1. Show that recursive enumerable languages are closed under the union and Kleene star operations.

Solution:

Union: If L_1 and L_2 are r.e and accepted by M_1 and M_2. For a string w, proceed as follows: Use the universal TM $\mathcal{U}$ to simulate M_1 and M_2 in parallel, one step at a time. If either M_1 or M_2 enters an accept state or reject state on w, have $\mathcal{U}$ enter an accepted or reject state, respectively.

Kleene star: Brute-force checking for all possible splits of the input using timesharing.

2. Consider a finite state automata that has access to a queue (instead of a stack). First, formally define such a *queue automata* and its acceptance condition (like in the case of PDA). Show that if a language L is accepted by a Turing machine M, then there is a queue automata Q that accepts L.

Solution: The queue automata $P = (Q, \Sigma, \Gamma, \delta, s, F)$. The transition function $\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$ gives the value of $\delta(q, \sigma, \gamma) = (q', \alpha)$ where γ is the symbol on the front of the queue and $\alpha \in \Gamma^*$ is the sequence of symbols that are enqueued. We will assume that P accepts via final state. We will use a special symbol $\dashv$ to indicate the tape.

We will show how a queue automata can simulate the behaviour of a TM M. On an input x, P will first enqueue the string on to the queue so that the queue looks as follows $\hat{\vdash}, x_1, x_2, x_3, \cdots, x_n, \dashv$. The left endmarker $\vdash$ is in the front of the queue and the hat denotes the head position of the TM.

While simulating one step of the TM M, the automaton P will start dequeueing until it reads the symbol with the hat above it. It will then use the transition function of M to decide its action. P will use its state to remember the previous symbol as well as the current symbol - this is necessary if the transition of the TM requires the tape head to be moved left.

Suppose that before the simulation of Step i of the TM M, the queue looks as follows $\vdash\ w_1\, w_2\ \cdots\ \hat{w}_k\ \cdots\ w_n\ \dashv$. The automaton P stores the state of M using its state. P starts dequeueing and enqueueing the queue until the queue looks like

$$\hat{w}_k\ w_{k+1}\ \cdots\ w_n\ \dashv\ \vdash\ w_1\ w_2\ \cdots\ w_{k-2}$$

and P is storing w_{k-1} using its state. If the transition of M is moving the tape head left, then P enqueues $\hat{w}_{k-1}$. Then it moves the queue to make it

$$\vdash\ w_1\ w_2\ \cdots\ \hat{w}_{k-1}\ w_k\ \cdots\ w_n\ \dashv .$$

Similarly, if M is moving the tape head right, then P moves the queue to make it

$$\vdash\ w_1\ w_2\ \cdots\ w_k\ \hat{w}_{k+1}\ \cdots\ w_n\ \dashv .$$

One corner case to take care of is when the queue looks as follows.

$$\vdash \ w_1 \, w_2 \ \cdots \ \hat{w}_n \ \dashv .$$

If the TM M now moves the tape head right, the automaton P will enqueue $\hat{\sqcup}$ before enqueueing $\dashv$. Thus, the queue will then look as follows:

$$\vdash \ w_1 \, w_2 \ \cdots \ w_n \, \hat{\sqcup} \ \dashv .$$

The automaton P will keep simulating M. If M enters an accept state, then P enters its final state.

3. Let $L \subseteq \{0,1\}^*$ be any infinite r.e language. Show that L is recursive iff there is an enumeration Turing machine that lists the strings in L in lexicographic order.

Solution:

($\Rightarrow$) If L is recursive, then there is a total TM M that halts on all inputs and accepts x iff $x \in L$. Now, run M with strings starting from ε in lexicographic order. Since M halts on all strings, print those strings that M accepted.

($\Leftarrow$) Let E be the enumeration machine for L that prints strings in L in lexicographic order. Given a string x, run E until it either prints x (in which case, accept and halt) or it prints a string y lexicographically larger than x before printing x (in which case, reject and halt).

Verify why this works.

4. Let $L_1 \subseteq \{0,1\}^*$ be any language. Show that L_1 is r.e iff there is a recursive language L_2 such that $L_1 = \{x \mid \exists y \text{ such that } \langle x, y\rangle \in L_2\}$.

