A_1Z \mid 1Z$$

$$Z \rightarrow A_2A_1$$

$$Z \rightarrow A_2A_1Z$$

Thus now we have obtained A_2 being in GNF. Now, consider production for A_1.

As $A_1 \rightarrow A_2A_2 \mid 0$

We will replace first A_2 on R.H.S. by its recently GNF rules, then we get,

$A_1 \rightarrow 0A_1A_2 \mid 1A_2 \mid 0A_1ZA_2 \mid 1ZA_2 \mid 0 \rightarrow$ using lemma 2

Thus now A_1 is also in GNF.

Now, consider Z rule.

$$\left.\begin{array}{l} Z \rightarrow A_2A_1 \\ Z \rightarrow A_2A_1Z \end{array}\right\} \begin{array}{l} \text{using lemma 2} \\ \text{replace } A_2 \text{ at R.H.S.} \end{array}$$

Hence we get,

$$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1ZA_1 \mid 1ZA_1$$

$$Z \rightarrow 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$$

Now let us rewrite the rules by converting back $A_1 = S$ and $A_2 = A$

$$S \rightarrow 0SA \mid 1A \mid 0SZA \mid 1ZA \mid 0$$

$$A \rightarrow 0S \mid 1 \mid 0SZ \mid 1Z$$

$$Z \rightarrow 0SS \mid 1S \mid 0SZS \mid 1ZS$$

$$Z \rightarrow 0SSZ \mid 1SZ \mid 0SZSZ \mid 1ZSZ$$

is equivalent GNF.

Example 3.12.14 *Convert the given CFG to GNF.*

$S \rightarrow CA$

$A \rightarrow a$

$C \rightarrow aB \mid b$

Solution : If you observe the given grammar A and C are in Greibach Normal Form. But S is not in GNF.

Let us apply lemma 1 for S

$$S \rightarrow CA \quad \text{can be written as}$$

$$S \rightarrow aBA \mid bA$$

Thus we can write equivalent GNF as

$$S \rightarrow aBA \mid bA$$

$$A \rightarrow a$$

$$C \rightarrow aB \mid b$$

Example 3.12.15 *Convert the grammar $S \rightarrow AB$, $A \rightarrow BS \mid b$, $B \rightarrow SA \mid a$ into Greibach normal form.*

SPPU : Dec.-06,07, May-07, Marks 8; Dec.-14, End Sem. Marks 4

Solution : Consider the grammar

$$S \rightarrow AB$$

$$A \rightarrow BS \mid b$$

$$B \rightarrow SA \mid a$$

Now assume $S = A_1$, $A = A_2$ and $B = A_3$ then we can rewrite the rules as-

$$A_1 \rightarrow A_2 A_3$$

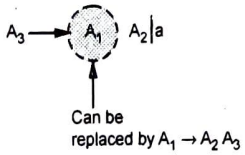
$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow A_1 A_2 \mid a$$

Now we will consider A_3 productions and let us apply two important lemma required to convert given CFG to GNF.

$$A_3 \rightarrow A_1 A_2 \mid a$$

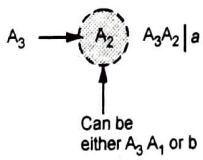
The lemma 1 suggests us to replace A_1 present in R.H.S. of rule A_3 by the rule of A_1. Hence



Hence we get

$$A_3 \rightarrow A_2 A_3 A_2 \mid a$$

Again in rule A_3 we get $A_2 A_3 A_2$ i.e. we can replace first A_2 by its R.H.S. production. By this we are actually applying lemma 1. Hence



Hence we get,

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid bA_3 A_2 \mid a$$

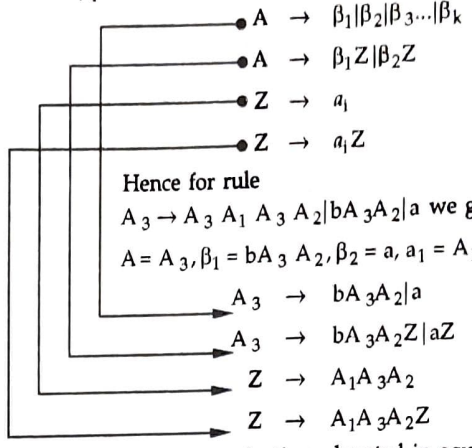
Now let us apply lemma 1 on this production.

