Aaiza Irfan

SP20-BCS-001

**THEORY OF AUTOMATA**

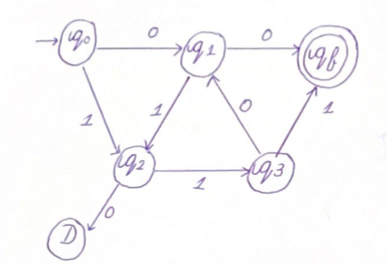
**ASSIGNMENT #1**

**Question #1:**

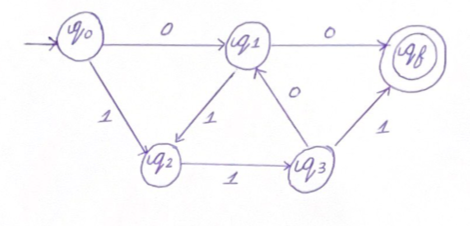
**PART A:**

**Language of all strings having double 0s or triple 1s over alphabet {0,1}**

DFA:



NFA:



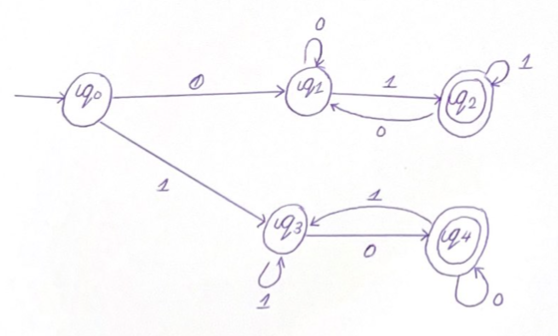
Regular Expression:

(00)1\*0\* + (111)0\*1\*

0\*1\* (00 + 111)

**Language of strings starting and ending with different letters over the alphabet {0,1}**

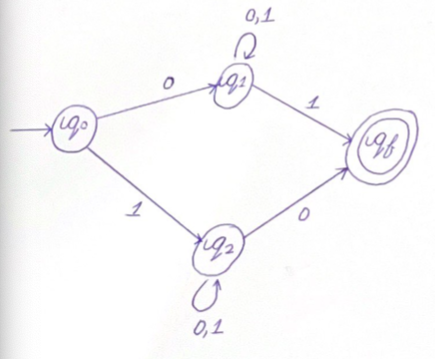
DFA:



Regular Expression for DFA:

(0+(10)\*1+) + (1+(10)\*0+)

NFA:

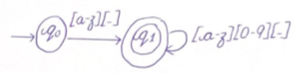


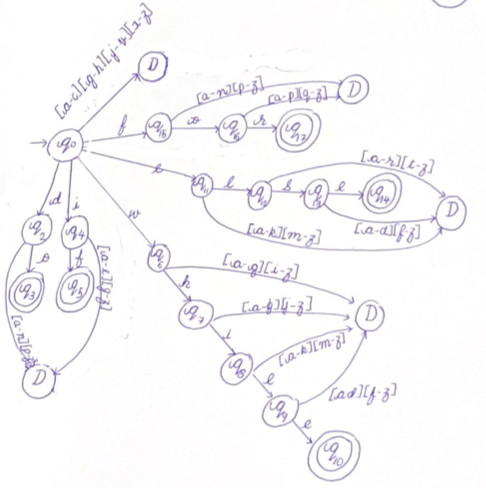
Regular Expression for NFA:

(0(0+1)\*1) + (1(0+1)\*0)

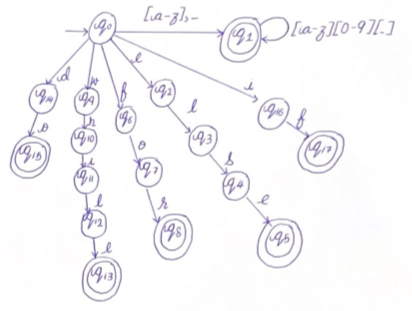
**Language that accepts all user defined variables/identifiers and any 5 special keywords of the C language over alphabet {a,b,c…z,0,1,….}**

DFA:





