### 



### FORMAL LANGUAGES AND AUTOMATA THEORY

A Supporting Material for Students

Page **0** of **39**

**Keywords:**

**Formal Languages and Automata Theory**

Finite Automata, Formal Languages, Chomsky hierarchy of Grammars, Regular Grammars, Context-Free Grammar, Transducers, Deterministic Finite Automata, Non - Deterministic Finite Automata, NFA- , Minimization/ Optimization of DFA, Regular Expressions, Arden’s Theorem, Simplification of Context - Free Grammar, Normal Forms, Push Down Automata, Turing Machine, Undecidable Problem, P and NP Classes of Languages.

|  |  |
| --- | --- |
| **Basic Terminology:** | |
| Computation | Computation can be defined as finding solution to a problem from given inputs  by means of algorithm. The theory of computation, a subfield of computer science and mathematics, deals with finding solutions to problems from the given inputs. |
| Automata | An automaton is a machine that has an input tape that can be put into any of the several states. Various symbols are written on the tape before execution. The automaton begins reading the symbols on the tape from left to right. Upon reading a symbol from the tape, the machine changes its state and advances the tape. After reading the input completely, the machine (halts). |
| A finite automaton (FA) is a simple idealized machine used to recognize  Finite automata patterns within input taken from some character set (or alphabet) C. The job of an FA is to accept or reject an input depending on whether the  pattern defined by the FA occurs in the input. | |
| A symbol is a single object and is an abstract entity that has no meaning by  Symbol itself. It is often called un-interpreted. It can be character. Normally, characters from a typical keyboard are used as symbols. | |
| Alphabet An alphabet is any finite, non-empty set of symbols/ characters. It is denoted by  . While distinguishing two alphabets, the second one is usually denoted by | |
| String | A string is a finite sequence of symbols chosen from some alphabet. In other words, a string is a finite sequence of symbols over an alphabet. It is usually denoted by W or S. |
| A formal language is an abstraction for general characteristics of programming  Formal language languages, that can be defined as a string set, all of which are chosen from some particular alphabet . | |
| Formal grammar | A formal grammar (or simply grammar) is a precise description of a formal language. In other words, a grammar is a notation for defining a language through a finite number of rules. |
| Generative Set of rules for generation of strings in a language.  grammars | |
| Analytic grammars Set of rules to determine whether a string is a member of the language. | |

## 1

|  |  |
| --- | --- |
| Deterministic finite automaton(DFA) | In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.** |
| Non-deterministic  finite automaton(NFA) | In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non- deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**. |
| Non-deterministic finite automaton  with ∈-transitions | **Nondeterministic finite automaton** with ε-**moves** (NFA-ε) is a further generalization to NFA. This **automaton** replaces the **transition** function with the one that allows the empty string ε as a possible input. The **transitions** without consuming an input symbol are called ε-**transitions**. |
| Moore machine | A Moore machine is a finite state automaton, where the outputs are determined  by the current state alone. |
| Mealy machine | A Mealy Machine is an FSM whose output depends on the present state as well  as the present input. |
| 2 DFA | A two-way DFA (2DFA) consists of a finite state control and a read-only input  tape that allows an input to be read back and forth. |
| Regular languages | Suppose ∑ is an alphabet. Then, the class of regular languages over ∑ is inductively defined in the following manner |
| Regular expressions | A regular expression is a string that describes the whole set of strings according to a certain syntax rule. These expressions are used by text editors and utilities,  to search bodies of text for certain patterns. |
| Regular set | These are the sets of which are accepted by finite automata. Any set represented by a regular expression is called regular set. |
| Arden’s theorem | Let p and Q be two regular expressions over Ʃ, and if P does not contain ϵ, then  R = Q+RP has a unique solution given by R=QP\* |
| Ambiguous  Grammar | An **ambiguous grammar** is a context-free **grammar** for which there exists a string that can have more than one leftmost derivation or parse tree, while an |
| Unambiguous Gram  mar | An **unambiguous grammar** is a context-free **grammar** for which every valid string has a unique |

2

|  |
| --- |
| In [formal language](https://en.wikipedia.org/wiki/Formal_language) theory, a [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar) *G* is said to be in **Chomsky normal form** (first described by [Noam Chomsky](https://en.wikipedia.org/wiki/Noam_Chomsky)) if all of its [production](https://en.wikipedia.org/wiki/Production_(computer_science)) [rules](https://en.wikipedia.org/wiki/Production_(computer_science)) are of the form:  Chomsky normal *A* → *BC*, or  *A* → *a*, or  form *S* → ε,  where *A*, *B*, and *C* are [nonterminal symbols,](https://en.wikipedia.org/wiki/Nonterminal_symbol) *a* is a [terminal symbol](https://en.wikipedia.org/wiki/Terminal_symbol) (a symbol that represents a constant value), *S* is the start symbol, and ε denotes the [empty string.](https://en.wikipedia.org/wiki/Empty_string) Also, neither *B* nor *C* may be the [start symbol](https://en.wikipedia.org/wiki/Start_symbol_(formal_languages)), and the third production rule can only appear if ε is in *L*(*G*), the language produced by the context-free grammar *G*. |
| In [formal language](https://en.wikipedia.org/wiki/Formal_language) theory, a [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar) is in **Greibach normal form** (GNF) if the right-hand sides of all [production](https://en.wikipedia.org/wiki/Production_(computer_science)) rules start with a [terminal](https://en.wikipedia.org/wiki/Terminal_symbol) [symbol,](https://en.wikipedia.org/wiki/Terminal_symbol) optionally followed by some variables. A non-strict form allows one exception to this format restriction for allowing the [empty word](https://en.wikipedia.org/wiki/Empty_word) (epsilon, ε) to be a member of the described language. The normal form was established by [Sheila Greibach](https://en.wikipedia.org/wiki/Sheila_Greibach) and it bears her name.  Greibach normal More precisely, a context-free grammar is in Greibach normal form, if all  form production rules are of the form:  AaA1A2A3……An  or S  Where A is a [nonterminal symbol,](https://en.wikipedia.org/wiki/Nonterminal_symbol) ‘a’ is a terminal symbol, A1A2A3……An is a (possibly empty) sequence of nonterminal symbols not including the start symbol, S is the start symbol, and *ε* is the [empty word](https://en.wikipedia.org/wiki/Empty_word). |
| Pushdown PDA is similar to finite state automata but the difference is it consists of auxiliary stack which provides an unlimited amount of memory.  automata(PDA) |
| Turing machine(TM) Turing machine was proposed by Alen Turing in 1936 which recognizes more languages than pushdown automata. |
| Undecidable Undecidable problem is a decision problem for which it is known to be  problem impossible to construct a single algorithm that always leads to a correct yes-or- no answer. |

**Unit wise concepts:**

**UNIT – I**



# Computation, Different models of Computation.

### Computation:

Computation can be defined as finding solution to a problem from given inputs by means of algorithm.

3

|  |  |
| --- | --- |
| The theory of computation, a subfield of computer science and mathematics, deals with finding solutions to problems from the given inputs.  **Models of computation:**  It means a formal, abstract definition of a computer.  There are two types of computations   1. Serial models 2. Parallel models    1. **Finite state machine (FSM)**   **Definition of Automata**  An automaton (plural: automata) is a self-operating machine.  An automaton is a machine that has an input tape that can be put into any of the several states. Various symbols are written on the tape before execution. The automaton begins reading the symbols on the tape from left to right. Upon reading a symbol from the tape, the machine changes its state and advances the tape. After reading the input completely, the machine (halts).   * 1. **Components of FSM**   **Components of FSM:**  A finite state automaton is usually described as consisting of three components   * + - a control unit     - A read unit and     - Input tape (or input file)   1. **Elements of FSM**   Finite state machine consists of 4 main elements:  **State:**  A state is a complete set of properties, transmitted by an object to an observer via one or more channels. Any change in the nature or quantity of such properties in a state is detected by an observer and thus a transmission of information occurs.  The following are some of the types of states:   1. **Start state**: An initial state or condition of a finite state machine. 2. **Accepting state**: If a finite state machine finishes an input string and is in an accepting state, the string is accepted or considered to be valid. 3. **Next state**: The state immediately following the current state, defined by the transition function of a finite state machine and the input is the next state. 4. **Universal state:** A state is an alternating Turing machine, from which the machine only accepts all the possible moves leading to acceptance, is called universal state. 5. **Existential state**: A state is a nondeterministic Turing machine, from which the machine accepts any move that leads to acceptance, is the existential state. 6. **Dead/Trap state**: A no final state of a finite state machine, whose transition on every input symbol | |
|  |  |
| **FLAT Important Qs** |  |

terminates on itself.

### Transition:

The following are two equivalent definitions for transition of a finite state machine.

* + 1. Transition is the act of passing from one state to the next.
    2. A change from one place or state or subject or stage, to another.

Transitions are represented in the following ways:

* State diagram or Transition diagram
* State transition table
* Transition function

### State diagram:

|  |  |
| --- | --- |
| **Notation** | **Meaning** |
|  | {Start of process} |
|  | Final / Accepting / Absorption state{End of  process state} |
|  | Transition |
|  | Input {Guarded transition} |
|  | Output |

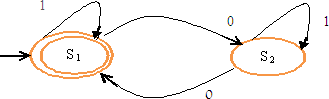
###### Table 1: State diagram notation

A diagram consists of circle to represent state and directed line segment to represent transition between the states, is called a state diagram. The symbols used in state diagrams are enlisted in Table 1

For a finite state machine, a state diagram is a directed graph where,

1. Each edge is a transition between two states
   * For a DFA, NFA and Moore machine, input is labelled on each edge.
   * For a mealy machine, input and outputs are labelled on each edge.
2. Each vertex is a state
   * For a Moore machine, output is signified for each state.

**Example 1:** (state diagram)



###### Fig 1: sample state diagram

S1 and S2 are states and S1 in an accept state. Each edge is labelled with the input.

**State transition table:**

## 5

A state transition table can be described, in general, as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs  States | Present inputs | | | |
| a1 | a2 | …. | an |
| S0 |  |  |  |  |
| S1 |  |  |  |  |
| S2 |  |  |  |  |
| …… |  |  |  |  |
| \* Sn |  |  |  |  |

###### Table: State transition table

* + Tabular representation of Transition that take two arguments and return a value
  + Rows correspond to states and columns correspond to inputs,
  + Entries correspond to next states
  + The start state (S0) is marked with an arrow (→).
  + The accepting states (Sn) are marked with a star, (\*).

For a finite state machine, a state transition table is a table describing the transition function, of a finite automaton this function governs that state to which the automaton will move, given an input to the machine.

# Mathematical representation of Finite State Machine

FSM can be described by a 5 tuple M = (Q, q0,F ) where −

* + - **Q** is a finite set of states.
    - ∑ is a finite set of symbols called the input alphabet.
    - **δ** is the input transition function where δ: Q × ∑ → Q .
    - **q0** is the initial state from where any input is processed (q0 ∈ Q).
    - **F** is the finite set of final states (F⊂Q) .

# Advantages of FSM

* Their simplicity makes it easy for inexperienced developers to implement them with no extra knowledge (low entry level)
* Predictability (in deterministic FSM): given a set of inputs and a known current state, the state transition can be predicated, allowing easy testing.
* Due to their simplicity, FSM are quick to design, implement and execute.
* FSMs are relatively flexible. There are a number of ways to implement a FSM-based system in terms of topology and also, it is easy to incorporate many other techniques.
* It enables easy transfer from a meaningful abstract representation to a coded implementation.
* Low processor overhead makes it well suited to domains, where execution time is shared between modules or subsystems. Only the code for current state needs to be executed. Perhaps, a small amount of logic to determine the current state is needed.
* FSM is a old knowledge representation and system modelling technique. It has been around for a long time. It is well known, even as an artificial intelligence technique, with lots of examples to learn from.

# Disadvantages of FSM

* The predictable nature of deterministic finite state machine can be unwanted in some domains such as Computer games.

## 6



|  |  |
| --- | --- |
| * Large systems, implemented using a FSM, can be difficult to manage and maintain without a well thought out design. The state transition can cause a fair degree of ‘spaghetti-factor’, when trying to follow line of execution. * Unsuitable for all problem domains, FSM should only be used when a system’s behavior can be decomposed into separate states with well-defined conditions for state transitions. This means that all states ,transitions and conditions need to be known and well-defined * The conditions for state transitions are rigid. In other word, they are fixed (this can be overcome by using a fuzzy state machine (FUSM)).   1. **Applications of FSM**      + FSM are extensively used in the video game industry .the latest games ,like warcraft 111,take advantage of complex FSM systems to control the AI .chat dialogues, where the user is prompted with choices, can also be run using FSMs.      + Besides controlling bots, dialogue and environmental conditions within video games, FSMs also have a large role outside the video game industry. For example, cars ,airplanes ,and robotics(machinery) employ complex FSMs. even websites have a FSM .websites that offer menus for you to traverse other detailed sections of the website ,act much like a FSM with transitions between states.      + The applications of FSMs are also found in language processing: parsing, morphological representations, spelling, information retrieval, tagging, stemming, image compressions, cryptography and designing of compilers.      + The cellular automata are used for making pretty pictures and animations      + A geographic automaton is used to simulate the behaviour and distribution of objects in space such as householders, pedestrians, vehicles, shops, roads, land parcels, sidewalks etc., with respect to their properties and their locations.      + FSMs applied to biological and biomedical problem solving, is the most significant improvement in the recent years. Fourier-transform nuclear magnetic resonance (NMR), NMR imaging (or tomography), x-ray tomography, x-ray diffraction, high performance liquid chromatography, differential scanning calorimetry and mass spectrometry are techniques, which employ FSMs.   **UNIT – II**   * 1. **Symbols, Alphabets , String and Languages**      1. **SYMBOLS:**   A symbol is a single object and is an abstract entity that has no meaning by itself. It is often called un-interpreted. It can be character. Normally, characters from a typical keyboard are used as symbols.   * + 1. **ALPHABETS:**   An alphabet is any finite, non-empty set of symbols/ characters. It is denoted by. While distinguishing two alphabets, the second one is usually denoted by .  Example:  a. = {0, 1} is the binary alphabet, consisting of the symbols 0 and 1.  b. = {a, b, c} is an alphabet of three symbols.  c. = {A, B,………, Z} is the uppercase English alphabet.   * + 1. **STRING:** | |
|  |  |
| **FLAT Important Qs** |  |

A string is a finite sequence of symbols chosen from some alphabet. In other words, a string is a finite sequence of symbols over an alphabet. It is usually denoted by W or S .Quite often, the letters u,v,w,x,y,z are used to denote strings while the letters i,j,k,l,m,n are used to denote the natural numbers.

1. The length of a string is defined as the number of symbols in the string and is denoted by **|**x**|** for the string x.
2. A null string is a string with no symbols. In other words, it’s a string of length zero and is denoted by or or .

Example:

* 1. If = {0, 1} then 01010, 1111, 11, 11, 10, 01… are some of the strings chosen from this alphabet.
  2. If = {a, b} then ab, abab, aabb, are the words chosen from this alphabet.

### Language:

A formal language is a set of finite length words drawn from some finite alphabet.

Examples:

* + - 1. Let ={Z} then the language of all possible strings is given by L1= {z, zz, zzz, zzzz,…}. Here L1 does not contain null string.
      2. Language containing the null string is { }
      3. Null language: { } =.

Language notation:

1. L(M) is a language defined by a machine M, that accepts a certain set of strings.
2. L (G) is a language defined by grammar G, that accepts a certain string set.
3. L(r) is a language defined by a regular expression.

# 2.2 Operations on Strings:

Basically, there are a number of operations on strings. Some of them are

1. Concatenation of Strings

Mathematically, If s1 and s2 are two strings then the concatenation‘s’ of s1 and s2 is given by

S’=S1◦S2

={xy |x ϵ S1,y ϵ S2}

**EXAMPLE: (Concatenation of strings)**

* 1. If S1=para and S2=graph, then S1◦S2=paragraph
  2. If ∑={a,b}, S1=ab,S2=baa, then S=S1◦S2=abbaa

1. Kleene Closure

Mathematically,

* If S is a set of strings, then S\* is defined as the smallest super set of S that contain ϵ and is closed under the string concatenation operation.
* If S is a set of symbols or characters, then S is the set of all strings over the symbols in S, including the empty string.

8



S+ =

i

where i ϵ N i≥0

i={ϵ} U S U S2 U S3 U ......

Thus,

where si refers to the ith string of S.

###### Example: (Kleene Closure)

* Kleene closure applied to the set of strings the: If S={cc, d} then

S={ϵ or any word composed of factors of cc and d}

={ϵ or all strings of c’s and d’s in which c occurs in pairs}

= U Si , I ϵ N, N=0, 1, ..

Where

S0=ϵ

S1={cc, d}

S2=SoS = {cc, d}o{d, cc}={cccc, ccd, dcc,dd}

S3= S2oS = {cccd, ccd, dcc, dd}o{cc, d}={cccccc, ccccd,ccdcc,ccdd, dcccc, dccd, ddcc, ddd} Thus

S\* ={ ϵ, d, cc, dd, ccd, ddd, cccc, ccdd, }

1. Positive Closure

Mathematically, if S is a set of strings/words, then S+ is called the positive closure of S with



S+ = i , i ϵ 1,. N.

###### Example: (Positive Closure)

If S ={a, ba}, then S+ ={a, ba}𝗁 , i.e.,

S+ =U Si , i=1,. ,n

Where

S1={a, ba}

S2=SoS={a, ba}o{ a, ba } = {aa, aba, baa, baba}

S3=S2oS={ aa, aba, baa, baba }o{a, ba}={aaa, aaba, abaa, ababa, baaa, baaba, babaa, babababa} Thus

S𝗁 ={a, aa, ba, aaa, aba, baa, }

1. String Reversal

Mathematically, when a string x is reversed i.e. spelt backwards, then xR is the resulting string which satisfies:

* XR(i)= x(n+1-i), for 1 ≤ i ≤ n
* |xR|=|x|

###### Example :( String Reversal)

* 1. If x=mus, then xR=sum.
  2. Let x=bottle. Then,xR=elttob

1. Parts of Strings

It is important to talk about the various parts of string, for example, a leading part, a middle segment and a trailing part. These concepts are used at several occasions and are referred to as prefix, substring and

## 9





|  |  |
| --- | --- |
| suffix respectively.  **2.3 Operations on languages**  There are totally 10 operations on languages. They are:   1. Union 2. Intersection 3. Complementation 4. Symmetric difference 5. Concatenation of language 6. Reversal of languages 7. Palindrome languages 8. Length subsetting of a language 9. Kleene star/closure of languages 10. De Morgan’s Laws 11. ***Union :***   This is one of the simplest operation on two languages. As discussed earlier languages are set of strings, and hence the union of two languages L1 and L2 is the set L1UL2. Mathematically, this is represented as  X L1UL2, if and only if x L1 or x L2  **Example**  Let L1={0,01,011} and L2={ ,001} then L1 U L2 = { ,0,01,011}   1. ***Intersection :***   Suppose L1 and L2 are two languages over an alphabet .Then intersection of L1 and L2 is denoted by L1 L2, if and only if ,x L1 and x L2.  **Example**  Let L1={ } and L2= .Then L1 L2=   1. ***Complementation :***   It is an operation performed on a single language. Suppose L is a language over an alphabet , then the compliment of L, denoted by L’, is the language consisiting of all those strings that are not in L over the alphabet. Mathematically, this is expressed as  x L’ if and only if x \* - L  **Example:**   * 1. If ={a,b} and L={a,b,aa} then the complement of L i.e., L’= \*-L =   { ,a,b,aa,bb,ab,ba,aaa,bbb,……}-{a,b,aa}  = { , bb, ab, ba…}   1. ***Symmetric difference :***   The operation, symmetric difference is no doubt an important but less familiaroperation.Suppose,L1 and L2 are two languages defined over an alphabet .Then the symmetric difference of L1 and L2 is denoted by L1 L2,and is defined as L2 L2=(L1 | |
|  |  |
| **FLAT Important Qs** |  |



|  |  |
| --- | --- |
| Accordingly, elements of L1 L2 are contained either in L1 or L2 but not in both.  **Example:**  Let L be a language over an alphabet .Then clearly L=L, LL= , L\* = L’ and LL’=\*  **5. *Concatenation of languages :***  Concatenation of two languages L1 and L2 is the language L1 L2, each element of which is a string formed by gluing one string of L1 with another string of L2.Mathematically, this is expressed as:  L L2=L1L2= { y|x L1 and y L2}  **Example:**   1. Let L1={bc,bcc,cc} and L2={cc,ccc} then, L1 L2= {bccc, bcccc, bcccccc, cccc, ccccc} 2. ***Reversal of languages :***   The reversal of a language is an important operation that can be used in illustrating the working procedure of various machines. This operation is similar to the reversal of a string operation. Mathematically, this can be written as LR= {WR|W L}  **Example:**   * 1. If L={ab,bc,cd} then, LR= {ba, cb, dc}  1. ***Palindrome languages :***   A language called Palindrome over  = {a, b} is defined as: Palindrome= { , all strings W, such that WR=W and W }.  Equivalently,  Palindrome={ ,a,b,aa,bb,aaa,bbb,aba,abba,aaaa,…}.   1. ***Length subsetting of a language :***   Suppose L is a language over a fixed alphabet  . Then for sake of convenience or compactness, it may be required to specify the strings in L, of length less than or equal to a specific value or a fixed size.  **Example:**  a. Let L={1,11,111,1111,…} then  L=4= {1111} and L<=4 = {1, 11, 111, 1111}.   1. ***Kleene star/Closure of languages :***   The Kleene star of language L is the language L\*, consisting of all the strings produced by concatenating any finite number of strings from L, together. In other words, kleene star denotes the set of all words/strings of any length of any length from given language. Mathematically, this is expressed as  L\*= {X| 0X10X2…0XR , with R 0 and XR L}  **Example:**   * 1. If L={ ,z,zz,zzz,…} over = {z} then   L\*= { , z, zz, zzz,…}  Where L\*=L0 U L1 U L2 U… b. \*={ }. | |
|  |  |
| **FLAT Important Qs** |  |

10. De Morgan’s Laws :

It allows one to express the intersection of two languages over ,in terms of the two operation viz., union and complementation.These are the fundamental laws in set theory.

