UNIT-V

AMBIGUITY IN CONTEXT FREE GRAMMARS:

Sometimes a grammar can generate the same string in several different ways. Such a string will have several different parse trees and thus several different meanings. This result may be undesirable for certain applications, such as programming languages, where a given program should have a unique interpretation.

If a grammar generates the same string in several different ways, we say that the string is derived ambiguously in that grammar. If a grammar generates some string ambiguously we say that the grammar is ambiguous.

Definition:

- A terminal string w Є L(G) is ambiguous if there exists two or more derivation trees for w (or there exist two or more leftmost derivation of w).
- Consider for example G = ({S}, {a, b, +, *}, P, S), where P consists of S → S + S | S * S|a|b. We have two derivation trees for a + a * b given in **Figure 5.1**:
- The leftmost derivations of **a + a * b** induced by the two derivation trees are

S → S + S	S → S * S
S → a + S	S → S + S * S
S → a + S * S	S → a + S * S
S → a + a * S	S → a + a * S
S → a + a * b	S → a + a * b

- ✓ Therefore a + a * b is ambiguous

Definition:

- A Context Free Grammar G is ambiguous if there exists some w Є L(G), which is ambiguous.

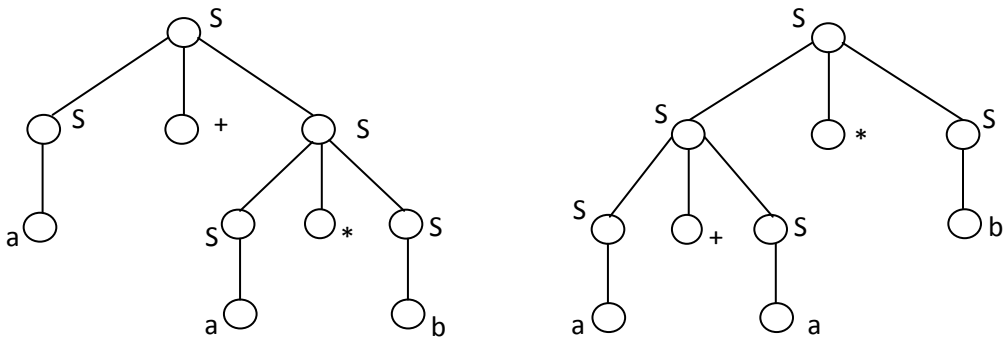


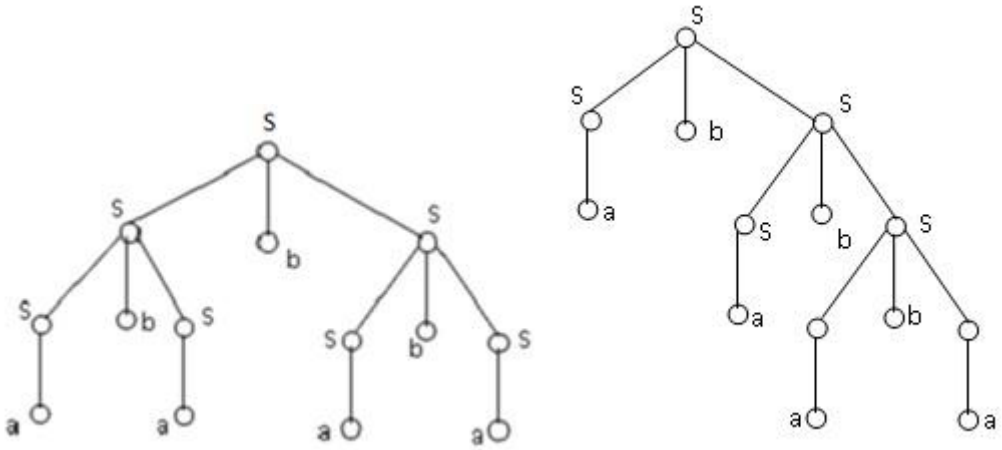
Figure 5.1: Derivation trees for a + a * b

Example:

If G is the grammar S → SbS | a, show that G is ambiguous

Solution:

- To prove that G is ambiguous, we have to find a w Є L(G), which is ambiguous.
- Consider w = **abababa** Є L(G). Then we get two different derivation trees for w as shown in figure given below: Thus G is ambiguous.



