A Practical File
of
Theory of Computation
(CD501)

Department of Computer Science Engineering

Submitted by

Name:_____

Enrollment No:_____

Semester:_____

Branch: _____

Session:_____

Department of Computer Science Engineering (Data Science)

Baderia Global Institute of Engineering and Management, Jabalpur (M.P.)

Global Square, Patan Bypass, Raigwan, Jabalpur, M.P., 482002

Theory of Computation (CD501)

Contents

Vision and Mission of the Institute

Vision

Transforming life by providing professional education with excellence.

Mission

1. *Quality Education*: Providing Education with quality and shaping up Technocrats and building managers with a focus on adapting to changing technologies.
2. *Focused Research & Innovation*: Focusing on Research and Development and fostering Innovation among the academic community of the Institution.
3. *People Focused*: Accountable and committed to institutional operations for effective functioning by Faculty members, Staff and Students.
4. *Holistic Learning*: Focus on conceptual learning with practical experience an experimental learning with strong Industrial connections and collaborations.
5. *Service to Society*: Providing Technical and Managerial services to society for betterment of their quality of life with best of the skills, compassion and empathy.

Vision and Mission of the Department

Vision

Cultivating leaders in Data Science who drive innovation and informed decision-making through the power of data analytics and scientific discovery.

Mission

The program strives to:

- Foster a student-centric learning environment that equips graduates with cutting-edge knowledge and skills in data science, analytics, and related technologies.
- Develop professionals who can apply data-driven approaches to solve complex problems across diverse industries, with a commitment to ethical standards and societal well-being.
- Continuously enhance faculty expertise through ongoing training, ensuring the delivery of high-quality education and mentorship.
- Promote collaboration between academia and industry through research and consultancy projects, providing students with practical experiences and opportunities for conceptual learning.

Program Education Objectives

1. To impart in depth concepts of the subject in terms of both theoretical and practical aspects to achieve excellent University Outputs.
2. To produce technically sound engineering graduates who would fulfill the latest requirements of computer science and IT industry at modest cost with the calibre of solving intricacies of deeper programming concepts.
3. To inculcate the lessons of communication skills, teamwork spirit and professional attitude to the budding engineering graduates.
4. In order to get versatile growth of the students, participation of students in extracurricular activities is also made compulsory; and also to develop ethical attitudes in the graduates so that they can also become good citizens of the nation.
5. To develop leadership and entrepreneurship qualities in budding graduates so that they will cherish and nourish the society and the nation with modern trends of digitization and blossom it with their unparalleled technical expertise.

Program Outcomes

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

Graduates will be able to-

PSO 1: Proficiency in Computer Systems: Capability to understand, analyze and develop computer programs in the areas related algorithms, system software, multimedia, web design, cloud computing and networking enable designing of computer systems that are efficient and support the degree of complexity.

PSO 2: Proficiency in Software Development: Ability to understand the structure and methodologies of software systems with keen knowledge about software design process and practical expertise in the programming languages and the open system platform.

PSO 3: Proficiency in Mathematics and science concepts: Capable of applying mathematics and science concepts to solve computation tasks and model real world problems using suitable algorithms and data structures.

PSO 4: Successful career as a Computer Science Engineer: Ability to employ themselves as computer professionals, to be an entrepreneur having innovative ideas and sound knowledge in various domains leads to enthusiasm in higher studies and research.

Course Outcomes

After Completing the course student should be able to:

CO1 – Explain the basic concepts of switching and finite automata theory & languages**.**

CO2 – Relate practical problems to languages, automata, computability and complexity.

CO3 –Construct abstract models of computing and check their power to recognize the languages.

CO4 – Analyze the grammar, its types, simplification and normal form. .

CO5 – Interpret rigorously formal mathematical methods to prove properties of languages, grammars and automata.

CO6 - Develop an overview of how automata theory, languages and computation are applicable in engineering application.

Laboratory Regulations and Safety Rules

The following Regulations and Safety Rules must be observed in all concerned laboratory locations.

