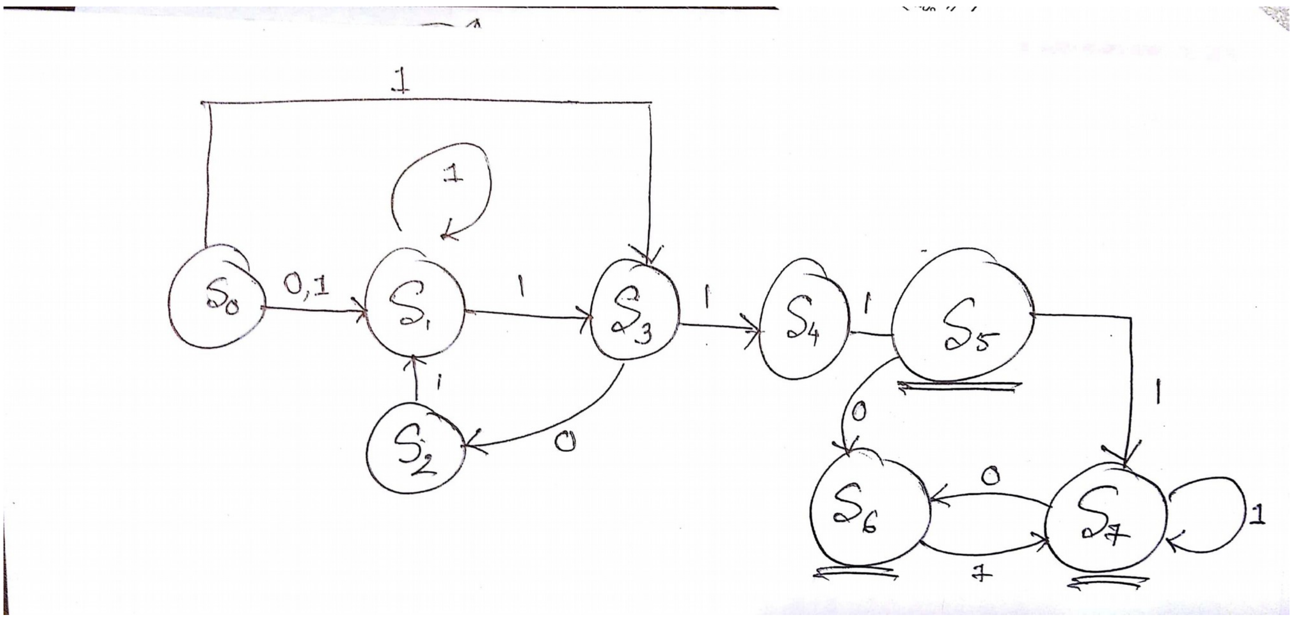
1B.
The informal definition of the accepted string is defined below:
it will accept any of the following strings:

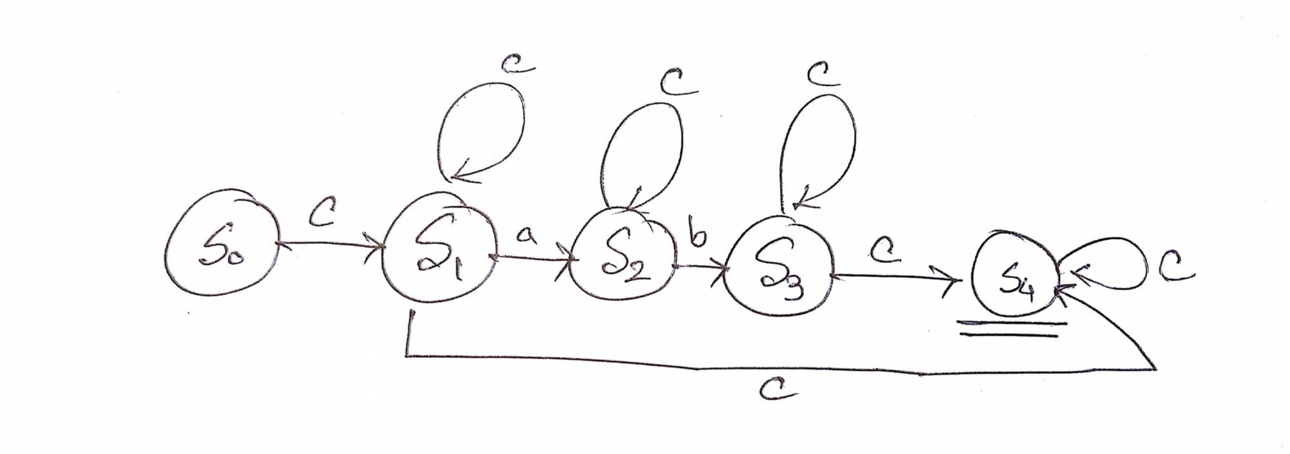
- starting with one or more 01, then a single 0 and finally zero or more 0 or 1. regular expression will be like (01)+0(0|1)*
- starting with two or more 0 and finally zero or more 0 or 1. regular expression will be like 00+(0|1)*
- starting with one or more 10, then a single 1 and finally zero or more 0 or 1. regular expression will be like (10)+1(0|1)*
- starting with two or more 1 and finally zero or more 0 or 1. regular expression will be like 11+(0|1)*
- combination of the aforementioned possibilities above as transition can happen like $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$ or $s_0 \rightarrow s_2 \rightarrow s_0 \rightarrow s_1$ or recursively. Regular expression will be like
  ((01)*(10)*|(10)*(01)*)(01)+0(0|1)*
  ((01)*(10)*|(10)*(01)*)00+(0|1)*
  ((01)*(10)*|(10)*(01)*)(10)+1(0|1)*
  ((01)*(10)*|(10)*(01)*)11+(0|1)*

1C
starts with zero or more b, then a, then optionally zero or more b and finally with a at end, then one or more a, then b, then optionally zero or more a, then one or more b, then a, then optionally one or more b and finally with a at end, then finishes with a or more than zero a or b
b*ab*a+ba*b(ab)*aa(a|b)*

2B



2C



However, this is the DFA of the language that accepts both 0 a and b or one a and b, starting and ending with arbitrary number of c. if there is 2n|2n+1 number of a in this language, then there should be 3n|3n+1 number of b in the language. Drawing such kind of DFA is not possible because we need to count the number of a and b.

4C
Suppose,
D = [0-9]
d = [1-9]

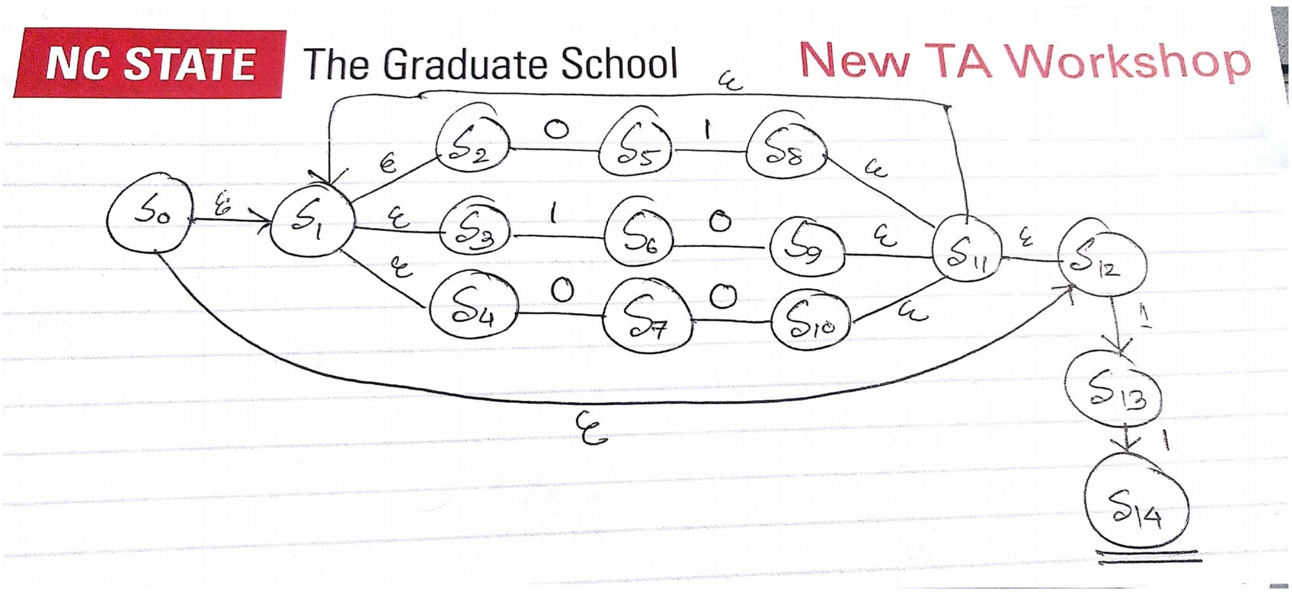
The regular expression is:
$((dDD|dD|d)(,))?(DDD,)*((dDD|dD|d))(.)(DD)