MINIMIZATION OF CONTEXT FREE GRAMMARS

SIMPLIFICATION OF CONTEXT FREE GRAMMARS

- In a context free grammar it may not be necessary to use all the symbols in V U Σ or all the productions in P for deriving sentences. So when we study the context free language L(G).

LANGUAGE OF A GRAMMAR:

- For the given context free grammar G = (V, Σ, P, S), the language of G denoted L(G) is the set of terminal strings that have derivations from the start symbol. That is
  - $L(G) = \{w \text{ in } \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$
- If a language L is the language of some context free grammar, then L is said to be a context free language or CFL.
- We try to eliminate symbols and productions in G which are not useful for derivation of sentences.
- For example consider the grammar G = ({S, A, B, C, E}, {a, b, c}, P, S) where
  P = {S → AB, A→ a, B →b, B → C, E → c | ε}
- It is easy to see that L(G) = {ab}. Let G' = ({S, A, B}, {a, b}, P', S), where P' consists of
  S → AB, A → a, B → b. i.e. L(G) = L(G').
- We have eliminated the symbols C, E and c and the productions B → C, E → c | ε.
- We note the following points regarding the symbols and productions which are eliminated:
  1. C does not derive any terminal string.
  2. E and c do not appear in any sentential form.

3. E → ε is a null production
4. B → C simply replaces B by C.

- Now we give the construction to eliminate
  - ✓ *Variables not deriving terminal strings.*
  - ✓ *Symbols not appearing in any sentential form.*
  - ✓ *ε productions and*
  - ✓ *Productions of the form A → B.*
- To get there, we need to make a number of preliminary simplifications, which are themselves useful in various ways:
  1. We must eliminate useless symbols, those variables or terminals that do not appear in any derivation of a terminal string from the start symbol.
  2. We must eliminate ε – productions, those of the form A → ε for some variable A.
  3. We must eliminate unit productions, those of the form A → B for variables A and B.

ELIMINATING USELESS SYMBOLS

- Any symbol is useful when it appears on right hand side, in the production rule and generates some terminal string.
- If no such derivations exist then it is supposed to be the useless symbol.
- A symbol p is useful if there exists some derivation in the following form:

$$S \overset{*}{\Rightarrow} \alpha p \beta$$

And

$$\alpha p \beta \overset{*}{\Rightarrow} w$$

- 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 an example given below:
  - **For example** consider the grammar G = (V, Σ, P, S)
    Where V = {S, T, X}, Σ ={0, 1}
    Productions are S → 0T | 1T | X | 0 | 1 and T → 00
    Start symbol is S.
  - **Sol:** To derive some string we should have to start with the start symbol (i.e. S)

    S → 0T

    S → 000

    Thus we reach certain string after following these rules.
  - But for the sentential form S → X there is no further rule as a definition to X.
  - Hence we declare X as a useless symbol. And we can remove the productions that contain X.
  - Now after the removal of useless production, the CFG becomes:

- G = (V, Σ, P, S)
  Where V = {S, T}, Σ ={0, 1}
  Productions are S → 0T | 1T | 0 | 1 and T → 00
- **Example (a):** Eliminate the useless symbols from the following grammar

S → aA | a | Bb |cC
A → aB
B → a |Aa
C → cCD
D → ddd

- **Solution:**
- Step -1: consider all the productions that are giving terminal symbols.