1. It is the duty of all concerned parties who use any electrical laboratory to take all reasonable steps to safeguard the HEALTH and SAFETY of themselves and all other users and visitors.
2. Make sure that all equipment is properly working before using for laboratory exercises. Any defective equipment must be reported immediately to the Lab. Instructors or Lab. Technical Staff.
3. Students are allowed to use only the equipment provided in the experiment manual or equipment used for senior project laboratory.
4. Power supply terminals connected to any circuit are only energized with the presence of the Instructor or Lab. Staff.
5. Students should keep a safe distance from the circuit breakers, electric circuits or any moving parts during the experiment.
6. Avoid any part of your body to be connected to the energized circuit and ground.
7. Switch off the equipment and disconnect the power supplies from the circuit before leaving the laboratory.
8. Observe cleanliness and proper laboratory housekeeping of the equipment and other related accessories.
9. Wear proper clothes and safety gloves or goggles required in working areas that involves fabrications of printed circuit boards, chemicals process control system, antenna communication equipment and laser facility laboratories.
10. Double check your circuit connections specifically in handling electrical power machines, AC motors and generators before switching “ON” the power supply.
11. Make sure that the last connection to be made in your circuit is the power supply and first thing to be disconnected is also the power supply.
12. Equipment should not be removed, transferred to any location without permission from the laboratory staff.
13. Software installation in any computer laboratory is not allowed without the permission from the Laboratory Staff.
14. Computer games are strictly prohibited in the computer laboratory.
15. Students are not allowed to use any equipment without proper orientation and actual hands on equipment operation.

INDEX

S. No.	Experiment	Date of Completion	Sign.
1.	Design an Algorithm to convert NDFA to DFA.		
2.	Write a C program to construct a DFA with ∑ = {0, 1} that accepts the languages ending with "01" over the characters {0, 1} –		
3.	Design a Program for creating machine that accepts the string always ending with 101.		
4.	Design a program for accepting decimal number divisible by 2.		
5.	Design a Program to find 2's complement of a given binary number.		
6.	Design a PDA machine that accepts the well-formed parenthesis.		
7.	Design a non deterministic PDA for accepting the language L = {ww^R w ∈ (a, b)$^+$}, i.e.,		
8.	Design a Turing machine that's accepts the following language $0^n1^n2^n$ where n>=1		

Experiment 1: Design an Algorithm to convert NDFA to DFA.

Let, M = (Q, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', ∑', q0', δ', F') such that L(M) = L(M').

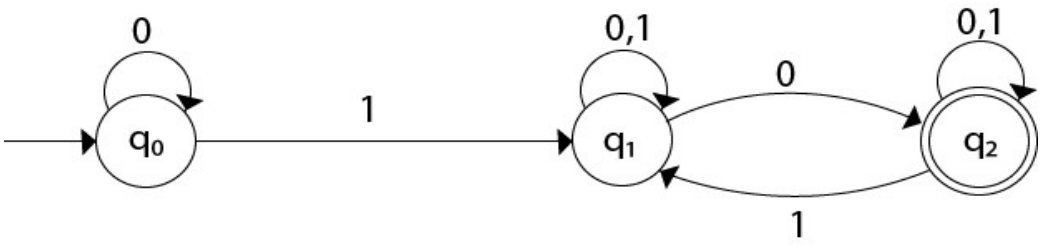
Step 1: Initially Q' = ϕ

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

Step 4: In DFA, the final state will be all the states which contain F(final states of NFA)

Example: Convert the given NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

The transition table for the constructed DFA will be:

State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q2]	[q2]	[q1, q2]
*[q1, q2]	[q1, q2]	[q1, q2]

The Transition diagram will be:

Output:

Experiment 2: Write a 'C' program to construct a DFA with ∑ = {0, 1} that accepts the languages ending with "01" over the characters {0, 1} –

```
#include<stdio.h>
#include<conio.h>
#define max 100
main() {
char str[max],f='a';
  int i;
  printf("enter the string to be checked: ");
  scanf("%s",str);
  for(i=0;str[i]!='\0';i++) {
    switch(f) {
      case 'a': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='a';
      break;
      case 'b': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='c';
      break;
      case 'c': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='a';
      break;
    }
  }
  if(f=='c')
    printf("\nString is accepted", f);
  else printf("\nString is not accepted", f);
    getch();
}
```