Solution:

($\Leftarrow$) Let L_1 be r.e and M be the TM such that $L = L(M)$. Define

$$L_2 = \{\langle x, y\rangle \mid M \text{ accepts } x \text{ in } y \text{ steps}\}.$$

Assume that the input alphabet is binary and y corresponds to the binary representation of a number.

Notice that L_2 is recursive because if we are given $\langle x, y\rangle$, we can use the universal TM to simulate M on x for y steps and check if M accepts x.

From the definition, verify that $L_1 = \{x \mid \exists y \text{ s.t } \langle x, y\rangle \in L_2\}$.

($\Rightarrow$) Suppose that $L_1 = \{x \mid \exists y \text{ s.t } \langle x, y\rangle \in L_2\}$ where L_2 is recursive. Let M be the total TM such that $L_2 = L(M)$. To obtain the TM M_1 for L_1, we will simulate M with input x and with strings y starting from ε. If for some y, M halts and accepts, M_1 will halt and accept.

5. Given a program in your favourite programming language, it would be nice to know if there are parts of the code that are never executed for any input. In the setting of Turing machines, let's phrase this question as follows: Given a TM machine M and a state q, does M ever reach the state q?

   Show that the language

   $$L = \{(\langle M\rangle, q) \mid \exists w \text{ such that } M \text{ on input } w \text{ enters the state } q\}$$

   is not recursive.

   Solution: To show that a language is not recursive, you can take any non-recursive language L' and show that $L' \leq_m L$. In this case let's try $L' = \mathsf{HP}$. Given $\langle M\rangle$ and a string x, we construct M' as follows. Let t' be the accept state of M'. The TM M' on input y simulates M on x. If M halts on x, then M' accepts by going to state t'. Now, $(\langle M'\rangle, t') \in L$ iff M halts on x. Thus $\mathsf{HP} \leq_m L$.

   To show that L is r.e, we can do the timesharing simulation of M on all inputs: Using the universal TM, we will simulate M on the first i inputs for i steps in Phase i. If for the string w_j, M enters the state q in step ℓ, then in Phase $\max\{j, \ell\}$, this will be observed and the simulation will halt and accept.

6. Recall that a TM M is *total* if for every input x, M halts on x. Consider the following language $L = \{\langle M\rangle \mid M \text{ is total}\}$. Show that L is not recursively enumerable. Is L co-r.e? Does the undecidability of L from Rice's theorem?

   Solution:

   To show that L is not r.e, we will show that $\overline{\mathsf{HP}} \leq_m L$. This is similar to the reduction for FIN and INFIN. For a TM $\langle M\rangle$ and string x, construct M' as follows: M' on input y, simulates M on x for $|y|$ steps and if it has not halted within $|y|$ steps, M' accepts y and halts. Otherwise M' enters an infinite loop on input y.

   Now, if M does not halt on x, then for every y, M accepts and halts and hence $\langle M\rangle \in L$. If M halts on x in k steps, then for all string y such that $|y| > k$, M' on input y never halts. Thus $\langle M'\rangle \notin L$.

   To show that L is not co-r.e, we have to show that $\overline{\mathsf{HP}} \leq_m \overline{L}$ or equivalently that $\mathsf{HP} \leq_m L$. This reduction is more straightforward: Given $\langle M\rangle$ and input x, construct M' as follows. The TM M' on input y simulates M on x. If M halts on x, then accept y and halt.

   Once again, if M halts on x, then M' halts on every string y and $\langle M'\rangle \in L$. If M does not halt on x, then M' does not halt on any string y, and hence $\langle M'\rangle \notin L$.

   You **cannot use** Rice's theorem to prove the undecidability of L. Suppose that M and M' are two TMs such that M rejects all strings and M' enters into an infinite loop on all strings. Then $L(M) = L(M') = \emptyset$, but $\langle M\rangle \in L$ and $\langle M'\rangle \notin L$.

7. Let $L = \{\langle M\rangle \mid w \in L(M) \text{ iff } w^R \in L(M)\}$. Show that L is not recursive. Does Rice's theorem apply here? Try to prove the theorem using a many-one reduction also.