5E
Writing a regular expression for all possible valid algebraic expression is impossible due to the usage of parenthesis. There can be infinitely nested parenthesis in the expression which can't be written as regular expression. However, simpler algebraic statements can be represented by regular expression:

([-]?id?)*([+-]? id [+-/*] [+-]? id ([+-/*] [+-]? id)*)+

7C

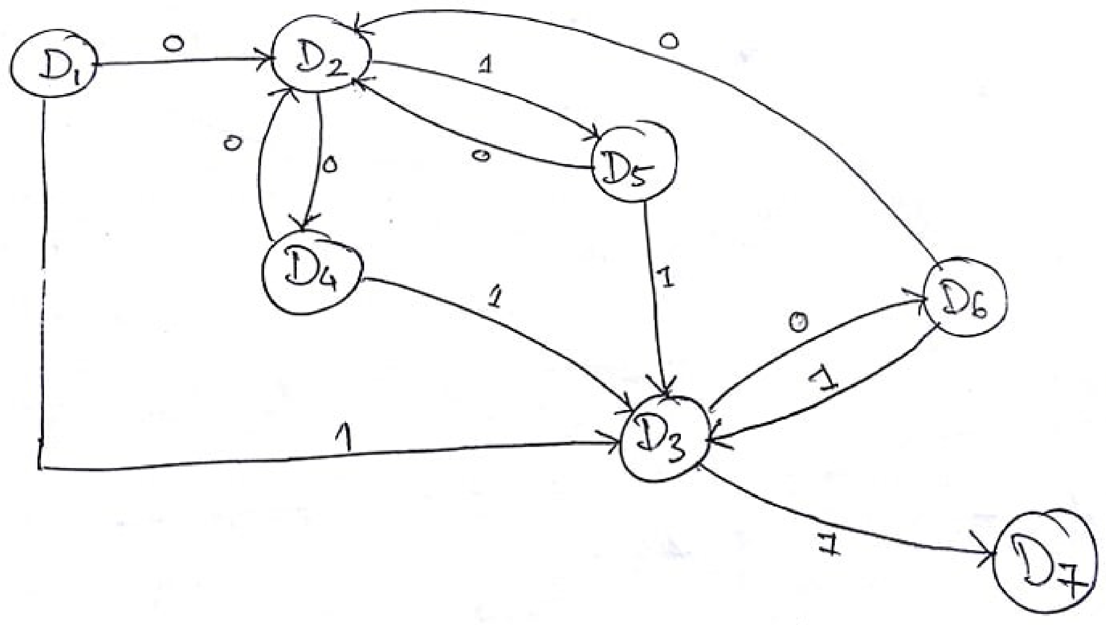
Regular Expression: (01|10|00)*11. Hence the corresponding NFA is given below:



Here is the chart for NFA → DFA conversion:

DFA States	NFA States	€-closure(move(q,*))	
		0	1
D1	S0 S1 S2 S3 S4 S12	S5 S7	S6 S13
D2	S5 S7	S10 S11 S12 S1 S2 S3 S4	S8 S11 S12 S1 S2 S3 S4
D3	S6 S13	S9 S11 S12 S1 S2 S3 S4	S14
D4	S10 S11 S12 S1 S2 S3 S4	S5 S7	S6 S13
D5	S8 S11 S12 S1 S2 S3 S4	S5 S7	S6 S13
D6	S9 S11 S12 S1 S2 S3 S4	S5 S7	S6 S13
D7	S14	-	-

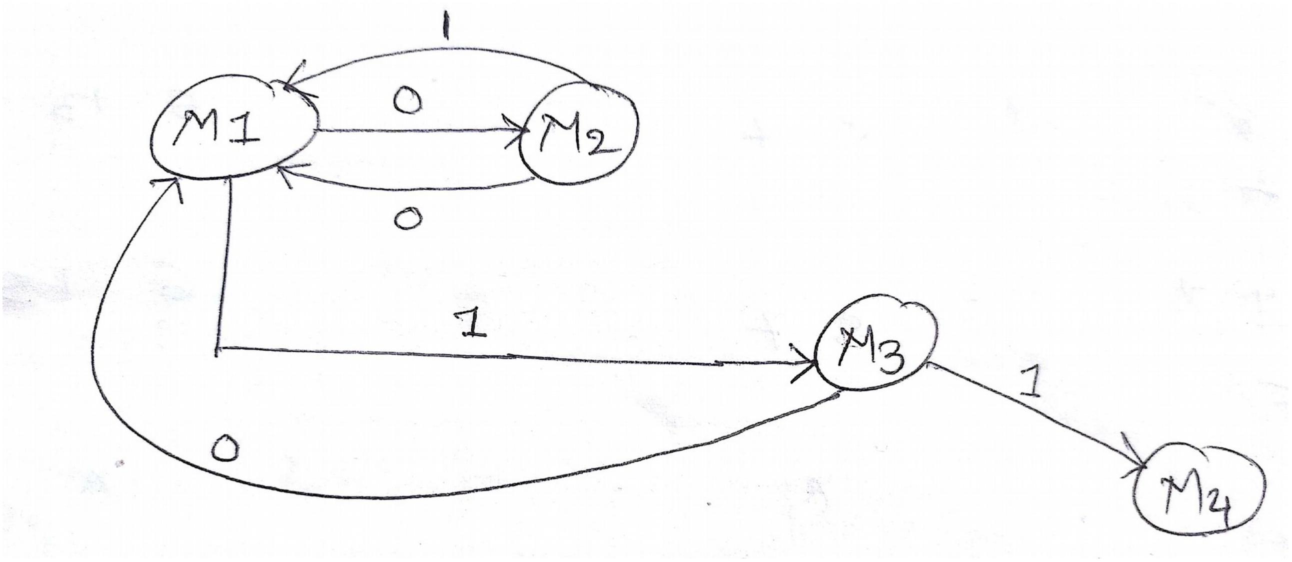
The corresponding DFA is:



DFA Minimization

Current Partition	Split on 0	Split on 1
{D1, D2, D3, D4, D5, D6}, {D7}	-	{D1, D2, D4, D5, D6}, {D3}, {D7}
{D1, D2, D4, D5, D6}, {D3}, {D7}	-	{D1, D4, D5, D6}, {D2}, {D3}, {D7}
{D1, D4, D5, D6}, {D2}, {D3}, {D7}	-	-

Hence the partitioned sets are
{D1, D4, D5, D6} = M1
{D2} = M2
{D3} = M3
{D7} = M4



Note for the Programming Assignment:
The identifier checking function in the regular expression class has been implemented. Still, there were some other problems in the given program

1. It could not process meta statement such as comments starting with //.
   The program was not taking newline character. Hence, the meta statement checking for comments was always returning false for comments. It has been fixed.
2. It could not show error message given the wrong symbol such as @.
   If there were still some characters left with no identifiable tokens and it reaches the EOF – that means there are some invalid inputs there. It has been fixed applying this logic.