CS 4510: Automata and Complexity

Spring 2015

Home work 2 // Due: Friday, January 30, 2015

Sagar Laud, slaud3, 902792910

1. (10 points) (Problem from Sipser text.)

   An **all**-NFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ that accepts a string $x \in \Sigma^*$ if *every* possible state that M could be in after reading input x is a state from F. Note, in contrast, that an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove that **all**-NFAs recognize the class of regular languages.

   To prove this solution, we actually have to prove two different directions. We need to prove that all-NFAs only accept regular languages and that every regular language is recognized by some all-NFA.

   There are two different ways to prove the first direction. I will attempt both of them. One way to prove the first direction is by relying on the closure properties of regular languages. We begin by establishing an NFA $X = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$. We now examine the closure properties of regular languages. The class of regular languages are closed under complement, this was proved in class (and is actually used in the next solution). Now, because of this, if we can somehow design a second NFA that recognizes all of the languages that X does not, then because of closure properties, we have actually proven that the set of languages that X does recognize is the set of regular languages. Basically, if we have an NFA which recognizes a language L, then our new NFA, let us call it K, will recognize $\bar{L}$. This is crucial. Because $\bar{L}$ can be recognized by an NFA, it is by definition regular. In turn, because $\bar{L}$ is regular, L is also regular simply because the class of regular languages is closed under complement! To make this concrete, we simply need to fully define K. Let $K = (Q, \Sigma, \delta, q_0, F)$ such that:

   - $Q = Q_1$
   - $\Sigma = \Sigma_1$
   - $\delta = \delta_1$
   - $q_0 = q_1$
   - $F = Q_1 - F_1$

   Essentially, what we are doing is simply flipping the accept and reject states from before, in the same manner that we converted a DFA which recognized a language L to recognize $\bar{L}$. That is all that has been done. Now, looking at the definition for our machine K, we see that it will behave identically to our machine X except that it will reject every single string that X recognizes. As a result, we have shown that if X recognizes a language L, then K must recognize $\bar{L}$. Because of this, $\bar{L}$ is indeed regular and through closure properties, L must be regular as well. Therefore, we have shown that each and every regular language that is used is either recognized by X or K. Given that one recognizes the complement of the other and both are still finite automatons, L and $\bar{L}$ must be regular. We have therefore proven that all-NFAs recognize the class of regular languages.

   Now for the second way to prove the first direction. To prove that all-NFAs only accept regular languages, we will convert an all-NFA into a DFA. If we can show that every all-NFA can be converted to a DFA, we prove that all-NFAs recognize regular languages. So, to do this, let us first define an all-NFA $N = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$. Now, we simply need to show how to convert this into a DFA. For this, let us define our DFA $M = (Q, \Sigma, \delta, q_0, F)$

   - $Q = P(Q_1)$
   - $\Sigma = \Sigma_1$
   - For the transition function, we must take the union over all subsets for each character that we could possibly transition with. So, for any $K \subseteq Q_1$, with a being the character read, $\delta(K, a) = \cup\, \delta_1(k, a)$ for all $k \in K$
   - $q_0 = q_1$
   - This also requires some thought. Because this is an all-NFA, for our accept states, we must choose all states that are in Q as well as those that were in F_1. So, $F = \{x \text{ where } x \in Q \text{ and } x \subseteq F_1\}$.

   Now, we must prove the other direction. This is actually significantly easier simply because of the definition of a DFA. We wish to prove that all regular languages are accepted by some all-NFA. This is obviously true as a DFA is an all-NFA by definition. In order for a DFA to accept a string, it must end in a single accept state. In doing so, if a string is accepted by a DFA, by definition every possible state that it could be in is an accept state. As a result, this half of the proof is complete.

Sagar Laud, slaud3, 902792910

2. (10 points)

Let L be a regular language. Is the language L_1 defined below regular? Why?

$L_1 = \{w \in \{0,1\}^* \mid w \text{ is either in } L^R \text{ or in } \bar{L} \text{ but not in both}\}$.

