Problems for Week Seven

DFAs, States, and Information

DFA design is a skill you can build over time. It takes a bit of practice to get the hang of it. To assist you in getting started, we'd like you to walk through the following problems. Consider the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$ and consider the following language L_1:

$$L_1 = \{ w \in \Sigma^* \mid |w| \equiv_5 3 \}$$

(Recall that $|w|$ denotes the length of a string).

i. Describe the language L_1 in your own words. Give three examples of strings over Σ that are in L_1 and three examples of strings over Σ that are not in L_1.

ii. Design a DFA for L_1. To test it, try running it on the six test cases you developed in part (i) of this problem and make sure it works correctly.

One of the intuitions we gave for DFA design is that the only "memory" the DFA has is what current state it's in. This means that it's often useful to design DFAs so that every state corresponds to some piece of information the automaton remembers about the string it's seen so far.

iii. Look over your DFA from part (ii). For each state in that DFA, think about what strings will end up in that state. What do they all have in common? Based on this, see if you can come up with an explanation for what piece of information each state in the DFA remembers.

iv. The smallest DFA that you can build for this language has exactly five states. If your DFA from part (ii) has fewer than five states, you should double-check it because it means that your automaton is incorrect. ☺ If you have more than five states, see if you can find a smaller DFA for the language. If you do, repeat your analysis from part (iii). What piece of information does each state correspond to?

Now, let's consider another language over Σ, which we'll call L_2:

$$L_2 = \{ w \in \Sigma^* \mid w \text{ has an odd number of } a\text{'s and a number of } b\text{'s that's a multiple of three } \}$$

Previously, we had you design a DFA for a language, then reason about what each of the states means. Now, we're going to ask you to do this in reverse order.

v. Any DFA for L_2 will use it states to remember some piece of information about the string it's seen so far. What pieces of information would a DFA for L_2 need to remember about the string that it's seen so far? As a hint, think about our DFA from class for the language of all strings with a number of 1's that's congruent to two modulo three.

vi. Based on your answer to part (v), try designing a DFA for L_2 in the following way: create one state for each possible piece of information the DFA would need to remember about the string it's seen so far, then add in transitions based on how the information changes upon seeing one more character. As a hint, the DFA should only need to remember one of six different pieces of information.

Designing DFAs

Below are a list of alphabets Σ and languages over those alphabets. For each language, design a DFA for that language.

i. Let Σ = {`a`, `b`} and let L = {`baa`}. Design a DFA for L.

ii. Let Σ = {`a`, `b`} and let L = { $w \in \Sigma^*$ | $w \neq \varepsilon$ and the first and last character of w are the same }. Design a DFA for L.

iii. Let Σ = {`a`, `b`} and let L = { $w \in \Sigma^*$ | w is a nonempty string whose characters alternate between `a`'s and `b`'s }. Design a DFA whose language is L.

Designing NFAs

Below are a list of alphabets Σ and languages over those alphabets. For each language, design an NFA for that language.

i. Let Σ = {`a`, `b`, `c`} and let L = { $w \in \Sigma^*$ | w ends in `cab` }. Design an NFA for L.

ii. Let Σ = {`a`, `b`, `c`} and let L = { $w \in \Sigma^*$ | some character in Σ appears at most twice in w }. Design an NFA for L. As a note, you're going to want to use the guess-and-check model here; any DFA for this language will require at least 64 states. *(Hint: What would you do if you knew what character was going to appear at most twice?)*

Automata Transformations

In lecture, we saw three transformations on automata and regular expressions. This problem explores two of those transformations.

i. Let Σ = {`a`, `b`} and let L = { $w \in \Sigma^*$ | the third-from-last character of w is `a` }. Design an NFA for L. Your NFA should use at most four states.

ii. Using the subset construction, convert your NFA from part (i) into an equivalent DFA. (Although you could just directly design a DFA for this language, we want you to practice using the subset construction to get a sense of how it works.) You may find it useful to construct the transition table for the DFA rather than to write out a state-transition diagram.

Closure Properties

We've seen many examples of closure properties of regular languages. The proofs of those closure properties tend to fall into one of two different categories. First, there are proofs that work by directly converting automata or regexes for some languages into an automaton or regex for the resulting language. Second, there are proofs that work by applying other closure properties in various combinations. In this problem, we'd like you to prove another closure property of the regular languages. Remember that you have lots of different tools at your disposal!

Prove that the regular languages are closed under subtraction. That is, prove that if L_1 and L_2 are regular languages over some alphabet Σ, then the language $L_1 - L_2$ is also regular.