Problem 1. [10 points] Construct a regular expression $R$ over the alphabet \{0, 1\} such that $L(R)$ is the set of words that do not contain 100 as three consecutive characters.

Solution:

$$0^* (1(\varepsilon|0))^*$$
Problem 2. [15 points] Let

\[ \Sigma = \{ \begin{bmatrix} a \\ b \\ c \end{bmatrix} : a, b, c \in \{0, 1\} \} \].

A string \( w \in \Sigma^* \), which can be written as

\[ \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} \cdots \begin{bmatrix} a_0 \\ b_0 \\ c_0 \end{bmatrix} = \prod_{i=n-1}^{0} \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix}, \]

encodes three natural numbers

\[ a_w = \sum_{i=0}^{n-1} 2^i a_i, \quad b_w = \sum_{i=0}^{n-1} 2^i b_i \quad \text{and} \quad c_w = \sum_{i=0}^{n-1} 2^i c_i. \]

Prove that the language \( \text{PLUS} = \{ w \in \Sigma^* : a_w + b_w = c_w \} \) is regular. If you utilize DFA, NFA or regular expressions, you may simply state their language instead of proving they describe that language (but you need to formally define them).

**Solution:** We construct a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) such that \( L(M) = \text{PLUS}^T \) as follows: \( Q = \{0, 1, \bot\} \), \( q_0 = 0 \), \( F = \{0\} \) and \( \delta : Q \times \Sigma \rightarrow Q \) is given by

\[ \delta \left( q, \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right) = \begin{cases} \bot, & q = \bot \\ \bot, & q \in \{0, 1\} \text{ but } (a + b + q) \mod 2 \neq c \\ \frac{a + b + q}{2}, & q \in \{0, 1\} \text{ and } (a + b + q) \mod 2 = c. \end{cases} \]

Intuitively, the automaton remembers whether the output has been different than expected with the error state \( \bot \) and, if not, what is the addition carry bit. It accepts if the output has always matched the sum and if no carry bit is left.

Since \( L(M) = \text{PLUS}^T \), the language \( \text{PLUS}^T \) is regular and thus so is \( (\text{PLUS}^T)^T = \text{PLUS} \) (recall Assignment 2 Problem 1).
Problem 3. [20 points] The substrings of a string are the strings that can be obtained by erasing characters from the original string. This can be defined inductively as follows, where $\Sigma$ is the strings’ alphabet: $\varepsilon$ is a substring of any string; and, for $a \in \Sigma$ and $w \in \Sigma^*$, the string $aw$ is a substring of a string $w' \in \Sigma^*$ if we can write $w' = xay$ with $x, y \in \Sigma^*$ and $w$ a substring of $y$.

Prove that, whenever $A \subseteq \Sigma^*$ is a regular language, so are the languages

$$\text{SUB}(A) = \{ w \in \Sigma^* : w \text{ is a substring of a string in } A \}$$

and

$$\text{SUPER}(A) = \{ w \in \Sigma^* : \text{there is a substring of } w \text{ in } A \}.$$ 

If you utilize DFA, NFA or regular expressions, you may simply state their language instead of proving they describe that language (but you need to formally define them).

**Solution:** Since $A$ is regular, there must be a DFA $M$ such that $L(M) = A$. Let $M' = (Q, \Sigma, \delta', q_0, F)$ be the NFA where $\delta': Q \times (\Sigma \cup \{ \varepsilon \}) \rightarrow 2^Q$ is given by

$$\delta'(q, x) = \begin{cases} \{ \delta(q, x) \}, & x \in \Sigma \\ \{ \delta(q, a) : a \in \Sigma \}, & x = \varepsilon. \end{cases}$$

Intuitively, $M'$ is allowed to not consume input characters. Because of this, $L(M') = \text{SUB}(A)$.