This language is indeed regular. The reasoning for this is rather simple and we can explain it verbally or use a proof by construction. Let us start with a verbal explanation and then move to a proof by construction to bring it home. We proved in class that the set of regular languages is closed under the reverse and complement operations. We will define an NFA and a DFA for L^R and $\bar{L}$ verbally since the proofs of closure for these operations were done in class. We start by defining one machine which recognizes the reverse of a regular language. If we have a DFA $M = (Q, \Sigma, \delta, q_0, F)$ which accepts L, we can easily create an NFA which recognizes its reversal. To do this, we take our DFA and convert it to an NFA. We reverse our start and end states, in that our initial state from M is now our accept state. Likewise, all of our accept states in M become our new start states. From there, we simply create a new start state which epsilon transitions to all of our new start states. Lastly, we reverse all of the transitions for our DFA so that they go in the opposite direction on the same input as before. It is simple to visualize that this will work, as now if we reverse a string, all paths from the initial DFAs end states to initial states will now be explored nondeterminstically. Our new machine, lets call it R, will only accept if our string s is the reverse of a string which was accepted by M, which means it accepts the reverse of L. Therefore, the class of regular languages is closed under reversal.

Now let us look at the complement operation. This is an even simpler task. Let us consider a DFA N which recognizes L. Now, looking at this DFA, we can easily construct a DFA which recognizes the complement of L. To do this, we simply take every state in N that is an accept state, and make it a non-accepting state. We then take all states that are non-accepting states, and make them into accepting states. In this way, we construct a DFA, let us call it C, that accepts the complement of L.

Finally, we can use a little bit of nondeterminism to our advantage. We create an NFA X which has a start state which episilon transitions to the start states for R and C which we defined previously. In this way, we have constructed an NFA which recognizes the reverse of L and the complement of L simultaneously. An important distinction however is that we accept a string s if and only if it is accepted by one of the machines at a time, not both.

Formally, let $R = (Q_1, \Sigma_1, \delta_1, q_r, F_1)$. Let $C = (Q_2, \Sigma_2, \delta_2, q_c, F_2)$. Then, $X = (Q, \Sigma, \delta, q_0, F)$

- $Q = Q_1 \times Q_2$
- $\Sigma = \Sigma_1 \times \Sigma_2$
- Let a be an element of Σ and be the symbol read in at every step. Let x and y be elements of Q and be the current states of R and C respectively. Then, $\delta((x, y), a) = (\delta_1(x, a), \delta_2(y, a))$
- $q_0 = (q_r, q_c)$
- $F = (F_1 \times (Q_2 - F_2)) \cup ((Q_1 - F_1) \times F_2)$

In doing this, we have constructed an NFA which can recognize our language L_1. Because we have designed an NFA which can recognize L_1, we have proven that L_1 is regular.

Sagar Laud, slaud3, 902792910

3. (10 points) Consider the DFA $M = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$, where δ is as defined below:

$$\begin{aligned} \delta(q_1, 0) &= q_1 \\ \delta(q_1, 1) &= q_2 \\ \delta(q_2, 0) &= q_2 \\ \delta(q_2, 1) &= q_2 \end{aligned}$$

Convert this into a regular expression using the algorithm covered in class. In particular, show the following regular expressions: $R_{ij}^{(k)}$ for $1 \leq i, j \leq 2$ for $k = 0, 1$ and $R_{12}^{(2)}$. (Simplify the regular expressions using the rules they obey.)

$R_{1,1}^0 = 0 \cup \epsilon$
$R_{2,1}^0 = \emptyset$
$R_{1,2}^0 = 1$
$R_{2,2}^0 = 0 \cup 1 \cup \epsilon$

$R_{1,1}^1 = (0 \cup \epsilon) \cup ((0 \cup \epsilon) \circ (0 \cup \epsilon)^* \circ (0 \cup \epsilon))$
$R_{1,1}^1 = (0 \cup \epsilon) \cup ((0 \cup \epsilon)^*)$
$R_{1,1}^1 = (0 \cup \epsilon)^*$
$R_{1,1}^1 = 0^*$

