Solutions of homework 3

1. Give a family of languages $A_k$, where each $A_k$ can be recognized by a $O(k)$-state DFA whereas $A^R_k$ requires $2^{\Omega(k)}$ states on a DFA. Prove that your languages have this property.

Solution Outline: (20 points)

Fix $\Sigma = \{0, 1\}$. Consider $A_k$ to be the set of strings over $\{0, 1\}$ of length at least $k$ containing a 1 in the $k$ position. Clearly, $A_k$ can be recognized by a $O(k)$-state DFA. On the other hand, we know from the previous problem set that $A^R_k$ requires $2^{\Omega(k)}$ states on a DFA.

(8 points for specifying a correct language; 6 points for $O(k)$ upper bound, 6 points for $2^{\Omega(k)}$ lower bound.)

2. (a) Design a two-way finite automaton with $O(k)$ states accepting the language $L_k = \{0, 1\}^* 1 \{0, 1\}^k \$, where $\Sigma = \{0, 1, \$\}$.

(b) Suppose that $w, v \in \Sigma^*$, $\chi_w = \chi_v$ and $\theta_w = \theta_v$. Show that, for any $u \in \Sigma^*$, $M$ accepts $wu$ iff $M$ accepts $vu$.

(c) Show that if $L$ is the language accepted by a deterministic two-way automaton, then $L$ is accepted by some ordinary (one-way) DFA.

(d) Conclude that there is an exponential-time algorithm, which given a deterministic two-way automaton $M$, constructs an equivalent DFA.

Solution Outline: (15, 20, 15, 30 points)

(a) The idea is to keep moving right on the input tape until reading a $\$, and then move left $k+1$ steps. If the reading head reads a 1, move all the way right and accept\(^1\). Otherwise, keep looping in a reject state.

Formally, take $Q = \{q_0, q_1, \ldots, q_{2k+3}, q_R\}$, $F = \{q_{2k+3}\}$, and take the transition function

\[
\delta(q_i, b) = \begin{cases} 
(q_0, \rightarrow) & \text{if } b \in \{0, 1\}, i = 0 \\
(q_1, \leftarrow) & \text{if } b = \$, i = 0 \\
(q_{i+1}, \leftarrow) & \text{if } b \in \{0, 1\}, i = 1, 2, \ldots, k \\
(q_{k+2}, \rightarrow) & \text{if } b = 1, i = k + 1 \\
(q_{i+1}, \rightarrow) & \text{if } b \in \{0, 1\}, i = k + 2, \ldots, 2k + 1 \\
(q_{2k+3}, \rightarrow) & \text{if } b = \$, i = 2k + 2 \\
(q_R, \rightarrow) & \text{otherwise}
\end{cases}
\]

(b) We begin with the simple case $u = \epsilon$. If $M$ accepts $wu = w$, then $\chi_w(q_0) = \chi_w(q_0)$ is an accepting state. Therefore, $M$ also accepts $v = vu$. The converse holds via symmetry.

\(^1\)A common mistake is to omit moving all the way to the right. This is necessary because in the definition of the 2-way DFA, we decide whether to accept by looking at the state of the machine is run when the head has first moved off the rightmost end of the input. It is also necessary to check that there is no 0’s and 1’s after the $\$, but we didn’t penalize students for omitting that check.
Next, we prove the more general statement that for all $u \in \Sigma^*$, $\chi_{wu} = \chi_{vu}$, after which we may deduce from the proof for the special case $u = \epsilon$ that for any $u \in \Sigma^*$, $M$ accepts $wu$ iff $M$ accepts $vu$.