Output:

Experiment 3: Design a Program for creating machine that accepts the string always ending with 101

```
#include<stdio.h>
#include<conio.h>
#define max 100
main() {
  char str[max],f='a';
  int i;
  printf("enter the string to be checked: ");
  scanf("%s",str);
  for(i=0;str[i]!='\0';i++) {
    switch(f) {
      case 'a': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='a';
      break;
      case 'b': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='c';
      break;
      case 'c': if(str[i]=='0') f='b';
        else if(str[i]=='1') f='a';
      break;
    }
  }
  if(f=='c')
    printf("\nString is accepted", f);
  else printf("\nString is not accepted", f);
    getch();
}
```

Output:

Experiment 4: Design a program for accepting decimal number divisible by 2.

```
#include<stdio.h>
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
printf("Enter the number");
scanf("%d",&i);
if(i%2= =0)
{
printf("The number is divisible by 2");
}
else
{
printf("The number is not divisible by 2");;
}
getch();
}
```

Output:

Experiment 5: Design a Program to find 2's complement of a given binary number

```
#include<stdio.h>
#include<conio.h>
#define SIZE 8
int main(){
  int i, carry = 1;
  char num[SIZE + 1], one[SIZE + 1], two[SIZE + 1];
  printf("Enter the binary number\n");
  gets(num);
  for(i = 0; i < SIZE; i++){
    if(num[i] == '0'){
      one[i] = '1';
    }
    else if(num[i] == '1'){
      one[i] = '0';
    }
  }
  one[SIZE] = '\0';
  printf("Ones' complement of binary number %s is %s\n",num, one);
  for(i = SIZE - 1; i >= 0; i--){
    if(one[i] == '1' && carry == 1){
      two[i] = '0';
    }
    else if(one[i] == '0' && carry == 1){
      two[i] = '1';
      carry = 0;
    }
    else{
      two[i] = one[i];
    }
  }
  two[SIZE] = '\0';
  printf("Two's complement of binary number %s is %s\n",num, two);
  getch();
}
```

Output:

Experiment 6: Design a PDA machine that accepts the well-formed parenthesis.

Pushdown Automata (PDA) are the finite automata (FAs), but with the ability to push and pop symbols to/from a stack.

PDA accepts strings if there is a legal path from start state to acceptance state for input. Otherwise, the string is rejected.

A PDA can be represented by a 7-tuple

(Q, ∑, 𝕋, q_0, ha, Δ,δ)

Where

The PDA is to finite subsets of Q × (𝕋 ∪ {Δ})*.

Parentheses are balanced if

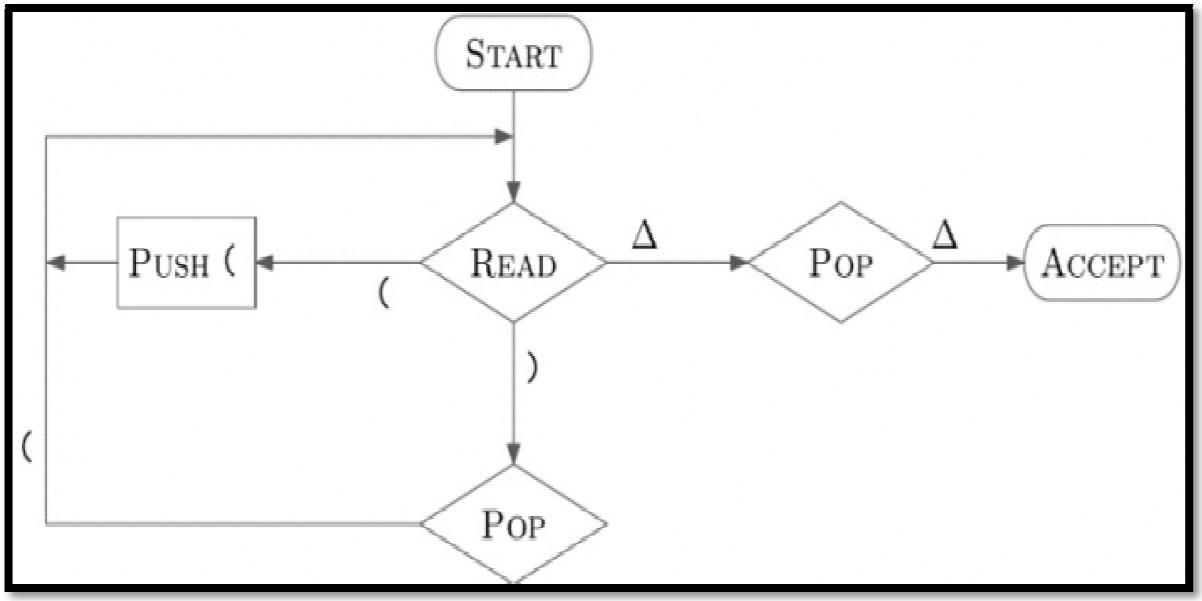
- While reading string, number of opening parentheses >= number of closing parentheses.
- When string is read, number of opening parentheses = number of closing parentheses.