Lemma 1 is -

If there is a rule

$A \rightarrow Aa_1 \mid Aa_2 \mid Aa_3 \mid \ldots Aa_n \ldots \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_k$

Such that β_i do not start with A then



Hence for rule

$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid bA_3 A_2 \mid a$ we get

$A = A_3, \beta_1 = bA_3 A_2, \beta_2 = a, a_1 = A_1 A_3 A_2$ we get,

$$A_3 \rightarrow bA_3 A_2 \mid a \quad \ldots (1)$$

$$A_3 \rightarrow bA_3 A_2 Z \mid aZ \quad \ldots (2)$$

$$Z \rightarrow A_1 A_3 A_2$$

$$Z \rightarrow A_1 A_3 A_2 Z$$

Now if we observe A_3 productions denoted in equations (1) and (2) we get it in GNF form i.e. NT → t · NT form. Finally we can rewrite A_3 as

$$\boxed{A_3 \rightarrow bA_3 A_2 \mid a \mid bA_3 A_2 Z \mid aZ} \quad \ldots (3)$$

Now consider A_2 production and let us apply lemma 1 for this rule.

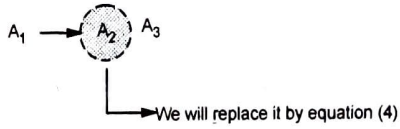
$$A_2 \rightarrow A_3 A_1 \mid b$$

Replace it by equation (3), we get,

$$\boxed{A_2 \rightarrow bA_3 A_2 A_1 \mid aA_1 \mid bA_3 A_2 Z A_1 \mid aZA_1 \mid b} \quad \ldots (4)$$

Clearly, A_2 is now in GNF.

Now let us concentrate on A_1. A_S



Hence

$$\boxed{A_1 \rightarrow bA_3 A_2 A_1 A_3 \mid aA_1 A_3 \mid bA_3 A_2 Z A_1 A_3 \mid aZA_1 A_3 \mid bA_3} \quad \ldots (5)$$

is in GNF form. Now the only remaining production of conversion is Z. Let us deal with it. As

$$Z \rightarrow A_1 A_3 A_2 \mid A_1 A_3 A_2 Z$$

At R.H.S. A_1 comes as the first symbol. Hence instead of A_1 if we put it R.H.S. we get,

$Z \rightarrow bA_3 A_3 A_2 bA_3 A_2 A_1 A_3 A_3 A_2 \mid aA_1 A_3 A_3 A_2$
$\mid bA_3 A_2 ZA_1 A_3 A_3 A_2 \mid aZA_1 A_3 A_3 A_2 \mid$

$Z \rightarrow bA_3 A_2 A_1 A_3 A_3 A_2 Z \mid aA_1 A_3 A_3 A_2 Z \mid bA_3 A_2 ZA_1 A_3 A_3 A_2 Z \mid$
$aZA_1 A_3 A_3 A_2 Z \mid bA_3 A_3 A_2 Z$

Thus we get Z production also in GNF form.

Example 3.12.16 *Reduce the following grammar to Greibach normal form.*

$S \rightarrow SS, \; S \rightarrow 0S1 \mid 01$ **SPPU : Dec.-11, Marks 8**

Solution : The Greibach normal form is -

NT → terminal (set of NT)

Let S → SS|0S1|01 be the given grammar.

Let S → 0SBS

$S \rightarrow 0SB$
$S \rightarrow 0B$
$B \rightarrow 1$

be equivalent GNF.

Review Question

1. *What is a normal form ? Explain Chomsky Normal Form (CNF) and Greibach Normal Form (GNF) with suitable example.*

SPPU : May-08, Marks 8

3.13 Closure Properties of CFL

In chapter 2, we have seen that regular languages are closed under union, concatenation and kleen closure. Now we will discuss closure properties of context free languages. In the sense, we will, check which are those properties for them CFLs are closed under. The context free languages are closed under some operation means after performing that particular operation on those CFLs the resultant language is context free language. These properties are as below

1. The context free languages are closed under union.
2. The context free languages are closed under concatenation.
3. The context free languages are closed under kleen closure.
4. The context free languages are not closed under intersection.
5. The context free languages are not closed under complement.