S → a
B → a
D → ddd

- Now consider the following productions:

S → cC
C → cCD
D → ddd

- When we try to derive the string using production rule for C we get

S → cC
S → ccCD
S → ccCddd
S → cccCDddd and so on…

- We will not get any terminal for C. Thus we get a useless symbol C. To reach to D the only rule available is by using C. But C gets eliminated, there is no point in keeping D. Hence D will also be removed.
- Therefore the following productions form the reduced grammar:

S → aA | a | Bb
A → aB
B → a |Aa

ELIMINATING ε PRODUCTIONS

- The productions of context-free grammars can be coerced into a variety of forms without affecting the expressive power of the grammars. If the empty string does not belong to a language, then there is a way to eliminate the productions of the form A → ε from the grammar.

- If the empty string belongs to a language, then we can eliminate ε from all productions save for the single production S → ε. In this case we can also eliminate any occurrences of S from the right-hand side of productions.
- Any production of a CFG of the form A → ε is called ε-production. Any variable A for which the derivation $A \overset{*}{\Rightarrow} \varepsilon$ is possible is called Nullable production.

Example:

- Given a Context Free Grammar with the following productions:

$$S \rightarrow aAb$$
$$A \rightarrow aAb \mid \varepsilon$$

Now obtain the new set of productions for a grammar same as the given CFG.

Solution

- The given grammar is

$$S \rightarrow aAb$$
$$A \rightarrow aAb \mid \varepsilon$$

- The ε-production in the given grammar is A → ε
- It is now removed after adding the new productions by substituting ε for A wherever it occurs in the right hand side. Hence we get the following

$$S \rightarrow aAb$$

  - After substituting the ε for A we get

$$S \rightarrow a\varepsilon b$$
$$S \rightarrow ab$$

  - Similarly we get

$$A \rightarrow a\varepsilon b$$
$$A \rightarrow ab$$

  - Now the following productions without ε are obtained

$$S \rightarrow aAb \mid ab$$
$$A \rightarrow aAb \mid ab$$

ELIMINATING UNIT PRODUCTIONS

- Any production of a CFG of the form A → B where {A, B} Є V is called a Unit production. These productions can be useful.
- However unit productions can complicate certain proof, and they also introduce extra steps into derivations that technically need not be there.
- Having the variable one on either side of a production is sometimes undesirable. Now we use the substitution rule for removing the unit productions.

- Given the context free grammar G = (V, Σ, P, S) with no ε productions, there exists a context free grammar G' = (V', Σ, P', S) that does not have any unit productions and that is equivalent to G.
- Let us illustrate the procedure to remove unit-production through an example.

Example:

- Eliminate unit productions from the grammar G given by productions as below:

$$S \rightarrow AB$$
$$A \rightarrow a$$
$$B \rightarrow C \mid b$$
$$C \rightarrow D$$
$$D \rightarrow E$$
$$E \rightarrow a$$

Solution:

- ✓ $A \rightarrow a$, $B \rightarrow b$ and $E \rightarrow a$ are the non unit productions.
- ✓ Therefore P' will contains the following productions:
- ✓ Since $B \overset{*}{\Rightarrow} E$ and $E \rightarrow a$ is a non unit production, $B \rightarrow a$ is in P'.
- ✓ Since $C \overset{*}{\Rightarrow} E$, $D \overset{*}{\Rightarrow} E$, and $E \rightarrow a$ is a non unit production, $C \rightarrow a$ and $D \rightarrow a$ is in P'.
- ✓ Hence we have the equivalent grammar without unit productions as G' defined by

$$G' = (\{S, A, B, C, D, E\}, \{a, b\}, P', S)$$

- ✓ With P' given by

$$S \rightarrow AB$$
$$A \rightarrow a$$
$$B \rightarrow b$$
$$B \rightarrow a$$
$$C \rightarrow a$$
$$D \rightarrow a$$

CHOMSKY NORMAL FORM

- The goal of this section is to show that every context free language (without ε) is generated by a context free grammar in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B and C are variables and *a* is a terminal. This form is called as Chomsky Normal Form
- In the Chomsky Normal Form, we have restrictions on the length of the Right hand side of the production and also the nature of the symbols in the right hand side of the productions.
- We now complete our study of grammatical simplifications (refer minimization of context free grammar) by showing that every nonempty context free language without ε has a grammar G in which all productions are in one of the two simple forms, either:
  1. $A \rightarrow BC$, where A, B and C are each variables, or