NFA:

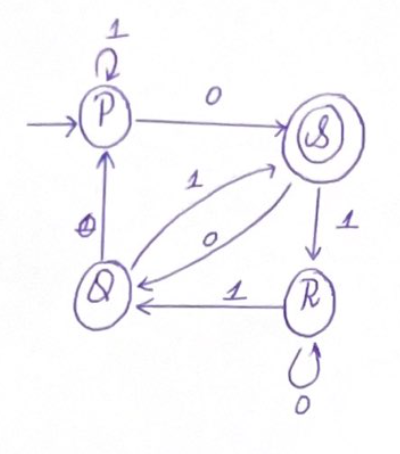


Regular Expression:

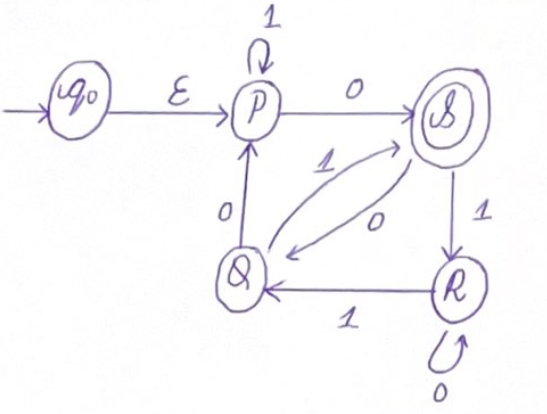
(([a-z]+\_ )([a-z]+[0-9]+[ \_ ])\*) + (if) + (else) + (while) + (for) + (do)

**PART B:**

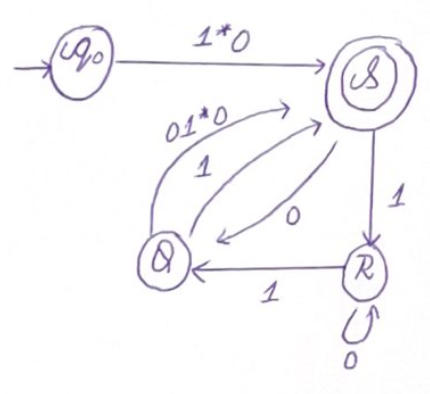
DFA for the given table:



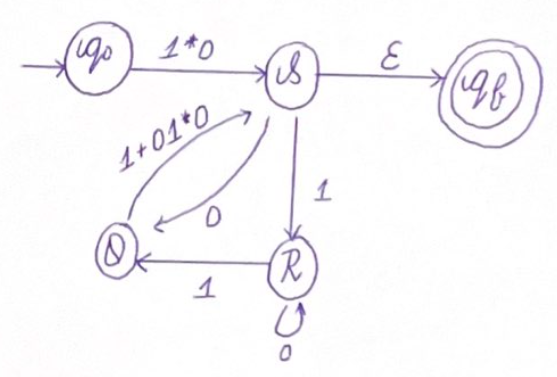
To deal with the incoming edge for the initial state, we will make a new initial state



We will eliminate state P



To deal with outgoing edge for final state, we will add a new final state



We will eliminate state R

Diagram, text

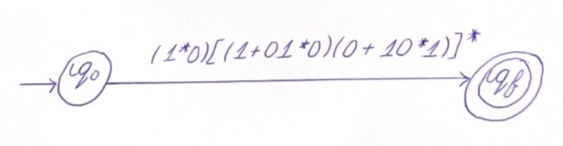
Description automatically generated

We will eliminate state Q

Text, letter

Description automatically generated

We will eliminate state S, leaving us with a direct path from the initial state to the final state



So, the regular expression we get is

(1\*0)[(1+01\*0)(0+10\*1)]\*

**PART C:**

Based on the regular expression (ɛ+0)(0\*10)\*(ɛ+11)