Let L1, L2, L3 be any three languages, then L1-(L2 U L3) = (L1-L3)

L1-(L2 L3) = (L1-L2) U (L1-L3) L1 L2 = (L1C U L2C) C.

# Chomsky hierarchy Grammars and their Relationship

Components of a Generative Grammar

The generative grammar *G* consists of the following components:

* + 1. A finite set *V* of non-terminal symbols.
    2. A finite set *T* of terminal symbols that is disjoint from *V*.
    3. A finite set *P* of production rules
    4. A distinguished symbol S ϵ V i.e. the start symbol.

Quad –Tuple Specification of Generative Grammar

The formal definition of generative grammars was first proposed by *Noam Chomsky* in 1950

**Formally, a generative grammar *G* is a quad-tuple**

***G= (V, T, P, S)***

**Where,**

***V* is a finite set of non-terminals**

***T* is a finite set of terminals**

***P* is the finite set of production rules, each of the form**

***(T U V)\*V (T U V)\****  ***(T U V)\****

**i.e., each production rule maps from one string of symbols to another, where, the first string contains at least one non-terminal symbol.**

***S* is the start symbol , *S ϵ V***

The language of a generative grammar *G*, denoted by L(G) ,is defined as all those strings over *T*, that can be generated by starting symbol S and then applying the production rules in *P* until non-terminal symbols are present.

# Types of Generative Grammar

In *1956*, *Noam Chomsky* classified the generative grammars into types known as the **Chomsky hierarchy.** Each type of generative grammar distinguishes the other types in the form of production rules. The following are the types:

The below table summarizes different types of generative grammars, discussed so far.

|  |  |  |  |
| --- | --- | --- | --- |
| **Chomsky’s hierarchy** | **Grammar** | **Restrictions on productions** | **Acceptor** |

## 12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type 0** | Unrestricted | (V +  (VUT)\*  UT)  (VUT) +  (VUT) +  A  (VUT)\*  A is a single non-terminal on left-hand side of productions.  A  t or A  tB or A  ϵ  Where A, B ϵ V and t ϵ T | TM |  |
| **Type 1** | Context-sensitive | LBA |
| **Type 2** | Context-free | PDA |
| **Type 3** | Regular | FA |

**Table: *Generative Grammars***

# Chomsky hierarchy of Formal Languages

The table below summarizes some of the formal languages:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Chomsky Hierarchy** | **Type of language** | **Language defined by** | **Corresponding acceptor** | **Does non- determinism give more**  **Power?** | **Set is closed under** |
| 3 | Regular language | Regular expression | Finite automaton | No | Union, Concatenation, Kleene star, intersection,  complement |
| 2 | Context- free  language | Context-free grammar | Pushdown automaton | Yes | Concatenation, Kleene star, union |
| 1 | Context- sensitive  language | Context- sensitive  grammar | Linear-bounded automaton | Yes | Union, Intersection, Concatenation, Complement. |
| Not defined | Recursive language | No grammar has been characterised by recursive languages so far | TM that never loops | No | Concatenation, Kleene star, Union,  Intersection, Complement. |
| 0 | Recursively enumaerable language | Unrestricted grammar | TM | No | Concatenation, Kleene star, Union, Intersection. |

**Table: Formal Languages**

**UNIT – III**

# Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is

## 13

called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

# Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q0, F) where −

* + - **Q** is a finite set of states.
    - **∑** is a finite set of symbols called the alphabet.
    - **δ** is the transition function where δ: Q × ∑ → Q
    - **q0** is the initial state from where any input is processed (q0 ∈ Q).
    - **F** is a set of final state/states of Q (F ⊆ Q).

###### Example

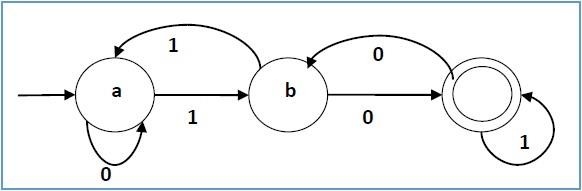
Let a deterministic finite automaton be →

* Q = {a, b, c},
* ∑ = {0, 1},
* q0 = {a},
* F = {c}, and

Transition function δ as shown by the following table −

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next State for Input 0** | **Next State for Input 1** |
| **a** | a | b |
| **b** | c | a |
| **c** | b | c |

Its graphical representation would be as follows −



# Non-deterministic Finite Automaton

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non- deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

# Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple (Q, ∑, δ, q0, F) where −

* + - **Q** is a finite set of states.
    - **∑** is a finite set of symbols called the alphabets.
    - **δ** is the transition function where δ: Q × ∑ → 2Q

(Here the power set of Q (2Q) has been taken because in case of NDFA, from a state, transition can

## 14

occur to any combination of Q states)

* + - **q0** is the initial state from where any input is processed (q0 ∈ Q).
    - **F** is a set of final state/states of Q (F ⊆ Q).

###### Example

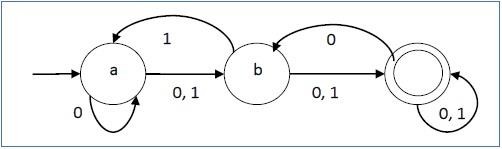
Let a non-deterministic finite automaton be →

* Q = {a, b, c}
* ∑ = {0, 1}
* q0 = {a}
* F = {c}

The transition function δ as shown below −

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next State for Input 0** | **Next State for Input 1** |
| a | a, b | b |
| b | c | a, c |
| c | b, c | c |

Its graphical representation would be as follows −



# Non-deterministic Finite Automaton with ∈ transitions

**Nondeterministic finite automaton** with ε-**moves** (NFA-ε) is a further generalization to NFA. This **automaton** replaces the **transition** function with the one that allows the empty string ε as a possible input. The **transitions** without consuming an input symbol are called ε-**transitions**.

###### Formal definition

An *NFA-ε* is represented formally by a [5-tuple,](https://en.wikipedia.org/wiki/N-tuple) (*Q*, [Σ](https://en.wikipedia.org/wiki/Sigma), [Δ](https://en.wikipedia.org/wiki/Delta_(letter)), *q0*, *F*), consisting of

* + - a finite [set](https://en.wikipedia.org/wiki/Set_(mathematics)) of [states](https://en.wikipedia.org/wiki/State_(computer_science)) *Q*
    - a finite set of [input symbols](https://en.wikipedia.org/wiki/Input_symbol) called the [alphabet](https://en.wikipedia.org/wiki/Alphabet_(computer_science)) Σ
    - a transition [function](https://en.wikipedia.org/wiki/Function_(mathematics)) δ : *Q* × (Σ 𝖴 {ε}) → *2Q*
    - an *initial* (or [*start*](https://en.wikipedia.org/wiki/Finite_state_machine#Start_state)) state *q*0 ∈ *Q*
    - a set of states *F* distinguished as [*accepting* (or *final*) *states*](https://en.wikipedia.org/wiki/Finite_state_machine#Accept_.28or_final.29_states) *F* ⊆ *Q*.

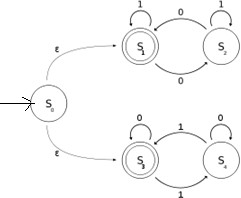
###### Example

The [state diagram](https://en.wikipedia.org/wiki/State_diagram) for *M*

Let *M* be a NFA-ε, with a binary alphabet, that determines if the input contains an even number of 0s or an even number of 1s. Note that 0 occurrences is an even number of occurrences as well.

In formal notation, let *M* = ({*s*0, *s*1, *s*2, *s*3, *s*4}, {0, 1}, δ, *s*0, {*s*1, *s*3}) where the transition relation Δ can be defined by this [state transition table](https://en.wikipedia.org/wiki/State_transition_table):

15



|  |  |  |  |
| --- | --- | --- | --- |
| **Input State** | **0** | **1** | **ε** |
| ***S*0** | ϕ | ϕ | {*S*1, *S*3} |
| ***S*1** | {*S*2} | {*S*1} | ϕ |
| ***S*2** | {*S*1} | {*S*2} | ϕ |
| ***S*3** | {*S*3} | {*S*4} | ϕ |
| ***S*4** | {*S*4} | {*S*3} | ϕ |

# Construct a DFA equivalent to NFA given in figure below. [NFA  DFA]

Let M={Q, q0,F} is a NFA which accepts the language L(M).There should be equivalent DFA by



,

M =(Q , , q0 ,F }.

###### Rule for conversions:



**(q0, q1), 0) = (q0, 0) (q1, 0)**

**Example:**

Let M = ({q0, q1},{0, 1}, ,q0,{q1}) be NFA

Where (q0, 0) = {q0,q1}

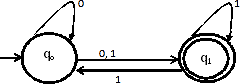
(q0, 1) = {q1} (q1, 0) =

(q1, 1) = {q0, q1}

Construct its equivalent DFA from the above transitions

###### Solution:

State Transition diagram:



Transition Table:

###### Diagram: NFA



|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| q0 | {q0,q1} | {q1} |
| q1 |  | {q0,q1} |

Here in the above table new state Occurred i.e, [q0, q1].We need to compute transitions from this new state [q0, q1].

Compute



:

16

[q0, q1], 0) = (q0, 0) (q1, 0)



= [q0, q1]

= [q0, q1] [q0,q1],1) = (q0,1) (q1,1)

= q1 { q0,q1}

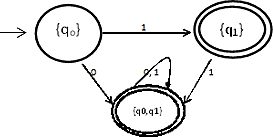
= [q0,q1]

New State Transition Table:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| q0 | [q0,q1] | [q1] |
| q1 |  | [q0,q1] |
| [q0,q1] | [q0,q1] | [q0,q1] |

Note: Here in the above table no new state occurred. So we stopped the procedure here. If any new state occurred we have to find out further transitions

New State diagram:



**Diagram: DFA for the above NFA**

# Construct NFA for the NFA- ε, given in figure. [NFA- ε to NFA]

Let M={Q, , q0,F} is a NFA- ε which accepts the language L(M).There should be equivalent NFA by M =(Q , , q0 ,F }.

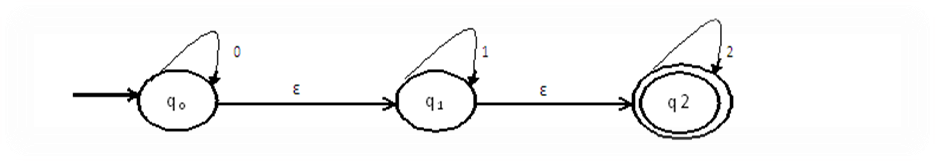
###### Rule for conversions:



**(q, a)= -closure( ( -closure(q),a))**



**Example:**

Construct an equivalalent NFA without ε for the below automaton

###### Dig: NFA with

**Solution:**

17

M=({q0,q1,q2},{0,1,2}, ,q0,{q2})

Transition Table:



###### Table: State Transition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 |  |
| q0 | {q0} |  |  | {q1} |
| q1 |  | {q1} |  | {q2} |
| q2 |  |  | {q2} |  |

Finding -closure for all states



-closure (q0) = {q0, q1, q2}

-closure (q1) = {q1, q2}

-closure (q2) = {q2}

Compute



:

(q0,0)= -closure( ( -closure(q0),0))

= -closure( (q0,q1,q2),o)

= -closure( (q0,0) (q1,0) (q2,0))

= -closure (q0 )

= -closure(q0)

={q0,q1,q2}

(q0,1)= -closure( ( -closure(q0),1))

= -closure( (q0,q1,q2),1)

= -closure( (q0,1) (q1,1) (q2,1))

= -closure()

= -closure(q1)

={q1,q2}

(q0,2)= -closure( ( -closure(q0,2))

= -closure( (q0,q1,q2),2))

= -closure( (q0,2) (q1,2) (q2,2))

= -closure( q2)

= -closure(q2)

={q2}

(q1,0)= -closure( ( -closure(q1),0))

= -closure( (q1,q2),0)

= -closure( (q1,0) (q2,0))

= -closure( )

=

(q1,1)= -closure( ( -closure(q1),1))

[ -closure (q) will never be empty]

**Table**



## 18

-closure( (q1,q2),1)



= -closure( (q1,1) (q2,1))

= -closure(q1, )

= -closure(q1)

={q1,q2}

(q1,2)= -closure( ( -closure(q1),2))

-closure( ( 1,q2),2)

= -closure( (q1,2) (q2,2))

= -closure( q2)

={q2}

(q2,0)= -closure( ( -closure(q2),0))

= -closure( (q2),0)

= -closure( (q2,0))

=

(q2,1)= -closure( ( -closure(q2),1))

= -closure( ( 2),0)

= -closure (q2,0)



(q2, 2) = -closure ( ( -closure (q2), 2))

-closure( (q2),2)

-closure(q2)

={q2}

Summary of



:

(q0,0) ={q0,q1,q2}

(q0,1) ={q1,q2}

(q0,2) ={q2} (q1,0) =

(q1,1) ={q1,q2}

(q1,2) ={q2} (q2,0) =

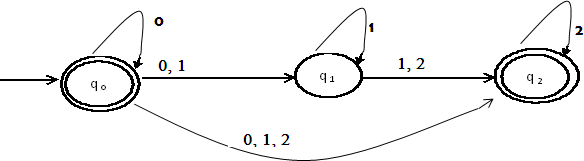
(q2,1) = (q2,2) ={q2}

State Transition Table for transition function is:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| q0 | {q0,q1,q2} | {q1,q2} | {q2} |
| q1 |  | {q1,q2} | {q2} |
| q2 |  |  | {q2} |

## 19

Final State diagram for NFA without :



**Dig: NFA with out**

# Moore machine? Example to convert Moore machine into mealy machine.

**Definition:** A Moore machine is a finite state automaton, where the outputs are determined by the current state alone.

A Moore machine can be described by a 6 tuple Mo = (Q, ∑, δ, q0, Г, λ) where −

* + - **Q** is a finite set of states.
    - **∑** is a finite set of symbols called the input alphabet.
    - Г is a finite set of symbols called the output alphabet.
    - **δ** is the input transition function where δ: Q × ∑ → Q
    - λ is the output transition function where λ: Q → Г
    - **q0** is the initial state from where any input is processed (q0 ∈ Q).

A Moore machine associates an output symbol with each state, and each time a state is entered, an output is obtained simultaneously. So, the first output always occurs as soon as the machine starts.

###### Example to convert Moore machine into mealy machine

**Example:** Convert the following Moore machine into an equivalent Mealy machine. Mo=({q0,q1,q2,q3},{a ,b},{0,1},δ, λ, q0) where, λ and δ are given below.

|  |  |  |  |
| --- | --- | --- | --- |
| ∑  Q | a | b | λ |
| q0 | q1 | q3 | 1 |
| q1 | q3 | q1 | 0 |
| q2 | q0 | q3 | 0 |
| q3 | q3 | q2 | 1 |

###### Fig: Transition and output for Moore Machine

**Solution:**

To construct a Mealy machine from Moore Machine Me= (Q, ∑, δ, Г, λ’, qo), define

**λ’ (q, a) =λ (δ (q, a))** ∀ q and a.

Based on the given data, the transition diagram of Moore machine is shown in above figure.

20

Thus,

* λ’(qo, a)=λ(δ(qo, a))=λ(q1)=0 ⇒a/0
* λ’(qo, b)=λ(δ(qo, b))=λ(q3)=1 ⇒b/1
* λ’(q1, a)=λ(δ(q1, a))=λ(q3)=1 ⇒a/1
* λ’(q1, b)=λ(δ(q1, b))=λ(q1)=0 ⇒b/0
* λ’(q2, a)=λ(δ(q2, a))=λ(qo)=0 ⇒a/0
* λ’(q2, b)=λ(δ(q2, b))=λ(q3)=0 ⇒b/0
* λ’(q3, a)=λ(δ(q3, a))=λ(q3)=0 ⇒a/0
* λ’(q3, b)=λ(δ(q3, b))=λ(q2)=0 ⇒b/0

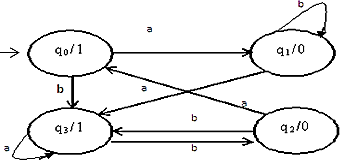
State Transition Table:

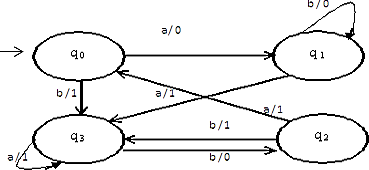
###### Figure: Moore Machine

|  |  |  |
| --- | --- | --- |
| ∑ Q | a | b |
| q0 | [q1, 0] | [q3, 1] |
| q1 | [q3, 1] | [q1, 0] |
| q2 | [q0, 0] | [q3, 0] |
| q3 | [q3, 0] | [q2, 0] |

State diagram:

###### Fig: Transition and output for Mealy Machine





**Figure: Mealy Machine**

# Mealy machine? Example to convert mealy machine into Moore machine.

**Definition:** A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

It can be described by a 6 tuple Me = (Q, ∑, δ, q0, Г, λ) where −

* + - **Q** is a finite set of states.

## 21

|  |  |
| --- | --- |
| * **∑** is a finite set of symbols called the input alphabet. * Г is a finite set of symbols called the output alphabet. * **δ** is the input transition function where δ: Q × ∑ → Q * λ is the output transition function where λ: Q → Г * **q0** is the initial state from where any input is processed (q0 ∈ Q)..   **Example to convert mealy machine into Moore machine**  **Example**: convert the mealy machine in below figure into an equivalent Moore machine.    **Fig: Mealy Machine**  **Solution**:  To construct a Moore Machine from Mealy machine Mo= (Q, ∑, δ, Г, λ’, qo), define  **δ’([q, b],a)=[ δ(q ,a), λ(q, a)] and λ’(q, b)=b.**  Given, Me= ({q0, q1},{a, b},{0,1},δ, qo, λ), Construct an equivalent Moore machine Mo= (Qx, ∑, Г, ∂’, λ’, [qo, b0]),  Where ,Qx={[q0,0],[q0,1],[q1,o],[q1,1]}, ∑={a ,b}.  Thus,   * + δ’([qo,0],a) = [ δ(qo, a),λ(q0,a)]   = [qo, 0]   * + δ’([qo,0],b) = [ δ(q0,b),λ(q0,b)]   = [q1, 1]   * + δ’([qo,1],a) = [ δ(q0,a),λ(q0,a)]   = [q0, 0]   * + δ’([qo,1],b) = [ δ(q0,b),λ(q0,b)]   = [q1, 1]   * + δ’([q1,0],a) = [ δ(q1,a),λ(q1,a)]   = [q0, 1]  Similarly δ’ ([q1, 0], b) = [ q1,0]   * + δ’([q1,1],a) = [ δ(q1,a),λ(q1,a)]   = [q0, 1]   * + δ’([q1,1],b) = [ δ(q1,b),λ(q1,b)]   = [q1, 0]  We also find: λ’ (q, b) =b states i.e.   * + λ’(q0,0)=0   + λ’(q0,1)=1   + λ’(q1,1)=1 | |
|  |  |
| **FLAT Important Qs** |  |

* + - * λ’(q1,0)=0

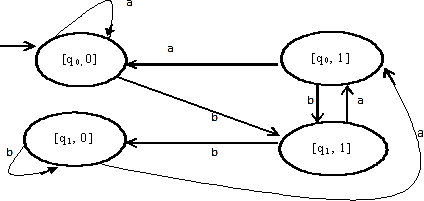
Finally, we draw the transition table and diagram for the Moore machine as follows:

State Transition Table:

|  |  |  |
| --- | --- | --- |
| ∑  Qx | a | b |
| [q0,0] | [q0,0] | [q1,0] |
| [q0,1] | [q0,0] | [q1,1] |
| [q1,0] | [q0,1] | [q1,0] |
| [q1,1] | [q0,1] | [q1,0] |

State diagram:

**Table: Transition Table for Moore machine**



# Minimize the DFA

###### Rules for Minimization:

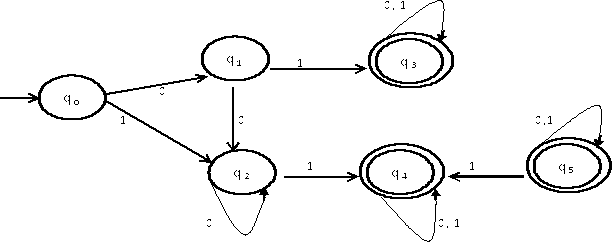
**Figure: Equivalent Mo for the above Me**

**UNIT – IV**

The states that can be **eliminated from automata**, without affecting the language accepted by automata are:

* + 1. Unreachable or inaccessible states
    2. Dead states
    3. Non-distinguishable or indistinguishable state or equivalent states.

**Example:** Minimize the given DFA.



###### 1. Unreachable or inaccessible states

[A state is unreachable state, if it cannot be reached from the start state.]

23



|  |  |
| --- | --- |
| Here q5 is unreachable state. So remove q5. Now the DFA becomes.     1. **Dead states**   [A state is dead, if it is not accepting state and has no out-going transitions, except to itself. Here there is no dead state]  Here there is no dead state.   1. **Non-distinguishable or indistinguishable state or equivalent states.**    1. П=({q0, q1,q2}, {q3, q4}) [i.e. **All Final states into one group and all Non-final states**   **into another Group**]  **Partitions:**  q0 q1 q2 q3 q4  Form Пnew, with subset {q0, q1, q2}.   * **Input: 0 Input:1**   (q0,0)=q1 (q0,1)= q2  δ(q1,0)=q2 δ(q1,1)=q3  δ(q2,0)= q2 δ(q2,1)=q4  Since for input ‘1’, states q1 and q2 give transitions to the final state and q0 is not.  So {q0, q1, q2} is partitioned into {q0}, {q1, q2} With subset {q3, q4},   * **Input: 0 Input:1**   (q3, 0)=q3 (q3,1)= q3  δ(q4, 0)=q4 δ(q4,1)=q4  i.e. {q3, q4} is not partitioned.  So Пnew = ({q0}, {q1, q2}, {q3, q4})  П ≠ Пnew  b. Now assign П new to П П=({q0}, {q1,q2}, {q3, q4}) | |
|  |  |
| **FLAT Important Qs** |  |

###### Partitions:

q0

q1 q2

q3 q4

Form Пnew, with subset {q1, q2}.