2. A → a, where A is a variable and a is a terminal.

- Further G has no useless symbols. Such a grammar is said to be in Chomsky Normal Form, or CNF.
- To put a grammar in CNF, start with one that satisfies the restrictions i.e. , the grammar has no ε – productions , unit productions or useless symbols.
- Every productions of such a grammar is either of the form A → a, which is already in a form allowed by CNF or it has a body of length 2 or more.
- Our tasks are to:
  1. **Arrange that all bodies of length 2 or more consist only of variables**
  2. **Break bodies of length 3 or more into a cascade of productions, each with a body consisting of two variables.**
- **The construction of 1. is as follows:**
  1. **For every terminal *a* that appears in the body of length 2 or more, create a new variable, say A. this variable has only one production, A → a.**
  2. **Now we use A in place of *a* everywhere *a* appears in a body of length 2 or more.**
  3. **At this point, every production has the body i.e. either a single terminal or at least two variables and no terminals.**
- **For step 2. We must break those productions $A \rightarrow B_1B_2 \ldots B_k$, for $k \geq 3$, in to a group of productions with two variables in each body. We introduce k-2 new variables, $C_1C_2 \ldots C_{k-2}$. The original production is replaced by the k-1 productions.**

$$A \rightarrow B_1C_1, \quad C_1 \rightarrow B_2C_2, \quad \ldots \quad C_{k-3} \rightarrow B_{k-2}C_{k-2}, \quad C_{k-2} \rightarrow B_{k-1}B_k$$

Example:

- Convert the given grammar G in to the CNF.

E → E + T | T * F | (E) | a | b | Ia | Ib | I0 | I1
T → T * F | (E) | a | b | Ia | Ib | I0 | I1
F → (E) | a | b | Ia | Ib | I0 | I1
I → a | b | Ia | Ib | I0 | I1

Solution:

- For part 1. Notice that there are eight terminals a, b, 0, 1, +, *, (, and), each of which appears in a body that is not a single terminal.
- Thus we must introduce eight new variables, corresponding to these terminals and eight productions in which the new variable is replaced by its terminal.
- Using the obvious initials as the new variables, we introduce:

A → a B → b Z → 0 O → 1
P → + M → * L → (R →)

- If we introduce these productions, and replace every terminal in a body that is other than a single terminal by the corresponding variable, we get the grammar as shown as follows:

E → EPT | TMF | LER | a | b| IA | IB | IZ | IO
T → TMF | LER | a | b| IA | IB | IZ | IO
F → LER | a | b| IA | IB | IZ | IO
I → a | b| IA | IB | IZ | IO
A → a
B → b
Z → 0
O → 1
P → +
M → *
L → (
R →)

- Now all productions are in Chomsky Normal Form except for those with the bodies of length 3: EPT, TMF, and LER.
- Some of these bodies appear in more than one production, but we can deal with each body once, introducing one extra variable for each.
- For EPT, we introduce new variable C_1 and replace the one production, E → EPT, where it appears, E → EC_1 and C_1 → PT.
- For TMF, we introduce new variable C_2. The two productions that use this body, E → TMF and T → TMF are replaced by E → TC_2, T → TC_2, and C_2 → MF.
- Then for LER we introduce a new variable C_3 and replace the three productions that use it, E → LER, T → LER and F → LER by E → LC_3, T → LC_3, F → LC_3, and C_3 → ER.
- The final grammar which is in CNF is shown as below:

E → EC_1 | TC_2 | LC_3 | a | b| IA | IB | IZ | IO
T → TC_2 | LC_3 | a | b| IA | IB | IZ | IO
F → LC_3 | a | b| IA | IB | IZ | IO
I → a | b| IA | IB | IZ | IO
A → a
B → b
Z → 0
O → 1
P → +
M → *
L → (
R →)
C_1 → PT
C_2 → MF
C_3 → ER

