Problem 1

Give a DFA for the following languages, specified by a transition diagram. For each one of them, give a short and clear description of how the machine works. The alphabet is $\Sigma = \{0, 1, 2\}$:

(a) $L_1 = \{w \in \Sigma^*: w$ begins with 0 or ends with 0 but not both$\}$.

(b) $L_2 = \{w \in \Sigma^*: w$ contains the pattern 01 at least twice$\}$.

(c) $L_3 = \{w \in \Sigma^*: the$ sum of the digits of $w$ is odd and is a multiple of 3$\}$.

(d) $L_4 = \{w \in \Sigma^*: w$ is not in the language of the regular expression $0^*1^*2^* + 2^*1^*0^*$\}$.

Solution

(a) We have 5 states $q_0, q_1, q_2, q_3, q_4$. For inputs that start with 0, after reading the 0 we go to a non-accepting state $q_1$ since 0 cannot be the last alphabet. After that, whenever a 1 or a 2 is read, we go to an accepting state $q_3$ and whenever we read a 0 we go back to $q_1$. For inputs that do not start with 0, we do the opposite at states $q_3$ and $q_4$. 
(b) We first construct a DFA that accepts if \( w \) contains the pattern 01 at least once. We stay at the initial state \( q_0 \) until we read a 0, on which we move to another state \( q_1 \). When we are at \( q_1 \), we go back to \( q_0 \) on 2, stay at \( q_1 \) on 0, and accept on 1. Upon reaching the accepting state, we do not leave. We duplicate the DFA to accept strings that contain the pattern 01 at least twice.

(c) Note that we accept if and only if the sum of digits is equal to \( 6k + 3 \) for \( k \geq 0 \). We keep track of the sum of digits at every step, and go to \( q_k \) when its remainder is equal to \( k \) modulo 6. \( q_3 \) is the accepting state since \( 6k + 3 \equiv 3 \pmod{6} \).

(d) We first construct a DFA for strings that are in the language of the regular expression \( 0^*1^*2^* + 2^*1^*0^* \). There are four cases:

(i) The input is the empty string. In this case we accept.
(ii) Inputs that begin with 1s. The 1s can only be followed by either 0s or 2s. Otherwise it switches to a “die” state.
(iii) Inputs that begin with 0s. It can only be followed by 2s, or 1s followed by 2s.
(iv) Inputs that begin with 2s. It can only be followed by 0s, or 1s followed by 0s.

To turn it into a DFA for \( L_4 \), we flip all the accepting states to non-accepting states and vice versa.
Problem 2

Convert the following NFA to a DFA using the method described in class. Specify the DFA by its transition diagram. The alphabet is $\Sigma = \{0, 1\}$.

Solution

First, we eliminate the $\epsilon$ transitions and obtain the following equivalent NFA:

Then we convert it to DFA. After eliminating the unreachable states we obtain the following DFA.
Problem 3

Consider the following languages over $\Sigma = \{0, 1\}$.

- $L_1$ is the language described by $1^*(0111^*)^*$.
- $L_2$ is the language of strings with at least one 0 and at least two 1s.
- $L_3$ is the language of the following NFA:

- $L_4$ is the language described by $(0 + 1)^*01(0 + 1)^*1$.
- $L_5$ is the language described by $(011 + 101 + 110)^*$.
- $L_6$ is the language of the following NFA:
• $L_7$ is the language of all strings that do **not** contain 00, 010 and do **not** end in 0 or 01.

• **(Optional)** $L_8$ is the language of the following DFA:

![DFA Diagram]

Which of these languages are the same and which are different? To show two languages are the same give a short argument, and to show two languages are different give a string that is in one but not in the other. (You must provide an explanation to get credit.)

**Solution**

We have the following partition \( \{L_1, L_3, L_7\}, \{L_2, L_8\}, \{L_4, L_6\}, \{L_5\} \). We need to argue that languages in the same group are equivalent and in different groups are not.

$L_1 = L_3 = L_7$: First, observe that $L_1$ is the language of all strings in which any 0 must be followed by at least two 1s. To see this, consider a string in such form. There can be any number of 1s before the first 0s; for each 0 in the string, it must be followed by at least two 1s. Therefore, it has the regular expression $1^*(0111^*)^*$.

Now we show that $L_3$ is equivalent to $L_1$. It suffices to show that $L_3$ is the language that describes all strings in which any 0 must be followed by at least two 1s. Consider a string in such form. There can be any number of 1s at the beginning, therefore it stays at an accepting state in the NFA. Whenever a 0 is read, we go to $q_1$, and we must read two 1s to reach an accepting state. Otherwise, we go to a “die” state.

To show that $L_7$ is equivalent to $L_1$. Clearly, any string in $L_1$ is in $L_7$. Conversely, consider a string in $L_7$. We show that any 0 in it must be followed by at least two 1s. By description, its last 2 characters in the string cannot be 0, this means a 0 in the string must be followed by at least two characters, and the first condition ensures that its following two characters must be 1s.