###### Input:0 Input:1

(q1,0)=q2 δ(q1,1)=q3

δ(q2,0)= q2 δ(q2,1)=q4

i.e. {q1, q2} is not partitioned.

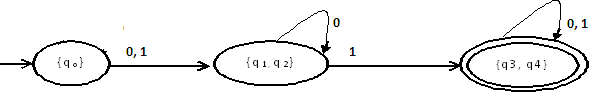
i.e. Пnew = П

Because Пnew = П we can stop here and considered the sets in Пnew as states in Minimized DFA. Merging of all indistinguishable states leads to minimum DFA, given in figure.

Q = {[q0], [q1, q2], [q3, q4]}



|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| **[q0]** | [q1, q2] | [q1, q2] |
| **[q1, q2]** | [q1, q2] | [q3, q4] |
| **\*[q3, q4]** | [q3, q4] | [q3, q4] |



* 1. **2DFA:**

###### Figure: Minimized DFA

A mathematical model of a machine, with the ability of a read-head to move left as well as right is a two way DFA.

A two-way DFA (2DFA) consists of a finite state control and a read-only input tape that allows an input to be read back and forth. The 2DFA decides whether a given input is accepted or rejected.

###### Elements of 2DFA:

A two-way DFA (2DFA) can be described by a 5 tuple Mo = (Q, ∑, δ, q0, λ) where −

* **Q** is a finite set of states.
* **∑** is a finite set of symbols called the input alphabet.
* **δ** is the input transition function where δ: Q **×** ∑ → Q **× {**L, R**}.**
* **q0** is the initial state from where any input is processed (q0 ∈ Q).
* F ⊆ Q is the set of accept states.

### Minimal algorithm for DFA

Let M = < Q, ∑, q0, , F > be a DFA that accepts a language L. Then the following algorithm produces the DFA, that has the smallest number of states among all the DFA‘s that accept L.

## 25

**Step 1**: Identify all unreachable or inaccessible states and eliminate them from the DFA, M.

**Step 2**: Identify all distinguishable states from the DFA and merge them all to form the DFA with smallest number of states.

**Step 3**: construct a DFA from π final .

**Step 4**: END.

### Regular languages:

Suppose ∑ is an alphabet. Then, the class of regular languages over ∑ is inductively defined in the following manner:

1. is a regular language.
2. For each a ∑ , {a} is a regular language.
3. If L1,L2,……Ln are regular languages ,then so Uni=1 Ln.
4. If L1,L2,……Ln are regular languages then so is L1 L2 …Ln.
5. If L is a regular language ,then so is L\*.
6. Nothing else is a regular language,unless it is constructed using points (i)-(v).

**Example:** The language { ,b,bb,…} is constructed by b\*.

### Regular Expressions:

Certain set of strings or languages can be represented in an algebraic fashion, and these algebraic expressions of languages are called regular expression.

A regular expression is a string that describes the whole set of strings according to a certain syntax rule. These expressions are used by text editors and utilities, to search bodies of text for certain patterns.

**Example:** If R1=c and R2 = then R1+ R2 is a regular expression.

### Regular Set:

These are the sets of which are accepted by finite automata. Any set represented by a regular expression is called regular set .

(or)

A set containing all the strings generated by a regular expression is known as regular set.

**Example:** {101},{ab,ba}

### Closure properties:

Assume that FA1and FA2 are two finite automata’s accepting languages L1and L2 defined by regular expressions r1 and r2 respectively as

FA1 = (Q1, 1, 1,q1,f1),

FA2 = (Q2, 2, 2,q2,f2).

###### Case 1: Construction of L1+L2



q1

FA1

f1

qf



q0

q2 FA2

26

f2

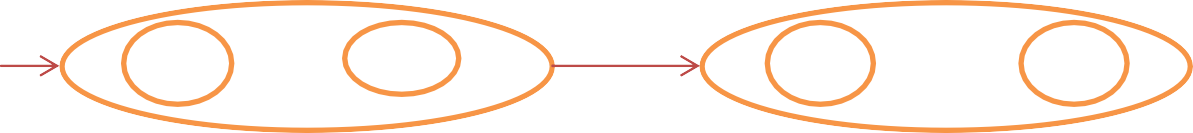
New start l

###### Fig: Automata for L1+L2

**Case 2: Construction of L1** . **L2**

New final

New Start



q1 FA1 f1

q2

New Final

###### Fig: Automata for L1 . L2

FA2 f2

###### Case 3: Construction of L1\*

𝑆𝑡𝑎𝑟𝑡∈



q1

FA1

f1

New start Original Start Original Final new final

**Fig: Automata for L1\***

### Arden’s theorem. Example. [DFA→ R.E]

|  |  |  |
| --- | --- | --- |
| Ʃ  Q | 0 | 1 |
| q1 | q1 | q2 |
| q2 | q3 | q2 |
| q3 | q1 | q2 |

**Arden’s theorem:**

Let p and Q be two regular expressions over Ʃ, and if P does not contain ϵ, then R = Q+RP has a unique solution given by

R=QP\*

R = Q+RP

= Q + (Q+RP) P

= Q + (Q+ (Q+RP) P) P

= Q + ((Q+ (Q + (Q+RP) P) P) P

…

…

= Q + QP + QPP + QPPP + ……

= Q ( + P + PP + PPP + ……) R= Q P\*

If Q=ϵ then, R=Q+RP

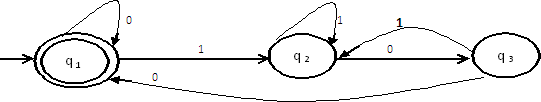
R=ϵ+RP

R=ϵP\* R=P\*

This theorem is used to find a RE, which can be recognized by transitions.

###### Example:

The state diagram for the above transition table is:



###### Fig: Transition Diagram

Now the equations for each state can be, written by considering all edges that enter into that state. Thus,

Substituting q3 in q2

Now,

q1 = q1 0+q3 0+ϵ (ϵ, only for initial state) q2 = q1 1+q2 1+q3 1

q3 =q2 0.

q2 = q1 1+q2 1+ q2 01

= q1 1+q2 1 (1+01)

=q1 1 (1+01)\* (R = Q+RP is R = QP\*)

q1 = q1 0+q3 0+ϵ

= q1 0+q2 0.0+ϵ

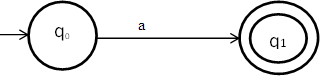
= q1 0+q1 1 (1+01)\*.00+ϵ

= q1 (0+1(1+01)\*.00) +ϵ

= ϵ (0+1(1+01)\*.00)\*

(R =Q + RP => R = QP\*) q1= (0+1(1+01)\*.00)\*

Since, q1 is the final state, RE for DFA will be



**RE= (0+1(1+01)\*00).**

### Find NFA-€ for regular expression (a+bb)\*

1. Consider NFA for ‘a’

|  |  |
| --- | --- |
| Consider NFA for ‘b’    Consider NFA for ‘bb’    Consider NFA for ‘a+bb’    Consider NFA for ‘(a+bb)\*’    **UNIT – V**   * 1. **Ambiguous grammar & Unambiguous grammar**   An **ambiguous grammar** is a context-free **grammar** for which there exists a string that can have more than one leftmost derivation or parse tree, while an  An **unambiguous grammar** is a context-free **grammar** for which every valid string has a unique leftmost derivation or parse tree.   * 1. **Reduction/Simplify of CFG with example:** | |
|  |  |
| **FLAT Important Qs** |  |



|  |  |
| --- | --- |
| 1. Removal of null productions(ɛ-productions) 2. Removal of Unit Productions 3. Removal of useless productions   **Simplify the following grammar**  **G:** S  a**|**aA**|**B**|**C A  aB  B  Aa C  cCD D  ddd  **Solution:**  In a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Besides, there may be some null productions (ɛ-productions), unit productions and useless productions. Elimination of these productions and symbols is called simplification of CFGs. Simplification essentially comprises of the following steps −   1. **Removal of Null Productions**   In the above example ɛ-production is **A**  . We have to remove this production without change the meaning of Grammar. For that we have substitute this production into grammar.  G1: S  a**|**aA**|**B**|**C A  aB  B  Aa **|**a after substitution of A  the above grammar changed  C  cCD Like this  D  ddd   1. **Removal of Unit Productions**   After removing the ɛ-production, we have to check for unit productions from above Grammar. Here unit productions are **S**  **B, S**  **C**   1. **S**  **B**   Substitute the Non unit productions of B on RHS of S  B and eliminate S  B, i.e., replace **S**  **B** with S Aa | a   1. **S**  **C**   Substitute the Non unit productions of C on RHS of S  C and eliminate S  C, i.e., replace **S**  **C** with S  cCD  G2: S  a**|**aA**|**Aa|cCD A  aB  B  Aa **|**a after the elimination of **S**  **B and S** **C** ,  C  cCD the above grammar G1is changed to G2. D  ddd  **C. Removal of useless Variables/ Symbols/Productions**   * 1. After removing unit productions, check for non-generating variables if any in G2.   Non generating variables are those which donot generate any all non-generating variables are useless | |
|  |  |
| **FLAT Important Qs** |  |

|  |  |
| --- | --- |
| and so the productions based on those. We have to eliminate useless variables and the productions based on those.   1. S a   Here S is generating terminal/terminal string. This production is usefull   1. S aA   aaB  aaa  Here S is generating terminal/terminal string. This production is usefull   1. S aaa   Here S is generating terminal/terminal string. This production is usefull   1. A  aB    aB   aa  Here A is generating terminal/terminal string. This production is usefull   1. B  Aa   aBa  aaa  Here B is generating terminal/terminal string. This production is usefull   1. B  Aa    aBa   aaa  Here B is generating terminal/terminal string. This production is usefull   1. C  cCD    c cCD D   c c cCD D D   c c c cCD D D D  ………  Here C is not generating terminal/terminal string. This production is **useless.** So **remove this production**.   1. D  ddd   Here D is generating terminal/terminal string. This production is usefull  G3: S  a**|**aA**|**Aa  A  aB After, removing the variable C and productions based  B  Aa **|**a on that the grammar G2 is changed to G3 as D  ddd  b. Now identify the variables which are not reachable from start variable i.e S, with the help of dependency graph. Dependency graph has to be drawn here.  Since D is not reachable from S, it is useless. After eliminating  The variable D and productions based on that the grammar G3 is changed to G4 as G4: S  a**|**aA**|**Aa  A  aB  B  Aa **|**a The simplified grammar is  G4: S  a**|**aA**|**Aa  A  aB | |
|  |  |
| **FLAT Important Qs** |  |

|  |  |
| --- | --- |
| B  Aa **|**a  L(G4)= {a, aaa,……}  L(G)={a,aaa,…….} L(G)=L(G4)   * 1. **Chomsky normal form**   In [formal language](https://en.wikipedia.org/wiki/Formal_language) theory, a [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar) *G* is said to be in **Chomsky normal form** (first described by [Noam Chomsky](https://en.wikipedia.org/wiki/Noam_Chomsky)) if all of its [production rules](https://en.wikipedia.org/wiki/Production_(computer_science)) are of the form:  *A* → *BC*, or  *A* → *a*, or  *S* → ε,  where *A*, *B*, and *C* are [nonterminal symbols](https://en.wikipedia.org/wiki/Nonterminal_symbol), *a* is a [terminal symbol](https://en.wikipedia.org/wiki/Terminal_symbol) (a symbol that represents a constant value), *S* is the start symbol, and ε denotes the [empty string.](https://en.wikipedia.org/wiki/Empty_string) Also, neither *B* nor *C* may be the [start](https://en.wikipedia.org/wiki/Start_symbol_(formal_languages)) [symbol,](https://en.wikipedia.org/wiki/Start_symbol_(formal_languages)) and the third production rule can only appear if ε is in *L*(*G*), the language produced by the context- free grammar *G*.   * 1. **Greibach normal form**   In [formal language](https://en.wikipedia.org/wiki/Formal_language) theory, a [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar) is in **Greibach normal form** (GNF) if the right- hand sides of all [production](https://en.wikipedia.org/wiki/Production_(computer_science)) rules start with a [terminal symbol](https://en.wikipedia.org/wiki/Terminal_symbol), optionally followed by some variables. A non-strict form allows one exception to this format restriction for allowing the [empty word](https://en.wikipedia.org/wiki/Empty_word) (epsilon, ε) to be a member of the described language. The normal form was established by [Sheila Greibach](https://en.wikipedia.org/wiki/Sheila_Greibach) and it bears her name.  More precisely, a context-free grammar is in Greibach normal form, if all production rules are of the  form:  AaA1A2A3……An or  S  Where A is a [nonterminal symbol,](https://en.wikipedia.org/wiki/Nonterminal_symbol) a is a terminal symbol, A1A2A3……An is a (possibly empty) sequence of nonterminal symbols not including the start symbol, S is the start symbol, and *ε* is the [empty](https://en.wikipedia.org/wiki/Empty_word) [word.](https://en.wikipedia.org/wiki/Empty_word)   * 1. **Pushdown Automata(PDA):**   **Definition:-**  PDA is similar to finite state automata but the difference is it consists of auxiliary stack which provides an unlimited amount of memory.  Context Free Grammar (CFG) is recognised by Pushdown Automaton. PDA is used in different areas like parsing and compiler construction.  PDA is a way to represent the language class called the Context Free Language (CFL).  PDA is a generalisation of Finite Stat Machine(FSA) and a PDA changes from state to state, reading input symbols .Unlike FSA transitions also update the stack either popping(or)pushing symbols.   * 1. **Elements, Components and operations of PDA**   **Elements of PDA:-**  A PDA is a 7-tuple specification | |
|  |  |
| **FLAT Important Qs** |  |

M= (Q, Σ, Γ, δ, q0, z0, F)

Where

Q=set of finite state

Σ=set of finite input symbols Γ= finite set of stack symbol q0 ∈ Q set of final state

F ⊆ Q set of final state

Z0= initial stack symbol placed on the top of the stack δ= is the transition function

δ: Q ×(Σ𝖴{ε})×Γ  Q×Γ\*

### Components of PDA:-

1. Input tape

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | a | b | a | a | ……..... |

1. Read unit **1. Input Tape**
2. Control unit
3. Stack

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

* 1. **Read unit**

CU

**Input tape:-**

* 1. **Control unit**
  2. **Stack**

It is an infinitely long tape on which input is written.

The tape is divided into sequence of cells. Each cell begins from left end and extends to right without an end. Each cell of the holds one input letter or a blank, epsilon. Input string is written on the tape prior to the beginning of the operation of the PDA

**Read unit:-**

Read unit of PDA reads words from the cells of the input tape, beginning with the first letter in the left most cell and then moves to the right. It cannot go back

## 33



|  |  |
| --- | --- |
| **Control unit:-**  It governs the operations of PDA by performing a sequence of transitions between internal states available to it.  The control unit executes a transition, whenever a letter of the input string is provided to it by the read unit. These transitions are determined by transition function of the PDA. Internal states are available to the control unit will have start, accept & reject states.  **Stack:-**  A PDA has an infinitely tall push down stack which has a last in first out discipline.  Stack always start with stack empty  **Operations on stack:-**   1. Push: - adds the input alphabet to the top of the stack 2. Pop: - Removes the top input alphabet from the top of the stack. If the stack is empty the basic pop does not changes the state of the stack. 3. No-Operation (NOP):- Does nothing to the stack.   **UNIT-VI**   * 1. **TURING MACHINE:**   Turing machine was proposed by Alen Turing in 1936 which recognizes more languages than pushdown automata.  **Definition 1**:  Turing machines are simple abstract computation devices intended to help investigate the extent and limitations of what can be computed.  **Definition 2:**  A Turing machine is a kind of state machine at any time the machine is in anyone of the final number of states. Instructions for a turing machine include the specification of conditions under which the machine will make transitions from one state to another.  Turing Machine can be described by a 7 tuple Me = (Q, q0, B, h) where −   * + - **Q** is a finite set of states.     - ∑ is a finite set of symbols called the input alphabet.     - **Г** is a finite set of symbols called the output alphabet.     - **δ** is the input transition function where δ: Q × ∑ → Q×Γ×{L,R,N} .     - **q0** is the initial state from where any input is processed (q0 ∈ Q).     - **B** is the blank symbol. | |
|  |  |
| **FLAT Important Qs** |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| * **h** is a halt state.     **Fig : The Language Hierarchy**   * 1. **Components of Turing Machine:**   A Turing machine is usually described as consisting of the following three components.   * + 1. Input Tape     2. Read/ Write Head     3. Control unit        1. **Input Tape**   ….. B a b a b a b B …..     * + - 1. **Read / Write head** (movement in both direction)       2. **Control Unit**     **Fig : Components of a turing machine**  **a) Input Tape**  A tape is divided into a sequence of numbered cells or sqaures , one next to another . Each cell | |
|  |  |
| **FLAT Important Qs** |  |



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| containsa symbol from some finite alphabet. The alphabet contains a blank symbol(B) . The set of symbols of tape is denoted by .  ….. B a b a b a b B …..  Read-write head Blank symbol   1. **Read or Write head :**   Read or write head moves in two directions (left write).These head reads the data or well as write the data. At each step the head doing:   * 1. Reads the symbol   2. Writes the symbol   3. moves left or right or doesn’t move.   ….. B a b a b a b B …..  Head starts at the left most position of the input string   1. **Control unit :**   Reading from the tape or writing to the tape by the tape . It Contains set of finite states .   1. Initial states 2. Halt state,it stops all further operations. 3. Other states.    1. **Instantaneous description of TM**   All symbols to left of head, State of machine, symbol head is scanning and all symbols to right of head, *i.e. (x1, x2, …..xn)*  Example of Turing machine accepting a string with equal numbers of zeros and ones - this can't be done with FA, as was previous shown.  Programming Turing machine can be done entirely in finite state logic, but can also be done with information on tape.  Finite state logic can also be used to store information, by including tape symbol dependent states.   * 1. **Language accepted by a TM**   A TM accepts a language if it enters into a final state for any input string w. A language is recursively enumerable (generated by Type-0 grammar) if it is accepted by a Turing machine.  A TM decides a language if it accepts it and enters into a rejecting state for any input not in the language. A language is recursive if it is decided by a Turing machine.  There may be some cases where a TM does not stop. Such TM accepts the language, but it does not decide it.   * 1. **Undecidable Problem**   In computability theory and computational complexity theory, an undecidable problem is a decision problem for which it is known to be impossible to construct a single algorithm that always leads to a correct yes-or-no answer. | |
|  |  |
| **FLAT Important Qs** |  |

|  |  |
| --- | --- |
| **Decision Problem**  There are many problems for which the answer is a Yes or a No. These types of problems are known as **decision problems**. For example,   * Whether a given graph can be colored by only 4-colors. * Finding Hamiltonian cycle in a graph is not a decision problem, whereas checking a graph is Hamiltonian or not is a decision problem.   Every decision problem can have only two answers, yes or no. Hence, a decision problem may belong to a language if it provides an answer ‘yes’ for a specific input. A language is the totality of inputs for which the answer is Yes. Most of the algorithms discussed in the previous chapters are **polynomial time algorithms**.  Algorithms such as Matrix Chain Multiplication, Single Source Shortest Path, All Pair Shortest Path, Minimum Spanning Tree, etc. run in polynomial time. However there are many problems, such as traveling salesperson, optimal graph coloring, Hamiltonian cycles, finding the longest path in a graph, and satisfying a Boolean formula, for which no polynomial time algorithms is known. These problems belong to an interesting class of problems, called the **NP-Complete** problems, whose status is unknown.  In this context, we can categorize the problems as follows −  **6.6 P-Class language:**  The class P consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time ***O(nk)*** in worst-case, where **k** is constant.  These problems are called **tractable**, while others are called **intractable or superpolynomial**.  Formally, an algorithm is polynomial time algorithm, if there exists a polynomial ***p(n)*** such that the algorithm can solve any instance of size **n** in a time ***O(p(n))***.  Problem requiring ***Ω(n50)*** time to solve are essentially intractable for large ***n***. Most known polynomial time algorithm run in time ***O(nk)*** for fairly low value of ***k***.  The advantages in considering the class of polynomial-time algorithms is that all reasonable **deterministic single processor model of computation** can be simulated on each other with at most a polynomial slow-d  **NP-Class language:**  The class NP consists of those problems that are verifiable in polynomial time. NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information. Hence, we aren’t asking for a way to find a solution, but only to verify that an alleged solution really is correct.  Every problem in this class can be solved in exponential time using exhaustive search.  **P versus NP**  Every decision problem that is solvable by a deterministic polynomial time algorithm is also solvable by a polynomial time non-deterministic algorithm.  All problems in P can be solved with polynomial time algorithms, whereas all problems in *NP - P* are | |
|  |  |
| **FLAT Important Qs** |  |

|  |  |
| --- | --- |
| intractable.  It is not known whether ***P = NP***. However, many problems are known in NP with the property that if they belong to P, then it can be proved that P = NP.  If ***P ≠ NP***, there are problems in NP that are neither in P nor in NP-Complete.  The problem belongs to class **P** if it’s easy to find a solution for the problem. The problem belongs to **NP**, if it’s easy to check a solution that may have been very tedious to find. | |
|  |  |
| **FLAT Important Qs** |  |

**FORMAL LANGUAGES AND AUTOMATA THEORY**

**Unit wise Question and Answers:**

### UNIT – I

**1. What is computation? What are the different models of Computation? Explain.**

### Computation:

Computation can be defined as finding solution to a problem from given inputs by means of algorithm.

The theory of computation, a subfield of computer science and mathematics, deals with finding solutions to problems from the given inputs.

### Model of computation:

It means a formal, abstract definition of a computer.