We will discuss the above mentioned closure properties of CFL with the help of proofs and examples.

Theorem 1 : If L_1 and L_2 are context free languages then $L = L_1 \cup L_2$ is also context free. That is, the CFLs are closed under union.

Proof : We will consider two languages L_1 and L_2 which are context free languages. We can give these languages using context free grammars G_1 and G_2 such that $G_1 \in L_1$ and $G_2 \in L_2$. The G_1 can be given as $G_1 = \{V_1, \Sigma, P_1, S_1\}$ where P_1 can be given as

$P_1 = \{$
$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \varepsilon$
$A_1 \rightarrow a$
$B_1 \rightarrow b$
$\}$

Here $V_1 = \{S_1, A_1, B_1\}$ and S_1 is a start symbol.

Similarly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$

$N_2 = \{S_2, A_2, B_2\}$ and S_2 is a start symbol.

P_2 can be given as :

$P_2 = \{$
$S_2 \rightarrow a\ A_2\ A_2 \mid b\ B_2\ B_2$
$A_2 \rightarrow b$
$B_2 \rightarrow a$
$\}$

Now $L = L_1 \cup L_2$ gives $G \in L$. This G can be written as

$G = \{V, \Sigma, P, S\}$
$V = \{S_1, A_1, B_1, S_2, A_2, B_2\}$
$P = \{P_1 \cup P_2\}$

S is a start symbol

$P = \{$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \varepsilon$
$A_1 \rightarrow a$
$B_1 \rightarrow b$
$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$
$A_2 \rightarrow b$
$B_2 \rightarrow a$
$\}$

Thus grammar G is a context free grammar which produces languages L which is context free language.

Theorem 2 : If L_1 and L_2 are two context free languages then L_1L_2 is CFG. That means context free languages are closed under concatenation.

Proof : Let L_1 is a context free language which can be represented by a context free grammar G_1, such that $G_1 \in L_1$ and

$G_1 = \{V_1, \Sigma, P_1, S_1\}$
$V_1 = \{S_1, A_1, B_1\}$
$\Sigma = \{a, b\}$

S_1 is a start symbol and P_1 is a set of production rules,

$P_1 = \{$
$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$
$A_1 \rightarrow a$
$B_1 \rightarrow b$
$\}$

Similarly, L_2 is a context free language which can be represented by a context free grammar G_2, such that $G_2 \in L_2$ and

$G_2 = \{V_2, \Sigma, P_2, S_2\}$
$V_2 = \{S_2, A_2, B_2\}$
$\Sigma = \{a, b\}$

S_2 is a start symbol and P_2 is a set of production rules,

$P_2 = \{$
$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$
$A_2 \rightarrow b$
$B_2 \rightarrow a$
$\}$

Now $L = L_1 L_2$ can be obtained by G such that $G = G_1 \cup G_2$ Therefore

$G = \{V, \Sigma, P, S\}$
$V = \{S, S_1, A_1, B_1, S_2, A_2, B_2\}$

where S is a start symbol. The production rules, P can be given as

$P = \{$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$
$A_1 \rightarrow a$
$B_1 \rightarrow b$
$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$
$A_2 \rightarrow b$
$B_2 \rightarrow a$
$\}$

As grammar G is context free grammar the language L produced by G is also context free language. Hence context free languages are closed under concatenation.

Theorem 3 : If L_1 is context free language then L_1^* is also context free. That means CFL is closed under kleen closure.

Proof : Let, L_1 be a context free language represented by G_1 such that $G_1 \rightarrow \varepsilon\ L_1$.

The CFG G_1 can be given as

$G_1 = \{V_1, \Sigma, P_1, S_1\}$ where S_1 is a start symbol

$P_1 = \{ S_1 \rightarrow A_1S_1A_1 \mid B_1S_1B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

Now $L = L_1^*$ can be represented by a grammar G such that

$G = \{ (V, \Sigma, P, S)$

$V = \{S, S_1, A_1, B_1\}$

and $P = S \rightarrow S_1\ S \mid \varepsilon$

$S_1 \rightarrow A_1\ S_1\ A_1 \mid B_1\ S_1\ B_1$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

Thus grammar G is a context free grammar and language L produced by G is also context free language. Hence context free language are closed under kleen closure.

