Problem 1 [40 points]: Consider the alphabet $\Sigma = \{a, b\}$.

a) Construct a Non-Deterministic-Finite Automaton (NFA) using the Thompson construction that is able to recognize the sentences generated by the regular expression $RE = (ab)^*.(a^+)$. 

b) Do the sentences $w_1 = “aba”$ and $w_2 = “aab”$ belong to the language generated by this regular expression? Justify.

c) Convert the NFA in part a) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.

d) Minimize the DFA using the iterative refinement algorithm discussed in class. Show your intermediate partition results and double check the DFA using the sentences $w_1$ and $w_2$.

Solution:

a) A possible construction (already simplified to have only a single $\varepsilon$-transition between states) would result in the NFA shown below and where the start state is labeled 0.

![NFA Diagram]

b) Regarding the word “aba” there is a path from state 0 to the accepting state 6, namely: 0,1,2,4,6,1,3,5,6,7,8,9. Regarding the word “aab” the automaton will never be able to reach the state 9 as in order to spell out the “b” character will necessarily be in state 5 and from that state there is no empty-string path to state 9. The accepting words can thus never terminate with the “b” character which is the case with this word.

c) Using the subset construction we arrive at the following subsets and transitions.

\[
\begin{align*}
S_0 &= \varepsilon\text{-closure (0)} = \{0, 1, 2, 3, 6, 7\} - \text{this is not a final state.} \\
S_1 &= \text{DFAedge}(S_0, a) = \varepsilon\text{-closure (goto(S0, a))} = \{1, 2, 3, 4, 6, 7, 8, 9\} - \text{final state} \\
S_2 &= \text{DFAedge}(S_0, b) = \varepsilon\text{-closure (goto(S0, b))} = \{1, 2, 3, 5, 6, 7\} \\
\text{DFAedge}(S_1, a) &= \varepsilon\text{-closure (goto(S1, a))} = \{1, 2, 3, 4, 6, 7, 8, 9\} = S_1 \\
\text{DFAedge}(S_1, b) &= \varepsilon\text{-closure (goto(S1, b))} = \{1, 2, 3, 5, 6, 7\} = S_2 \\
\text{DFAedge}(S_2, a) &= \varepsilon\text{-closure (goto(S2, a))} = \{1, 2, 3, 4, 6, 7, 8, 9\} = S_1 \\
\text{DFAedge}(S_2, b) &= \varepsilon\text{-closure (goto(S2, b))} = \{1, 2, 3, 5, 6, 7\} = S_2
\end{align*}
\]
This results in the DFA shown below with starting state S0.

\[
\begin{align*}
\text{S0} & \quad \text{a} \quad \text{S1} \\
\text{a} \quad \text{S0} & \quad \text{b} \quad \text{S2} \\
\text{b} & \quad \text{S2} \\
\end{align*}
\]

d) This DFA can be further minimize by using the iterative refinement partitioning yielding the sequence of partitions indicated below. In this particular case the initial partition is also the final partition as it is not possible to discriminate any states based on input characters.

Initial partitioning on final states (no further partition is possible)  

final minimal DFA
Problem 2 [30 points]: Consider the DFA below with starting state 1 and accepting state 2:

![DFA Diagram]

a) Describe in English the set of strings accepted by this DFA.
b) Using the Kleene construction algorithm derive the regular expression recognized by this automaton simplifying as much as possible.

Solution:

a) This automaton recognizes all the strings over the \{a,b,\} alphabet that are not empty either are the singleton “a”, or end with an “b” and can have any combination of pairs of “aa” sequences or an sequence of “b”s of arbitrary length.
b) The derivations are shown below with the obvious simplification.

Expressions for \(k = 0\)

\[
R_{11}^0 = (b|\epsilon)
\]
\[
R_{12}^0 = (a)
\]
\[
R_{21}^0 = (a)
\]
\[
R_{22}^0 = (b|\epsilon)
\]

