Problem 1 [50 points]: Consider the alphabet $\Sigma = \{0,1\}$.

a) Consider the Non-Deterministic Finite Automaton (NFA) depicted below. Why is this automaton non-deterministic? Explain the various source on indeterminacy.

b) Do the sentences $w_1 = "111"$ and $w_2 = "10"$ belong to the language generated by this FA? 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) This FA is a NFA as some states have more than one transition on the same alphabet symbol. For example, state 1, has two outgoing edges labelled '0' and two outgoing edges labelled '1'.

b) Regarding the word “111” there is a path from state 0 to the accepting state 4, namely: 0,1,2,4. Regarding the word “10” the automaton will either end up in state 2 or in state 3, neither of which is an accepting state.

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

\[
\begin{align*}
S_0 &= \epsilon\text{-closure } \{0\} = \{0\} \text{ -- this is not a final state.} \\
DFA_{\text{edge}}(S_0,0) &= \epsilon\text{-closure } (\text{goto}(S_0, 0)) = \{0\} = S_0 \\
S_1 &= DFA_{\text{edge}}(S_0,1) = \epsilon\text{-closure } (\text{goto}(S_0, 1)) = \{1\} \\
S_2 &= DFA_{\text{edge}}(S_1,0) = \epsilon\text{-closure } (\text{goto}(S_1, 0)) = \{2,3\} \\
DFA_{\text{edge}}(S_1,1) &= \epsilon\text{-closure } (\text{goto}(S_1, 1)) = \{2,3\} = S_2 \\
S_3 &= DFA_{\text{edge}}(S_2,0) = \epsilon\text{-closure } (\text{goto}(S_2, 0)) = \{2,3,5\} \text{ -- final state} \\
S_4 &= DFA_{\text{edge}}(S_2,1) = \epsilon\text{-closure } (\text{goto}(S_2, 1)) = \{2,3,4\} \text{ -- final state} \\
S_5 &= DFA_{\text{edge}}(S_3,0) = \epsilon\text{-closure } (\text{goto}(S_3, 0)) = \{2,3,5,6\} \text{ -- final state} \\
DFA_{\text{edge}}(S_3,1) &= \epsilon\text{-closure } (\text{goto}(S_3, 1)) = \{2,3,4,6\} = S_6 \text{ -- final state}
\end{align*}
\]
S6 = DFAedge(S4,0) = ε-closure (goto(S4, 0)) = \{2, 3, 5, 6\} = S5 – final state
DFAedge(S4,1) = ε-closure (goto(S4, 1)) = \{2, 3, 4, 6\} = S6 – final state

DFAedge(S5,0) = ε-closure (goto(S5, 0)) = \{2, 3, 5, 6\} = S5 – final state
DFAedge(S5,1) = ε-closure (goto(S5, 1)) = \{2, 3, 4, 6\} = S6 – final state
DFAedge(S6,0) = ε-closure (goto(S6, 0)) = \{2, 3, 5, 6\} = S5 – final state
DFAedge(S6,1) = ε-closure (goto(S6, 1)) = \{2, 3, 4, 6\} = S6 – final state

This results in the DFA shown below with starting state S0 = \{0\}.

(d) We can try to minimize this DFA by using the iterative refinement partitioning described in class. The figure below depicts a possible sequence of refinements. For each step we indicate the criteria used to discriminate between states in the previous partition.

As can be seen this DFA also accepts the input string "111" and rejects the input string "10".
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 it as much as possible.

Solution:

a) This automaton recognizes all non-null strings over the \{a,b\} alphabet that end with and odd number of "b" characters.
b) The derivations are shown below with the obvious simplification performed as early as possible to reduce expression length.

Expressions for k = 0

- \( R^0_{11} = (a | \epsilon) \)
- \( R^0_{12} = (b) \)
- \( R^0_{13} = \emptyset \)
- \( R^0_{21} = (a) \)
- \( R^0_{22} = (\epsilon) \)
- \( R^0_{23} = (b) \)
- \( R^0_{31} = (a) \)
- \( R^0_{32} = (b) \)
- \( R^0_{33} = (\epsilon) \)

Expressions for k = 1

- \( R^1_{11} = R^0_{11} (R^0_{11})* R^0_{11} | R^0_{11} \)
- \( R^1_{12} = R^0_{12} (R^0_{11})* R^0_{12} | R^0_{12} \)
- \( R^1_{13} = R^0_{13} (R^0_{11})* R^0_{13} | R^0_{13} \)
- \( R^1_{21} = R^0_{21} (R^0_{11})* R^0_{21} | R^0_{21} \)
- \( R^1_{22} = R^0_{22} (R^0_{11})* R^0_{22} | R^0_{22} \)
- \( R^1_{23} = R^0_{23} (R^0_{11})* R^0_{23} | R^0_{23} \)
- \( R^1_{31} = R^0_{31} (R^0_{11})* R^0_{31} | R^0_{31} \)
- \( R^1_{32} = R^0_{32} (R^0_{11})* R^0_{32} | R^0_{32} \)
- \( R^1_{33} = R^0_{33} (R^0_{11})* R^0_{33} | R^0_{33} \)