Theorem 4 : If L_1 and L_2 are two CFLs then $L = L_1 \cap L_2$ may be CFL or may not be CFL. That means L is not closed under intersection.

Proof : Let, $L_1 = \{0^n\ 1^n\ 2^i \mid n \geq 1, i \geq 1\}$

$L_2 = \{0^n\ 1^n\ 2^n \mid n \geq, i \geq 1\}$

The grammar for L_1 is

$S \rightarrow AB$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

Similarly L_2 can be represented by grammar.

$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 1B2 \mid 12$

Now if we try to obtain

$L = L_1 \cap L_2$ then we get sometimes context free languages and sometimes non context free languages. Thus we can say that CFLs are not closed under intersection.

Theorem 5 : If L_1 is a CFL then L_1' may or may not be CFL. That means CFL is not closed under complement.

Proof : Let L_1 and L_2 are two CFLs. We will assume that complement of a context free language is a CFL itself. Hence L_1' and L_2' both are CFLs. We can also state that $(L_1' \cup L_2')$ is context free (since CFLs are closed under union). But $(L_1' \cup L_2') = L_1 \cap L_2$ i.e. $L = L_1 \cap L_2$ may or may not be CFL. The L_1 and L_2 are arbitrary CFLs, there may exist L_1' and L_2' which are not CFL. Hence complement of certain language may be context free or may not be. Therefore we can say that CFL is not closed under complement operation.

Theorem 6 : Context Free Languages are closed under homomorphism.

Proof : Let, G = (V,T,P,S) be the context free grammar for generating the language L. The input set is $\sum$ and let Γ be the set of substitution symbols. The $\sum^* \rightarrow \Gamma^*$ be a homomorphism.

Let $a \in L$ and w be the substitution that replaces every $a \in \sum$.

Then $h(w) = h(a_1)h(a_2)h(a_3)$ which ultimately generates h(L). Thus $h(a_i)$ is homomorphism for every $a_i \in \sum$.

For example : Consider

$P = S \rightarrow aSb \mid bSb \mid \varepsilon$

Let $h(a) = 00$

$h(b) = 11$

Then $h(w) = S \rightarrow 00S11 \mid 11S00 \mid \varepsilon$

is again a GFG.

Hence it is proved that context free languages are closed under homomorphism.

Review Question

1. *Write a note on closure properties of CFL.*

SPPU : May 09, Marks 4

3.14 Decision Properties of CFL

SPPU : Dec.-11, Marks 5

Various decision properties of CFLs are

1. Emptiness
2. Finiteness
3. Membership

These properties are discussed in detail as follows

1. Emptiness

Theorem : There exists an algorithm which can determine whether or not the given context free grammar can generate any word at all.

Proof : The algorithm to determine whether or not a grammar generates any word is given below -

i) Put the dot (•) above every terminal symbol. If there is any null string ε then put dot over ε as well. These symbols occur at right hand side of production rule.

ii) If all the symbols of right hand side of the production rule are dotted then dot the corresponding non terminal (which is at left hand side of production rule) throughout the grammar. Repeat this step as long as possible.

iii) If the stating symbol gets dotted then answer is yes otherwise answer is no.

Let us apply this algorithm on some grammar.

Let,

$S \rightarrow PQ$

$P \rightarrow AP \mid AA$

$A \rightarrow a$

$$Q \rightarrow BQ \mid BB$$
$$B \rightarrow b$$

Now if we want to put dots we will put over terminals at RHS of rule.

Step 1 :

1) $S \rightarrow PQ$
2) $P \rightarrow AP \mid AA$
3) $A \rightarrow \dot{a}$
4) $Q \rightarrow BQ \mid BB$
5) $B \rightarrow \dot{b}$

Step 2 :

As we RHS of rule 3) and RHS of rule 5) is dotted we will dot A and B throughout the grammar.

1) $S \rightarrow PQ$
2) $P \rightarrow \dot{A}P \mid \dot{A}\dot{A}$
3) $\dot{A} \rightarrow \dot{a}$
4) $Q \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$
5) $\dot{B} \rightarrow \dot{b}$

Now, in rule 2) the alternate production at RHS is dotted $\left(\dot{A}\dot{A}\right)$. Hence dot P. Similarly dot Q.