There are two types of computations

1. Serial models
2. Parallel models

###### Serial models

\*Turing machine

\*Random access machine

\*Primitive recursive

\*Cellular automaton

\*Finite state machine

\*Cell probe machine

\*Pointer machine

###### Parallel models

\*Multiprocessor model

\*Work-depth model

\*Parallel random-access machine

### Serial models

###### Turing machine:

\*Alternation

\*Alternating Turing machine

\*Nondeterministic Turing machine

\*Oracle Turing machine

\*Probabilistic Turing machine

\*Universal Turing machine

\*Quantum computation

This is a model of computation consisting of a finite state machine controller, a read-write head and an unbounded sequential tape. Depending on the current state and symbol read on the tape, the machine can change its state and then move the head left or right. Unless otherwise specified, a Turing machine is deterministic. Turing machine stores characters or an infinitely long tape, with one square area being scanned by a read/write head at any given time.

###### Random access machine:

This is a model of computation whose memory consists of an unbounded sequence of registers, each of which may hold an integer. In this model, arithmetic operations are allowed to compute the address of a memory register.

###### Primitive recursive:

A total function which can be written, using only nested conditional (if-then-else) statements and fixed iteration (for) loops. These models use functions and function compositon to operate on numbers.

###### Cellular automaton:

This is a two-dimensional organisation of simple, finite state machine whose next state depends on their own state and the states of their eight closest neighbours. In general the machines may be arranged in meshes of higher or lower dimensions, having larger neighbourhoods or arbitrarily complex processors.

###### Finite state machine:

This is a model of computation consisting of a set of states, a start state, an input alphabet and a transition function that maps input symbol and current state to next State. Computation begins in the start state with an input string. It changes to new states, Depending on the transition function. There are the many variants, For example:

* 1. Moore machine, b) Mealy machine, c) ∈-transitions, or null, d) recogniser

###### Cell probe model:

This is a model of computation where the cost of a computation is measured by the total number of memory access to a random access memory, with cell size cell of log n bits.

###### Pointer machine:

This is a model of computation memory consists of an unbounded collection of registers, connected by pointers.

###### Alternation:

This is a model of computation proposed by A. k. chandra, L. Stockmeyere, and D .kozen, which has two kinds of states-AND and OR.

###### Alternating Turing machine:

This is a nondeterministic Turing machine having universal states, from which the machine accepts possible moves of only that state which lead to acceptance.

###### Nondeterministic Turing machine:

This is a Turing machine which has more than one next state for some combination s of contents of the current cell and current state.

###### Oracle Turing machine:

This is a Turing machine with an extra Oracle tape and three extra states q?, qy, qn. when the machine enters q?, control goes to state qy, of the Oracle tape content is in the Oracle set; otherwise control goes to state an.

###### Probabilistic Turing machine:

This is an Turing machine in which some transitions are random choices among finitely many alternatives.

###### Universal Turing machine:

This is a Turing machine that is capable of simulating any other Turing machine by encoding the

latter.

###### Quantum computation:

Here the computation is based on quantum mechanical effects, such as superposition, in addition to classical digital manipulation.

### Parallel models

Page **2** of **63**

###### Multiprocessor model:

This is a model of parallel computation, based on a set of communicating sequential processor.

###### Work-depth model:

This is a model of parallel computation in which one keeps track of the work and depth of computation, without worrying about how it maps onto a machine.

###### Parallel random-access machine:

This is a shared memory model of computation, where the processors typically execute the same instruction synchronously and access to any memory location occurs in unit time.

###### Shared memory:

Here all the processors have the same global image of the whole memory.

### 2. What are the different classes of Automata? How they are classified? Explain in detail.

An automaton is a system which obtains, transforms, transmits and uses information to perform its functions without direct human participation. It is self-operational. The following is the classification of automata:

### Based on Modeling:

**Probabilistic automata:** In a probabilistic automata there is a predetermined probability of each of the next states, given the current state and input.

**Non-probabilistic automata:** In a non-probabilistic automata there is no predetermined probability of next state, given the current state and input.

### Based on Implementation

**Deterministic automata:** In the case of deterministic automata, the output is uniquely determined by the input sequences, i.e., we get a definite output given any input. The behaviour of such automata can be accurately predicted, if the transfer operator is known and given in the form of a table of a logical function. Also, we need to know the initial state and the input sequence.

**Statistical automata:** In statistical automata, random output sequences are generated, given any fixed input. The amount of randomness can be set by applying a probability of any output, given the systems current state and input sequence.

**Memoryless automata:** Memoryless automata, a concept in automata theory, recognizes only one input at a time and produces the output based on that input. The output is not influenced by any additional inputs which arrived. The reaction time of the automaton is constant for all input signals. The internal state of such an automaton is independent of any external action.

**Finite memory automaton:** Finite memory automaton, a concept in automata theory, refers to the type of automata where the group of output signals generated at a given quantised time depends not only on the signals applied at the same moment but also on those which arrived earlier. These preceding external actions (or fragments of them) are recorded in the automaton by a variation of its internal state. The reaction of such an automaton is uniquely determined by the group of input signals that has arrived and by its internal state at a given time. These factors also determine the state into which the automaton goes.

**Infinite memory automata:** It refers to an abstract circuit of a logical automaton which, in principle, is suitable for realising any information-processing algorithm. Turing machine belongs to this class of automata.

### BASED ON PROCESSING

Page **3** of **63**

 **Acceptors and Recognisers**

Acceptors are those which either accept the input or do not. Recognisers are those which either recognise the input or do not. The following automata fall in this category:

* 1. Deterministic finite state machine
  2. Nondeterministic finite state machine
  3. Pushdown automata (PDA)

**Deterministic finite state machine:** A deterministic finite state machine or deterministic finite state automata (DFA) is a finite state machine where for each pair of state and input symbol there is a unique next state. In other words for each state there is at most one transition for each possible input.

**Nondeterministic finite automata:** In nondeterministic finite automata (NFA), there can be more than one transition from a given state, for a given possible input (there may be several possible next states).

In nondeterministic automata the next state depends not only on the current input event, but also on a arbitrary number of subsequent input events. Until these subsequent events occur, it is not possible to determine which state the machine is in.

**Pushdown automata:** Pushdown automata (PDA), is a generalization of a finite state automaton. Like a FSA, a PDA changes from state to state, reading symbols of the input. Unlike an FSA, transitions also update the stack either by popping symbols or pushing them.

 **Transducers**

Transducers generate output from the given input. The following automata are in this category:

1. Mealy machine
2. Moore machine

**Mealy machine:** Mealy machine is a finite state machine, where the outputs are determine by the current state and the input.

**Moore machine:** Moore machine is a finite state machine, where the outputs are determined by the current state alone.

### Based on Input

**Tree automata:** A tree automata is a type of finite state machine. It deals with tree structures, rather than the strings, like moore conventional finite state machines.

**Linear automata:** Finite automata that operates on languages of infinite words. Rabin/Buchi automata: Finite automata that operate on languages of infinite words.

### Based on Applications

**Cellular Automaton:** The cellular automaton consists of a line of cells, each coloured either black or white. At every step, there is a definite rule that determines the colour of a given cell from the colour of its immediate left and right neighbours on the step before. These automata are used for making pretty pictures and animations.

**Geographic automata:** A system that is used to stimulate the behavior and distribution of objects in space such as householdes, pedestrians, vehicles, shops, roads, land parcels, sidewalks etc., with respect to their properties and their locations, is a geographic automata system.

In developing geographic automata systems, the aim is to infuse spatial properties into automata tools and adopt object-based view of urban system in terms of their activities.

Page **4** of **63**

The geographic automata can be of a fixed or non-fixed type. The fixed geographic automata represent the objects that do not change their location over time. In context of urban systems, these objects are road links, buildings footprints, parks etc.

Non-fixed geographic automata symbolize the entities that change their location, overtime, such as pedestrians, vehicles, household etc.

### Give the formal definition of finite state machine (FSM)? What are the components of FSM?

**Definition of Automata**

An automaton (plural: automata) is a self-operating machine.

An automaton is a machine that has an input tape that can be put into any of the several states. Various symbols are written on the tape before execution. The automaton begins reading the symbols on the tape from left to right. Upon reading a symbol from the tape, the machine changes its state and advances the tape. After reading the input completely, the machine (halts). Tate diagram

### Components of FSM:

A finite state automaton is usually described as consisting of three components

* + a control unit
  + A read unit and
  + Input tape (or input file)

The nature and operations of these components are as follows:

###### Input tape:

The input tape, sometimes called the data tape of the automaton, consisting of a sequence of cells. Each of the cells beginning from the left hand end of the tape, (which is considered as the start of the tape) contains one of the words of the string to be processed by the machine. The words themselves are usually identified as comprising input/data symbols, with the individual words treated as if they constituted single symbols. The identification of the medium, containing the input string as a tape, is a historical artifact. Now, it only serves to convey the notion, that the words of the input string are made available to the automaton one at a time in a sequence, from left to right (beginning with the first or the leftmost word).

###### Read unit:

The read unit of the automaton reads words from the cells of the input tape (beginning with the first word in the leftmost cell of the input tape) and provides the individual words, one at a time, to the control unit.

###### Control unit:

The control unit governs the operations of the automaton by performing a sequence of transitions between the internal states available to it. Beginning with its initial state, the control unit executes a transition as each word is provided to it by the read unit, where these transitions are determined by the transition function of the automaton. If, immediately after the last word of the input string has been read, the control unit moves into an accepting state, then the automaton is said to accept or recognise the string of words. Otherwise, if the control unit is not in an accepting state after the last word is read, the automaton rejects the string.

In other words a finite automaton can also be thought of as a device, which satisfies the following conditions:

Page **5** of **63**

* + The tape has the left end extends to the right, without an end.
  + The tape is divided into squares, in each of which a symbol can be written, prior to the start of the operation of the automaton.
  + The tape has read-only head.
  + The head is always at the leftmost square, at the beginning of the operation.
  + The head moves to the right by one square, every time it reads a symbol. It never moves to the left. When it sees no symbol, it stops and the automaton terminates its operation.
  + There is a finite control which determines the state of the automaton as well as controls the movement of the head.

### What are the elements of FSM? What are its advantages and disadvantages of FSM?

**Elements of finite state machine:**

Finite state machine consists of 4 main elements:

### State:

A state is a complete set of properties, transmitted by an object to an observer via one or more channels. Any change in the nature or quantity of such properties in a state is detected by an observer and thus a transmission of information occurs.

The following are some of the types of states:

1. **Start state**: An initial state or condition of a finite state machine.
2. **Accepting state**: If a finite state machine finishes an input string and is in an accepting state, the string is accepted or considered to be valid.
3. **Next state**: The state immediately following the current state, defined by the transition function of a finite state machine and the input is the next state.
4. **Universal state:** A state is an alternating Turing machine, from which the machine only accepts all the possible moves leading to acceptance, is called universal state.
5. **Existential state**: A state is a nondeterministic Turing machine, from which the machine accepts any move that leads to acceptance, is the existential state.
6. **Dead/Trap state**: A no final state of a finite state machine, whose transition on every input symbol terminates on itself.

### Transition:

The following are two equivalent definitions for transition of a finite state machine.

* 1. Transition is the act of passing from one state to the next.
  2. A change from one place or state or subject or stage, to another.

Transitions are represented in the following ways:

* State diagram or Transition diagram
* State transition table
* Transition function

Page **6** of **63**

### State diagram:

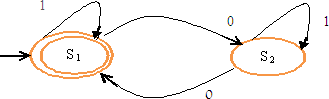
|  |  |
| --- | --- |
| **Notation** | **Meaning** |
|  | {Start of process} |
|  | Final / Accepting / Absorption state{End of  process state} |
|  | Transition |
|  | Input {Guarded transition} |
|  | Output |

###### Table 1: State diagram notation

A diagram consists of circle to represent state and directed line segment to represent transition between the states, is called a state diagram. The symbols used in state diagrams are enlisted in Table 1

For a finite state machine, a state diagram is a directed graph where,

1. Each edge is a transition between two states
   * For a DFA, NFA and Moore machine, input is labelled on each edge.
   * For a mealy machine, input and outputs are labelled on each edge.
2. Each vertex is a state
   * For a Moore machine, output is signified for each state.

**Example 1:** (state diagram)

###### Fig 1: sample state diagram

S1 and S2 are states and S1 in an accept state. Each edge is labelled with the input.

### State transition table:

A state transition table can be described, in general, as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs  States | Present inputs | | | |
| a1 | a2 | …. | an |
| S0 |  |  |  |  |
| S1 |  |  |  |  |
| S2 |  |  |  |  |
| …… |  |  |  |  |
| \* Sn |  |  |  |  |

###### Table: State transition table

* + Tabular representation of Transition that take two arguments and return a value

Page **7** of **63**

* + Rows correspond to states and columns correspond to inputs,
  + Entries correspond to next states
  + The start state (S0) is marked with an arrow (→).
  + The accepting states (Sn) are marked with a star, (\*).

For a finite state machine, a state transition table is a table describing the transition function, of a finite automaton this function governs that state to which the automaton will move, given an input to the machine.

###### Example 2: (State Transition Table)

An example of state transition table of machine, M (fig 1) is given below

|  |  |  |
| --- | --- | --- |
| Inputs States | Present inputs | |
| 1 | 0 |
| \* S1 | S1 | S2 |
| S2 | S2 | S1 |

### Advantages of FSM

###### Table 1.3 state transition table

* Their simplicity makes it easy for inexperienced developers to implement them with no extra knowledge (low entry level)
* Predictability (in deterministic FSM): given a set of inputs and a known current state, the state transition can be predicated, allowing easy testing.
* Due to their simplicity, FSM are quick to design, implement and execute.
* FSMs are relatively flexible. There are a number of ways to implement a FSM-based system in terms of topology and also, it is easy to incorporate many other techniques.
* It enables easy transfer from a meaningful abstract representation to a coded implementation.
* Low processor overhead makes it well suited to domains, where execution time is shared between modules or subsystems. Only the code for current state needs to be executed. Perhaps, a small amount of logic to determine the current state is needed.
* FSM is a old knowledge representation and system modelling technique. It has been around for a long time. It is well known, even as an artificial intelligence technique, with lots of examples to learn from.

### Disadvantages of FSM

* The predictable nature of deterministic finite state machine can be unwanted in some domains such as Computer games.
* Large systems, implemented using a FSM, can be difficult to manage and maintain without a well thought out design. The state transition can cause a fair degree of ‘spaghetti-factor’, when trying to follow line of execution.
* Unsuitable for all problem domains, FSM should only be used when a system’s behavior can be decomposed into separate states with well-defined conditions for state transitions. This means that all states ,transitions and conditions need to be known and well-defined
* The conditions for state transitions are rigid. In other word, they are fixed (this can be overcome by using a fuzzy state machine (FUSM)).

Page **8** of **63**

### Applications of FSM

* FSM are extensively used in the video game industry .the latest games ,like warcraft 111,take advantage of complex FSM systems to control the AI .chat dialogues, where the user is prompted with choices, can also be run using FSMs.
* Besides controlling bots, dialogue and environmental conditions within video games, FSMs also have a large role outside the video game industry. For example, cars ,airplanes ,and robotics(machinery) employ complex FSMs. even websites have a FSM .websites that offer menus for you to traverse other detailed sections of the website ,act much like a FSM with transitions between states.
* The applications of FSMs are also found in language processing: parsing, morphological representations, spelling, information retrieval, tagging, stemming, image compressions, cryptography and designing of compilers.
* The cellular automata are used for making pretty pictures and animations
* A geographic automaton is used to simulate the behaviour and distribution of objects in space such as householders, pedestrians, vehicles, shops, roads, land parcels, sidewalks etc., with respect to their properties and their locations.
* FSMs applied to biological and biomedical problem solving, is the most significant improvement in the recent years. Fourier-transform nuclear magnetic resonance (NMR), NMR imaging (or tomography), x-ray tomography, x-ray diffraction, high performance liquid chromatography, differential scanning calorimetry and mass spectrometry are techniques, which employ FSMs.

Page **9** of **63**

### Write short notes on

**UNIT – II**

### Symbols ii. Alphabets iii. Strings

**SYMBOLS:**

A symbol is a single object and is an abstract entity that has no meaning by itself. It is often called un-interpreted. It can be character. Normally, characters from a typical keyboard are used as symbols.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Greek alphabeti: |  |  | Mathematical iymboli: |
| A | 𝛼 | alpha Ξ | ξ | Xi | ≡ is identically equal to |
| B | 𝛽 | beta Ο | ο | Omicron | ≈ is approximately |
| Γ | 𝛾 | gamma Π | π | Pi | ∉ does not belongs to |
| Δ | 𝛿 | Delta Ρ | ρ | Rho | ≪ is much less than |
| Ε | 𝜀 | Epsilon Σ | σ, ς \* | Sigma | ≫ is much greater than |
| Ζ | 𝜁 | Zeta Τ | τ | Tau | 𝖺 propotional to |
| Η | 𝜂 | Eta Υ | υ | Upsilon | N Natural set |
| Θ | 𝜃 | Theta Φ | φ | Phi |  Implies |
| Ι | 𝖯 | Iota Χ | χ | Chi | ∋ Their exits |
| Κ | 𝜅 | Kappa Ψ | ψ | Psi | C Complex set |
| Λ | 𝜆 | Lambda Ω | ω | Omega | R Real set |
| Μ | μ | Mu ν | ν | Nu |  |

[From [http://www.rapidtables.com/math/]](http://www.rapidtables.com/math/)

### ALPHABETS:

An alphabet is any finite, non-empty set of symbols/ characters. It is denoted byΣ. While distinguishing two alphabets, the second one is usually denoted by Γ.

Example:

1. Σ= {0, 1} is the binary alphabet, consisting of the symbols 0 and 1.
2. Σ= {a, b, c} is an alphabet of three symbols.
3. Σ= {A, B,………, Z} is the uppercase English alphabet.

### STRINGS:

A string is a finite sequence of symbols chosen from some alphabet. In other words, a string is a finite sequence of symbols over an alphabet. It is usually denoted by W or S .Quite often, the letters u,v,w,x,y,z are used to denote strings while the letters i,j,k,l,m,n are used to denote the natural numbers.

1. The length of a string is defined as the number of symbols in the string and is denoted by **|**x**|** for the string x.

Page **10** of **63**

1. A null string is a string with no symbols. In other words, it’s a string of length zero and is denoted by

Λ or 𝜆 or 𝜖.

Example:

* 1. If Σ= {0, 1} then 01010, 1111, 11, 11, 10, 01… are some of the strings chosen from this alphabet.
  2. If Σ= {a, b} then ab, abab, aabb, are the words chosen from this alphabet.

### 2. What are different operations on Strings? Explain with examples.

**Operations of Strings:**

Basically, there are a number of operations on strings. Some of them are

1. Concatenation of Strings

Concatenation of strings is simply the ‘gluing’ of strings together and placing them adjacent to each other in order to form a new string.

Mathematically, If s1 and s2 are two strings then the concatenation‘s’ of s1 and s2 is given by

S’=S1◦S2

={xy |x ϵ S1,y ϵ S2}

**EXAMPLE: (Concatenation of strings)**

* 1. If S1=para and S2=graph, then S1◦S2=paragraph
  2. If ∑={a,b}, S1=ab,S2=baa, then S=S1◦S2=abbaa

c. Let x=0100101 and y=1111, then x ○ y=01001011111

1. abba◦ ϵ=abba. Here **|** ϵ **|**=0
2. Let z be any string then z○ ϵ =ϵ ○z
3. Associativity: (x ○ y) ○ z=x ○ (y ○ z), where x, y, and z are three strings.
4. Kleene Closure

The Kleene star or Kleene closure was introduced by *Stephen Kleene*. It is a unary operation, either on set of strings or on sets of symbols or characters. The application of the Kleene star to a set S is written as S\*. It is widely used for regular expressions.

1. If S is a set of strings, then S\* is defined as the smallest super set of S that contain ϵ and is closed under the string concatenation operation.
2. If S is a set of symbols or characters, then S is the set of all strings over the symbols in S, including the empty string.

Thus,

###### Example: (Kleene Closure)

S+ =⋃𝑖≥0 𝑆i

where i ϵ N i≥0

i={ϵ} U S U S2 U S3 U ......

where si refers to the ith string of S.

* Kleene closure applied to the set of strings the:
  1. If S={cc, d} then

S={ϵ or any word composed of factors of cc and d}

={ϵ or all strings of c’s and d’s in which c occurs in pairs}

= U Si , I ϵ N, N=0, 1, ..

Where

S0=ϵ

S1={cc, d}

Page **11** of **63**

S2=SoS = {cc, d}o{d, cc}={cccc, ccd, dcc,dd}

S3= S2oS = {cccd, ccd, dcc, dd}o{cc, d}={cccccc, ccccd,ccdcc,ccdd, dcccc, dccd, ddcc, ddd} Thus

Where S0= ϵ,

S\* ={ ϵ, d, cc, dd, ccd, ddd, cccc, ccdd, }

* 1. If S={aa, b} then S ={aa, b} , i.e., S = U Si , i ϵ N

S1={aa, b},

S2={aa,b}o{aa,b} = {aaaa, aab, baa, bb},

S3 = S2oS = {aaaa, aab, baa, bb}o{aa, b} = {aaa, aaba, abaa, ababa, baaa, baaba, babaa, bababa}

Thus

Where S0={ϵ}

S1={a}

S\* ={ϵ, b, aa, bb, aab, bbb, baa, aaaa, }

* 1. Let S={a,ba}, then S ={a,ba} , ie., S\* = ⋃𝑖≥0 𝑆i , i ϵ N

S2={aa,ba} S3={aaa,aba,baa} S4={aaaa,aaba,baaa,baba} Thus

S={ϵ,a, aa, ba, aaa, aba, baa, aaaa, aaba, }

* Kleene closure as applied to set of characters. Let S={a,b,c}, then S\* ={ϵ, a, b, c, aa, ab, ac, ba, bb, bc, }

1. Positive Closure

The closure property, extended to set of all words of any length, except the null string i.e., ϵ, is the operation called positive closure. Thus, if S is a set of strings/words, then S is called the positive closure of S with

S+ =⋃𝑖>1 𝑆 i , i ϵ 1,. N.