Expressions for \(k = 1\)

\[
R_{11}^1 = R_{11}^0 (R_{11}^0)^* R_{11}^0 | R_{11}^1 = (b|\epsilon) . (b|\epsilon)^* . (b|\epsilon) | (b|\epsilon) = b^*
\]
\[
R_{12}^1 = R_{11}^0 (R_{11}^0)^* R_{12}^0 | R_{12}^1 = (b|\epsilon) . (b|\epsilon)^* . (a) | (a) = (b|\epsilon)^* (a) = b^*a
\]
\[
R_{21}^1 = R_{21}^0 (R_{11}^0)^* R_{11}^0 | R_{21}^1 = (a) . (b|\epsilon)^* . (b|\epsilon) | (a) = ab^*
\]
\[
R_{22}^1 = R_{21}^0 (R_{11}^0)^* R_{12}^0 | R_{22}^1 = (a) . (b|\epsilon)^* . (a) | (b|\epsilon) = a(b^*a | \epsilon) | (a|\epsilon) = ab^*a | b | \epsilon
\]

Expressions for \(k = 2\)

\[
R_{11}^2 = R_{12}^1 (R_{22}^1)^* R_{21}^1 | R_{11}^2 = (b^*a) . (ab^*a | b | \epsilon)^* . ab^* | b^*
\]
\[
R_{12}^2 = R_{12}^1 (R_{22}^1)^* R_{12}^1 | R_{12}^2 = (b^*a) . (ab^*a | b | \epsilon)^* . (ab^*a | b | \epsilon) | b^*a
\]
\[
R_{21}^2 = R_{12}^1 (R_{22}^1)^* R_{21}^1 | R_{21}^2 = (b^*a) . (ab^*a | b | \epsilon)^* . ab^* | ab^*
\]
\[
R_{22}^2 = R_{12}^1 (R_{22}^1)^* R_{22}^1 | R_{22}^2 = (b^*a) . (ab^*a | b | \epsilon)^* . (ab^*a | b | \epsilon) | (ab^*a | b | \epsilon)
\]

\[
L = R_{12}^2 = (b^*a) . (ab^*a | b | \epsilon)^* . (ab^*a | b | \epsilon) | b^*a
\]

As can be seen the simplification of any of these regular expressions beyond the expressions for \(k=1\) is fairly complicated. This method, although correct by design leads to regular expressions that are far from being a minimal or most compact representation of the regular language a given finite automaton can recognize.
Problem 3 [10 points] Let \( L \) be a regular language over a finite alphabet \( \Sigma \). Show that the language consisting of all strings not in \( L \) over the same alphabet is also regular.

Solution: If \( L \) is a regular language then there exists a DFA \( M_1 \) that recognizes it. Now given \( M_1 \) we can construct a DFA \( M_2 \) that recognizes the complement of \( L \) with respect to the input alphabet \( \Sigma \) thus showing that the language recognized by \( M_2 \) is also regular. The DFA \( M_2 \) is a replica of \( M_1 \) but converting all accepting states to non-accepting states and vice versa. The starting state of \( M_2 \) is the same state as in the original \( M_1 \) DFA. Each input string is only accepted by \( M_2 \) iff the path spelled by its characters in the DFA leads to an accepting state. This means that by construction the same string would not be accepted in \( M_1 \). Conversely, the input string spells out a path to a non-accepting state in \( M_2 \), than it would spell out a path to an accepting state in \( M_1 \). In effect \( M_2 \) recognized the string not in the language recognized by the original DFA \( M_1 \) thus proving our claim.

Problem 4 [20 points]: Draw a DFA capable of recognizing the set of all binary-valued string (i.e., over the alphabet \( \Sigma = \{0,1\} \)) which when interpreted as a decimal number correspond to multiples of 4. Note than the string may have an arbitrary number of leading zero characters.

Solution: A structured approach to this problem consists in tracking the “remainder” of the division of the string seen so far by 4. For example, when the DFA has seen the string “10” the current interpretation of the number is the decimal 2 and so exhibiting a remainder of 2. If a “1” is seen the “value” is doubled as it corresponds to a shift to the left of the decimal interpretation of the input string and a digit “1” will correspond to an addition. This means that the value 2 is now \( 2 \times 2 + 1 \) that is 5 and the remainder is 1. As such the DFA should not accept the current string and thus be in a non-accepting state. If, however, from the state where we have seen “10” we observe a “0” in terms of the remainder we would have \( 2 \times 2 + 0 \) that is 4 and thus a remainder of 0. We would have to be in an accepting state.

The figure above depicts to DFA implementations for two different interpretations of this problem. The DFA on the LHS assumes the number to be recognized is non-zero, whereas the DFA on the RHS accepts a string whose decimal value interpretation is zero. As such the later DFA will accept the string “0”, “00”, “000” and so forth.