Hence,

1) $S \rightarrow \dot{P}\dot{Q}$
2) $\dot{P} \rightarrow \dot{A}P \mid \dot{A}\dot{A}$
3) $\dot{A} \rightarrow \dot{a}$
4) $\dot{Q} \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$
5) $\dot{B} \rightarrow \dot{b}$

Thus in rule 1) the R.H.S. is dotted. Hence dot S. But since S being dotted is a start symbol, the algorithm generates answer yes. Hence we can prove that an algorithm exists which determines whether or not the given grammar generates any string.

Finiteness

Theorem : There exists an algorithm to determine whether the given context free grammar generates a finite or infinite language. **SPPU : Dec.-11, Marks 5**

Proof : We will write following algorithm -

1. Make the grammar simplified by removing unit productions and useless symbols.
2. Check whether there is any self-embedded non terminal or not by following steps.
   a) Underline some non terminal say X which is at LHS of production rule.
   b) Then put dot over all the X which are at RHS throughout the grammar. The dotted X and underlined X are treated as two different symbols.
   c) Put dot over any non terminal at LHS whose RHS contains any dotted symbols. Dot this non terminal throughout the grammar.
   d) If underlined X gets dotted then X is called self embedded otherwise it is not self embedded.
3. If the grammar contains any self embedded non terminal then it generates infinite languages otherwise it generates finite languages.

To support this algorithm, let us take one grammar.

$$S \rightarrow PQa \mid bpz \mid b$$
$$P \rightarrow Xb \mid bZa$$
$$Q \rightarrow bPP$$
$$X \rightarrow aZa \mid aaa$$
$$Z \rightarrow ZPbP$$

The above grammar has Z as useless symbol. Hence by removing it we get,

1) $S \rightarrow PQa \mid b$
2) $P \rightarrow Xb$
3) $Q \rightarrow bPP$
4) $X \rightarrow aaa$

Now we will mark X of rule 4) underlined and put dot over x in rule 2)

1) $S \rightarrow PQa \mid b$
2) $P \rightarrow \dot{X}b$
3) $Q \rightarrow bPP$
4) $\underline{X} \rightarrow aaa$

As R.H.S. of rule 2) contains one dotted symbol dot P throughout the grammar

$$S \rightarrow \dot{P}Qa \mid b$$
$$\dot{P} \rightarrow \dot{X}b$$
$$Q \rightarrow b\dot{P}\dot{P}$$
$$\underline{X} \rightarrow aaa$$

Then Q gets dotted

$$S \rightarrow \dot{P}\dot{Q}a \mid b$$
$$\dot{P} \rightarrow \dot{X}b$$
$$\dot{Q} \rightarrow b\dot{P}\dot{P}$$
$$\underline{X} \rightarrow aaa$$

The S gets dotted

$$\dot{S} \rightarrow \dot{P}\dot{Q}a \mid b$$
$$\dot{P} \rightarrow \dot{X}b$$
$$\dot{Q} \rightarrow b\dot{P}\dot{P}$$

$$X \rightarrow aaa$$

Now all the non terminals are dotted but X is not. That means X is not self embedded symbol. Hence the grammar generates finite language.

Membership : There exists an algorithm which tells whether given string belongs to given grammar.
To check whether the given grammar generates desired string the derivation tree can be drawn.

Consider the grammar,

$$S \rightarrow PQa \mid b$$
$$P \rightarrow Xb$$
$$Q \rightarrow bPP$$
$$X \rightarrow aaa$$

Check whether it generates string aaabbaaabaaaba.

The leaves generated string aaabbaaabaaaba.

3.15 Chomsky Hierarchy

The Chomsky's Hierarchy represents the class of languages that are accepted by different machine. The category of languages in Chomsky's Hierarchy is as given below -

Language class	Language	Grammar	Machine	One example
Type 3	Regular	Regular grammar	FSM i.e. NFA or DFA	a^*b^*
Type 2	Context free	Context free grammar	PDA	$a^n b^n$
Type 1	Decidable languages	Context sensitive grammar	Linear bounded automata	$a^n b^n c^n$
Type 0	Computable languages	Unrestricted grammar	Turing machine	n!

This is a hierarchy therefore every language of type 3 is also of type 2, 1 and 0. Similarly every language of type 2 is also of type 1 and 0 etc.