###### Example: (Positive Closure)

* 1. If S ={a, ba}, then S+ ={a, ba}𝗁 , i.e.,

S+ =U Si , i=1,. ,n

Where

S1={a, ba}

S2=SoS={a, ba}o{ a, ba } = {aa, aba, baa, baba}

S3=S2oS={ aa, aba, baa, baba }o{a, ba}={aaa, aaba, abaa, ababa, baaa, baaba, babaa, babababa} Thus

S𝗁 ={a, aa, ba, aaa, aba, baa, }

1. String Reversal

Intuitively, when a string x is reversed i.e. spelt backwards, then xR is the resulting string which satisfies:

Page **12** of **63**

* XR(i)= x(n+1-i), for 1 ≤ i ≤ n
* |xR|=|x|

###### Example :( String Reversal)

* 1. If x=mus, then xR=sum.
  2. Let x=bottle. Then,xR=elttob

1. Parts of Strings

It is important to talk about the various parts of string, for example, a leading part, a middle segment and a trailing part. These concepts are used at several occasions and are referred to as prefix, substring and suffix respectively.

### Define formal language? Explain in detail about Operations on languages.

**FORMAL LANGUAGES:**

* 1. A formal language is an abstraction for general characteristics of programming languages, that can be defined as a string set, all of which are chosen from some particular alphabet Σ.
  2. A formal language is a set of finite length words drawn from some finite alphabet.
  3. The languages are denoted by letter L, with or without a subscript. Language notation:
     1. L(M) is a language defined by a machine M, that accepts a certain set of strings.
     2. L (G) is a language defined by grammar G, that accepts a certain string set.
     3. L(r) is a language defined by a regular expression. Examples:

1. Let Σ={Z} then the language of all possible strings is given by L1= {z, zz, zzz, zzzz,…}. Here L1 does not contain null string.
2. Language containing the null string is { 𝜖}
3. Null language: { } =𝜙.

### Operations on languages

There are totally 10 operations on languages. They are:

1. Union
2. Intersection
3. Complementation
4. Symmetric difference
5. Concatenation of language
6. Reversal of languages
7. Palindrome languages
8. Length subsetting of a language
9. Kleene star/closure of languages
10. De Morgan’s Laws
11. Union :

This is one of the simplest operation on two languages. As discussed earlier languages are set of strings, and hence the union of two languages L1 and L2 is the set L1UL2. Mathematically, this is represented as

X ∈L1UL2, if and only if x ∈L1 or x ∈L2

**Example**

a. Let L1={0,01,011} and L2={∈,001} then L1 U L2 = {∈,0,01,011}

Page **13** of **63**

b. Let L1={0,000,00000,…} and L2={ ∈,00,0000,…}

then L1 U L2 = {0}\*

1. Intersection :

Suppose L1 and L2 are two languages over an alphabet Γ.Then intersection of L1 and L2 is denoted by L1 ∩ L2, if and only if ,x ∈ L1 and x ∈ L2.

###### Example

1. Let L1={∈} and L2=∅.Then L1∩ L2=∅.
2. Let L be any language, then L ∩ L’=∅.
3. Complementation :

It is an operation performed on a single language. Suppose L is a language over an alphabet ∑, then the compliment of L, denoted by L’, is the language consisiting of all those strings that are not in L over the alphabet. Mathematically, this is expressed as

x ∈ L’ if and only if x ∈ ∑\* - L

###### Example:

* 1. If ∑={a,b} and L={a,b,aa} then the complement of L i.e., L’= ∑\*-L =

{∈,a,b,aa,bb,ab,ba,aaa,bbb,……}-{a,b,aa}

= {∈, bb, ab, ba…}

* 1. Let L= {∈, 1, 11…} be a language over {0, 1}. Then, clearly L’ consists of all strings, containing at least one 0.

1. Symmetric difference :

The operation, symmetric difference is no doubt an important but less familiaroperation.Suppose,L1 and L2 are two languages defined over an alphabet ∑.Then the symmetric difference of L1 and L2 is denoted by L1⨁L2,and is defined as L2⨁L2=(L1𝖴 𝐿2) − (𝐿1 ∩ 𝐿2).

Accordingly, elements of L1⨁L2 are contained either in L1 or L2 but not in both.

###### Example:

a. Let L be a language over an alphabet .Then clearly L⨁∅=L, L⨁L=∅ , L⨁∑\* = L’ and L⨁L’=∑\*

b. Let L1= {00, 0000,..} and L2= {11, 1111,..}.Then , L1⨁L2= {00, 11, 0000, 1111,…} =L1 U L2.

1. Concatenation of languages :

Concatenation of two languages L1 and L2 is the language L1∘L2, each element of which is a string formed by gluing one string of L1 with another string of L2.Mathematically, this is expressed as:

L1 ∘L2=L1L2= {𝑥 ∘y|x ∈L1 and y ∈ L2}

###### Example:

* 1. Let L1={bc,bcc,cc} and L2={cc,ccc} then, L1∘L2= {bccc, bcccc, bcccccc, cccc, ccccc}

Page **14** of **63**

* 1. Let L1={zizz,…} and L2={0,1} then, L1∘L2={zzz,zzzzz,…}.

1. Reversal of languages :

The reversal of a language is an important operation that can be used in illustrating the working procedure of various machines. This operation is similar to the reversal of a string operation. Mathematically, this can be written as LR= {WR|W∈L}

###### Example:

* 1. If L={ab,bc,cd} then, LR= {ba, cb, dc}

b. If L={10,12,34} then, LR= {01, 21, 43}

1. Palindrome languages :

A language called Palindrome over ∑ = {a, b} is defined as: Palindrome= {∈, all strings W, such that WR=W and W∈ ∑}.

Equivalently,

Palindrome={∈,a,b,aa,bb,aaa,bbb,aba,abba,aaaa,…}.

1. Length subsetting of a language :

Suppose L is a language over a fixed alphabet ∑ . Then for sake of convenience or compactness, it may be required to specify the strings in L, of length less than or equal to a specific value or a fixed size.

###### Example:

a. Let L={1,11,111,1111,…} then

L=4= {1111} and L<=4 = {1, 11, 111, 1111}.

1. Kleene star/Closure of languages :

The Kleene star of language L is the language L\*, consisting of all the strings produced by concatenating any finite number of strings from L, together. In other words, kleene star denotes the set of all words/strings of any length of any length from given language. Mathematically, this is expressed as

L\*= {X|∈0X10X2…0XR , with R≥0 and XR∈L}

###### Example:

* 1. If L={∈,z,zz,zzz,…} over ∑= {z} then L\*= {∈, z, zz, zzz,…}

Where L\*=L0 U L1 U L2 U…

b. ∅\*={∈}.

1. De Morgan’s Laws :

It allows one to express the intersection of two languages over ∑,in terms of the two operation viz., union and complementation.These are the fundamental laws in set theory.

Let L1, L2, L3 be any three languages, then L1-(L2 U L3) = (L1-L3)

L1-(L2 ∩ L3) = (L1-L2) U (L1-L3)

Page **15** of **63**

L1 ∩ L2 = (L1C U L2C) C.

### Explain in detail about Chomsky hierarchy Grammars and their Relationship.

**Formal Grammar:**

A formal grammar (or simply grammar) is a precise description of a formal language. In other words, a grammar is a notation for defining a language through a finite number of rules.

Two main categories of formal grammar are:

* + Generative grammars – are set of rules for generation of strings in a language.
  + Analytic grammars – are set of rules to determine whether a string is a member of the language.

###### Generative Grammar

A generative grammar consists of a set of rules for transforming strings. To generate a string in the language, one begins with a string consisting of only a single start symbol and then applies the rules successively (any number of times) to rewrite the string .The language consists of all the strings that can be generated in this manner. Any particular sequence of legal choices, taken during this rewriting this process, yields one particular string, then the grammar is said to be *ambiguous.*

###### Analytic Grammar

Analytic grammars are used to design a parser for the language. Examples of analytic grammars include the following:

1. Top-down parsing language: a highly minimalist analytic grammar formalism, developed in the early 1970s, to study the behaviour of top-down parsers.
2. Link grammars: a form of analytic grammar, designed for linguistics, which derives syntactic structures by examining the positional relationships between pairs of words.
3. Parsing expression grammars: a more recent generalisation of top-down parsing language, designed around the practical expressiveness needs of programming language and compiler writers.

Components of a Generative Grammar

The generative grammar *G* consists of the following components:

* 1. A finite set *V* of non-terminal symbols.
  2. A finite set *T* of terminal symbols that is disjoint from *V*.
  3. A finite set *P* of production rules
  4. A distinguished symbol S ϵ V i.e. the start symbol.

Quad –Tuple Specification of Generative Grammar

The formal definition of generative grammars was first proposed by *Noam Chomsky* in 1950

**Formally, a generative grammar *G* is a quad-tuple**

***G= (V, T, P, S)***

**Where,**

***V* is a finite set of non-terminals**

***T* is a finite set of terminals**

***P* is the finite set of production rules, each of the form**

***(T U V)\*V (T U V)\****  ***(T U V)\****

**i.e., each production rule maps from one string of symbols to another, where, the first string contains at least one non-terminal symbol.**

***S* is the start symbol , *S ϵ V***

Page **16** of **63**

The language of a generative grammar *G*, denoted by L(G) ,is defined as all those strings over *T*, that can be generated by starting symbol S and then applying the production rules in *P* until non-terminal symbols are present.

### Types of Generative Grammar

In *1956*, *Noam Chomsky* classified the generative grammars into types known as the **Chomsky hierarchy.** Each type of generative grammar distinguishes the other types in the form of production rules. The following are the types:

##### Unrestricted or Phrase Structured Grammar

An unrestricted grammar is a formal grammar, on which no restrictions are made on the left and right-hand side of the production.

**Formally, unrestricted grammar *G* is a quad-tuple**

**G= (V, T, P, S)**

**Where, *V, T, S* are same as in the case of generative grammar and *P* is the finite set of production rules, each of the form**

**(V U T )+**  **(V U T)\***

**i.e., no the left-hand side of any productions. However, can appear on the right-hand side of the productions.**

**EXAMPLE:** Let *G = (V,T,P,S)* be unrestricted grammar whose production rules are

*P* ={S  AS | 𝜀

A  aA | a aaaA  ba}.

The derivation for *w=baba* is follows:

S=>AS

=>AAS

=>AAAS

=>AAA

=>aAAA

=>aaAAA

=>aaaAAA

=>baAA

=>baaAA

=>baaaAA

=>bbaA

=>bbaa.

In Chomsky’s Hierarchy, an unrestricted grammar is called **type 0 grammar**. The language generated by this grammar is recursively enumerable language and the recognisers are **Turing machines**. In other words, for every unrestricted grammar G, there exists some TM capable of recognising L (G) and vice versa.

**EXAMPLE:** Recall that the language {anbnan|n>=0} is not a context free language .This language can be generated with an unrestricted grammar.

*S*  a*SBA*|ab*A AB*  *BA*

b*B*  bb

b*A*  ba

Page **17** of **63**

aA  aa

**EXAMPLE:** {ww|w ϵ {a,b}, {anbn|n>=0} and {an!|n>=0} are some of the languages, that are generated by unrestricted grammar.

##### Context-Sensitive Grammar

A Context-sensitive grammar is a formal grammar, in which the left-hand sides and right-hand sides of any production rule may be surrounded by a context of terminal and non-terminal symbols.

**Formally, a context sensitive grammar *G* is a quad-tuple**

***G= (V, T, P, S)***

**Where, V,T,S are same as in the case of generative grammar and P is the finite set of productions, each of the form**

**(V U T) → (V U T)**

**i.e., no ϵ on the left and right- hand sides of any productions. Hence, this grammar is -free.**

**EXAMPLE:** Let *G = (V, T, P, S)* be a context-sensitive grammar whose production rules are

*P*={ *S*  a*B*b a*B*  b*BB*

*B*b  aa

*B*  b }

The derivation for *w=aaabb* is as follows:

S=>aBb

=>bBBb

=>aaBb

=>abBBb

=>aaaBb

=>aaabb

In Chomsky’s hierarchy, context-sensitive grammar is called **type 1 grammar**. The language generated by this grammar is context-sensitive language and the recogniser is **linear bounded automata** (**LBA)**.

**EXAMPLE:** The language {anbncn|n>=1} is a context-sensitive language. This language can be generated with a context-sensitive grammar:

S  abc **|** aSBC CB  BC

Bb  bb.

**EXAMPLE:** The language {an : *P* is a prime number} is generated with context-sensitive grammar.

##### Context- Free Grammar

A context- free grammar is a grammar in which the left-hand side of each production rule consists of only a single non-terminal symbol. This restriction does not make all languages generate context-free grammar.

**Formally, a context free grammar *G* is a quad-tuple**

***G= (V, T, P, S)***

**Where, *V,T,S* are same as in the case of generative grammar and P is the finite set of productions, each of the form**

***A***  ***(V* U *T)***

**i.e., a single non-terminal symbol (A in this case) on left-hand side and any number of terminals and non-terminals(including ϵ ) on the right-hand side of production.**

Page **18** of **63**

**EXAMPLE:** Let *G= (V, T, P, S)* be a context- free grammar whose production rules are:

|  |  |  |  |
| --- | --- | --- | --- |
| *P* ={ |  | | |
|  | S |  | aSa |
|  | S |  | bSb |
|  | S |  | a|b| |

The derivation for *w=ababa* is as follows:

}.

*S*=>*aSa*

=>*abSba*

=>*ababa*

In Chomsky’s hierarchy, context-free grammar is called **type 2 grammar**. The language generated by this grammar is context-free language and the corresponding recogniser is **Push Down Automata (PDA)**.

**EXAMPLE:** The language {anbn|n>=1} is context-free language and this language can be generated with a context-free grammar:

*S*  *aSb*

*S*  *ab*.

**EXAMPLE:** The language {*wwR|w* ϵ {a,b}\* } is generated with a context-free grammar.

##### Regular Grammar

Recall that a grammar said to be regular, iff it is right-linear or left-linear. Informally, a regular grammar is a formal grammar with restrictions on both left and right-hand sides of the productions.

* + Left-hand side of each production rule consists of only a single non-terminal symbol.
  + Right-hand side of each production rule may be nothing , or a single terminal symbol, or a single terminal symbol followed by a non-terminal symbol, but nothing else. (Longer strings of terminals or single terminal without anything else are also allowed.)

**Formally, a regular grammar *G* is a quad-tuple**

***G= (V, T, P, S)***

**Where, *V,T,S* are same as that for generative grammar and *P* is the finite set of productions, each of the form**

***A***  ***t or A***  ***t or B or A***

**Where *A,B, ϵ V* and *t ϵ T***

**EXAMPLE 3.3.0:** Let G=(V,T,P,S) be a regular grammar whose production rules are

*P*= {

*S*  *aS*

*S*  *a*

}.

The derivation for *w=aaa* is follows:

S=>aS

=>aaS

=>aaa.

Page **19** of **63**

In Chomsky’s hierarchy, regular grammar is called a ***type 3 grammar****.* The language generated by this grammar is a regular language and the corresponding recogniser is **FA.**

**EXAMPLE:** The language {anbm **|**m, n>=1} is a regular language and this language can be generated with the regular grammar.

S  aA A  aA A  bB B  bB B  ϵ

Regular grammars are commonly expressed using regular expressions.

The below table summarises different types of generative grammars, discussed so far.

|  |  |  |  |
| --- | --- | --- | --- |
| **Chomsky’s hierarchy** | **Grammar** | **Restrictions on productions** | **Acceptor** |
| **Type 0** | Unrestricted | (V +  (VUT)\*  UT) | TM |
| **Type 1** | Context-sensitive | (VUT) +  (VUT) + | LBA |
| **Type 2** | Context-free | A  (VUT)\*  A is a single non-terminal on left-hand side of productions. | PDA |
| **Type 3** | Regular | A  t or A  tB or A  ϵ  Where A, B ϵ V and t ϵ T | FA |

**Table: *Generative Grammars***

### What are different types of Formal Languages? What are the properties of Formal Languages?

**Formal Languages**

A formal language is a set of finite-length words, drawn from some finite alphabet. In the following sections, we briefly discuss the following types of formal languages and their prosperities.

1. Regular language
2. Context-free language
3. Context-Sensitive language
4. Recursive language
5. Recursively enumerable language
6. Deterministic context-free language
7. Indexed language

##### Regular Language

*A regular language is a formal language* that satisfies the following equivalent properties:

* + It can be accepted by a DFA.

Page **20** of **63**

* + It can be accepted by an NFA
  + It can be described by a regular expression
  + It can be generated by a regular grammar.
  + It can be accepted by a read- only TM. (2DFA is also known as read-only TM).

###### Closure Properties

The regular languages are closed under the following operations: If L, L1, and *L2* are regular languages, then

* the complementation L‾= ∑ -L is also regular
* the Kleene closure L of L is also regular
* the Union *L1 U L2* is also regular.
* the intersection L1 ᴒ L2 is also regular.
* the difference L1- L2 is regular.
* The concatenation L1◦L2 is regular.
* The reverse LR of L is also regular.

###### Examples of Regular Languages

1. The language, consisting of all strings over the alphabet {a ,b}, that accepts even number of a’s.
2. The language, consisting of all strings over the alphabet {0, 1}, that accepts odd number of 1’s and even number 0’s
3. The language, consisting of all strings over ∑ ={0,1}, that contains set of strings of 0’s and 1’s except those containing substring 001.

**EXAMPLE:** Consider a regular language to accept those strings of binary numbers, that are divisible by three, over the alphabet ∑={0,1}.There exist three equivalence classes of strings for this:

* + a number that give 0 as remainder.
  + a number that give 1 as remainder.
  + a number that give 2 as remainder.

Thus the minimal automaton, accepting this language, would have three states corresponding to the equivalence classes, as shown in figure 3.1.

##### Context-Free Language

There exist the following equivalent definitions for the concept of context-free language.

1. A formal language, that is generated by a context-free grammar, is a context-free language.
2. Context-free languages are those that are identical to the set of languages, accepted by PDA.

###### Closure Properties

Context-free language are closed under the following operations. If L, L1, and L2 are the context-free languages and R is the regular language, then:

* + The kleene closure L of L is also a context-free language
  + The cancatenation L1 ○ L2 is a context-free language
  + The union L1 U l2 is context-free
  + The intersection of a context-free language L and a regular language R is always context-free i.e., LᴒR is context-free
  + The reverse LR of L is also context-free

Page **21** of **63**

*Properties of Context-Free Languages*

* + An alternative and equivalent definition of context-free languages employs non-deterministic PDA: a language is context-free, if and only if, it can be accepted by such an automaton.
  + A language can also be modelled as a set of all sequences of terminals, which are accepted by the grammar. This model is helpful in understanding **set operations** an languages.
  + Context-free languages are not closed under complementation, intersection or difference.

###### Examples of Context-Free Languages

* 1. The language of palindrome, over the alphabet ∑={a,b,c}, is a context-free language.
  2. The language L={w|na(w)=nb(w)} is a context-free language.
  3. The language L={anbn|n≥0} is acontext-free language.

##### Context-Sensitive Language

A formal language, that can be described by a context-sensitive grammar, is called a context-sensitive language.

###### Closure Properties

* + The union, intersection, and concatenation of two context-sensitive languages are all context- sensitive.
  + The complementation of a context-sensitive language is also context-sensitive.

*Computational Properties*

Computationally, the context-sensitive languages are equivalent to linear, bounded non-deterministic TMs (i.e., a non-deterministic TM with a tape of Cn cells, where n is the size of the input and C is a constant associated with the machine ). This means that every formal language, that can be decided by such a machine, is a context-sensitive language can be decided by such a machine.

###### Example of Context-Sensitive Language

* + The language L={anbncn: n≥1} is context-sensitive.
  + The language L={anbncndn: n≥1} is context-sensitive.
  + L={ap : p is primenumber} is context-sensitive

##### Recursive Language

The following are the equivalent definitions for the concept of a recursive language:

1. A recursive formal language is a recursive subset in the set of all possible words, over the alphabet of the language.
2. A recursive language is a formal language, for which there exists a TM which, when presented with any input string
   * Halts and accepts if the string is in the language
   * Halts and rejects otherwise.

###### Closure Properties

Recursive language are closed under the following operations. If L,L1 and L2 are recursive languages, then

* + The kleene closure L\* of L is recursive
  + The concatenation L1 ○ L2 is recursive
  + The union L1 U L2 is recursive

Page **22** of **63**

* + The intersection L1 ᴒL2 is recursive
  + The complement L‾=∑ -L is also recursive.

##### Recursively Enumerable Language

There exists three equivalent definitions for the concept of a recursively enumerable language.

1. A recursively enumerable formal language is a recursively enumerable subset in the set of all possible words, over the alphabet of the language.
2. A recursively enumerable language is a formal language, for which ther exists a TM which will enumerate all valid strings of the language.
3. A recursively enumerable language is a formal language for which there exists a TM, that will halt and accept , when presented with any string in the language as input, but may halt and reject or loop forever, when presented with a string not in the language.

###### Closure Properties

Recursively enumerable languages are closed under the following operations. If L, L1 and L2 are recursively enumerable languages, then:

* + The Kleene closure L of L is recursively enumerable
  + The concatenation L1 ○ L2 is recursively enumerable
  + The union L1 U L2 is recursively enumerable
  + The intersection L1 ᴒ L2 is recursively enumerable.

Recursively enumerable languages are not closed under set difference and complementation.

*Recursive vs. Recursively Enumerable Languages*

* + Recursively enumerable languages are accepted, when a TM enters a final state and rejected otherwise (even if TM loops).

Recursive languages are accepted by a TM that always halts, even when it rejects the input.

* + Recursively enumerable languages are known as **type 0 language** in the Chomsky ‘s hierarchy of final languages.
  + The acceptors of recursively enumerable languages are TMs.

The corresponding acceptors of recursive languages are TMs that never loop.

##### Deterministic Context-Free Language

It is a formal language that is a proper subset of languages defined by context-free grammars. The set of deterministic context-free languages is identical to the set of languages accepted by a deterministic PDA.