For $u \neq \epsilon$, write $u = au'$, where $a \in \Sigma, u' \in \Sigma^*$. Fix $q \in Q$. Let us now examine the executions of $M$ on input $wu$ and on input $vu$, with initial configurations $(\epsilon, q, vu)$ and $(\epsilon, q, vu)$ respectively. Observe that $(\epsilon, q, vu) \vdash^*_M (w, \chi_{wu}(q), au')$ and that $(\epsilon, q, vu) \vdash^*_M (v, \chi_{w}(q), au')$. Now, advance both executions to the configurations $(w, \chi_{w}(q), au')$ and $(v, \chi_{w}(q), au')$, where $\chi_{w}(q_0) = \chi_{v}(q)$. Therefore, in both executions, $M$ is in the same state, and the reading head is reading the same input $a$, and about to move to the right.

The more general claim then follows from the following observations:

- As long as the reading head stays at $a$ or to the right of $a$, then $M$ makes the same transitions and stays in the same state in both executions. In particular, if the execution on input $wu$ halts, then both executions terminates in the same state.
- If the reading head moves back to $a$ in the execution on input $wu$, and is going to make a transition with direction $\leftarrow$, $M$ must be in the same state $q$ in both executions just before that transition. Now, if $q' = \theta_{wu}(q, a) = \theta_{v}(q, a) = t$, then we have $\chi_{wu}(q) = \chi_{vu}(q) = t$. Otherwise (that is, $q' \neq t$), advance the execution for input $wu$ to the configuration $(w, q', u)$ and for input $vu$ to the configuration $(v, q', u)$. At this point, we’re back in this case or the previous case, depending on whether the reading head is about to move left or right.\(^2\)

(c) Suppose $L$ is a language accepted by a deterministic two-way automaton $M$. We define an equivalence relation $\approx_M$ on strings in $\Sigma^*$, where $w \approx_M v$ iff $\chi_w = \chi_v$ and $\theta_w = \theta_v$. It is clear from (b) that $\approx_M$ is a refinement of $\approx_L$ (that is, $w \approx_M v \Rightarrow w \approx_L v$). Observe that there are at most $|Q|^{|\Sigma|}$ possible (distinct) functions for $\chi_w$ and $|Q|^{|\Sigma|}$ possible (distinct) functions for $\theta_w$. Therefore, $\approx_M$ has at most $|Q|^{|\Sigma|}$ equivalence classes. Since $\approx_M$ is a refinement of $\approx_L$, $\approx_L$ also has $O(|Q|^{|\Sigma|})$ (that is, finitely) equivalence classes. The Myhill-Nerode theorem then yields a DFA for the language $L$.

(d) METHOD I (due to David Songtag) We construct a DFA $(Q', \Sigma, \delta', q_0', F')$ that accepts the same language as the 2-way DFA:

- $Q'$ is set of pairs of functions $(\chi, \theta)$, where $\chi : Q \rightarrow (Q \cup \{t\})$ and $\theta : Q \times \Sigma \rightarrow Q \cup \{t\}$. In particular, $|Q'| = (|Q| + 1)^{|\Sigma|} \cdot (|Q| + 1)^{|\Sigma|}$.
- $q_0' = (\chi_0, \theta_0)$.
- For each $(\chi, \theta) \in Q'$ and $a \in \Sigma$, we set $\delta_0((\chi, \theta), a) = (\chi', \theta')$, where $\chi', \theta'$ are computed as follows as in part (b). The details for $\chi'$ is as follows: On input $q \in Q$:
  i. Set $p = \chi(q)$.
  ii. If $p = t$, return $t$. Else, set $(q', D) = \delta(p, a)$. If $D = \rightarrow$, return $q'$; else, set $p = \theta(q', a)$. If we detect a loop or have run for more than $|w| \cdot |Q|$ steps, return $t$ (refer to step 1 in method II). Return to ii.
- $F' = \{(\chi, \theta) \in Q' \mid \theta(q_0) \in F\}$.

We omit the details for proving correctness and efficiency.

METHOD II We divide the proof into several steps.