Examples

- (())() − Balanced
- (((() − Not balanced
-)()(() − Not balanced

PDA for balancing brackets

Each (is pushed, each) causes an (to be popped). This is shown below



Experiment 7: Design a non deterministic PDA for accepting the language L = {wwR w ∈ (a, b)$^+$}, i.e.,

L = {aa, bb, abba, aabbaa, abaaba, ...}

Solution :

In this type of input string, one input has more than one transition states, hence it is called non-deterministic PDA, and the input string contain any order of 'a' and 'b'. Each input alphabet has more than one possibility to move next state. And finally when the stack is empty then the string is accepted by the NPDA. In this NPDA we used some symbols which are given below:

Γ = { a, b, z }

Where, Γ = set of all the stack alphabet
z = stack start symbol
a = input alphabet
b = input alphabet

The approach used in the construction of PDA –

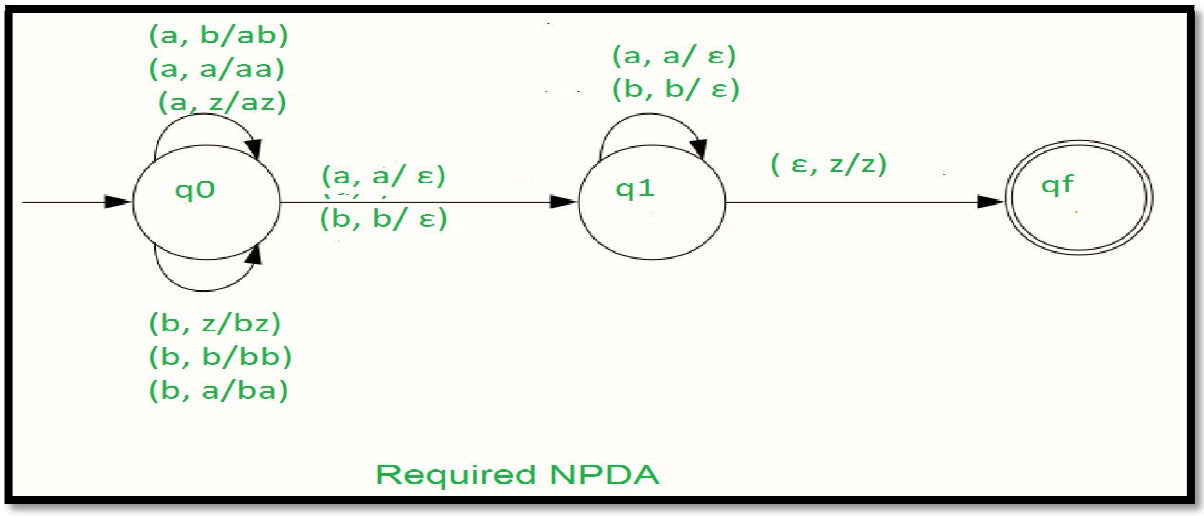
As we want to design an NPDA, thus every times 'a' or 'b' comes then either push into the stack or move into the next state. It is dependent on a string. When we see the input alphabet which is equal to the top of the stack then that time pop operation applies on the stack and move to the next step.