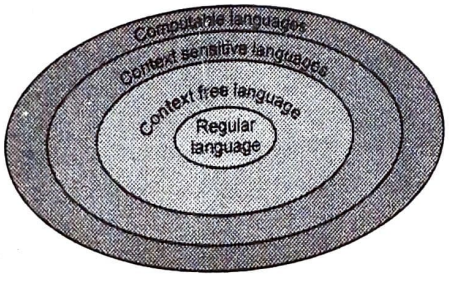


Fig. 3.15.1 Chomsky hierarchy

Type 3 - Regular languages

Regular languages are those languages which can be described using regular expressions. These languages can be modelled by NFA or DFA.

Type 2 - Context free languages

The context free languages are the languages which can be represented by Context Free Grammar (CFG). The production rule is of the form

$$A \rightarrow \alpha$$

where A is any single non-terminal and α is any combination of terminals and non-terminals.

A NFA or DFA cannot recognize strings of this language because these automata are not having "stack" to memorize. Instead of it the Push Down Automata can be used to represent these languages.

Type 1 - Context sensitive languages

The context sensitive grammars are used to represent context sensitive languages. The context sensitive grammar is follows the following rules -

1. The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
2. The number of symbols on the left hand side must not exceed the number of symbols on the right hand side.
3. The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The automaton which recognizes context sensitive languages is called linear bounded automaton. While deriving using context sensitive grammar the sentential form must always increase in length every time a production rule is applied. Thus the size of a sentential form is bounded by a length of the sentence we are deriving.

Type 0 - Unrestricted languages

There is no restriction on the grammar rules of these type of languages. These languages can be effectively modeled by turing machines.

Review Questions

1. *Define Type 0 and Type 1 grammer.* SPPU : May-06, Marks 4
2. *Write short note on Chomsky hierarchy.* SPPU : Dec.-11, Marks 8

3.16 Application of CFG

When any high level program like C or Pascal is compiled, the compiler checks the syntax of every programming statement by constructing syntax tree. And for building the syntax tree, it is necessary to write context free grammar for each statement in the program.

For example, if in your C program the statement is

$$x = y + z;$$

Then the x, y and z are identified as 'id' by lexical analyzer. It will be interpreted as

$$'id' = 'id' + 'id' ;$$

The CFG for this will be

$$S \rightarrow id = ET$$
$$E \rightarrow E + F$$
$$E \rightarrow F$$
$$F \rightarrow id$$
$$T \rightarrow ;$$

Let us build the parse tree accordingly.

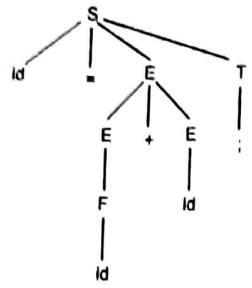


Fig. 3.16.1 Parse tree

Read out the leaf nodes and you will get id = id + id ; If ; is missing in your statement rule won't get matched with T branch and hence it will generate syntax error as 'statement ; missing'.

Thus context free grammar is a powerful tool used in compilers for syntax analysis.

1. Parsing techniques

Context free grammar is used for parsing the programming constructs and finding the syntactical errors from source program. There are two types of parser that are used by compiler top down parser and bottom up parser.

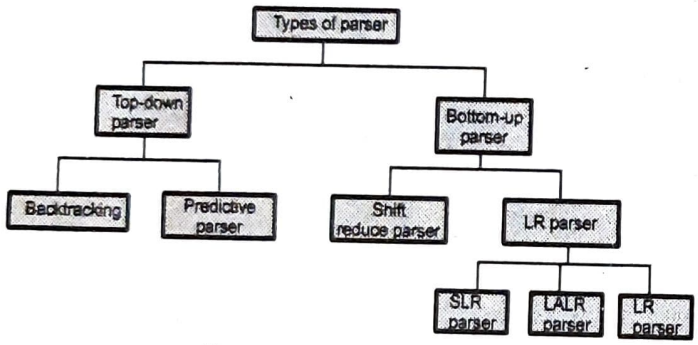


Fig. 3.16.2 Parsing techniques

When parse tree is constructed from root and expanded to leaves then such type of parser is called **Top-down parser.**

When parse tree is constructed from leaves to root, then such type of parser is called **Bottom-up parser.**

2. Markup languages

Markup languages is a family of languages in which the certain strings have special meaning. These strings are referred as **tags.** Thus tags tell us about the semantics of various strings within the document. The most commonly used example of markup language is **Hyper Text Markup Language** (HTML). The context free language is used to describe the structure of HTML document. Consider following HTML document -

* Wonderful colors *

<h1>Red</h1>

<h2>Blue</h2>

<h3>Green</h3>

The above scripting document will help in displaying the text on the web page with special formatting. In order to check this HTML document syntactically, a context free grammar is used. Following are some variables/non terminals are used to parse this HTML document -

1. Text

It represents the collection of strings.