These type of languages are not defined in Chomsky hierarchy.

##### Indexed language

An indexed language is a formal language, discovered by *Alfred Aho,* which is a proper subset of context-sensitive languages and a proper superset of context-free languages.

An indexed language is minimally characterised by an indexed grammar, or by a **nested stack automaton**. An indexed grammar may have a stack attached to non-terminals, which gets copied to sub- nonterminal. A nested stack automaton may read its stack, in addition to pushing or popping it. Also, a stack may nest other stacks inside it.

The indexed language is not defined in Chomsky hierarchy.

Page **23** of **63**

The table below summarises some of the formal languages:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Chomsky hierarchy** | **Type of language** | **Language defined by** | **Correspond ing acceptor** | **Does non- determinism**  **give more Power?** | **Set is closed under** |
| 3 | Regular language | Regular expression | Finite automaton | No | Union, Concatenation, Kleene star, intersection,  complement |
| 2 | Context-free language | Context-free grammar | Pushdown automaton | Yes | Concatenation, Kleene star,  union |
| 1 | Context- sensitive language | Context-sensitive grammar | Linear- bounded automaton | Yes | Union, Intersection, Concatenation,  Complement. |
| Not defined | Recursive language | No grammar has been characterised by recursive languages so far | TM that never loops | No | Concatenation, Kleene star, Union, Intersection, Complement. |
| 0 | Recursively enumaerable langugae | Unrestricted grammar | TM | No | Concatenation, Kleene star, Union, Intersection. |

###### Table: Formal Languages

Page **24** of **63**

### UNIT – III

1. **Construct NFA for the NFA- ε, given in figure. [NFA- ϵ to NFA]**

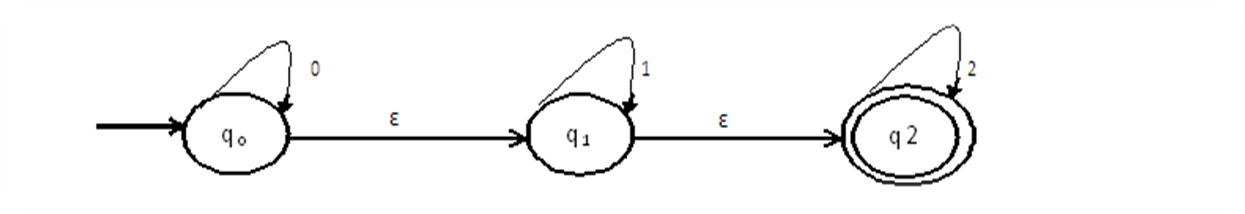
Let M={Q,∑,𝛿,q0,F} is a NFA- ε which accepts the language L(M).There should be equivalent NFA by M′=(Q′, ∑′,𝛿′,q0′,F′}.

###### Rule for conversions:

𝜹′**(q, a)=**∈**-closure(**𝜹**(**∈**-closure(q),a))**

**Example:**

Construct an equivalalent NFA without ε for the below automaton



###### Solution:

M=({q0,q1,q2},{0,1,2},𝛿,q0,{q2})

Transition Table:

Finding ∈-closure for all states

###### Dig: NFA with ∈

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 𝛿 | 0 | 1 | 2 | ∈ |
| q0 | {q0} | ∅ | ∅ | {q1} |
| q1 | ∅ | {q1} | ∅ | {q2} |
| q2 | ∅ | ∅ | {q2} | ∅ |

**Table: State Transition Table**

∈ -closure (q0) = {q0, q1, q2} [∈-closure (q) will never be empty]

∈ -closure (q1) = {q1, q2}

∈ -closure (q2) = {q2} Compute 𝛿′:

𝛿′ (q0,0)=∈-closure(𝛿(∈-closure(q0),0))

=∈-closure(𝛿(q0,q1,q2),o)

=∈-closure(𝛿(q0,0)𝖴 𝛿(q1,0)𝖴 𝛿(q2,0))

=∈-closure (q0𝖴 ∅ 𝖴 ∅)

=∈-closure(q0)

={q0,q1,q2}

Page **25** of **63**

𝛿′ (q0,1)=∈-closure(𝛿(∈-closure(q0),1))

=∈-closure(𝛿(q0,q1,q2),1)

=∈-closure(𝛿(q0,1)𝖴 𝛿(q1,1)𝖴 𝛿(q2,1))

=∈-closure(∅ 𝖴 {𝑞1} 𝖴 ∅)

=∈-closure(q1)

={q1,q2}

𝛿′ (q0,2)=∈-closure(𝛿 (∈-closure(q0,2))

=∈-closure(𝛿(q0,q1,q2),2))

=∈-closure(𝛿(q0,2) 𝖴(q1,2) 𝖴(q2,2))

=∈-closure(∅ 𝖴 ∅ 𝖴q2)

=∈-closure(q2)

={q2}

𝛿′ (q1,0)= ∈-closure(𝛿(∈-closure(q1),0))

= ∈-closure(𝛿(q1,q2),0)

= ∈-closure(𝛿(q1,0) 𝖴 𝛿(q2,0))

= ∈-closure(∅ 𝖴 ∅)

=∅

𝛿′ (q1,1)= ∈-closure(𝛿(∈-closure(q1),1))

=∈-closure(𝛿(q1,q2),1)

= ∈-closure(𝛿(q1,1) 𝖴 𝛿(q2,1))

= ∈-closure(q1, 𝖴 ∅)

= ∈-closure(q1)

={q1,q2}

𝛿′ (q1,2)= ∈-closure(𝛿(∈-closure(q1),2))

=∈-closure(𝛿(𝑞1,q2),2)

= ∈-closure(𝛿(q1,2) 𝖴 𝛿(q2,2))

= ∈-closure(∅ 𝖴 q2)

={q2}

𝛿′ (q2,0)= ∈-closure(𝛿(∈-closure(q2),0))

Page **26** of **63**

= ∈-closure(𝛿(q2),0)

= ∈-closure(𝛿(q2,0))

=∅

𝛿′ (q2,1)= ∈-closure(𝛿(∈-closure(q2),1))

= ∈-closure(𝛿(𝑞2),0)

= ∈-closure 𝛿(q2,0)

= ∅

𝛿′ (q2, 2) = ∈-closure (𝛿(∈-closure (q2), 2))

=∈-closure(𝛿(q2),2)

=∈-closure(q2)

={q2}

Summary of δ′:

𝛿(q0,0) ={q0,q1,q2}

𝛿(q0,1) ={q1,q2}

𝛿(q0,2) ={q2}

𝛿(q1,0) =∅

𝛿(q1,1) ={q1,q2}

𝛿(q1,2) ={q2}

𝛿(q2,0) =∅

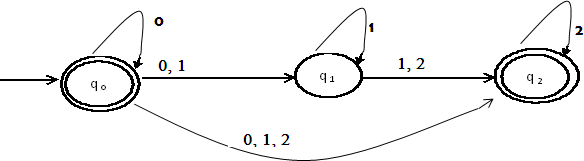
𝛿(q2,1) =∅

𝛿(q2,2) ={q2}

State Transition Table for transition function δ′ is:

Final State diagram for NFA without :

|  |  |  |  |
| --- | --- | --- | --- |
| 𝛿′ | 0 | 1 | 2 |
| q0 | {q0,q1,q2} | {q1,q2} | {q2} |
| q1 | ∅ | {q1,q2} | {q2} |
| q2 | ∅ | ∅ | {q2} |



###### Dig: NFA with out ∈

Page **27** of **63**

### Construct a DFA equivalent to NFA given in figure below. [NFA  DFA]

Let M={Q,∑,𝛿,q0,F} is a NFA which accepts the language L(M).There should be equivalent DFA by M′=(Q′, ∑′,𝛿′,q0′,F′}.

###### Rule for conversions:

𝜹′(**(q0, q1), 0) =** 𝜹**(q0, 0)** 𝖴 𝜹**(q1, 0)**

**Example:**

Let M = ({q0, q1},{0, 1}, 𝛿,q0,{q1}) be NFA

Where 𝛿(q0, 0) = {q0,q1}

𝛿(q0, 1) = {q1}

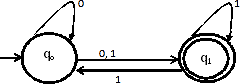
𝛿(q1, 0) = ∅

𝛿(q1, 1) = {q0, q1}

Construct its equivalent DFA from the above transitions

###### Solution:

State Transition diagram:



Transition Table:

###### Diagram: NFA

|  |  |  |
| --- | --- | --- |
| 𝛿 | 0 | 1 |
| q0 | {q0,q1} | {q1} |
| q1 | ∅ | {q0,q1} |

Here in the above table new state Occurred i.e, [q0, q1].We need to compute transitions from this new state [q0, q1].

Compute 𝛿′:

𝛿′([q0, q1], 0) = 𝛿(q0, 0) 𝖴 𝛿(q1, 0)

= [q0, q1] 𝖴 ∅

= [q0, q1]

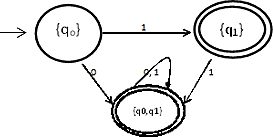
𝛿′([q0,q1],1) = 𝛿(q0,1)𝖴 𝛿(q1,1)

= q1𝖴{ q0,q1}

= [q0,q1]

Page **28** of **63**

New State Transition Table:



|  |  |  |
| --- | --- | --- |
| 𝛿′ | 0 | 1 |
| q0 | [q0,q1] | [q1] |
| q1 | ∅ | [q0,q1] |
| [q0,q1] | [q0,q1] | [q0,q1] |

Note: Here in the above table no new state occurred. So we stopped the procedure here. If any new state occurred we have to find out further transitions

New State diagram:

**Diagram: DFA for the above NFA**

### What is Moore machine? Write example convert Moore machine into mealy machine.

**Definition:** A Moore machine is a finite state automaton, where the outputs are determined by the current state alone.

A Moore machine can be described by a 6 tuple Mo = (Q, ∑, δ, q0, Г, λ) where −

* + **Q** is a finite set of states.
  + **∑** is a finite set of symbols called the input alphabet.
  + Г is a finite set of symbols called the output alphabet.
  + **δ** is the input transition function where δ: Q × ∑ → Q
  + λ is the output transition function where λ: Q → Г
  + **q0** is the initial state from where any input is processed (q0 ∈ Q).

A Moore machine associates an output symbol with each state, and each time a state is entered, an output is obtained simultaneously. So, the first output always occurs as soon as the machine starts.

###### Example to convert Moore machine into mealy machine

**Example:** Convert the following Moore machine into an equivalent Mealy machine. Mo=({q0,q1,q2,q3},{a ,b},{0,1},δ, λ, q0) where, λ and δ are given below.

|  |  |  |  |
| --- | --- | --- | --- |
| ∑  Q | a | b | λ |
| q0 | q1 | q3 | 1 |
| q1 | q3 | q1 | 0 |

Page **29** of **63**

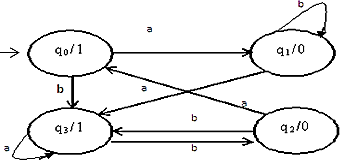
|  |  |  |  |
| --- | --- | --- | --- |
| q2 | q0 | q3 | 0 |
| q3 | q3 | q2 | 1 |

###### Fig: Transition and output for Moore Machine

**Solution:**

To construct a Mealy machine from Moore Machine Me= (Q, ∑, δ, Г, λ’, qo), define

**λ’ (q, a) =λ (δ (q, a))** ∀ q and a.

Based on the given data, the transition diagram of Moore machine is shown in above figure.

Thus,

* λ’(qo, a)=λ(δ(qo, a))=λ(q1)=0 ⇒a/0
* λ’(qo, b)=λ(δ(qo, b))=λ(q3)=1 ⇒b/1
* λ’(q1, a)=λ(δ(q1, a))=λ(q3)=1 ⇒a/1
* λ’(q1, b)=λ(δ(q1, b))=λ(q1)=0 ⇒b/0
* λ’(q2, a)=λ(δ(q2, a))=λ(qo)=0 ⇒a/0
* λ’(q2, b)=λ(δ(q2, b))=λ(q3)=0 ⇒b/0
* λ’(q3, a)=λ(δ(q3, a))=λ(q3)=0 ⇒a/0
* λ’(q3, b)=λ(δ(q3, b))=λ(q2)=0 ⇒b/0

State Transition Table:

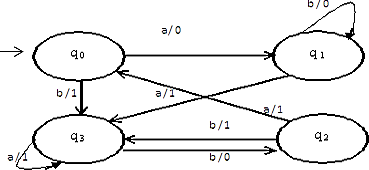
###### Figure: Moore Machine

|  |  |  |
| --- | --- | --- |
| ∑ Q | a | b |
| q0 | [q1, 0] | [q3, 1] |
| q1 | [q3, 1] | [q1, 0] |
| q2 | [q0, 0] | [q3, 0] |
| q3 | [q3, 0] | [q2, 0] |

State diagram:

###### Fig: Transition and output for Mealy Machine

Page **30** of **63**

**Figure: Mealy Machine**

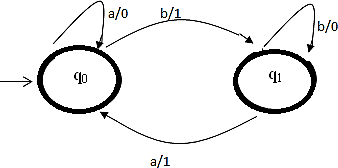
### What is Mealy machine? Write example convert mealy machine into Moore machine.

**Definition:** A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

It can be described by a 6 tuple Me = (Q, ∑, δ, q0, Г, λ) where −

* + **Q** is a finite set of states.
  + **∑** is a finite set of symbols called the input alphabet.
  + Г is a finite set of symbols called the output alphabet.
  + **δ** is the input transition function where δ: Q × ∑ → Q
  + λ is the output transition function where λ: Q → Г
  + **q0** is the initial state from where any input is processed (q0 ∈ Q)..

###### Example to convert mealy machine into Moore machine

**Example**: convert the mealy machine in below figure into an equivalent Moore machine.

###### Solution:

**Fig: Mealy Machine**

To construct a Moore Machine from Mealy machine Mo= (Q, ∑, δ, Г, λ’, qo), define

**δ’([q, b],a)=[ δ(q ,a), λ(q, a)] and λ’(q, b)=b.**

Given, Me= ({q0, q1},{a, b},{0,1},δ, qo, λ), Construct an equivalent Moore machine Mo= (Qx, ∑, Г, ∂’, λ’, [qo, b0]),

Where ,Qx={[q0,0],[q0,1],[q1,o],[q1,1]}, ∑={a ,b}.

Thus,

* + - δ’([qo,0],a) = [ δ(qo, a),λ(q0,a)]

Page **31** of **63**

= [qo, 0]

* + - δ’([qo,0],b) = [ δ(q0,b),λ(q0,b)]

= [q1, 1]

* + - δ’([qo,1],a) = [ δ(q0,a),λ(q0,a)]

= [q0, 0]

* + - δ’([qo,1],b) = [ δ(q0,b),λ(q0,b)]

= [q1, 1]

* + - δ’([q1,0],a) = [ δ(q1,a),λ(q1,a)]

= [q0, 1]

Similarly δ’ ([q1, 0], b) = [ q1,0]

* + - δ’([q1,1],a) = [ δ(q1,a),λ(q1,a)]

= [q0, 1]

* + - δ’([q1,1],b) = [ δ(q1,b),λ(q1,b)]

= [q1, 0]

We also find: λ’ (q, b) =b states i.e.

* + - λ’(q0,0)=0
    - λ’(q0,1)=1
    - λ’(q1,1)=1
    - λ’(q1,0)=0

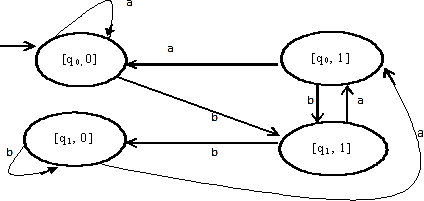
Finally, we draw the transition table and diagram for the Moore machine as follows:

State Transition Table:

|  |  |  |
| --- | --- | --- |
| ∑  Qx | a | b |
| [q0,0] | [q0,0] | [q1,0] |
| [q0,1] | [q0,0] | [q1,1] |
| [q1,0] | [q0,1] | [q1,0] |
| [q1,1] | [q0,1] | [q1,0] |

State diagram:

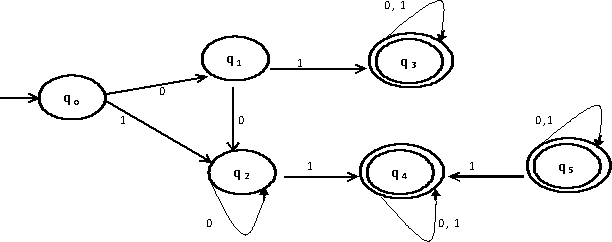
###### Table: Transition Table for Moore machine

**Figure: Equivalent Mo for the above Me**

Page **32** of **63**

### UNIT – IV

**1. Minimize the DFA**

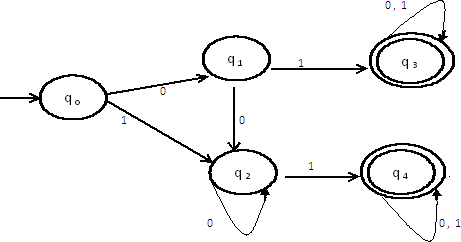


###### Rules for Minimization:

1. Unreachable or inaccessible states
2. Dead states
3. Non-distinguishable or indistinguishable state or equivalent states.

The states that can be **eliminated from automata**, without affecting the language accepted by automata are:

1. **Unreachable or inaccessible states**

[A state is unreachable state, if it cannot be reached from the start state.] Here q5 is unreachable state. So remove q5. Now the DFA becomes.

###### Dead states

[A state is dead, if it is not accepting state and has no out-going transitions, except to itself. Here there is no dead state]

Here there is no dead state.

###### Non-distinguishable or indistinguishable state or equivalent states.

* 1. П=({q0, q1,q2}, {q3, q4}) [i.e. **All Final states into one group and all Non-final states**

###### into another Group]

**Partitions:**

q0 q1 q2

q3 q4

Page **33** of **63**

Form Пnew, with subset {q0, q1, q2}.

###### Input: 0 Input:1

𝛿(q0,0)=q1 𝛿(q0,1)= q2

δ(q1,0)=q2 δ(q1,1)=q3

δ(q2,0)= q2 δ(q2,1)=q4

Since for input ‘1’, states q1 and q2 give transitions to the final state and q0 is not.

So {q0, q1, q2} is partitioned into {q0}, {q1, q2} With subset {q3, q4},

###### Input: 0 Input:1

𝛿(q3, 0)=q3 𝛿(q3,1)= q3

δ(q4, 0)=q4 δ(q4,1)=q4

i.e. {q3, q4} is not partitioned.

So Пnew = ({q0}, {q1, q2}, {q3, q4}) П ≠ Пnew

* 1. Now assign П new to П П=({q0}, {q1,q2}, {q3, q4})

###### Partitions:

q

q1 q2

q3 q4

Form Пnew, with subset {q1, q2}.

###### Input:0 Input:1

𝛿(q1,0)=q2 δ(q1,1)=q3

δ(q2,0)= q2 δ(q2,1)=q4

i.e. {q1, q2} is not partitioned.

i.e. Пnew = П

Because Пnew = П we can stop here and considered the sets in Пnew as states in Minimized DFA. Merging of all indistinguishable states leads to minimum DFA, given in figure.

Q = {[q0], [q1, q2], [q3, q4]}

Σ = {0, 1}

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| **[q0]** | [q1, q2] | [q1, q2] |
| **[q1, q2]** | [q1, q2] | [q3, q4] |
| **\*[q3, q4]** | [q3, q4] | [q3, q4] |

Page **34** of **63**

### A) What is 2DFA?

**Figure: Minimized DFA**

### Write the Minimal algorithm for DFA?

1. **What is regular Expression, regular Language, and regular Set.**

### Explain closure properties of Regular language.

###### Solution:

1. **2DFA:**

A mathematical model of a machine, with the ability of a read-head to move left as well as right is a two way DFA.

A two-way DFA (2DFA) consists of a finite state control and a read-only input tape that allows an input to be read back and forth. The 2DFA decides whether a given input is accepted or rejected.

###### Elements of 2DFA:

A two-way DFA (2DFA) can be described by a 5 tuple Mo = (Q, ∑, δ, q0, λ) where −

* + **Q** is a finite set of states.
  + **∑** is a finite set of symbols called the input alphabet.
  + **δ** is the input transition function where δ: Q **×** ∑ → Q **× {**L, R**}.**
  + **q0** is the initial state from where any input is processed (q0 ∈ Q).
  + F ⊆ Q is the set of accept states.

###### Minimal Algorithm for DFA:

Let M = < Q, ∑, q0,𝛿, F > be a DFA that accepts a language L. Then the following algorithm produces the DFA, that has the smallest number of states among all the DFA‘s that accept L.

**Step 1**: Identify all unreachable or inaccessible states and eliminate them from the DFA, M.

**Step 2**: Identify all distinguishable states from the DFA and merge them all to form the DFA with smallest number of states.

**Step 3**: construct a DFA from π final .

**Step 4**: END.

###### Regular languages:

Suppose ∑ is an alphabet. Then, the class of regular languages over ∑ is inductively defined in the following manner:

1. ∅ is a regular language.
2. For each a ∈ ∑ , {a} is a regular language.

Page **35** of **63**

1. If L1,L2,……Ln are regular languages ,then so Uni=1 Ln.
2. If L1,L2,……Ln are regular languages then so is L1∘L2∘…Ln.
3. If L is a regular language ,then so is L\*.
4. Nothing else is a regular language,unless it is constructed using points (i)-(v).

**Example:** The language {𝜖,b,bb,…} is constructed by b\*.

###### Regular Expressions:

Certain set of strings or languages can be represented in an algebraic fashion, and these algebraic expressions of languages are called regular expression.

A regular expression is a string that describes the whole set of strings according to a certain syntax rule. These expressions are used by text editors and utilities, to search bodies of text for certain patterns.

**Example:** If R1=c and R2 =𝜙 then R1+ R2 is a regular expression.

###### Regular Set:

These are the sets of which are accepted by finite automata. Any set represented by a regular expression is called regular set .

(or)

A set containing all the strings generated by a regular expression is known as regular set.