Let now $M'' = (Q, \Sigma, \delta'', q_0, F)$ be the NFA where $\delta'': Q \times (\Sigma \cup \{ \varepsilon \}) \rightarrow 2^Q$ is given by

$$\delta''(q, x) = \begin{cases} \{ \}, & x = \varepsilon \\ \{ q, \delta(q, x) \}, & x \in \Sigma. \end{cases}$$

Intuitively, $M''$ is allowed to ignore input characters. Thus $L(M'') = \text{SUPER}(A)$. 


Problem 4. [10 points] Let $A$ be a finite language over an alphabet $\Sigma$. We have seen that $A$ is regular. Prove that any DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = A$ satisfies $|Q| > |w|$ for each string $w \in A$.

Solution: We have seen from the proof of the Pumping Lemma that $|Q|$ is a pumping length for $A$. Therefore, if there were a string $w \in A$ such that $|w| \geq |Q|$, we could apply the Pumping Lemma and write $w = xyz$ with $x, y, z \in \Sigma^*$, $y \neq \varepsilon$ and $xy^iz \in A$ for all $i \in \mathbb{N}$ (we ignore $|xy| \leq |Q|$). But then $A$ would have infinitely many strings, one for each $i$. 
Problem 5. [20 points] Recall our scheme to encode DFA as strings over an alphabet $\Sigma$ with $|\Sigma| \geq 2$. Prove that the language 

$$\text{EQ}_{\text{DFA}} = \{ \langle M_0 \rangle \langle M_1 \rangle : M_0 \text{ and } M_1 \text{ are DFA such that } L(M_0) = L(M_1) \}$$

is decidable. If you utilize DFA, NFA or regular expressions, you may simply state their language instead of proving they describe that language (but you need to formally define them). If you utilize Turing machines, a high-level description of them is sufficient.

Hint: start by showing that the language 

$$\text{NULL}_{\text{DFA}} = \{ \langle M \rangle : M \text{ is a DFA such that } L(M) = \{ \} \}$$

is decidable. Then reduce the decidability of $\text{EQ}_{\text{DFA}}$ to that of $\text{NULL}_{\text{DFA}}$.

Solution: As suggested, we start by showing that $\text{NULL}_{\text{DFA}}$ is decidable. Our Turing machine will start by checking that the input has the form $\langle M \rangle$ for some DFA $M = (Q, \Sigma, \delta, q_0, F)$. If it does not have this form, the Turing machine rejects, as it should. There are a few ways to complete this Turing machine. One is to test whether $\delta(q_0, w) \in F$ for any strings $w$ of length less than $|Q|$, which is correct due to Assignment 2 Problem 3. Another way is to first build a directed graph $G = (Q, E)$ with vertex set $Q$ and edge set $E = \{ uv : \exists a \in \Sigma : \delta(u, a) = v \}$ and then check whether a final state is reachable from the initial state using, say, depth-first search.

Now we show how to build a Turing machine to decide $\text{EQ}_{\text{DFA}}$ from a Turing machine to decide $\text{NULL}_{\text{DFA}}$. First it checks whether the input is of the form $\langle M_0 \rangle \langle M_1 \rangle$ for some DFA $M_0 = (Q_0, \Sigma, \delta_0, q_{00}, F_0)$ and $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, rejecting otherwise, as it should. It then builds the DFA $M = (Q, \Sigma, \delta, q_0, F)$ from $M_0$ and $M_1$ where $Q = Q_0 \times Q_1$, $q_0 = (q_{00}, q_{01})$, 

$$F = \{ (q_0, q_1) \in Q : (q_0 \in F_0 \text{ and } q_1 \notin F_1) \text{ or } (q_0 \notin F_0 \text{ and } q_1 \in F_1) \}$$