GREIBACH NORMAL FORM

- There is another interesting normal form for grammars that we shall not prove.
- Every nonempty language without ε is L(G) for some grammar G each of whose productions are of the form **A → aα**, where a is **a** is a terminal and **α** is a string of zero or more variables.
- Converting a grammar to this form is complex, even if we simplify the task by, say, starting with a Chomsky-Normal-Form grammar.
- Roughly we expand the first variable of each production until we get a terminal. However because there can be cycles, where we never reach a terminal, it is necessary to short-circuit the process, creating a production that introduces a terminal as the first symbol of the body and has variables following it to generate all the sequences of variables that might have been generated on the way to generation of that terminal.
- This form is called Greibach Normal Form, after Sheila Greibach, who first gave a way to construct such grammars, has several interesting consequences.

PUMPING LEMMA FOR CONTEXT FREE LANGUAGES

- Now we shall develop a tool for showing that certain languages are not context free.
- The theorem called the pumping lemma for context free languages, says that in any sufficiently long string in a context free language, it is possible to find at most two short nearby substrings that we can pump in tandem.
- That is we may repeat both of the strings i times for any integer i, and resulting string will still be in the language.
- We may contrast this theorem with the analogous pumping lemma for regular languages which says we can always find one small string to pump.
- The difference is seen when we consider a language like $L = \{0^n1^n \mid n \geq 1\}$. We can show it is not regular by fixing n and pumping a substring of 0's thus getting a string with more 0's than 1's.
- The context free language pumping lemma states only that we can find two small strings, so we might be forced to use a string of 0's and a strings of 1's thus generating only strings in L when we pump.
- That outcome is fortunate because L is a context free language and thus we should not be able to use CFL pumping lemma to construct strings not in L.

THE SIZE OF PARSE TREES

- Our first step in deriving a pumping lemma for CFL's is to examine the shape and size of parse trees. One of the uses of CNF is to turn parse trees into binary trees.
- These trees have some convenient properties, one of which we make use of here.

Theorem:

- Suppose we have a parse tree according to a CNF grammar G = (V, Σ, P, S) and suppose that the yield of the tree is a terminal string w. if the length of the longest path is n then $|w| \leq 2^{n-1}$.

Basis:

- Let n=1. Recall that the length of the path in a tree is the number of edges i.e. one less than the number of nodes.
- Thus a tree with a maximum path length of 1 consists of only a root node and a leaf node labeled by a terminal. String w is this terminal, so |w| = 1.
- Since $2^{n-1} = 2^0 = 1$ in this case, we have proved the basis.

Induction:

- Suppose the longest path has length n and n>1. The root of the tree uses a production which must be of the form A → BC, since n > 1 i.e. we could not start the tree using a production with a terminal.
- No path in the sub trees rooted at B and C can have length greater than n-1. Since these paths exclude the edge from the root to its child labeled B or C.
- Thus by inductive hypothesis, these two sub trees each have yields of length at most 2^{n-2}. The yield of the entire tree is the concatenation of these two yields and therefore has length at most $2^{n-2} + 2^{n-2} = 2^{n-1}$. Thus the inductive step is proved.

STATEMENT OF PUMPING LEMMA

- The pumping lemma for CFL's is quite similar to the pumping lemma for regular languages, but we break each string z in the CFL L into five parts, and we pump the second and fourth in tandem.

Theorem: (The pumping lemma for context free languages)

- Let L be a CFL then there exists a constant n such that if z is any string in L such that |z| is at least n, then we can write z = uvwxy, subject to the following conditions:
  1. |vwx| ≤ n. i.e. the middle portion is not too long.
  2. vx ≠ ε. Since v and x are the pieces to be pumped, this condition says that at least one of the strings we pump must not be empty.
  3. For all i ≥ 0, uv^iwx^iy is in L. i.e. the two strings v and x may be pumped any number of times, including 0, and the resulting string will still be a member of L.