**Example:** {101},{ab,ba}

###### D. Closure properties:

Assume that FA1and FA2 are two finite automata’s accepting languages L1and L2 defined by regular expressions r1 and r2 respectively as

FA1 = (Q1,Σ1,𝛿1,q1,f1),

FA2 = (Q2,Σ2,𝛿2,q2,f2).

###### Case 1: Construction of L1+L2

∈

FA1

f

∈

New start ∈

FA2

∈

∈ l

∈

New final

###### Fig: Automata for L1+L2

Page **36** of **63**

###### Case 2: Construction of L1 . L2

New Start ∈

q FA1 q FA2 f

New Final

###### Fig: Automata for L1 . L2

**Case 3: Construction of L1\*** ∈

𝑆𝑡𝑎𝑟𝑡 ∈

∈

∈

∈

q

FA1

∈

f1

New start Original Start Original Final new final

**Fig: Automata for L1\***

1. **State Arden’s theorem. Construct a R.E corresponding to the DFA, representing the table given below; q1 is both the initial and final state transition table is given below. [DFA→R.E]**

|  |  |  |
| --- | --- | --- |
| Ʃ  Q | 0 | 1 |
| q1 | q1 | q2 |
| q2 | q3 | q2 |
| q3 | q1 | q2 |

**Arden’s theorem:**

Let p and Q be two regular expressions over Ʃ, and if P does not contain ϵ, then R = Q+RP has a unique solution given by

R=QP\*

R = Q+RP

= Q + (Q+RP) P

= Q + (Q+ (Q+RP) P) P

= Q + ((Q+ (Q + (Q+RP) P) P) P

…

…

= Q + QP + QPP + QPPP + ……

Page **37** of **63**

= Q ( + P + PP + PPP + ……)

R= Q P\*

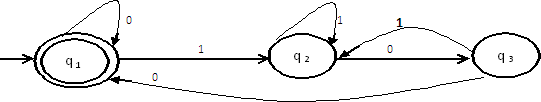
If Q=ϵ then, R=Q+RP

R=ϵ+RP

R=ϵP\* R=P\*

This theorem is used to find a RE, which can be recognized by transitions.

###### Example:

The state diagram for the above transition table is:

###### Fig: Transition Diagram

Now the equations for each state can be, written by considering all edges that enter into that state. Thus,

Substituting q3 in q2

Now,

q1 = q1 0+q3 0+ϵ (ϵ, only for initial state) q2 = q1 1+q2 1+q3 1

q3 =q2 0.

q2 = q1 1+q2 1+ q2 01

= q1 1+q2 1 (1+01)

=q1 1 (1+01)\* (R = Q+RP is R = QP\*)

q1 = q1 0+q3 0+ϵ

= q1 0+q2 0.0+ϵ

= q1 0+q1 1 (1+01)\*.00+ϵ

= q1 (0+1(1+01)\*.00) +ϵ

= ϵ (0+1(1+01)\*.00)\*

(R =Q + RP => R = QP\*)

Page **38** of **63**

q1= (0+1(1+01)\*.00)\*

Since, q1 is the final state, RE for DFA will be

**RE= (0+1(1+01)\*00).**

### A) Find the shortest string that is not in the language represented by the regular expression a\*(ab)\*b\*

1. **For the two regular expressions given below,**

### Find a string corresponding to r2 but not to r1 and

* 1. **Find a string corresponding to both r1 and r2. r1=a\*+b\* r2=ab\*+ba\*+b\*a+ (a\*b)\***

###### Solution:

1. Regular expression=a\*(ab)\*b\* L(R) = L (a\*(ab)\*b\*)

= L (a\*). L(ab)\*.L(b\*)

= {ϵ,a, aa, aaa.....}.{ ϵ,ab, abab, ababab,.....}.{ ϵ,b,bb,bbb. }

= {ϵ,ab,abab,aaab,aabab,aa,aaabab,aaa....}{ ϵ,b, bb. }

= {ϵ,b,bb,bbb,ab,abb,abbb,abab,ababb,a,ab,abb,aab,aabb,aabbb,aabab,aa,aab,aabb,aaabb. }

= {ϵ, a, b, aa, bb, ab, aaa, aab. }

Here in the above language the missing shortest string is ba.

1. For the two expressions given below
   1. Find a string corresponding to R2 but not in R1.
   2. Find a string corresponding to both R1 and R2. r1=a\*+b\* r2=ab\*+ba\*+b\*a+(a\*b)\*.

Ans: L (r1) = L(a\*+b\*)

= L(a\*)𝖴L(b\*)

= {ϵ,a,aa, aaa,....} 𝖴 {ϵ,b,bb,bbb,. }

= {ϵ,a,b,aa, bb, aaa, bbb. }

L(r2) = L(ab\*+ba\*+b\*a+(a\*b)\*)

= L(ab\*)𝖴 L(ba\*)𝖴 L(b\*a)𝖴 L(a\*b)\*

= {a(ϵ ,b,bb,bbb,.....)} 𝖴 {b(ϵ ,a,aa,aaa)}𝖴 {ϵ ,b,bb,bbb,.....}.a} 𝖴{{ϵ,a,aa,aaa }b}\*

= {(a,ab,abb) 𝖴 (b,ba,baa) 𝖴 (a,ba,bba) 𝖴 (b,ab,aa,b)\*}

= {(a,ab,abb,b,ba,baa,a,ba,bba) 𝖴 (ϵ,b,ab,aab,bb,abab,aabaab)}

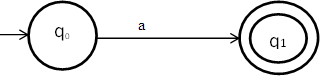
= {∈,a,b,ab,ba,bb,aab,baa,bba,aab. }

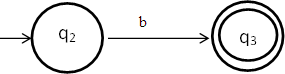
1. String corresponding to R2 but not in R1 is {ab, ba }

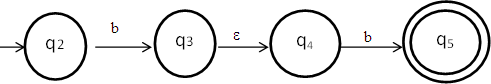
Page **39** of **63**

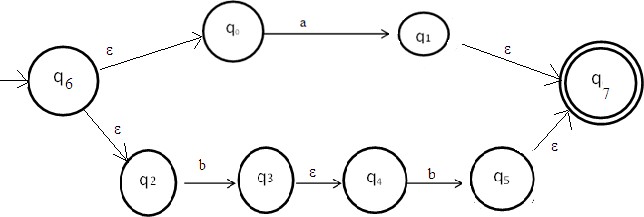
1. String corresponding to both R1 and R2 is {∈,a ,b, bb. }

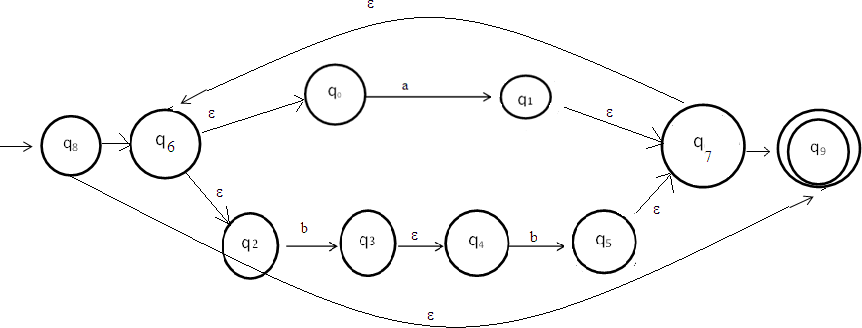
### Find NFA-Ԑ for regular expression (a+bb)\*

* 1. Consider NFA for ‘a’

Consider NFA for ‘b’

Consider NFA for ‘bb’

Consider NFA for ‘a+bb’

Consider NFA for ‘(a+bb)\*’

Page **40** of **63**

### UNIT – V

1. **Simplify the following grammar G:** S  a**|**aA**|**B**|**C

A  aB| ∈

B  Aa C  cCD D  ddd

###### Solution:

In a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Besides, there may be some null productions (ɛ-productions), unit productions and useless productions. Elimination of these productions and symbols is called simplification of CFGs. Simplification essentially comprises of the following steps −

Reduction of CFG

* 1. Removal of null productions(ɛ-productions)
  2. Removal of Unit Productions
  3. Removal of useless productions

###### Removal of Null Productions

In the above example ɛ-production is **A**  𝜀. We have to remove this production without change the meaning of Grammar. For that we have substitute this production into grammar.

G1: S  a**|**aA**|**B**|**C A  aB

B  Aa **|**a after substitution of A  𝜀 the above grammar changed

C  cCD Like this

D  ddd

###### Removal of Unit Productions

After removing the ɛ-production, we have to check for unit productions from above Grammar. Here unit productions are **S**  **B, S**  **C**

1. **S**  **B**

Substitute the Non unit productions of B on RHS of S  B and eliminate S  B, i.e., replace **S**  **B** with S Aa | a

1. **S**  **C**

Substitute the Non unit productions of C on RHS of S  C and eliminate S  C, i.e., replace **S**  **C** with S  cCD

G2: S  a**|**aA**|**Aa|cCD A  aB

B  Aa **|**a after the elimination of **S**  **B and S** **C** ,

C  cCD the above grammar G1is changed to G2.

Page **41** of **63**

D  ddd

###### Removal of useless Variables/ Symbols/Productions

* 1. After removing unit productions, check for non-generating variables if any in G2.

Non generating variables are those which donot generate any all non-generating variables are useless and so the productions based on those. We have to eliminate useless variables and the productions based on those.

* + 1. S a

Here S is generating terminal/terminal string. This production is usefull

* + 1. S aA

aaB

aaa

Here S is generating terminal/terminal string. This production is usefull

* + 1. S aaa

Here S is generating terminal/terminal string. This production is usefull

* + 1. A  aB

 aB

 aa

Here A is generating terminal/terminal string. This production is usefull

* + 1. B  Aa

aBa

aaa

Here B is generating terminal/terminal string. This production is usefull

* + 1. B  Aa

 aBa

 aaa

Here B is generating terminal/terminal string. This production is usefull

* + 1. C  cCD

 c cCD D

 c c cCD D D

 c c c cCD D D D

………

Here C is not generating terminal/terminal string. This production is **useless.** So **remove this production**.

* + 1. D  ddd

Here D is generating terminal/terminal string. This production is usefull

G3: S  a**|**aA**|**Aa

A  aB After, removing the variable C and productions based

B  Aa **|**a on that the grammar G2 is changed to G3 as D  ddd

* 1. Now identify the variables which are not reachable from start variable i.e S, with the help of dependency graph. Dependency graph has to be drawn here.

Since D is not reachable from S, it is useless. After eliminating

The variable D and productions based on that the grammar G3 is changed to G4 as G4: S  a**|**aA**|**Aa

Page **42** of **63**

A  aB

B  Aa **|**a The simplified grammar is

G4: S  a**|**aA**|**Aa

A  aB

B  Aa **|**a

L(G4)= {a, aaa,……}

L(G)={a,aaa,…….} L(G)=L(G4)

### Convert the following Context Free Grammar [CFG] grammar into Chomsky normal form [CNF]

S  bA **|** aB

A  bAA**|**aS**|**a B  aBB**|**bS**|**b

###### Solution:

Step-1: There are no null and unit productions. Step-2: P’= {A  a, B b} is in CNF

S bA **|** aB AbAA **|** aS BaBB **|** bS.

Replace the terminals on RHS by variables.

Thus,

S  bA =>S  TbA, where Tb  b S  aB =>S TaB, Where Ta  a A  bAA =>A TbAA, Where Tb  b A  aS =>A  TaS, where Ta  a B  aBB =>B TaBB, where Ta  a B  bS =>B TbS, where Tb  b.

Step-3: P’{

A a, B b,

S  TbA, S  TaB, A  TaS,

B  TbS} are now in CNF.

Step-4: Restrict the number of variables on RHS for production: A  TbAA B  TaBB.

So that,

A  TbAA =>A  TbK1, Where K1  AA.

Page **43** of **63**

B  TaBB =>B  TaK2, Where K2  BB.

Thus, the final Grammar is in Chomsky normal form (CNF) is G = ({S, A, B, Ta, Tb, K1, K2}{a,b},P,S).

P”{

}

S  TbA**|**TaB

A  TbK1**|**TaS**|**a B  TaK2**|**TbS**|**b K1 AA

K2  BB, Ta  a and Tb  b

### Convert the Context Free Grammar (CFG), G=({A1,A2,A3},{a, b}, p, S) into greibach Normal Form (GNF) where p consist of :

### A1A2A3

**A2****A3A1|b A3****A1A2|a**

###### Solution:

1. The grammar is already simplified.
2. The grammar already in CNF form.
3. The variables already renamed.
4. Consider the productions

[A1A2A3 1](#_TOC_250000)

A2A3A1 **|**b 2

A3A1A2**|**a 3

Consider Ai Aj…… & 4 point in procedure For condition **j>i**, leave the production as it is.

For condition **j=i** apply lemma 2. (Elimination of Left Recursion) For condition j<i apply lemma 1. (Substitution Rule)

Where A3 A1A2**|**a, Here j<i, then apply lemma 1(**Substitution Rule**)

So substitute eq1 in eq3 (Substitutions should be made to right hand side Left most variables only) A3 A2A3A2**|**a

Here j<i, apply lemma1

Now Substitute eq 2 in eq 3 A3 A3A1A3A2**|**bA3A2**|**a

Here j=i, apply lemma2 i.e by introducing new variable B3, the productions are

A3 A3A1A3A2**|**bA3A2**|**a

α β1 β2 A3 bA3A2**|**a 6

Page **44** of **63**

A3 bA3A2B3**|**aB3

B3 A1A3A2 7

B3 A1A3A2B3

Now the variable A3 is in GNF

1. Consider eq2

Here j<i, because A3 is in GNF substitute eq6 in eq2, to bring A2 also into GNF A2A3A1**|**b

A2bA3A2A1

A2aA1

A2bA3A2B3A1 8

A2aB3A1

A2b

Here A2 is also in GNF Consider eq1 A1A2A3

Where j>i, because A2 in GNF substitute eq8 in eq1 to bring A1 also into GNF

A1bA3A2A1A3 A1aA1A3

GNF

A1bA3A2B3A1A3 9

A1aB3A1A3

A1bA3

Now to bring B3 also in Substitute eq9 in eq7 B3bA3A2A1A3A3A2 B3aA1A3 A3A2 B3bA3A2B3A1A3 A3A2 B3aB3A1A3 A3A2 B3bA3 A3A2 B3bA3A2A1A3A3A2B3 B3aA1A3 A3A2B3 B3bA3A2B3A1A3 A3A2B3 B3aB3A1A3 A3A2B3 B3bA3 A3A2B3

Now all the variables are including new variable B3 is in GNF

###### Summary:-

Page **45** of **63**

|  |  |  |
| --- | --- | --- |
| A1bA3A2A1A3 | A2bA3A2B3A1 | B3aB3A1A3 A3A2 |
| A1aA1A3 | A2aB3A1 | B3bA3 A3A2 |
| A1bA3A2B3A1A3 | A2b | B3bA3A2A1A3A3A2B3 |
| A1aB3A1A3 | A3 bA3A2|a | B3aA1A3 A3A2B3 |
| A1bA3 | A3 bA3A2B3|aB3 | B3bA3A2B3A1A3 |

A2A3A1|b A2bA3A2A1

A2aA1

B3bA3A2A1A3A3A2 B3aA1A3 A3A2 B3bA3A2B3A1A3 A3A2

A3A2B3

B3aB3A1A3 A3A2B3 B3bA3 A3A2B3

Finally the given grammar is converted into GNF

### Construct a Greibach normal form [GNF] grammar equivalent to the following CFG

###### Solution:

SAA**|**0 ASS**|**1

**Step-1:** Simplify G

There are no useless productions, no epsilon-productions, and no unit productions. The given grammar is a simplified grammar.

**Step-2:** Transform G into equivalent CNF. Given CFG is already in CNF.

**Step-3:** Transform G into equivalent grammar G’ in GNF. Rename the variables in the grammar to A1, A2,…..

Let S=A1; A=A2 A1 A2A2 **|** 0 A2 A1A1**|**1

**Step-4:** Order the productions and remove recursion, where ever necessary.

A1  A2A2**|**0 ………1 No modification needed j > i A2  A1A1**|**1 …........2 Modification needed j < i

Consider equation 2, Here j<i, then apply lemma 1**(Substitution Rule)**

So substitute eq1 in eq2 (Substitutions should be made to right hand side Left most variables only) A2  A2A2A1 **|**0A1 **|**1 3

Here j = i, apply Lemma 2 **(Elimination of Left Recursion)**

A2  A2A2A1 **|** 0A1 **|** 1

Resolving left recursion

A2  A2A2A1 **|** 0A1**|** 1 needs to be resolved.

Introduce a new variable B2 and add new production to B2 Now,

Page **46** of **63**

A2  0A1 **|** 1, A2  0A1B2 | 1B2 4

Now A2 is in GNF.

B2 A2A1, B2 A2A1B2 5

the production rule in a compact form is: A1A2A2**|**0 A20A1B2**|**1B2**|**0A1**|**1 B2A2A1**|**A2A1B2

**Step-5:** to bring A1 also into GNF Substitute eq 4 in eq1

A1A2A2**|**0 needs to be modified A10A1B2A2**|**1B2A2**|** 0A1A2**|** 1A2**|** 0 6

Now A1 is also in GNF form

**Step-6:** to bring B2 also into GNF Substitute eq 4 in eq5

B20A1B2A1**|**1B2A1**|**0A1A1**|**1A1 B20A1B2A1B2**|**1B2A1B2**|**0A1A1B2**|**1A1B2 7

Now B2 is also in GNF form

**Step-7:** Finally, All the productions are in perfect order and the grammar is in GNF: A10A1B2A2**|**1B2A2**|**0A1A2**|**1A2**|**0

A20A1B2**|**1B2**|**0A1**|**1 8

B0A1B2A1**|**1B2A1**|**0A1A1**|**1A1**|**0A1B2A1B2**|**1B2A1B2**|**0A1A1B2**|**1A1B2

### Pushdown Automata

PDA is similar to finite state automata but the difference is it consists of auxiliary stack which provides an unlimited amount of memory.

Context Free Grammar (CFG) is recognised by Pushdown Automaton. PDA is used in different areas like parsing and compiler construction.

### Define PDA? Explain about components and operations of PDA

###### Definition:-

PDA is a way to represent the language class called the Context Free Language (CFL).

PDA is a generalisation of Finite Stat Machine(FSA) and a PDA changes from state to state, reading input symbols .Unlike FSA transitions also update the stack either popping(or)pushing symbols.

###### Elements of PDA:-

A PDA is a 7-tuple specification M= (Q, Σ, Γ, δ, q0, z0, F)

Where

Q=set of finite state

Σ=set of finite input symbols Γ= finite set of stack symbol

q0 ∈ Q set of final state F ⊆ Q set of final state

Page **47** of **63**

Z0= initial stack symbol placed on the top of the stack

### Components of PDA:-

1. Input tape

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | a | b | a | a | ……..... |

1. Read unit **1. Input Tape**



1. Control unit
2. Stack

###### Read unit

δ= is the transition function δ: Q ×(Σ𝖴{ε})×Γ  Q×Γ\*

###### Input tape:-

* 1. **Control unit**

CU

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

###### Stack

It is an infinitely long tape on which input is written.

The tape is divided into sequence of cells. Each cell begins from left end and extends to right without an end. Each cell of the holds one input letter or a blank, epsilon. Input string is written on the tape prior to the beginning of the operation of the PDA

###### Read unit:-

Read unit of PDA reads words from the cells of the input tape, beginning with the first letter in the left most cell and then moves to the right. It cannot go back

###### Control unit:-

It governs the operations of PDA by performing a sequence of transitions between internal states available to it.

The control unit executes a transition, whenever a letter of the input string is provided to it by the read unit. These transitions are determined by transition function of the PDA. Internal states are available to the control unit will have start, accept & reject states.

###### Stack:-

A PDA has an infinitely tall push down stack which has a last in first out discipline.

Stack always start with stack empty

### Operations on stack:-

1. Push: - adds the input alphabet to the top of the stack
2. Pop: - Removes the top input alphabet from the top of the stack. If the stack is empty the basic pop does not changes the state of the stack.
3. No-Operation (NOP):- Does nothing to the stack.