Solution: Rice's theorem **can be applied** here. If $L(M) = L(M')$, then $w \in L(M)$ iff $w \in L(M')$ iff $w^R \in L(M')$ iff $w^R \in L(M)$. Thus, L is undecidable by appealing to Rice's theorem.

To show undecidability using reductions, we will show that $\overline{\mathsf{HP}} \leq_m L$. Given $\langle M \rangle$ and x, construct M' as follows. On input y, M' simulates M on x. If M halts on x, then M' accepts y iff $y = 10$.

If M does not halt on x, then $L(M') = \emptyset$ and hence $\langle M' \rangle \in L$. If M halts on x, then $L(M') = \{10\}$ and hence $\langle M' \rangle \notin L$.

Remark: Using Rice's theorem we showed the undecidability. But using reductions, we actually showed that L is not even r.e.

8. The *busy-beaver function* $\mathsf{BB} : \mathbb{N} \to \mathbb{N}$ is defined as follows: Consider all the Turing machines with n states that halt when given the input string ϵ, and among them look at the TM $M_{\max}$ that runs for the maximum number of steps. The number of steps that $M_{\max}$ runs for on the empty string ϵ is equal to $\mathsf{BB}(n)$. You can assume that the input alphabet and the tape alphabet is $\{0, 1\}$.

   Informally, we want to know what is the running time of the longest-running program (that does not loop forever) that you can write, with say 100 lines of code.

   Let $L_{\mathsf{BB}} = \{(\langle M \rangle, k) \mid \mathsf{BB}(n) \leq k, \text{ where } M \text{ has } n \text{ states}\}$. Show that L_{BB} is not recursive.

   Show that there does not exist a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\mathsf{BB}(n) \leq g(n)$ for every $n \in \mathbb{N}$.

   Solution:

   This is a slightly complicated solution compared to the ones seen in class. This not a many-reduction as seen in class.

   Suppose that there is an algorithm $\mathcal{A}$ for L_{BB}. We will use $\mathcal{A}$ to design an algorithm for HP. Suppose that we have a TM $\langle M \rangle$ and a string x and we wish to know if $(\langle M \rangle, x) \in \mathsf{HP}$. We will construct a TM M' where M' on an input y, simulates M on x and accept y iff M halts on x. Suppose that M' has n states. We will now use the algorithm $\mathcal{A}$ to find the smallest number k such that $(\langle M' \rangle, k) \in L_{\mathsf{BB}}$. This will require multiple calls to $\mathcal{A}$ with k starting from 1. Since $\mathsf{BB}(n)$ is finite, we will find the such a smallest k after a finite number of calls to $\mathcal{A}$. Now, we will use the universal TM to simulate M' on the input ε for $\mathsf{BB}(n)$ many steps. If the simulation has not halted by then, we know that M' will never halt on ε, and consequently M does not halt on x. On the other hand, if M' has halted on ε within $\mathsf{BB}(n)$ steps, then that implies that M halts on x.

   For the second part, let g be a computable function such that $\mathsf{BB}(n) \leq g(n)$ for every $n \in \mathbb{N}$. Let K be the TM that computes g. We will use M to design an algorithm for HP. Given an input $\langle M \rangle$ and a string x where M has n states, we use K to compute $g(n)$. Now, we simulate M on x for $g(n)$ steps. If M halts by then, we halt and accept. If M has not halted within $g(n)$ steps, then we know that M will not halt because $\mathsf{BB}(n) \leq g(n)$.

9. (**Challenge question**)

I will not be handing out a solution for this problem. Attempt this problem after you have tried all the other problems and the basic concepts are clear.

A *tag*-Turing machine has two tape-heads, one for reading and another for writing. The read-head is at the left end of the input at the beginning, and the write-head is at the first blank symbol after the input. In every step of the computation, the read-head can either stay in the current cell or move one cell right. It cannot move left or write anything on the cell it is on. The write-head must always write something on the cell it is on, and *must* move one cell to the write.

Show that the tag-Turing machines computer the set of recursively enumerable languages. In other words, show that an arbitrary one-tape Turing machine can be simulated by a one-tape tag-Turing machine.