ENUMERATION OF PROPERTIES OF CFL

- We shall now consider some of the operations o context free languages that are guaranteed to produce a context free language.
- Many of these closure properties will parallel the theorems we had for regular languages. However there are some differences.

- First we introduce an operation called **substitution**, in which we replace each symbol in the strings of one language by an entire language. This operation, a generalization of homomorphism which is useful in proving some other closure properties of CFL's like the regular expression operations: union, concatenation and closure.
- We show that CFL's are closed under homomorphisms and inverse homomorphisms. Unlike the regular languages, the CFL's are not closed under intersection or difference.
- However the intersection or difference of a CFL and a regular language is always a CFL.

Substitution

- Let Σ be an alphabet and suppose that for every symbol a in Σ, we choose a language L_a. These chosen languages can be over any alphabets, not necessarily Σ and not necessarily the same.
- This choice of languages defines a function **s(a substitution)** on Σ and we shall refer to L_a as **s(*a*)** for each symbol a.
- If $w = a_1a_2 \ldots a_n$, is a string in Σ* then s(w) is the language of all the strings $x_1x_2 \ldots x_n$ such that string x_i is in the language $s(a_i)$ for i =1, 2, . . . n.
- Put another way, s(w) is the concatenation of languages $s(a_1)s(a_2) \ldots s(a_n)$. we can further extend the definition of s to apply to languages: s(L) is the union of s(w) for all strings w in L.

Example:

- Suppose $s(0) = \{a^nb^n \mid n \geq 1\}$ and s(1) = {aa, bb}.That is, s is a substitution on alphabet Σ = {0, 1}.
- Languages s(0) is the set of strings with one or more *a*'s followed by an equal number of b's, while s(1) is the finite language consisting of the two strings aa and bb.
- Let w = 01. Then s(w) is the concatenation of the language s(0)s(1). To be exact, s(w) consists of all strings of the forms a^nb^naa and a^nb^{n+2}, where n ≥ 1.
- Now suppose L = L(0*), that is , the set of all strings of 0's. Then s(L) = (s(0))*. This language is the set of all strings of the form $a^{n1}b^{n1}a^{n2}b^{n2} \ldots a^{nk}b^{nk}$, for some k ≥ 0 and any sequence of choices of positive integers n_1, n_2, . . . n_k. it includes strings like ε, aabbaaabbb, and abaabbabab.

APPLICATIONS OF THE SUBSTITUTION THEOREM

- There are several familiar closure properties which we studied for regular languages that we can show for CFL's. we shall list them all in one theorem given as follows:

Theorem:

- The context free languages are closed under the following operations:
  1. Union
  2. Concatenation
  3. Closure (*), and positive closure (+)

4. Homomorphism

Proof:

- Each requires only that we set up the proper substitution. The proofs below each involve substitution of context free languages in to other context free languages.
  1. **Union:** Let L_1 and L_2 be CFL's. Then $L_1 \cup L_2$ is the language s(L), where L is the language {1, 2} and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$.
  2. **Concatenation:** Let L_1 and L_2 be CFL's. Then L_1L_2 is the language s(L), where L is the language {12} and s is the same substitution as in case (1).
  3. **Closure and positive closure:** if L_1 is a CFL, L is the language $\{1\}^*$, and s is the substitution $s(1) = L_1$ then $L_1^* = s(L)$. Similarly if L is instead the language $\{1\}^+$, then $L_1^+ = s(L)$.
  4. Suppose L is CFL over an alphabet Σ, and h is a homomorphism on Σ. Let s be the substitution that replaces each symbol *a* in Σ by the language consisting of the one string that is h(*a*). That is s(*a*) = {h(*a*)}, for all *a* in Σ. Then h(L) = s(L).