Page **48** of **63**

### Check whether the language is accepted by PDA or not L= {0n12n | n>0}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | ……..... |

q0 q0 q0 q1 q1 q1 q2

###### Transitions:

δ(q0,0,z0)={(q0,0z0)} push

δ(q0,0,0) ={(q0,00)} push

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
| 0 |
| 0 |
| Z0 |

δ(q0,1,0) ={(q1,0)} nop

δ(q1,1,0)={(q2,ε)} pop

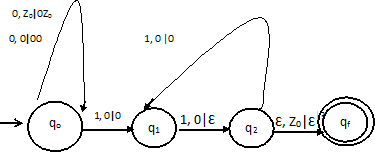
δ(q2,1,0)={(q1,0)}nop

δ(qf, ε, z0)={(qf, ε)} pop Final Stack

Here stack is empty and input string completed. So given language is accepted by PDA. M=({q0,q1,q2,qf},{0,1},{z0,0},δ,q0,z0,{qf})

###### State transition table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Unread input** | **Transition**  [current state, input , Top symbol of stack, operation on stack, new state] | **Stack** | **New State** |
| 001111 | - | Z0 | q0 |
| 01111 | (q0, 0, Z0, Push(0), q0) | 0Z0 | q0 |
| 1111 | (q0, 0, 0, Push(0), q0) | 00 Z0 | q0 |
| 111 | (q0, 1, 0, Nop, q1) | 00 Z0 | q1 |
| 11 | (q1, 1, 0, Pop(a), q2) | 0 Z0 | q2 |
| 1 | (q2, 1, 0, Nop, q1) | 0Z0 | q1 |
| ε | (q1, 1, 0, Pop, q2) | Z0 | q2 |
| - | (q2, ε, Z0, Pop, qf) | - | qf |

**State Diagram:**

Page **49** of **63**

### Design PDA for the Language L={WCWR | w ∈ (a, b)}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Top Plate** | **State** | **Input** | | |
| **0** | **1** | **2** |
| Blue | q1 | Add blue Plate, stay in state q1 | Add green Plate stay in state q1 | Go to state q2 |
| q2 | Remove top Plate stay in q2 | - | - |
| Green | q1 | Add blue plate, Stay in state q1 | Add green plate, stay in state q1 | Go to state q2 |
| q2 | - | Remove top plate stay in state q2 | - |
| Red | q1 | Add blue plate, stay in state q1 | Add Green Plate, stay in state q1 | Go to state q2 |
| q2 | Without waiting for next input remove top plate | | |

Let W=110 WR =011

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | C | 0 | 1 | 1 |

**Transitions:**

q0 q0 q0 q0 q1 q1 q1 q1

(q0, 1, R) = (q0, GR) Push

(q0, 1, G) = (q0, GG) Push

(q0, 0, G) = (q0, BG) Push

(q0, C, B) = (q1, B) Nop

(q1, 0, B) = (q1, ε) pop

(q1, 1, G) = (q1, ε) pop

(q1, ε, R) = (qf, ε) pop

**Other possibilities**

(q0, 1, B) = (q0, GB) Push

(q0, 0, R) = (q0, BR) Push

(q0, 0, B) = (q0, BB) Push

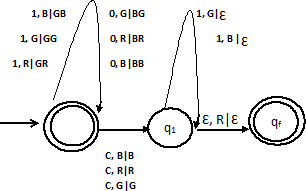
(q0, C, R) = (q1, R) Nop

(q0, C, G) = {(q1, G)} Push

**State Diagram:**

Initially Top of the stack

|  |
| --- |
| B |
| G |
| G |
| R |



### Design a PDA for the language L = { WWR | W ∈ (0+1)\*}from the above table information

M=({q1, q2},{0,1},{R,B,G},δ,q1,R,q2) W=001 WR=100

Page **50** of **63**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 0 | …… |

###### Transitions:

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
| G |
| B |
| B |
| R |

q1 q1 q1 q1 q2 q2 q2

δ (q1,0,R)={(q1,BR)}

δ (q1,0,B)={(q1,BB),(q2,ε)} δ (q1,1,B)={(q1,GB)}

δ (q1,1,G)={ (q2,ε ), (q1,GG)} δ (q2,0,B)={(q2,ε)}

δ (q2,ε,R)={(q2 ,ε)}

δ (q1,0,G)={(q1,BG)}

δ (q1,1,R)={(q1,GR)}

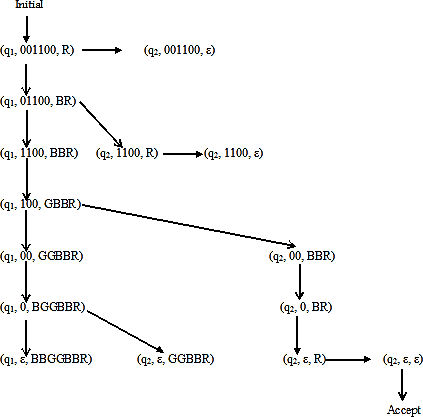
δ (q2,1,G)={(q2,ε)}

δ (q1,ε,R)={(q2,ε)}

Finally stack is empty

So here input is completed and stack also empty. Hence the language L=WWR is accepted by the PDA. And this is nondeterministic PDA

**ID (Instantaneous Description):-** [To verify the working of a PDA at any instance of time, Instantaneous Description is used]



Page **51** of **63**

### Derive leftmost and right most derivations and derivation trees for the input string a= b \* c + d / e for the given Grammar and check whether the grammar is ambiguous or not.

**Definition** − A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple **(V, T, P, S)** where

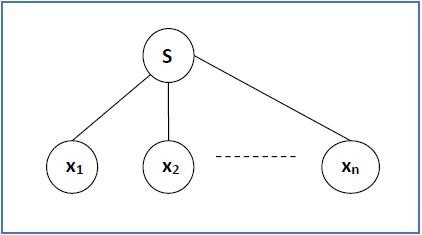
* + **V** is a set of non-terminal symbols.
  + **T** is a set of terminals where **V ∩ T = NULL.**
  + **P** is a set of rules, **P: V → (V** 𝖴 **T)\***, i.e., the left-hand side of the production rule **P** does have any right context or left context.
  + **S** is the start symbol.

**Generation of Derivation Tree**

A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a context-free grammar.

###### Representation Technique

* **Root vertex** − Must be labeled by the start symbol.
* **Vertex** − Labeled by a non-terminal symbol.
* **Leaves** − Labeled by a terminal symbol or ε.

If S → x1x2 …… xn is a production rule in a CFG, then the parse tree / derivation tree will be as follows −

###### Derivation or Yield of a Tree

The derivation or the yield of a parse tree is the final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the Nulls. However, if all the leaves are Null, derivation is Null.

###### Example

Let a CFG {V,T,P,S} be

V = {S}, T = {a, b}, Starting symbol = S, P = S → SS | aSb | ε One derivation from the above CFG is “abaabb”

S → SS → aSbS → abS → abaSb → abaaSbb → abaabb

###### Sentential Form and Partial Derivation Tree

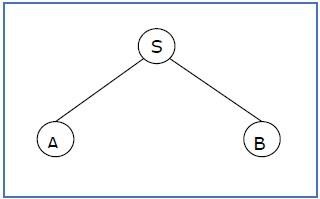
A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

###### Example

Page **52** of **63**

If in any CFG the productions are −

S → AB, A → aaA | ε, B → Bb| ε

the partial derivation tree can be the following −

If a partial derivation tree contains the root S, it is called a **sentential form**. The above sub-tree is also in sentential form.

Leftmost and Rightmost Derivation of a String

* **Leftmost derivation** − A leftmost derivation is obtained by applying production to the leftmost variable in each step.
* **Rightmost derivation** − A rightmost derivation is obtained by applying production to the rightmost variable in each step.

**Derivations:** First let's look at a simple grammar that describes a limited collection of arithmetic expressions.

**E > E + E**

**E > E \* E**

**E > ( E )**

**E ----> a | b | c | ...**

**Grammar: Arith. Exp.**

Here is a sequence of replacements that leads to a sequence of terminal symbols.

###### E ===> E \* E ===> ( E ) \* E ===> ( E + E) \* E ===> ( a + E ) \* E ===> ( a + b ) \* E ===> ( a + b ) \* c

In this case, we obtained the final sentence ( a + b ) \* c starting from E. One says that "E derives ( a

+ b ) \* c" or (to use passive voice) "( a + b ) \* c is derived from E". The sequence is called a derivation sequence. In this case it is a leftmost derivation sequence, because at each stage the leftmost non-terminal was replaced. Each non-terminal that is replaced is colored red above. Sometimes there was only one non- terminal, so it is leftmost on an honorary basis.) Here is another derivation:

###### E ===> E \* E ===> E \* c ===> ( E ) \* c ===> ( E + E ) \* c ===> ( E + b ) \* c ===> ( a + b ) \* c

This was a rightmost derivation sequence, because at each stage the rightmost non-terminal was replaced.

The sentence **( a + b ) \* c** has a unique leftmost derivation, a unique (different) rightmost derivation and the unique parse tree shown below:

**Parse Tree: ( a + b ) \* c**

**E**

**/|\**

**E \* E**

**/|\ \**

**( E ) c**

**/|\**

**E + E**

**| |**

**a b**

Page **53** of **63**

**E**

**/|\**

**/ | \**

**/ | \**

**/ | \**

**/ | \**

|  |  |  |
| --- | --- | --- |
| **E** | **|** | **E** |
| **/|\** | **|** | **|** |
| **/ | \** | **|** | **|** |
| **/ E \** | **|** | **|** |
| **/ /|\ \ |** | | **|** |
| **| / | \ ||** | | **|** |
| **| E | E ||** | | **|** |
| **| | | | ||** | | **|** |
| **( a + b )\*** | | **c** |

**Ambiguity**: There are other sentences derived from E above that have more than one parse tree, and corresponding left- and rightmost derivations. For example, the very simple sentence a + b \* c. The table looks at leftmost derivations and parse trees:

|  |  |
| --- | --- |
| **1st Leftmost Der.** | **2nd Leftmost Der.** |
| **E ===> E + E**  **===> a + E**  **===> a + E \* E**  **===> a + b \* E**  **===> a + b \* c** | **E ===> E \* E**  **===> E + E \* E**  **===> a + E \* E**  **===> a + b \* E**  **===> a + b \* c** |
| **1st Parse Tree** | **2nd Parse Tree** |
| **E**  **/ | \**  **/ | \ E + E**  **| / | \**  **| / | \ a E \* E**  **| |**  **b c** | **E**  **/ | \**  **/ | \ E \* E**  **/ | \ |**  **/ | \ | E + E c**  **| |**  **a b** |

Even if some parse trees are unique, if there are multiple parse trees for any sentence, then the grammar is called **ambiguous**.

Here is a new grammar that is not ambiguous and accepts the same language as the previous grammar.:

**E ----> E + T | T**

**T ----> T \* F | F**

**F ----> ( E ) | a | b | c ...**

**Grammar: Arith. Exp.**

It's can be very hard (or impossible) to prove results such as determining whether or not a grammar is ambiguous. In practice there are usually no problems. With this grammar every sentence has a unique leftmost and rightmost derivation and a unique parse tree. For example, the above sentence **a + b \* c** that caused a problem has the following leftmost derivation and parse tree on the left (along with the its twin **a \* b + c** on the right):

Page **54** of **63**

|  |  |
| --- | --- |
| **Leftmost Derivations** | |
| **a + b \* c** | **a \* b + c** |
| **E ===> E + T** | **E ===> E + T** |
| **===> T + T** | **===> T + T** |
| **===> F + T** | **===> T \* F + T** |
| **===> a + T** | **===> F \* F + T** |
| **===> a + T \* F** | **===> a \* F + T** |
| **===> a + F \* F** | **===> a \* b + T** |
| **===> a + b \* F** | **===> a \* b + F** |
| **===> a + b \* c** | **===> a \* b + c** |
| **1st Parse Tree** | **2nd Parse Tree** |
| **E**  **/|\**  **/ | \**  **E + T**  **| /|\**  **T T \* F**  **| | |**  **F F c**  **| |**  **a b** | **E**  **/|\**  **/ | \**  **E + T**  **| |**  **T F**  **/|\ |**  **/ | \ c T \* F**  **| |**  **F b**  **| a** |

Page **55** of **63**

### UNIT – VI

TURING MACHINE:

Turing machine was proposed by Alen Turing in 1936 which recognizes more languages than pushdown automata.

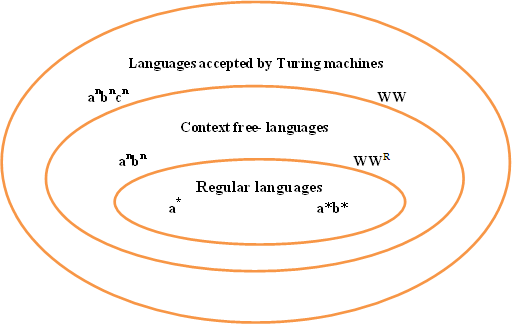
###### Definition 1:

Turing machines are simple abstract computation devices intended to help investigate the extent and limitations of what can be computed.

###### Definition 2:

A Turing machine is a kind of state machine at any time the machine is in anyone of the final number of states. Instructions for a turing machine include the specification of conditions under which the machine will make transitions from one state to another.

Turing Machine can be described by a 7 tuple Me = (Q, Σ, Γ, 𝛿,q0, B, h) where −

* **Q** is a finite set of states.
* ∑ is a finite set of symbols called the input alphabet.
* **Г** is a finite set of symbols called the output alphabet.
* **δ** is the input transition function where δ: Q × ∑ → Q×Γ×{L,R,N} .
* **q0** is the initial state from where any input is processed (q0 ∈ Q).
* **B** is the blank symbol.
* **h** is a halt state.

**Fig : The Language Hierarchy**

### Components of Turing Machine:

A Turing machine is usually described as consisting of the following three components.

1. Input Tape
2. Read/ Write Head
3. Control unit

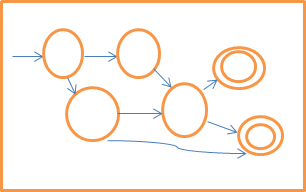
###### Input Tape

Page **56** of **63**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ….. | B | a | b | a | b | a | b | B | ….. |

1. **Read / Write head** (movement in both direction)

###### Control Unit

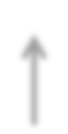
**Fig : Components of a turing machine**

###### Input Tape

A tape is divided into a sequence of numbered cells or sqaures , one next to another . Each cell containsa symbol from some finite alphabet. The alphabet contains a blank symbol(B) . The set of symbols of tape is denoted by Γ.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ….. | B | a | b | a | b | a | b | B | ….. |

Read-write head Blank symbol



###### Read or Write head :

Read or write head moves in two directions (left write).These head reads the data or well as write the data. At each step the head doing:

1. Reads the symbol
2. Writes the symbol
3. moves left or right or doesn’t move.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ….. | B | a | b | a | b | a | b | B | ….. |

###### Control unit :

Head starts at the left most position of the input string

Reading from the tape or writing to the tape by the tape . It Contains set of finite states .

1. Initial states
2. Halt state,it stops all further operations.
3. Other states.

### Design a Turing Machine which is accepted by all Strings of the form anbn.

###### Solutions:

Page **57** of **63**

M = (Q, Σ, Γ, 𝛿, q0, B, h)

({q0, q1, q2, q3, q4}, {a, b, x, y}, 𝛿, q0, B, q4)

X X Y Y

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ………… | B | a | a | b | b | B | ………… |

###### Transition Table:

q0 q1 q1

q2 q2

q0 q1 q1

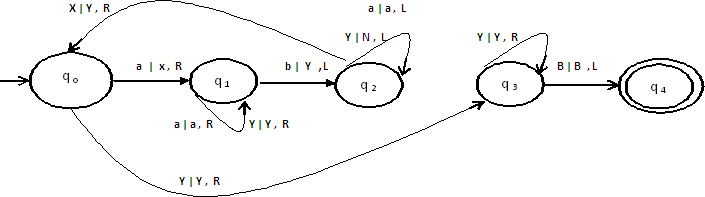
q2 q2

q0 q3 q3

q4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | X | Y | B |
| q0 | (q1,X,R) | - | - | (q3,Y,R) | - |
| q1 | (q1,a,R) | (q2,X,L) | - | (q1,X,R) | - |
| q2 | (q2,a,L) | - | (q0,X,R) | (q2,X,L) | - |
| q3 | - | - | - | (q3,Y,R) | (q4,B,N) |
| q4 | - | - | - | - | - |

###### State Diagram:

Instantaneous Description (ID): q0aabbB ├ Xq1abbB

├ Xaq1bbB

├ Xq2aYbB

├ Xq0YbB

├ XXq1Yb

├ XXYq1bB

├ XXYYq2B

Page **58** of **63**

├ XXYq2YB

├ XXq2YYB

├ XXq0YYB

├ XXYq3YB

├ XXYYq3B

├ XXYYq4

### Design a Turing Machine which accepts all Strings of the form anbncn for n>=1.

###### Solution:

M = (Q, Σ, Γ, 𝛿,q0, B, h)

({q0, q1, q2, q3, q4, q5, q6}, {a, b, c, X, Y, Z}, 𝛿, q0,B, q6)

X X Y Y Z Z

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ……. | B | a | a | b | b | c | c | B | ……. |

###### Transition Table:

q0 q1 q1 q2 q2 q3 q3 q3 q3

q0 q1 q1 q2 q2

q3 q3 q3 q3

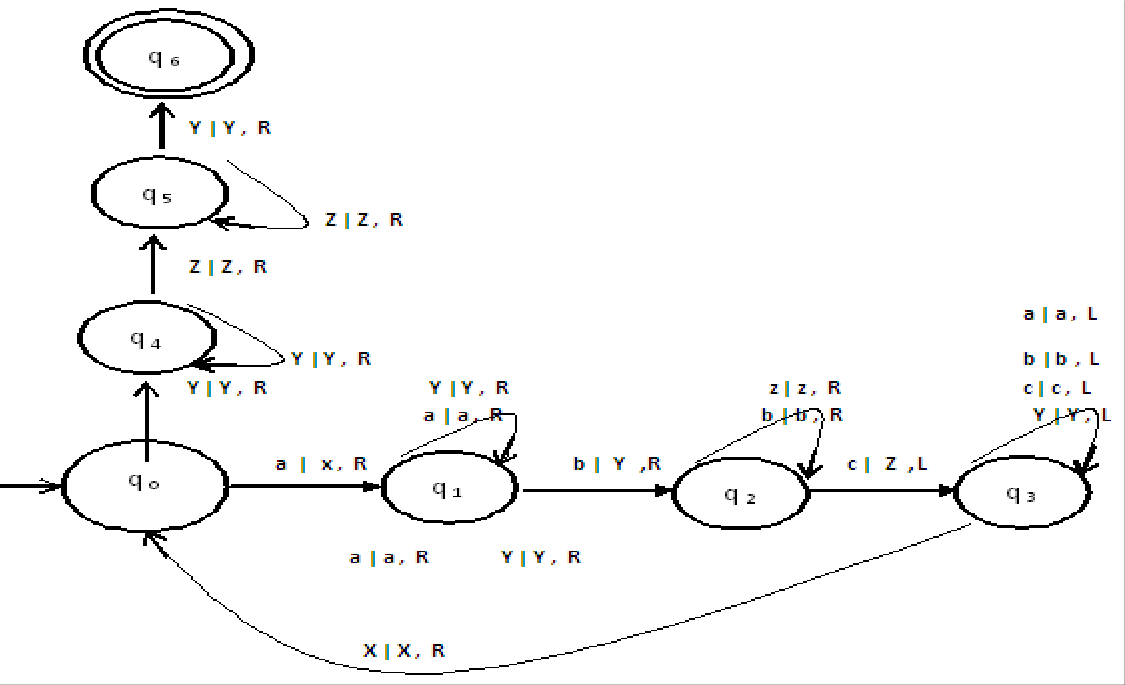
q0 q4 q4 q5 q5

q6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | X | Y | Z | B |
| q0 | (q1,X,R) | - | - | - | (q4,Y,R) | - | - |
| q1 | (q1,a,R) | (q2,Y,R) | - | - | (q1,Y,R) | - | - |
| q2 | - | (q2,b,R) | (q3,Z,L) | - | - | (q2,Z,R) | - |
| q3 | (q3,a,L) | (q3,b,L) | - | (q0,X,R) | (q3,Y,L) | (q3,Z,L) | - |
| q4 | - | - | - | - | (q4,Y,R) | (q5,Z,R) | - |
| q5 | - | - | - | - |  | (q5,Z,R) | (q6,B,R) |
| q6 | - | - | - | - | - | - | - |

Page **59** of **63**

###### State Diagram:



Instantaneous Description: q0aabbccB

q0aabbccB ├ Xq1abbccB

├ Xaq1bbccB

├ XaYq2bccB

├ XaYbq2ccB

├ XaYq3bZcB

├ Xaq3YbZcB

├ Xq3aYbZcB

├ Xq0aYbZcB

├ XXq1YbZcB

├ XXYq1bZcB

├ XXYq1bZcB

├ XXYYq2ZcB

├ XXYYZq2cB

├ XXYYq2ZZB

├ XXYq3YZZB

├ XXq3YYZZB

├ XXq0YYZZB

├ XXYq4YZZB

Page **60** of **63**

├ XXYYq4ZZB

├ XXYYZq5ZB

├ XXYYZZq5B

├ XXYYZZq6

### Design a Turing Machine that copies given string over ∑={a,b}.Find the computation of TM for the string abb.

###### Solution:

a b b

x y y a b b

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ……. | B | a | b | b | B | B | B | B | ……. |

q0 q1 q2 q2 q2 q4

q6 q6 q6 q6

q1 q3 q3 q5 q5

q7 q7 q7 q7

q1 q3 q5 q5 q5 q7 q7 q7 q7 q1

h q8 q8 q8 q8

###### Transition Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | X | Y | B |
| q0 |  |  |  |  | (q1,B,R) |
| q1 | (q2,X,R) | (q3,Y,R) |  |  | (q8,B,L) |
| q2 | (q2,a,R) | (q2,b,R) |  |  | (q4,B,R) |
| q3 | (q3,a,R) | (q3,b,R) |  |  | (q5,B,R) |
| q4 | (q4,a,R) | (q4,b,R) |  |  | (q6,a,L) |
| q5 | (q5,a,R) | (q5,b,R) |  |  | (q7,b,L) |
| q6 | (q6,a,L) | (q6,b,L) | (q1,X,R) | (q1,Y,R) | (q6,B,L) |
| q7 | (q7,a,L) | (q7,b,L) | (q1,X,R) | (q1,Y,R) | (q7,B,L) |
| q8 |  |  | (q8,a,L) | (q8,b,L) | (h, B, N) |

Page **61** of **63**

###### State Diagram:

**Instantaneous Description**:

├q0BabbBBBBB

├Bq1abbBBBBB

├BXq2bbBBBBB

├BXbq2bBBBBB

├BXbbq2BBBBB

├BXbbBq4BBBB

├BXbbBq6aBBB

├BXbbq7BaBBB

├BXbq7bBaBBB

├BXq7bbBaBBB

├BXq1bbBaBBB

├BXYq3bBaBBB

├BXYbq3BaBBB

├BXYbBq5aBBB

├BXYbBaq5BBB

Page **62** of **63**

├BXYbBaq6bBB

├BXYbBq6abBB

├BXYb q7BabBB

├BXYq7bBabBB

├BXYq1bBabBB

├BXYYq3BabBB

├BXYYBq5abBB

├BXYYBaq5bBB

├BXYYBabq5BB

├BXYYBabq6bB

├BXYYBaq6bbB

├BXYYBq6abbB

├BXYYq7BabbB

├BXYYq1BabbB

├BXYYq8BabbB

├BXYq8bBabbB

├BXq8bbBabbB

├Bq8abbBabbB

├q8BabbBabbB

├hq8BabbBabbB

Page **63** of **63**