$R_{2,1}^1 = \emptyset \cup (\emptyset \circ (0 \cup \epsilon)^* \circ (0 \cup \epsilon))$
$R_{2,1}^1 = \emptyset \cup (\emptyset \circ 0^* \circ (0 \cup \epsilon))$
$R_{2,1}^1 = \emptyset \cup (\emptyset \circ (0 \cup \epsilon))$
$R_{2,1}^1 = \emptyset \cup (\emptyset)$
$R_{2,1}^1 = \emptyset$

$R_{1,2}^1 = 1 \cup ((0 \cup \epsilon) \circ (0 \cup \epsilon)^* \circ 1)$
$R_{1,2}^1 = 1 \cup ((0 \cup \epsilon)^* \circ 1)$
$R_{1,2}^1 = 1 \cup (0^* \circ 1)$
$R_{1,2}^1 = 0^* \circ 1$

$R_{2,2}^1 = (0 \cup 1 \cup \epsilon) \cup (\emptyset \circ (0 \cup \epsilon)^* \circ 1)$
$R_{2,2}^1 = (0 \cup 1 \cup \epsilon) \cup (\emptyset \circ 0^* \circ 1)$
$R_{2,2}^1 = (0 \cup 1 \cup \epsilon) \cup (\emptyset \circ 1)$
$R_{2,2}^1 = (0 \cup 1 \cup \epsilon) \cup (\emptyset)$
$R_{2,2}^1 = (0 \cup 1 \cup \epsilon)$

$R_{2,2}^2 = (0^* \circ 1) \cup ((0^* \circ 1) \circ (0 \cup 1 \cup \epsilon)^* \circ (0 \cup 1 \cup \epsilon))$
$R_{2,2}^2 = (0^* \circ 1) \cup ((0^* \circ 1) \circ (0 \cup 1 \cup \epsilon)^*)$
$R_{2,2}^2 = (0^* \circ 1) \cup ((0^* \circ 1) \circ (0 \cup 1)^*)$
$R_{2,2}^2 = (0^* \circ 1) \circ (0 \cup 1)^*$

Sagar Laud, slaud3, 902792910

4. (15 points) Let L be a regular language. Suppose there exists a set of n pairs of strings $\{(x_i, y_i) \mid 1 \leq i \leq n\}$ such that the following two conditions hold:

   (a) $x_i y_i \in L$ for all $1 \leq i \leq n$, and

   (b) $x_i y_j \notin L$ for all $1 \leq i \neq j \leq n$.

   Show that any NFA for L must have n states.

   We start by defining an NFA for this situation. We then prove that it must have n states. So, let us define an NFA $X = (Q, \Sigma, \delta, q_0, F)$ as an NFA which recognizes our language L. Now, let us take a string from our language in the form $s = x_i y_i$. Now, let us look at our states in conjunction with our string. We know that s is accepted by our machine X. Therefore, we know that taking our start state q_0 and taking characters from x_i one by one will get us to one state. And then from that state, we can take y_i character by character and bring us to some accepting state. Formally put, $\hat{\delta}(q_0, x_i)$ ends with q_k. Then, if we start with q_k, $\hat{\delta}(q_k, y_i)$ ends with some accepting state. Now let us examine one more thing in order to prove that we must have n states in X. Let us consider strings $x_i y_i$ and $x_j y_i$ and refer to them as s_1 and s_2 respectively. If somehow we did not have distinct states for the result of $\hat{\delta}(q_0, x_i)$ and $\hat{\delta}(q_0, x_j)$, then these would end at the same state. But, the second half of the strings are now identical. So, our machine X would accept both s_1 and s_2 which is a problem as s_2 is not in our language L. As a result, for every single pairing in L, (essentially for every possible value of i), we must have a distinct state. Therefore, X must have atleast n states to accomplish this task.

5. (15 points) Identify the regular set and the non-regular set from the following two subsets of $\{0,1\}^*$. Prove your answer.

   (a) $B = \{1^k y \mid y \text{ contains at least } k \text{ 1s, for } k \geq 1\}$.