- Tips for constructing NFA and DFA:
  1. First understand the language clearly

     For example:

     L = {w ε {0, 1}* | w consists of a 0 and 1}

     w = {01}

     L = {w ε {0, 1}* | w consists of a 0 and at least one 1}

     w = {01, 011, 0111 . . .}

     L = {w ε {0, 1}* | w consists of exactly one 0 and 1}

     w = {01}

     L = {w ε {0, 1}* | w consists of exactly one 0 or 1}

     w = {0, 1}

1) Design DFA for the language L = {w | w accepts string 1100 only}
   w = {1100}
   Note: Dummy states are non – accepting states whose transition for every input character terminates on themselves.
2) Design DFA for the language L = { w Є {0, 1}* | w contains set of all strings contain 1100 as substring}
   Since 1100 is the substring itself so construct the NFA and DFA for it.
3) Design DFA for the language L ={ x Є {0, 1}* | x has even number of zeros}
   W = {ε, 1, 00, 1111, 0000, 0011111, ... }
4) Design DFA for the language L = { x Є {0, 1}* | x has even no. of zero's and even no. of one's }
   W = {ε, 11, 0011, 1100, . . . }

5) Design DFA for the language L that accepts the strings start with 011 over binary alphabet
   W = {011, 0110, 0111, 011011, 011000, . . . }
   Find out minimum length string in language (e.g. 011)
   Draw min length + 1 number of states
   Mark first state as initial state and last state as final state.
6) Design DFA for the language L which accepts set of strings containing exactly 4 1's in every string over alphabet Σ = {0, 1}.
   W = {1111, 111100, 10101010, 110011, 11001100, . . . }
7) Design DFA for the language L that accepts strings containing exactly 1 over an alphabet {0, 1}.
   W ={1, 01, 010, 0010, 00100, ... }
8) Design NFA for the language L = all strings over {0, 1} that have at least two consecutive 0's or 1's
   W = { 00, 11, 10100001, ...}
9) Design NFA that accepts string 1100 only.
10) Design NFA that accepts all strings that end in 01
   W = {01, 001, 101, 1001, 11101, ...}
11) Design NFA to accept set of strings over an alphabet {0, 1} and ending with two consecutive 0's.
   W ={00, 000, 100, 1100, 0100, ... }
12) Design NFA which accepts set of all strings containing 1100 as substring.
   W = { 1100, 01100, 11100, 011000, ...}
13) Design NFA which accepts set of all strings containing 3rd symbol from right side is 1.
   W ={100, 111, 101, 0101, 0100, 110, ...}
14) Construct the DFA for the following language L = {w | w doesn't contain the substring ab}
   W = {a, b, aa, aaa, bb, ba, ... }
15) Construct DFA for the language L = {w | w doesn't contain the substring baba}
   W= {a, aa, b, ba, bb, bab, babb, ...}
16) L = {w | w doesn't contain substring 110}. Construct the DFA for L.
   W = {0, 1, 11, 111, 10, 01, 100, 100011, ... }
17) L = {w | w is any string accept 11 and 111} construct the FA for the language L.
   W = {0, 1, 00, 01, 011, 1110, 110, ... }
18) Construct DFA that accepts all strings over {a, b} that have length 3
   W = {aaa, aab, aba, abb, baa, bab, bba, bbb}
19) Construct DFA that accepts all strings over {0, 1} that end with 111
   W = {111, 0111, 11111, 1110111, 110111, 10111, . . . }
20) Construct DFA for the language L that accepts all strings over {0, 1} that begin with 111
   W = {111, 1110, 1111, 11100, 11110, ... }
21) Construct DFA that accepts all strings over {0, 1} that begin or end with 111
   W = {111, 0111, 1110, 010111, 10111, ...}
22) Construct DFA that accepts all strings over {0, 1} that begin and end with 111
   W = {111, 1110111, 1111, 1110111111, ... }
23) Design DFA for accepting the language of strings over {0, 1} that contain 10 as sub string. Process the string 0011.
   W = {10, 010, 0110, 0100, ...}