2. Char

It represents a single character which is allowed in the HTML document. The collection of various characters lead to a text.

3. Doc

It represents the sequence of elements.

4. Element

The element can be simple text or a text enclosed within the tags.

The production rules defining above HTML document will be -

Char→ a | A |...

Text→ Char Text | ε

Doc→ Element Doc | ε

Element→Text |

<h1>Doc</h1> |

<h2>Doc</h2> |

<h3>Doc</h3>|...

Thus CFG is used to describe the structure of HTML document.

3. XML and Document Type Definitions

The HTML is a markup language which makes use of some standard tags. There are some special meanings associated with these tags. But Extensible Markup Language (XML) is a special kind of language using which user can define his own tags. The purpose of XML is not to describe the formatting of HTML but to describe the semantics of the text. For example, consider following XML document

```
<Person>
        <Personal-Info>
            <Name>My Name is Jaya</Name>
            <City>I live in Pune</City>
        </Personal-Info>
        <Hobby>
            <first>I like reading</first>
            <second>I like programming</second>
            <third>I like singing</third>
        </Hobby>
</Person>
```

Note that the XML document allows user to use user-defined tags which are useful for describing the semantics of the text. For instance : if we want to know the city in which the person lives then it becomes convenient to find out the corresponding information. The **Document Type Definition (DTD)** is a kind of context free grammar used to define the structure of the XML document. The DTD has its own notations and variables.

Using DTD we can specify the various elements types, attributes and their relationship with one another. Following example shows how DTD is useful for describing the structure of XML document -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
<!ELEMENT student (name,address,std,marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT std (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
]>
<student>
        <name>Anand</name>
        <address>Pune</address>
        <std>Second</std>
        <marks>70 percent</marks>
</student>
```

Here we are defining the DTD

In DTD, The basic entity is **ELEMENT**. The elements are used for defining the tags. The elements typically consist of opening and closing tag. Mostly only one element is used to define a single tag. The PCDATA is used in above given XML document. It stands for Parsed Character **Data** (i.e. text). Any Parsable character data should not contain the markup characters. The markup characters are < or > or &. If we want to use less than, greater than or ampersand characters then make use of < , > or & The notations of context free grammar can also be used in DTD. For instance :

hobby → programming | reading | singing

can be denoted as :

<!ELEMENT hobby(programming | reading |singing)

In order to denote more than one occurrences the + sign can also be used in DTD.

Review Questions

1. *Discuss the applications of CFG in XML*
   SPPU : May-06, Dec.-06, Marks 4, May-11, Marks 6
2. *Explain the application of context free grammar to syntax analysis phase of compiler with suitable example* SPPU : Dec.-13, Marks 6
3. *Explain with examples any two applications of contest free grammar.*
   SPPU : Oct.-16, In Sem, Marks 4

3.17 Case Studies

CFG for Palindrome

- The palindrome is a condition in which we get the same string when read it from left to right or from right to left.
- Let G be CFG such that

  G = (V, T, P, S) where

  V = {S}

  T = {a, b}

S is a start symbol.

P - The production rules are

S → aSb | bSa

S → a | b | ε

- The simulation of above CFG can be shown for string abba

  S

  a S a

  a b S b a

  a b ε b a

  a b b a

CFG for Parenthesis Match

- Parenthesis match is a match made between opening and closing parenthesis.
- Let G be CFG such that

  G = (V, T, P, S) where

  V = {S}

  T = {(,)}

S is a start symbol.

P is a set of production rules

The rules are

S → SS | (S)| ( )

- The simulation above CFG can be shown for the string (( )) ( )

  S

  SS

  S( )

  (S)( )

  (( ))( )

❑❑❑