We get the following NFA-epsilon

**A picture containing text, whiteboard

Description automatically generated**

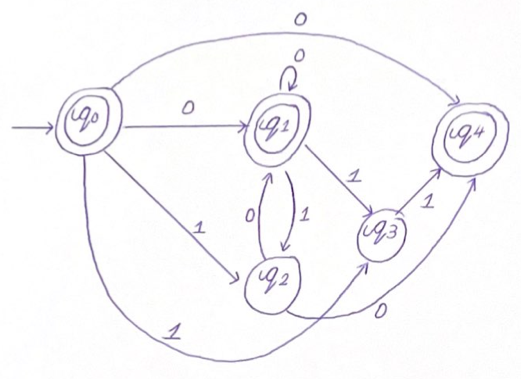
Null closure:

|  |  |
| --- | --- |
| q0 | {q0,q1,q4} |
| q1 | {q1,q4} |
| q2 | {q2} |
| q3 | {q3} |
| q4 | {q4} |

Based on the null closure we get the following state table for NFA:

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| **q0** | q1,q4 | q2,q3 |
| **q1** | q1 | q2,q3 |
| **q2** | q1,q4 | - |
| **q3** | - | q4 |
| **q4** | - | - |

NFA:

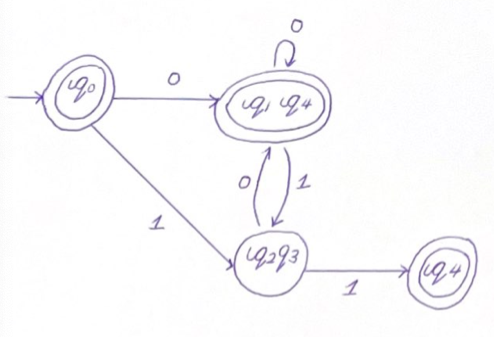


Based on the NFA, we can make the following state table for DFA:

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| **q0** | q1q4 | q2q3 |
| **q1q4** | q1q4 | q2q3 |
| **q2q3** | q1q4 | q4 |

So we get the following DFA diagram

DFA:



**Question #2:**

**Answer the following questions. You have to present these questions in your class. The supplementary material for preparing the answers will be shared with you.**

1. **How do computer registers store values? Can we use state diagrams to demonstrate their behavior?**

Registers are a type of memory in the computer which accepts, stores, and transfers data or instructions. Register uses flip flops to store values, where each flip flop can store a single bit. The composition of flip flops in registers, help store values in form of multiple bits.

Register’s behavior can be demonstrated through state diagrams. Depending on the type of register we are dealing with, we can find out the state, and the output. The state is based on the value of the flip-flops, and how the state changes are found out by effect of the input of the values of the flip-flops.

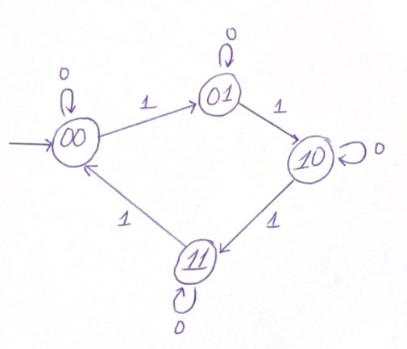
1. **Explain the working of 2-bit and 3-bit counters along with their DFAs. Also mention their use in computer electronics.**

Counters are examples of registers, where the main function is to produce a specified output pattern sequence.

**2-bit counter:**

Each time input 1 is given the value changes from 0 to 3 before it returns back to 0. The values of 0 and 3 are represented in binary.

The following is the state diagram for 2-bit counter



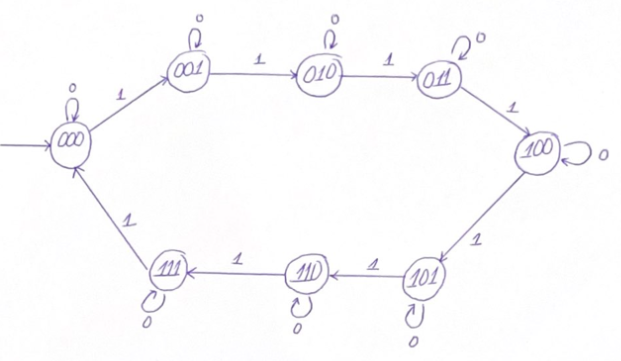
The following is the state transition table

|  |  |
| --- | --- |
| **Current State** | **Next State** |
| 00 | 01 |
| 01 | 10 |
| 10 | 11 |
| 11 | 00 |

**3-bit counter:**

The 3-bit counter deals with a total of 8 states (23) and consists of T flip-flops

The following is the state diagram for the 3-bit counter.