\(^2\)There is an inconsistency in the definition of $\theta_w(q, a)$ in the problem set, between the formal definition and the informal definition. The former is correct, and “last” should be replaced with “first” in the latter.
Step 1: It is easy to see that given \( w \in \Sigma^*, q \in Q \), \( \chi_w(q) \) can be computed in \( \text{poly}(|w|, |Q|, |\Sigma|) \) time for each \( q \in Q \) by simulating \( M \) on \( q \), and returning \( t \) if a loop is detected. Detecting a loop is simple since there are at most \( |w| \cdot |Q| \) distinct configurations for the computation of \( M \) on input \( w \). We can either keep track of all the configurations we have seen so far, or simulate up to at most \( |w| \cdot |Q| + 1 \) steps, and return \( t \) if the simulation has yet to complete. Hence, \( \chi_w \) and similarly \( \theta_w \) can be computed in \( \text{poly}(|w|, |Q|, |\Sigma|) \) time.

Step 2: Observe that we can extend the argument in (b) to prove that if \( w, v \in \Sigma^* \) satisfy \( \chi_w = \chi_v \) and \( \theta_w = \theta_v \) (that is, \( w \approx_M v \)), then for any \( u \in \Sigma^* \), we also have \( \chi_{wu} = \chi_{vu} \) and \( \theta_{wu} = \theta_{vu} \) (that is, \( wu \approx_M vu \)).

Step 3: It suffices to give an exponential-time algorithm that on input \( M \), computes a representative for each equivalence class for \( \approx_M \). The DFA will have a state for each equivalence class, the start state is the class \([\epsilon]\), and every state of the form \([x]\) for each representative \( x \) that \( M \) accepts. As shown in step 1, we can test whether \( M \) accepts \( x \) in time \( \text{poly}(|x|, |Q|, |\Sigma|) \). The transition function is \( \delta([x], a) = [xa] \). The proof of correctness is analogous that in the Myhill-Nerode theorem (the difference being that in the proof uses equivalence classes for \( \approx_L \); however, the same proof works because \( \approx_M \) is a refinement of \( \approx_L \), and because of the observation in step 2, needed to show that the transitions are well-defined).

Step 4: We construct representatives for \( \approx_M \) equivalence classes incrementally, where \( R_i \) contains the representatives of length \( i \). Let \( R_0 = \{\epsilon\} \). Given \( R_i \), we compute \( R_{i+1} \) by taking each string \( w \) in \( R_i \), and then computing \( \chi_{wu} \) and \( \theta_{wu} \) for all \( a \in \Sigma \). We add \( wa \) to to \( R_{i+1} \) if \( wa \) is not in the same \( \approx_M \) equivalence class as any string in \( R_0 \cup \cdots \cup R_i \) and any string previously added to \( R_{i+1} \). We terminate as soon as \( R_{i+1} = \emptyset \). The algorithm converges in \( O(2^{|Q|} |\Sigma| \log |Q|) \) steps, because it cannot create more than \( O(2^{|Q|} |\Sigma| \log |Q|) \) equivalence classes.

Now, we need to show that when the algorithm terminates, we have found representatives for all equivalence classes. Suppose the algorithm terminates with \( R_{i+1} = \emptyset \). By construction, every string in \( \Sigma^* \) of length at most \( i + 1 \) has a representative for its equivalence class. Assume on the contrary that there is some string \( w \in \Sigma^*, |w| > i + 1 \) for which there is no representative for \( |w| \). We may assume \( \text{WLOG} \) that \( w \) is the shortest such string (break ties arbitrarily). Write \( w = w_1 u \), where \( |w_1| = i + 1, u \in \Sigma^* \). Since \( R_{i+1} = \emptyset \), there exists some \( w_2, |w_2| < i + 1 \) such that \( w_2 \approx_M w_1 \). Then, by the observation in step 2, we have \( w = w_1 u \approx_M w_2 u \). However, \( |w_2 u| < |w_1 u| \), which contradicts the minimality of \( w \).

Finally, it is straight-forward to check that the overall algorithm runs in exponential time.

(6 points each for steps 1-3, 12 points for step 4)