   This language is regular. Proving this is as simple as generating a regular expression for it. We can represent our language by starting off with a 1, any number of 0's, then a 1, and then any combination of 0's and 1's from there on out. We use the regular expression $1 \circ 0^* \circ 1 \circ (0 \cup 1)^*$ to describe our language. By defining a regular expression for our language, we have proven that our language is regular.

   (b) $C = \{1^k y \mid y \text{ contains at most } k \text{ 1s, for } k \geq 1\}$.

   This language is non-regular. To use this, we will use a proof by contradiction in conjunction with the pumping lemma. Let us start by assuming that our language is regular. We will quickly find that this isn't the case. By starting with this, we can use the pumping lemma. If the pumping lemma is not satisfied, we know that our language cannot be regular. According to the pumping lemma, let us define a pumping length p. So, we now take a string from our language. Let our string be w $= 1^p 01^p$. Now that we have created our string, the pumping lemma tells us that we can split w into 3 parts: x, y, and z such that for $i \geq 0, xy^i z$ is an element of L, $|y| > 0$, and $|xy| \leq p$.

   Now, let us consider all possible splits of our string where we start out meeting all conditions. To make this concrete, let us define a constant k. We then go on to say that our string is $1^k 1^{p-k} 01^p$. Let us first consider the situation where $x = \epsilon$. In doing this, $y = 1^k$ and $|xy| \leq p$. However, we see that once we pump down y (set $i = 0$), the number of 1's at the beginning of the string decreases and because y from the original string does not change, it is no longer an element of L. Since we've removed the possibility of the empty string, $x = 1^k$ and $y = 1^{p-k}$. We now bound k by saying that $0 \leq k \leq p - 1$ (k cannot equal p because $|y|$ must be greater than 0). So now, we use the remainder of the pumping lemma. Now, we begin to pump down. As we pump down to $i = 0$, the the number of 1's on the left side of the zero decreases and as a result ends up being less than the number of 1's on the right side of the zero. There is no valid split for which all of the conditions of the pumping lemma are met. As a result, the language cannot possibly be regular and must in fact be non-regular.

Sagar Laud, slaud3, 902792910

6. Let L be a language defined over a finite alphabet Σ. Two strings $x, y \in \Sigma^*$ are said to be *distinguishable* by L iff there exists some string z such that exactly one of xz or yz is in L.

   (15 points) Consider the language $L = \{x \in \Sigma^* \mid x = x^R\}$. (Here x^R is the reverse of the string x.) Show that any two distinct strings $x, y \in \Sigma^*$ are distinguishable by L.

   We will assume that $|\Sigma| > 1$. With this assumption, we really only need to look at two cases. The first case that we must look at is if $|x| = |y|$. In this case it is easy to show that the strings are distinguishable by L. If we define our string $z = x^R$, we can show that this is the case. Because the strings are distinct, $xz = xx^R$ is indeed in our language L. However, if we do the same with y we see that $yz = yx^R$ which cannot possibly be an element of L because y and x are distinct.

   Now we look at the only other case, which is where $|x|$ and $|y|$ are not equal. Now this case is a little more interesting and we need to add some additional fields. Let us define one extra field z_1 which we will append to the shorter of the two strings. Essentially, we have x and y which are distinct but we need to have them be the same length to ensure that when we append their reverse to them, only one is in L. Otherwise we could have some situation where appending x^R allows both strings to be palindromes. So, we append z_1 to the shorter of the two fields such that now, both fields have the same length but are still distinct. However this is guaranteed to be possible as the strings are already distinct, one must simply choose a z_1 that does not make the strings identical. Now, let us assume for our purposes that y was the shorter of the two strings. We append z_1 to y such that $|x| = |yz_1|$ and $x \neq yz_1$. Because the two strings are still distinct, we choose $z = x^R$. Because the two strings are distinct, $xz = xx^R$ and $yz_1z = yz_1x^R$. It is clear that xz is in L while yz_1z is not and as a result, any two distinct strings x and y are distinguishable by L.