1. (Sipser, Problem 3.13) A Turing machine with stay put instead of left is similar to an ordinary Turing machine, but the transition function has the form

\[ \delta : Q \times T \rightarrow Q \times T \times \{R, S\} \]

At each point the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is not equivalent to the usual version. (Hint: Show that these machines only recognize regular languages). [20 points]

Solution: 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 \( q_{\text{accept}} \) upon reading a blank, and from states outside \( F \) to \( q_{\text{reject}} \) upon reading a blank.

Next, we start with a Turing machine \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \) 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.

- 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 \( q_{\text{accept}} \).

2. (Sipser, Problem 3.18) Show that a language is decidable iff some enumerator enumerates the language in lexicographic order. [15 points]

Solution: 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.

3. Say that string $x$ is a prefix of string $y$ if a string $z$ exists where $xz = y$, and say that $x$ is a proper prefix of $y$ if in addition $x \neq y$. A language is prefix-free if it doesn’t contain a proper prefix of any of its members. Let

$$\text{PrefixFree}_{REX} = \{R | R \text{ is a regular expression where } L(R) \text{ is prefix-free}\}$$

Show that $\text{PrefixFree}_{REX}$ is decidable. [15 points]

Solution: We construct a $TM$ that decides $\text{PrefixFree}_{REX}$ as follows. 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. Note that it is first required to remove all the states (actually, just accepting states) not reachable from $q_0$ as these states cannot lead to any string being in the language.

4. Let $Non – Empty$ be the following language

$$Non – Empty = \{<M> | M \text{ accepts some string}\}.$$

Show that $Non – Empty$ is Turing recognizable. [10 points]

Solution: We simply proceed as in the construction of an enumerator from a Turing machine: simulate $M$ on all strings of length at most $i$ for $i$ steps, and keep increasing $i$. We accept if the computation of $M$ accepts some string. If $L(M)$ is non-empty, we are certain that for some $i$ our machine will halt and accept.

---

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

---

2