So, in the end, if the stack becomes empty then we can say that the string is accepted by the PDA.

STACK Transition Function

$$\delta(q0, a, z) \vdash (q0, az)\delta(q0, a, a) \vdash (q0, aa)\delta(q0, b, z)$$
$$\vdash (q0, bz)\delta(q0, b, b) \vdash (q0, bb)\delta(q0, a, b) \vdash (q0, ab)\delta(q0, b, a)$$
$$\vdash (q0, ba)\delta(q0, a, a) \vdash (q1, \in)\delta(q0, b, b) \vdash (q1, \in)\delta(q1, a, a)$$
$$\vdash (q1, \in)\delta(q1, b, b) \vdash (q1, \in)\delta(q1, \in, z) \vdash (qf, z)$$

Where, q0 = Initial state
qf = Final state
∈ = indicates pop operation



So, this is our required non-deterministic PDA for accepting the language L = {wwR w ∈ (a, b)*}

Example:
We will take one input string: "abbbba".

- Scan string from left to right
- The first input is 'a' and follows the rule:
- on input 'a' and STACK alphabet Z, push the two 'a's into STACK as: (a, Z/aZ) and state will be q0
- on input 'b' and STACK alphabet 'a', push the 'b' into STACK as: (b, a/ba) and state will be q0
- on input 'b' and STACK alphabet 'b', push the 'b' into STACK as: (b, b/bb) and state will be q0
- on input 'b' and STACK alphabet 'b' (state is q1), pop one 'b' from STACK as: (b, b/∈) and state will be q1
- on input 'b' and STACK alphabet 'b' (state is q1), pop one 'b' from STACK as: (b, b/∈) and state will be q1
- on input 'a' and STACK alphabet 'a' and state q1, pop one 'a' from STACK as: (a, a/∈) and state will remain q1
- on input ∈ and STACK alphabet Z, go to the final state(qf) as : (∈, Z/Z)

So, at the end the stack becomes empty then we can say that the string is accepted by the PDA.

NOTE: ***This DPDA will not accept the empty language.***
Problem:
Design a deterministic PDA for accepting the language L = { wcwR w ∈ (a, b)*}, i.e.,

{aca, bcb, abcba, abacaba, aacaa, bbcbb, ...}

In each string, the substring which is present on the left side of c is the reverse of the substring which is the present right side of c.

Explanation:
Here we need to maintain string in such a way that, the substring which is present on the left side of c is exactly the reverse substring which is the right side of c. For doing this we used a stack. In string 'a' and 'b' are present any order and 'c' come only one time. When 'c' comes then the pop operation is started into the stack. And when a stack is empty then language is accepted.

$\Gamma = \{a, b, z\}$

Where, Γ = set of all the stack alphabet
z = stack start symbol
a = input alphabet
b = input alphabet

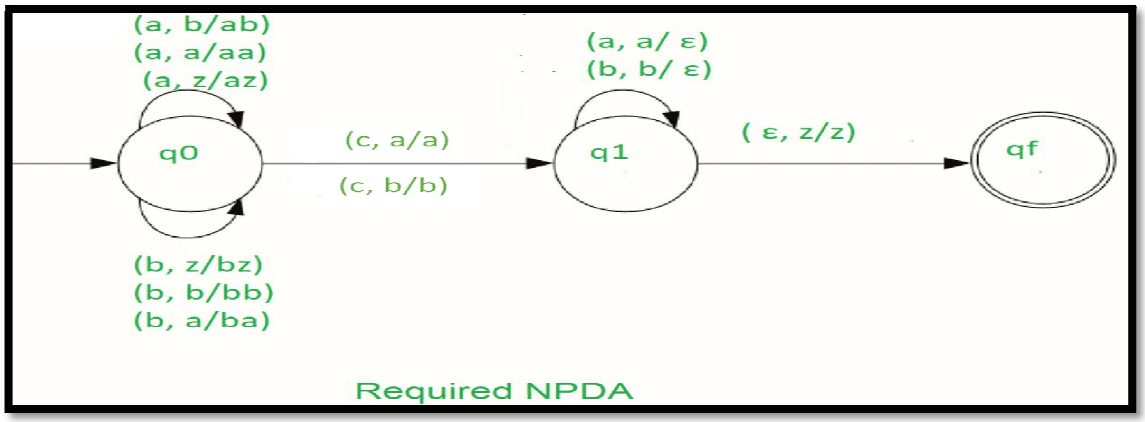
The approach used in the construction of PDA:
As we want to design PDA In every time when 'a' or 'b' comes we push into the stack and stay on the same state q0. And when 'c' comes then we move to the next state q1 without pushing 'c' into the stack. And after when comes an input which is the same as the top of the stack then pop from the stack and stay on the same state. POP operation is performed until the input string is ended. Finally when the input is ∈, then move to the final state qf.
When if the stack will become empty then the language is accepted.