Expressions for k = 2

- \( R^2_{11} = R^1_{12} (R^2_{22})* R^1_{21} | R^1_{11} \)
- \( R^2_{12} = R^1_{12} (R^2_{22})* R^1_{22} | R^1_{12} \)
- \( R^2_{13} = R^1_{12} (R^2_{13})* R^1_{23} | R^1_{13} \)
- \( R^2_{21} = R^1_{22} (R^2_{22})* R^1_{21} | R^1_{21} \)
- \( R^2_{22} = R^1_{22} (R^2_{22})* R^1_{22} | R^1_{22} \)
- \( R^2_{23} = R^1_{22} (R^2_{23})* R^1_{23} | R^1_{23} \)
- \( R^2_{31} = R^1_{32} (R^2_{22})* R^1_{31} | R^1_{31} \)
- \( R^2_{32} = R^1_{32} (R^2_{23})* R^1_{32} | R^1_{32} \)
- \( R^2_{33} = R^1_{32} (R^2_{23})* R^1_{23} | R^1_{33} \)

Expressions for k = 3

...
Expressions for \( k = 3 \)

\[
R_{11}^3 = R_{13}^2 (R_{33}^2) * R_{31}^2 | R_{11}^2 = ... \\
R_{12}^3 = R_{13}^2 (R_{33}^2) * R_{32}^2 | R_{12}^2 = (a)^*(b)((a)^*(b) | (\epsilon)) * (b)((a)^*(b) | (\epsilon)) * (b) | (\epsilon)) * (a)^*(b) \\
(\epsilon)) | (a)^*(b) | ((a)^*(b) | (\epsilon)) * (a)^*(b) | (\epsilon)) * (a)^*(b) | (\epsilon)) | (a)^*(b)) \\
R_{13}^3 = R_{13}^2 (R_{33}^2) * R_{33}^2 | R_{13}^2 = ... \\
R_{21}^3 = R_{23}^2 (R_{33}^2) * R_{31}^2 | R_{21}^2 = ... \\
R_{22}^3 = R_{23}^2 (R_{33}^2) * R_{32}^2 | R_{22}^2 = ... \\
R_{23}^3 = R_{23}^2 (R_{33}^2) * R_{33}^2 | R_{23}^2 = ... \\
R_{31}^3 = R_{33}^2 (R_{33}^2) * R_{31}^2 | R_{31}^2 = ... \\
R_{32}^3 = R_{33}^2 (R_{33}^2) * R_{32}^2 | R_{32}^2 = ... \\
R_{33}^3 = R_{33}^2 (R_{33}^2) * R_{33}^2 | R_{33}^2 = ... \\
\]

\[
L = R_{12}^3 = (a)^*(b)((a)^*(b) | (\epsilon)) * (b)((a)^*(b) | (\epsilon)) * (b) | (\epsilon)) * (a)^*(b) | (\epsilon)) | (a)^*(b) | ((a)^*(b) | (\epsilon)) * (a)^*(b) | (\epsilon)) * (a)^*(b) | (\epsilon)) | (a)^*(b)) \\
\]

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 even the most compact representation of the regular language the DFA recognizes.

**Problem 3 [20 points]** As mentioned in class there is an inherent limitation of DFAs to handle inputs with a specific structure, as is the case of recognizing valid instances of the "balanced parenthesis" problem. This is due to the limitations of finite automata to "count" or "keep track". Still, there seems to exist problems that involve counting that are within reach of finite automata. This is the case of the problem in which we would like to recognize that a specific input string has 3 consecutive identical characters (say over the alphabet \{0,1\}).

How do you explain this apparent paradox? Present a DFA (or NFA) that checks if an input string over the alphabet \{0,1\} of length greater than 3 has exactly 3 consecutive identical symbols and explain which can this FA "count" but there is no FA that can handle the matching balanced parenthesis problem.

**Solution:**

There is really no paradox here. FA can count a finite number of precise "quantities". For the example at hand where you are asked about determining if a given string has exactly 3 consecutive identical symbols, a state of the FA can "track" what is the current number of consecutive states and once the fixed value of 3 is reached, the FA simply enters an accepting state (see the DFA below).

![DFA Diagram](image-url)

On the other hand for the balanced parenthesis problem, an FA would have to continuously monitoring the number of unbalanced parenthesis counting down from zero to a possibly infinite number to keep track of the various distinct levels of "nesting" of parenthesis. As this number can be arbitrarily large, at a given point you would run out of states as this is the only way a FA has to store state. The number of thee distinct states is unbound making it impossible for a FA to keep track.