24) L_1 = {x Є {0, 1}* | the third bit from the left end is 1 in x}
25) L_2 = {x Є {0, 1}* | the third bit from the right end is in x} → is difficult to construct
26) Write the Regular expression for the language L ={ab^nw | n ≥ 3, w Є$(a, b)^+$}
    Sol: one a followed by at least 3 b's followed by at least a or b
    So we may get the following
    $ab^3b^*(a+b)^+$
    $ab^3b^*(a+b)^*(a+b)$
27) Write the Regular expression for L ={a^nb^m | n ≥ 4, m ≤ 3}
    $a^4a^*(b^0 + b^1 + b^2 + b^3)$ i.e. start with at least 4 a's and with atmost 3 b's
28) Write the regular expression for L = {w | |w| mod 3 = 0}
    $((a+b)(a+b)(a+b))^* = ((a+b)^3)^*$
29) Write the regular expression for L = {w Є {a, b}* | $n_a(w)$ mod 3 = 0}
    (b*ab*ab*ab*)*
30) Write the regular expression for L = {w | w contain at least two a's or exactly 2 b's}
    B*ab*a(a+b)* + a*ba*ba*
31) Write the regular expression for L = {w | w contains a single 1 }
    0*10*
32) Write the regular expression for L = {w | w contains at least 1 }
    (0+1)*1(0+1)*

Q) Write and explain properties of transition function? April – 2011 (R09)

Sol)

PROPERTIES OF TRANSITION FUNCTIONS

Property 1:

δ(q, ε) = q in a finite automaton. This means the state of the system can be changed only by an input symbol.

Property 2:

For all strings w and input symbols a,

$$\delta (q, \mathbf{aw}) = \delta (\delta (q, a), w)$$

$$\delta (q, \mathbf{wa}) = \delta (\delta (q, w), a)$$

This property gives the state after the automaton consumes or reads the first symbol of a string **aw** and the state after the automaton consumes a prefix of the string **wa**.

Example: **April – 2011 (R09)**

Prove that for any transition function δ and for any two input strings x and y,

$$\delta(q, xy) = \delta(\delta(q, x), y) \quad \text{- (a)}$$

Proof:

By the method of induction on |y|, i.e. length of y

Basis: when |y| = 1, y = a Є Σ.

LHS of **(a)** (i.e. $\delta(q, xy)$) = $\delta(q, xa)$ = $\delta(\delta(q, x), a)$ by property 2 = RHS of **(a)** (i.e. $\delta(\delta(q, x), y)$)

Assume the result for all strings x and strings y with |y| = n.

Let y be a string of length n+1.

Write $y = y_1a$ where $|y_1|=n$

LHS of **(a)** $= \delta(q, xy_1a) = \delta(q, x_1a)$, where $x_1 = xy_1$

$= \delta(\delta(q, x_1), a)$

$= \delta(\delta(q, xy_1), a)$

$= \delta(\delta(\delta(q, x), y_1), a)$ by induction hypothesis

RHS of **(a)** $= \delta(\delta(q, x), y_1a)$

$= \delta(\delta(\delta(q, x), y_1), a)$ by property 2

Hence LHS = RHS, this proves **(a)** for any string y of length n+1. By the principle of induction, **(a)** is true for all strings.

Q. Describe the following sets by regular expressions:

(a) {101} (b) {abba} (c) {01, 10} (d) {ε, ab} (e) {abb, a, b, bba}

(f) {ε, 0, 00, 000, . . .} and (g) {1, 11, 111, . . . } **April – 2011 (R09)**

Sol.

(a) Now, {0}, {1} are represented by **0** and **1**, respectively.
Similarly {101} is obtained by concatenating 1, 0 and 1. So {101} is represented by **101**

(b) **abba** represents {abba}

(c) As {01, 10} is the union of {01} and {10}, {01, 10} is represented by **01 + 10**

(d) The set {ε, ab} is represented by **ε + ab**.

(e) As {ε, 0, 00, 000, . . . } is simply {0}*, it is represented by **0***.

Any element in {1, 11, 111, . . . } can be obtained by concatenating 1 and any element of {1}*. Hence **1(1)*** represents {1, 11, 111, . . . }.