Where, q0 = Initial state
qf = Final state
z = stack start symbol
∈= indicates pop operation

Stack transition functions:

$$\delta(q0, a, z) \vdash (q0, az)\delta(q0, a, a) \vdash (q0, aa)\delta(q0, b, z)$$
$$\vdash (q0, bz)\delta(q0, b, b) \vdash (q0, bb)\delta(q0, a, b) \vdash (q0, ab)\delta(q0, b, a)$$
$$\vdash (q0, ba)\delta(q0, c, a) \vdash (q1, a)\delta(q0, c, b) \vdash (q1, b)\delta(q1, a, a)$$
$$\vdash (q1, \in)\delta(q1, b, b) \vdash (q1, \in)\ \delta(q1, \in, z) \vdash (qf, z)$$

Where, q0 = Initial state
qf = Final state
∈ = indicates pop operation



So, this is our required deterministic PDA for accepting the language,

L = { wcwR w ∈ (a, b)*}

Output for Design of PDA for L = {WW^R, where w ∈ $(a, b)^+$}

Experiment 8: Design a Turing machine that's accepts the following language $0^n 1^n 2^n$ where n>=1

The language L = $\{0^n1^n2^n \mid n\geq1\}$ represents a kind of language where we use only 3 character, i.e., 0, 1 and 2. In the beginning language has some number of 0's followed by equal number of 1's and then followed by equal number of 2's. Any such string which falls in this category will be accepted by this language. The beginning and end of string is marked by $ sign.

Example:

Input : 0 0 1 1 2 2

Output : Accepted

Input : 0 0 0 1 1 1 2 2 2 2

Output : Not accepted

Assumption: We will replace 0 by X, 1 by Y and 2 by Z

Approach used –

First replace a 0 from front by X, then keep moving right till you find a 1 and replace this 1 by Y. Again, keep moving right till you find a 2, replace it by Z and move left. Now keep moving left till you find a X. When you find it, move a right, then follow the same procedure as above.

A condition comes when you find a X immediately followed by a Y. At this point we keep moving right and keep on checking that all 1's and 2's have been converted to Y and Z. If not then string is not accepted. If we reach $ then string is accepted.

- **Step-1:**

Replace 0 by X and move right, Go to state Q1.

- **Step-2:**

Replace 0 by 0 and move right, Remain on same state
Replace Y by Y and move right, Remain on same state
Replace 1 by Y and move right, go to state Q2.

- **Step-3:**

Replace 1 by 1 and move right, Remain on same state
Replace Z by Z and move right, Remain on same state
Replace 2 by Z and move right, go to state Q3.

- **Step-4:**

Replace 1 by 1 and move left, Remain on same state
Replace 0 by 0 and move left, Remain on same state
Replace Z by Z and move left, Remain on same state

Replace Y by Y and move left, Remain on same state
Replace X by X and move right, go to state Q0.

- **Step-5:**

If symbol is Y replace it by Y and move right and Go to state Q4
Else go to step 1

- **Step-6:**

Replace Z by Z and move right, Remain on same state
Replace Y by Y and move right, Remain on same state
If symbol is $ replace it by $ and move left, STRING IS ACCEPTED, GO TO FINAL STATE Q5

Output for design of Turing Machine for L= {$0^n 1^n 2^n$} where n>=1