1. (a) Let $\Sigma = \{0, 1\}$. Draw an NFA that accepts the language $\{w \mid$ there are at least two block of 0s of even length$\}$. A “block of 0s” is a maximal non-empty substring of 0s. Examples of strings in the language are 0000100, 11001000100111 and example of strings not in the language are 001, 00010. Briefly explain the meaning of your states so that the construction/correctness of your machine is understandable.

(b) Draw an NFA that accepts all strings that contain cs374 as a subsequence. Assume that $\Sigma$ consists of all English letters and digits. You can label transitions with $\Sigma$ (or other sets) to avoid specifying all the labels and make your NFA description compact.

(c) Suppose you have an NFA $N_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ that accepts all strings that contains cs374 as a subsequence. You also have an NFA $N_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$ that accepts all strings that contain cs473 as a subsequence. Now you want to create an NFA $N = (Q, \Sigma, \delta, s, A)$ that accepts all strings that contain both cs374 and cs473 as subsequences. You can of course convert $N_1$ into a DFA $M_1$ and $N_2$ into a DFA $M_2$ via the subset construction and then do a product construction over $M_1$ and $M_2$ to create a DFA $M$ that accepts the desired set of strings. However this can create a large DFA with a number of states that is exponential in the number of states of $N_1$ and $N_2$. Instead we would like to create a more compact NFA. Show that the product construction can be generalized to create an NFA $N = (Q, \Sigma, \delta, s, A)$ such that $L(N) = L(N_1) \cap L(N_2)$. Your main task is to formally define $\delta, s, A$ in terms of the parameters of $N_1$ and $N_2$. Briefly justify why your construction works. A formal proof by induction is not needed. Note that $\delta$ for a NFA is different from that for a DFA. Be careful with set-theoretic notation.

(d) For practice, do not submit for grading:
   i. Draw an NFA for the regular expression $(010)^* + (01)^* + 0^*$.
   ii. Now using the powerset construction (also called the subset construction), design a DFA for the same language. Label the states of your DFA with names that are sets of states of your NFA. You should use the incremental construction so that you only generate the states that are reachable form the start state.

2. Let $L_1$ and $L_2$ be two languages. We define $\text{insert}(L_1, L_2)$ to be the language $\{xwy \mid xy \in L_1, w \in L_2\}$. In other words each string in $\text{insert}(L_1, L_2)$ is obtained by starting with a string $z \in L_1$ and inserting some string of $w$ of $L_2$ somewhere in $z$; thus the new string is $xwy$ where $z = xy$. For instance if $L_1 = \Sigma^*$ and $L_2 = \{\text{cs374}\}$ then $\text{insert}(L_1, L_2)$ is set of all strings that contain cs374 as a substring; if $L_2 = \{\text{cs374}, \text{cs473}\}$ then $\text{insert}(L_1, L_2)$ is the set of all strings that contain cs374 or cs473 as a substring. You can imagine more interesting examples.

(a) Prove that if $L_1, L_2$ are regular then $\text{insert}(L_1, L_2)$ is regular via the following technique. Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ be a DFA such that $L_1 = L(M_1)$, and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$ be a DFA such that $L_2 = L(M_2)$. Describe an NFA $N$ in terms of $M_1, M_2$ that accepts $\text{insert}(L_1, L_2)$. Explain the construction of your NFA.
(b) In this second part you will give a different proof. Let \( r_1 \) be a regular expression for \( L_1 \) and \( r_2 \) be a regular expression for \( L_2 \), that is \( L_1 = \text{L}(r_1) \) and \( L_2 = \text{L}(r_2) \). We will develop a recursive algorithm that given \( r_1, r_2 \) constructs a regular expression \( r' \) such that \( L(r') = \text{insert}(L_1, L_2) \). No correctness proof is required but a brief explanation of the derivation would help you get partial credit in case of mistakes.

i. For each of the base cases when \( r_1 \) is \( \emptyset \), \( e \) and \( a, a \in \Sigma \) describe a regular expression for \( \text{insert}(L(r_1), L(r_2)) \) in terms of \( r_2 \) and the letters in \( \Sigma \).

ii. Suppose \( r_1 = s + t \) where \( s \) and \( t \) are regular expressions. Moreover let \( s' \) be a regular expression for \( \text{insert}(L(s), L(r_2)) \) and \( t' \) be a regular expression for \( \text{insert}(L(t), L(r_2)) \). Describe a regular expression for the language \( \text{insert}(L(r_1), L(r_2)) \) using \( r_2, s, t, s', t' \).

iii. Same as the previous part but now consider \( r_1 = st \).

iv. Same as the previous part but now consider \( r_1 = (s)^* \).

v. Apply your construction to the regular expressions \( r_1 = 0^* + (01)^* + 011^*0 \) and \( r_2 = 101 \) to obtain a regular expression for \( \text{insert}(L(r_1), L(r_2)) \).

---

**Solved problem**

4. Let \( L \) be an arbitrary regular language. Prove that the language \( \text{half}(L) := \{ w \mid ww \in L \} \) is also regular.

**Solution:** Let \( M = (\Sigma, Q, s, A, \delta) \) be an arbitrary DFA that accepts \( L \). We define a new NFA \( M' = (\Sigma, Q', s', A', \delta') \) with \( \epsilon \)-transitions that accepts \( \text{half}(L) \), as follows:

\[
Q' = (Q \times Q \times Q) \cup \{s'\}
\]

\( s' \) is an explicit state in \( Q' \)

\[
A' = \{(h, h, q) \mid h \in Q \text{ and } q \in A\}
\]

\[
\delta'(s', \epsilon) = \{(s, h, h) \mid h \in Q\}
\]

\[
\delta'((p, h, q), a) = \{\delta(p, a), h, \delta(q, a)\}
\]

\( M' \) reads its input string \( w \) and simulates \( M \) reading the input string \( ww \). Specifically, \( M' \) simultaneously simulates two copies of \( M \), one reading the left half of \( ww \) starting at the usual start state \( s \), and the other reading the right half of \( ww \) starting at some intermediate state \( h \).

- The new start state \( s' \) non-deterministically guesses the “halfway” state \( h = \delta^*(s, w) \) without reading any input; this is the only non-determinism in \( M' \).
- State \( (p, h, q) \) means the following:
  - The left copy of \( M \) (which started at state \( s \)) is now in state \( p \).
  - The initial guess for the halfway state is \( h \).
  - The right copy of \( M \) (which started at state \( h \)) is now in state \( q \).
- \( M' \) accepts if and only if the left copy of \( M \) ends at state \( h \) (so the initial non-deterministic guess \( h = \delta^*(s, w) \) was correct) and the right copy of \( M \) ends in an accepting state.
Rubric: 5 points =
+ 1 for a formal, complete, and unambiguous description of a DFA or NFA
  – No points for the rest of the problem if this is missing.
+ 3 for a correct NFA
  – −1 for a single mistake in the description (for example a typo)
+ 1 for a brief English justification. We explicitly do not want a formal proof of correctness, but we do want one or two sentences explaining how the NFA works.