$L_2 = L_8$: We construct the DFA in $L_8$ from $L_2$. We keep track of the number of 0s and 1s we have read at each step. We reach

(a) $q_0$ when we have no 0s and 1s,

(b) $q_1$ when we have enough 0s and no 1s,
(c) $q_2$ when we have enough 0s and one 1,
(d) $q_3$ when we have no 0s and one 1,
(e) $q_4$ when we have no 0s and enough 1s,
(f) $q_5$ when we have enough 0s and 1s.

We begin at $q_0$ and transit to the other states correspondingly. $q_5$ is the only accepting states as we only accept when we have enough 0s and 1s, and we never leave $q_5$ after reaching it.

$L_4 = L_6$: First we show that the regular expression \((0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^* 1\) is equivalent to \((0 + 1)^* 0 1 (0 + 1)^* 1\). One direction is easy. To see the other direction, consider any string of the form \((0 + 1)^i 0 (0 + 1)^k 1 (0 + 1)^j 1\) for some \(i, j, k \geq 0\). There are three cases:

(a) \(k = 0\). Then it is of the form \((0 + 1)^* 0 1 (0 + 1)^* 1\).

(b) The substring in \((0 + 1)^k\) is of the form \(1 (0 + 1)^{k-1}\). Then the string is of the form \((0 + 1)^i 0 1 (0 + 1)^{k-1} 1 (0 + 1)^j 1\), which is also of the form \((0 + 1)^i 0 1 (0 + 1)^{k+j} 1\).

(c) The substring in \((0 + 1)^k\) is of the form \(0 (0 + 1)^{k-1}\). Then the string is of the form \((0 + 1)^i 0 0 (0 + 1)^{k-1} 1 (0 + 1)^j 1\), which is also of the form \((0 + 1)^{i+1} 0 (0 + 1)^{k-1} 1 (0 + 1)^j 1\).

The argument in (c) can be repeated until we reach the first and second cases, whose regular expressions are of the form \((0 + 1)^* 0 1 (0 + 1)^* 1\). Hence the two regular expressions are equivalent.

Now we show that there is a 1 to 1 correspondence between the regular expression \((0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^* 1\) and the NFA of $L_6$. The three \((0 + 1)^*\) in the regular expression represent the loops at $q_0$, $q_1$, and $q_2$. The 0 between the first two \((0 + 1)^*\) represents the transition from $q_0$ to $q_1$. The 1 between the second two \((0 + 1)^*\) represents the transition from $q_1$ to $q_2$. The 1 at the end represents the transition from $q_2$ to $q_3$.

$L_1 \neq L_2, L_4, L_5$: The string 1 is in $L_1$ but not in any of these other languages.

$L_2 \neq L_4, L_5$: The string 1100 is in $L_2$ but not in $L_4$ and $L_5$.

$L_4 \neq L_5$: The string 0011 is in $L_4$ but not in $L_5$.

**Problem 4**

In this problem you will design an NFA that checks if an arithmetic expression is formatted correctly. An arithmetic expression is made up of parans ‘(’, ‘)’, operations ‘+’, ‘-’, ‘*’ and the digits from 0 to 9. Here are some correctly formatted expressions:

\[
1 + 2 \\
(199 + 75) * (89 - 2036) \\
(88 + 0) / 1
\]
And here are some incorrectly formatted ones:

74 * 0
(7 * 1)(
10 *

You may assume the expression does not have nested parans like \((5 \ast (4 + 6))\). Feel free to make other simplifying assumptions, but make sure you state all such assumptions in your solution.

When drawing the transition diagram of your NFA, you can use the shorthand notation \([0-9]\) to describe transitions labeled by all the digits ‘0’, ‘1’, … ‘9’. You can also label a single transition by multiple symbols: For instance, a transition labeled by ‘671’ stands for three consecutive transitions labeled by ‘6’, ‘7’, and ‘1’ respectively.

This is a design problem, and part of your job is to figure out a way to distinguish among correct and incorrect description. There may not be a single right answer. You must describe your reasoning clearly in your solution. Solutions that only provide a picture of an NFA with no explanation will get no credit.

**Solution**

We assume leading zeros in each number are allowed but the empty string and negative numbers are not allowed. Note that a number has at least one digit; therefore whenever we transit to a state (e.g. \(q_3, q_4\)) on a digit \([0-9]\), there is a loop at the state which takes more \([0-9]\)s.

In the beginning we can read a number or a ‘(’. If we read a number, it can be followed by nothing and so \(q_4\) is an accepting state. Or the number is followed by an operator, then we go back to the initial state \(q_0\). If we read a ‘(’, we go to the state \(q_2\) to read a number. After that we can take an operator to reach \(q_2\) to take another number, or close the bracket using ‘)’ to go to the accepting state \(q_4\). If it is followed by an operator, we go back to the initial state \(q_0\).