and $\delta : Q \times \Sigma \to Q$ is given by 

$$\delta((q_0, q_1), a) = (\delta_0(q_0, a), \delta_1(q_1, a)) \text{.}$$

Thus, by simulating $M_0$ and $M_1$, we ensure that $L(M) = L(M_0) \oplus L(M_1)$, so $L(M) = \{ \}$ if, and only if, $L(M_0) = L(M_1)$. All that remains for our Turing machine to do is to use the Turing machine for $\text{NULL}_{\text{DFA}}$ to decide whether $L(M) = \{ \}$.
Problem 6. [15 points] Assume we agree on a scheme analogous to that of Problem 5 to encode context-free grammars as strings over an alphabet $\Sigma$ with $|\Sigma| \geq 2$. Consider then the language $\text{EQ}_{\text{CFG}} = \{ \langle G_0 \rangle \langle G_1 \rangle : G_0$ and $G_1$ are context-free grammars such that $L(G_0) = L(G_1) \}$, which is known to be undecidable (after the exam is over, you can see a proof in Chapter 5 of Sipser’s book).

Assuming this fact, prove or disprove that $\text{EQ}_{\text{CFG}}$ is enumerable. If you utilize Turing machines, a high-level description of them is sufficient.

Solution: We will prove that $\Sigma^* \setminus \text{EQ}_{\text{CFG}}$ is enumerable, which implies that $\text{EQ}_{\text{CFG}}$ is not enumerable by Assignment 4 Problem 2. Here is how a Turing machine $M$ with $L(M) = \Sigma^* \setminus \text{EQ}_{\text{CFG}}$ looks like:

<table>
<thead>
<tr>
<th>Procedure $M(s \in \Sigma^*)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check that the input has the form $\langle G_0 \rangle \langle G_1 \rangle$ with $G_0$ and $G_1$ context-free grammars;</td>
</tr>
<tr>
<td>if not then</td>
</tr>
<tr>
<td>Accept;</td>
</tr>
<tr>
<td>Convert $G_0$ to an equivalent context-free grammar $G'_0$ in Chomsky normal form;</td>
</tr>
<tr>
<td>Convert $G_1$ to an equivalent context-free grammar $G'_1$ in Chomsky normal form;</td>
</tr>
<tr>
<td>for $k$ from 0 onwards do</td>
</tr>
<tr>
<td>for each string $w$ among the $</td>
</tr>
<tr>
<td>Run the CYK algorithm to decide whether $w \in L(G'_0)$;</td>
</tr>
<tr>
<td>Run the CYK algorithm to decide whether $w \in L(G'_1)$;</td>
</tr>
<tr>
<td>if the verdicts differed then</td>
</tr>
<tr>
<td>Accept;</td>
</tr>
</tbody>
</table>

This Turing machine will accept any input not of the form $\langle G_0 \rangle \langle G_1 \rangle$ since those can never be in $\text{EQ}_{\text{CFG}}$. On the other hand, when an input does have the form $\langle G_0 \rangle \langle G_1 \rangle$, then if $L(G_0) = L(G_1)$, the Turing machine can never accept and will diverge. Finally, if $L(G_0) \neq L(G_1)$, then there must be at least one word $w \in \Sigma^*$ that would cause its iteration to accept. Either the machine accepts before considering $w$ or will eventually try out $w$ and accept.
Problem 7. [10 points] Develop a λ-term \textit{Reverse} such that, whenever a λ-term \( L \) normalizes to a list, the λ-term \((\text{Reverse} \ L)\) normalizes to the reverse of that list.

Solution: First we introduce a λ-term \textit{Append} to append two lists. This is a recursive term and we use the methodology for recursion we saw in class.

\[
\text{Append} = (\lambda x. x x) \lambda r. \lambda a, b. \text{If (IsEmpty} \ a) b \left( \text{Cons} (\text{Head} \ a) (r \ r (\text{Tail} \ a) b) \right)
\]

We use the same methodology again to define \textit{Reverse}:

\[
\text{Reverse} = (\lambda x. x x) \lambda r. \lambda \ell. \text{If (IsEmpty} \ \ell) \text{Nil} \left( \text{Append} (r \ r (\text{Tail} \ \ell)) (\text{Cons} (\text{Head} \ \ell) \text{Nil}) \right)
\]