1. The word "dermatoglyphics" is one of the longest words in English in which no letter appears more than once!

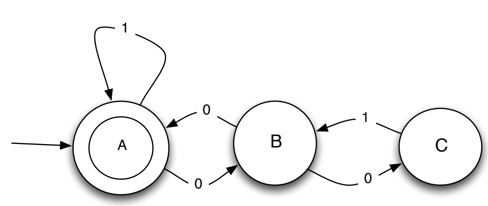
   For this problem, I would like you to finish describing two families of finite automata that recognize the languages, R_Σ, of strings over an alphabet Σ that are not like "dermatoglyphics". That is, the machines you describe should recognize the languages of all strings over their alphabets that include at least one symbol that appears repeatedly. You should not make any assumptions about the size or contents of the alphabet over which these strings are formed. As a result, it will not be possible to simply draw diagrams of these machines. Instead, you should give formal descriptions of the machines.

   (a) Last week, you described a family of deterministic finite automata, D_{R_Σ}, such that D_{R_Σ} recognized the language of all strings over the alphabet Σ that include at least two copies of some symbol in Σ.

   (b) This week, I want you to describe a family of non-deterministic finite automata, N_{R_Σ}, such that N_{R_Σ} recognizes the language of all strings over the alphabet Σ that include at least two copies of some symbol in Σ. In addition to describing the machine, provide a formula expressing the relationship between the size of the alphabet $|\Sigma|$ and the state set Q used by N_{R_Σ}. Try to design your machines so that its state sets is as small as possible.

   You should provide a formal description of the automata (i.e., specify the members of the tuple $(Q, \Sigma, \delta, s, F)$ that describe the automaton) rather than a state diagram. This formal description should be accompanied by a bit of concise prose (and possibly a diagram) that explains the intuition behind your construction.

2. Produce a DFA equivalent to the NFA shown below using the subset construction described on pages 55-58 of Sipser. Label the states of your DFA with the subset of the letters A, B, and C that corresponds to the subset they represent. Only show the reachable states (including the state corresponding to the empty set of NFA states), but show edges for all transitions.



3. In class I described a transformation

$$L_{\frac{1}{2}} = \{x \mid \text{for some } y \in \Sigma^*, xy \in L \ \& \ |x| = |y|\}$$

that can be applied to any language over the alphabet Σ. I stated that regular languages are closed under this transformation and outlined two approaches to proving this claim. Like most proofs of this sort, both approaches involved assuming we were given a DFA $D = (Q, \Sigma, \delta, s, F)$ that recognized some regular language L and then showing how to construct an NFA N based on D that recognized $L_{\frac{1}{2}}$. I provided a precise description of how to construct the desired NFA for the first approach in class.

I described a second approach (which is summarized below) informally, but I did not show how to formally describe the NFAs imagined in this second approach. For this problem, I want you to give a formal description of how to construct NFAs that use the second approach.

In the second approach, I suggested the machine N could initially guess a final state f which D could reach after processing all the symbols in x and a guessed string y whose length was equal to x. N, however, would not attempt to guess the contents of y at the same time that it guesses f. Instead, as the symbols of x were read one by one, the machine would guess the symbols of y one by one. While it would read the symbols of x in forward order it would guess the symbols of y in reverse order. In addition to guessing the symbols of y, for each letter of y guessed the machine would also guess the state the machine D would have been in just before reading that symbol in such a way that its guess was consistent with the state f and the symbols of y it had already guessed. Basically, the machine would simulate the execution of M running backward on y at the same time it simulates the execution of D running forward on x.

To verify that all of its guessing is correct, the machine must be designed to check that the two simulations of D end up in the same state.

4. For any language L over alphabet Σ, consider the derived languages

$$L_{--\frac{1}{3}} = \{w \mid \text{ for some } x, y \in \Sigma^*, |x| = |y| = |w| \text{ and } xyw \in L\}$$
$$L_{-\frac{1}{3}-} = \{w \mid \text{ for some } x, y \in \Sigma^*, |x| = |y| = |w| \text{ and } xwy \in L\}$$
$$L_{\frac{1}{3}-\frac{1}{3}} = \{w \mid \text{ for some } x, y, z \in \Sigma^*, |x| = |y| = |z|, w = xz \text{ and } xyz \in L\}$$

Regular languages are closed under at least one of these three transformations. At the same time, regular languages are not closed under at least one of these transformations. Identify one of the transformations that preserves regularity and justify your claim that regular languages are closed under this transformation by providing a formal description of how to transform a DFA, M, such that $L = L(M)$ into a new automaton (probably a NFA) M' that recognized the transformed language ($L_{--\frac{1}{3}}$, $L_{-\frac{1}{3}-}$, or $L_{\frac{1}{3}-\frac{1}{3}}$). Provide a brief but complete explanation of the intuition behind your construction.

5. (Sipser 1.18 f, i, and l) Give regular expressions generating the languages described in parts f, i, and l of Sipser's exercise 1.6. For the second problem, assume that the first digit of the input is considered an odd position.

f) $\{w \mid w$ does not contain the substring 110 $\}$

i) $\{w \mid$ every odd position of w is a 1 $\}$

l) $\{w \mid w$ contains an even number of 0s, or contains exactly two 1s $\}$