Solutions of homework 4


Solution Outline: (30 points)

We outline (at a high level) two possible ways to go about simulating an arbitrary Turing machine $M$ with a Turing machine with left reset; the only tricky part is simulating a left move (we can, by expanding the state space, use state information to remember the new state that the machine is supposed to transition to) and we shall only focus on that.\(^1\)

- First idea: make a new copy of the contents of the tape at every left move, and then do our computations on the new copy. We mark the cell we are at, go all the way to the end of the tape, and put a # symbol, before copying the contents after the # symbol. When copying a cell whose right neighbor is marked, we know that this should be our new head position, so we leave an additional head-marker on the copied cell. When the copying is done, we reset the head, invalid the old copy, then move right until we reach the cell with the head-marker.

- Second idea: incremental crossing. Mark the current cell, and reset to the left. After that, we make multiple passes over the input by resetting, crossing out one more cell on each pass, starting with the left-most one. Each time we cross out a cell, we remember (using state information) whether it is 1 right, 2 right, or more than 2 right moves before we reach the marked cell. We stop crossing out cells when we are 2 right moves before reaching the marked cells. When we next reset, we uncross each cell, until we reach the first cell that was not crossed; that is the cell to the left of the marked one. Some minor modifications are necessary to handle the boundary cases (if the marked cell is one of the first 3 cells on the tape).


Solution Outline: (25 points)

It is easy to see that we can simulate any DFA on a Turing machine with stay put instead of left. The only non-trivial modification is to add transitions from state in $F$ to $q_{accept}$ upon reading a blank, and from states outside $F$ to $q_{reject}$ upon reading a blank.\(^2\)

Next, we start with a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ with stay put instead of left, and show how we can construct a DFA $(Q', \Sigma', \delta', q'_0, F)$ that recognizes the same language. The intuition here is that $M$ cannot move left and cannot read anything it has written on the tape as soon as it moves right, and therefore it has essentially only one-way access to its input, much like a DFA.

First, we modify $M$ as follows; note that these changes do not affect the language it recognizes.

\(^1\)Note that we cannot immediately apply the Church-Turing thesis to a Turing machine with left reset, and thus we cannot assume that a Turing machine with left reset can count, do arithmetic operations, or perform other non-trivial algorithmic tasks.

\(^2\)Note that constructing a NFA $N$ with a single accept state by adding $\epsilon$-transitions from all the accept states in the DFA to a new accept state does not work. This is because the stay put Turing machine is deterministic, so we will need to convert $N$ to a DFA, which will not preserve the property of having a single accept state.
• Add a new symbol so that $M$ never writes blanks on the tape; instead, $M$ writes the new symbol when it’s going to write blanks, and we extend the transition function so that upon reading this new symbol, it behaves as though it read a blank.

• When $M$ transitions into $q_{\text{reject}}$ or $q_{\text{accept}}$, the reading head moves right (and never stays put).

Set $Q' = Q$, $\Sigma' = \Sigma$, $q'_0 = q_0$, and consider the transition function:

$$\delta'(q, \sigma) = \begin{cases} 
q, & \text{if } q \in \{q_{\text{accept}}, q_{\text{reject}}\} 
q_{\text{reject}}, & \text{if } M \text{ starting at state } q \text{ and reading } \sigma \text{ keeps staying put.} 
q', & \text{where } q' \text{ is the state the } M \text{ enters when it first moves right upon starting at state } q \text{ and reading } \sigma.
\end{cases}$$

(for $q \in Q$ and $\sigma \in \Sigma$). Observe that there are finitely many state-alphabet pairs, $M$ either ends up either staying put and looping, or eventually moves right, and thus $\delta'$ is well-defined. Finally, we define $F$ to be the set containing $q_{\text{accept}}$ and all states $q \in Q, q \neq q_{\text{accept}}, q_{\text{reject}}$ such that $M$ starting at $q$ and reading blanks, eventually enters $q_{\text{accept}}$.

3. **Sipser problem 3.16.**

**Solution Outline:** (20 points)

If $A$ is decidable by some TM $M$, the enumerator operates by generating the strings in lexicographic order, testing each in turn for membership in $A$ using $M$, and printing the string if it is in $A$.

If $A$ is enumerable by some enumerator $E$ in lexicographic order, we consider two cases. If $A$ is finite, it is decidable because all finite languages are decidable (just hardwire each of the strings into the $TM$). If $A$ is infinite, a TM $M$ that decides $A$ operates as follows. On receiving input $w$, $M$ runs $E$ to enumerate all strings in $A$ in lexicographic order until some string lexicographically after $w$ appears. This must occur eventually because $A$ is infinite. If $w$ has appeared in the enumeration already, then accept; else reject.

Note: It is necessary to consider the case where $A$ is finite separately because the enumerator may loop without producing additional output when it is enumerating a finite language. As a result, we end up showing that the language is decidable without using the enumerator for the language to construct a decider. This is a subtle, but essential point.

4. Let $\text{PrefixFree}_{\text{REX}} = \{R \mid R \text{ is a regular expression where } L(R) \text{ is prefix-free}\}$. Show that $\text{PrefixFree}_{\text{REX}}$ is decidable.

**Solution Outline:** (25 points)

We construct a TM that decides $\text{PrefixFree}_{\text{REX}}$ as follows\(^3\). On input $R$, reject if $R$ is not a valid regular expression. Otherwise, construct a DFA $D$ for the language $L(R)$ (refer to chapter 1 of Sipser for the algorithm that constructs an equivalent NFA for $L(R)$ from $R$, and for the algorithm that converts an NFA to a DFA). By running a DFS starting from $q_0$, we can remove all states that are not reachable from $q_0$ from the automaton. Finally, for each accept state $q$, we run a DFS starting from $q$ and check if another accept state (not equal to $q$) is reachable from $q$, or if there is a loop from $q$ to itself. If any such paths or loops are found, reject. Otherwise, accept.

\(^3\)Note that $\text{PrefixFree}_{\text{REX}}$ can contain infinite languages. For instance, take $R = 0^*1$.  

2