The following is the state transition table

|  |  |
| --- | --- |
| **Current State** | **Next State** |
| 000 | 001 |
| 001 | 010 |
| 010 | 011 |
| 011 | 100 |
| 100 | 101 |
| 101 | 110 |
| 110 | 111 |
| 111 | 000 |

**Use in cs:**

* Used for counting
* Measuring time and frequency
* To increment memory addresses

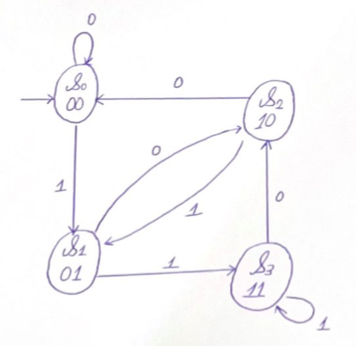
1. **Explain the working of 2-bit and 3-bit shift registers along with their DFAs. Also mention their use in computer electronics.**

The shift register is capable of shifting its binary contents in one or both directions. It shifts its contents a bit each clock plus. It uses a cascade of flip flops where the output of the flip flop is the input of the next flip flop.

An n-bit shift register is formed from n flip-flops, where each flip-flop stores one bit

**2-bit shift register:**

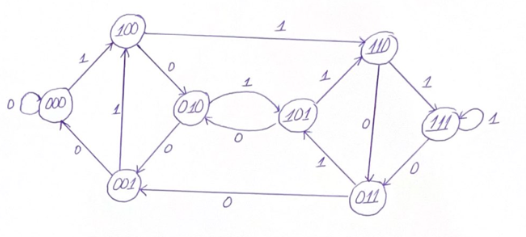
The following is the state diagram for the 2-bit shift register

****

|  |  |  |
| --- | --- | --- |
| **output** | **0** | **1** |
| 00 | 00 | 01 |
| 01 | 10 | 11 |
| 10 | 00 | 01 |
| 11 | 10 | 11 |

**3-bit shift register:**

The following is the state diagram for the 2-bit shift register

****

|  |  |  |
| --- | --- | --- |
| **output** | **0** | **1** |
| 000 | 000 | 100 |
| 001 | 000 | 100 |
| 010 | 001 | 101 |
| 011 | 001 | 101 |
| 100 | 010 | 110 |
| 101 | 010 | 110 |
| 110 | 011 | 111 |
| 111 | 011 | 111 |

**Use in CS:**

Since shift registers are used for data movement or data storage their use in computer science is inside calculators. To store two numbers before they are added or to convert data from serial to parallel and from parallel to serial.

1. **Explain the working of any Vending Machine along with its DFA with output.**

The vending machine I used, is one where you need 75 cents to dispense the object from the machine. The machine only takes a quarter (25 cents) or a dollar (100 cents). In case a person puts a dollar in the machine, along with the object being dispensed there is also 25 cents/ 1 quarter returned as change. On the other hand, if the person puts 3 quarters, there will be no change and only the object will be dispensed.

Based on this vending machine we get the following state table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Current State** | **Input** | | **Next State** | **Output** | |
| **Quarter** | **Dollar** | **Dispense** | **Change**  **(Quarter)** |
| wait | 0 | 0 | wait | 0 | 0 |
| wait | 0 | 1 | wait | 1 | 1 |
| wait | 1 | 0 | 25 cents | 0 | 0 |
| 25 cents | 0 | - | 25 cents | 0 | 0 |
| 25 cents | 1 | - | 50 cents | 0 | 0 |
| 50 cents | 0 | - | 50 cents | 0 | 0 |
| 50 cents | 1 | 0 | wait | 1 | 0 |

Based on the state diagram, and the working of the vending machine we get the following state diagram:

