Exercise (8.2.1 from textbook)

We explore equivalence between function computation and language recognition for Turing machines.

**Definition**

The graph of a function is the set of all strings
\[(x, f(x))\], where \(x\) is a non-negative integer in binary, and \(f(x)\) is the value of \(f\) evaluated on \(x\), again in binary.

**Definition**

A Turing machine computes function \(f\) if, starting with a string \(x\) on the tape, halts (in any state) with \(f(x)\) on the tape. \(x\) and \(f(x)\) are in binary.

Do the following:

(a) Show that, given a TM computing \(f\), we can construct a TM accepting the graph of \(f\).

(b) Show that, given a TM accepting the graph of a function \(f\), we can construct a TM that computes \(f\).

**Solution**

(a) Our TM uses two tracks: in the first track it stores the input \([x, y]\). The goal is to check whether \(y = f(x)\) so that \([x, y]\) belongs to the graph of \(f\). On the second track, the TM simulates the TM that computes \(f\), using \(x\) (first part of the first track) as working tape. When the simulated TM halts, there is \([f(x), y]\) on the first track, and our TM just checks whether \([f(x), y] = [y, y]\).
In this case we have a TM $M_0$ that accepts the graph of $f$, and we want to construct a TM $M$ that computes $f$. We cannot try all (infinite) $[x,i]$ to see whether $[x,i]$ belongs to the graph of $f$; in fact, $M_0$ may not terminate on some inputs.

Instead, we try all values of $i$ in a different fashion: we emulate $M_0$ with a second tape, executing step 1 with $i=0$, then step 2 with $i=1$, and after that step 1 with $i=2$, as shown in the figure, where we move diagonally on the grid; $i$ is denoted in decimal.

The visit of the grid in a diagonal fashion guarantees that, if $f(x)=y$ (and therefore $M_0$ terminates), at a certain point we reach the value of $i$ such that $i=y$ and $M$ can check (by emulating $M_0$) that $[x,y]$ belongs to the graph of $f$, therefore having the correct value for $f(x)$. Note that $M$ does always terminate if $f$ is defined for all inputs $x$.

Alternative solution:

We make use of non-determinism, and for each $i$, we guess whether to run $M_0$ on $[x,i]$ or whether to switch directly to $i+1$. 
Exercise (8.46 from textbook)

Design a 2-tape TM accepting strings over \{0,1\} having an equal number of 0's and 1's. The first tape reads the input from left to right; the second tape is a working tape, storing the excess of 0's over 1's or vice versa.

Solution

The idea is that our multitable Turing machine \( M \) writes initially a symbol \# on the working tape; the input tape is scanned sequentially, while the head on the working tape moves right (resp. left) if the input symbol read is 1 (resp. 0). If at the end of the input the head of the working tape is on \#, then the string is accepted.

The transition function is defined as follows:

\[
\begin{align*}
(q_0, [0, b]) & \rightarrow (q_2, [0, \#], [5, 5]) \\
(q_0, [1, b]) & \rightarrow (q_2, [1, \#], [5, 5])
\end{align*}
\]

Intuitively, the above rules make \( M \) write \# (in place of \( b \)), since the working tape is all blank) on the working tape, whatever symbol (0 or 1) is read on the input symbol; the heads do not move.

Then, the input is scanned in state \( q_2 \):

\[
\begin{align*}
(q_2, [0, \#]) & \rightarrow (q_2, [0, \#], [R, L]) \\
(q_2, [1, \#]) & \rightarrow (q_2, [1, \#], [R, R]) \\
(q_2, [0, b]) & \rightarrow (q_2, [0, b], [R, L]) \\
(q_2, [1, b]) & \rightarrow (q_2, [1, b], [R, R])
\end{align*}
\]
At the end of the input tape, we go in state $q_2$ (only final state) only if the head of the working tape reads #.

$$(q_2, [5, #]) \rightarrow (q_2, [5, #], [S, S])$$

Exercise (8.4.9 from textbook)

We consider a $k$-head Turing machine having a single tape and $k$ heads; more than one head can be on the same symbol. At each move, the TM can change state, write a symbol on each cell under a head, and move each cell, or keep it stationary. We number the heads with numbers $\{1, \ldots, k\}$: when there is more than one head on a single cell, the written symbol will be the one written by the head with highest number.

Prove that the languages accepted by $k$-head Turing machines are the same languages accepted by ordinary TM's.

Solution. We prove the assertion by showing that it is possible to simulate a $k$-head TM with a single-head (ordinary) TM.

Our TM will use additional symbols $H_1, \ldots, H_k$ to mark the portions of the $k$ heads on the tape, while the head will move back and forth from the leftmost head symbol to the rightmost one. At each head symbol $H_i$, our TM emulates the move of the $i$-th head, writing the correct symbol and making the corresponding move.
The problem is to deal with multiple heads on the same cell. In this case, we choose to put the head symbols one adjacent to the other, ordered so that the head symbol with the smallest number is the leftmost one. For example, we can have a configuration like

\[ Q_{3.4} H_{3} H_{4} O_{11} \hat{5}_{3} H_{11} \]

denoting that heads number 3.4 and 11 are on the same cell.

This complicates the things, but the emulation is still possible. No symbol is written after some \( H_i \) if the symbol on the right is another head symbol. When a head symbol is to be moved, it has to skip all other head symbols; moreover, if it is arriving on a cell "pointed" by other head symbols, it needs to be put in the correct place so as to respect the order (also this is easily feasible).
Consider the following languages over $\Sigma = \{0, 1\}$:

$$L_e = \{ \varepsilon(M) \mid \varepsilon(M) = \emptyset \}$$

$$L_{\neg e} = \{ \varepsilon(M) \mid \varepsilon(M) \neq \emptyset \}$$

Hence: $L_e$ is the set of all strings that encode T.M.s that accept the empty language.

$L_{\neg e}$ is the complement of $L_e$.

**Claim 1:** $L_{\neg e}$ is R.E.

**Proof:** construct N.T.M. $N$ for $L_{\neg e}$

(and then convert $N$ to an ordinary T.M.)

$N$ works as follows: on input $\varepsilon(M)$

1) guess a string $w \in \Sigma^*$

2) minitalize $M$ on $w$ (like a U.T.M.)

3) accept $\varepsilon(M)$ if $M$ accepts $w$

We have $\varepsilon(M) \in \varepsilon(N) \iff$ for s.t. $\langle M, w \rangle \in \varepsilon(U)$

$\iff$ for s.t. $w \in \varepsilon(M)$

$\iff \varepsilon(M) \in L_{\neg e}$
Claim 2: \( L_n \) is non-recursively enumerable.

Proof: by reduction from \( L_m \) to \( L_n \).

Reduction \( R \) is a function computable by a halting T.M. with:
- input: instance \( \langle M, w \rangle \) of \( L_m \)
- output: instance \( \Sigma(M') \) of \( L_n \)

end set: \( \langle M, w \rangle \in L_m \iff \Sigma(M') \in L_n \).

Description of \( M' \):
- \( M' \) ignores completely its own input string \( X \)
- instead, it replaces its input by the string \( \langle M, w \rangle \) and simulates \( M \) on \( w \) using UTM.
- if \( M \) accepts \( w \), then \( M' \) accepts \( X \)
- if \( M \) never halts on \( w \) or rejects \( w \),
  then \( M' \) also never halts on \( X \).

Note: if \( w \notin \Sigma(M) \Rightarrow \Sigma(M') = \Sigma \)
if \( w \notin \Sigma(M) \Rightarrow \Sigma(M') = \emptyset \)

hence \( \langle M, w \rangle \in L_m \iff \Sigma(M') \in L_n \).

We can construct a halting T.M. \( M_r \) that, given \( \langle M, w \rangle \) as input, reconstructs \( \Sigma(M') \) for an \( M' \) that behaves as above.

q.e.d.

To sum up, we have that \( L_n \) is R.E. but non-recursively enumerable, hence \( L_e \) must be non-R.E.
Exercise 3.2.1

The halting problem, \( L_{\text{H}} \), the set \( \langle M, w \rangle \) s.t. \( M \) halts on \( w \) (with or without accepting) is R.E., but not recursive.

To show R.E., we construct a T.M. \( H \) s.t. 
\[ L(H) = L_H = \{ \langle M, w \rangle \mid M \text{ halts on } w \} \]

To show that \( L_H \) is not recursive, we assume by contradiction it is \( \bar{R} \), and derive that \( L_H \) is recursive.

By contradiction, let \( V \) be an algorithm for \( L_H \) and \( U \) a procedure for \( L_{\text{H}} \).

\[ \langle M, w \rangle \rightarrow H \rightarrow \text{triggers} \rightarrow \text{yes} \rightarrow V \rightarrow \text{yes} \]

\[ \langle M, w \rangle \rightarrow H \rightarrow \text{no} \]

\( A_m \)

\( A_m \) would be an algorithm for \( L_H \), contradiction.
Let \( L \) be R.E. and \( \overline{L} \) be non-R.E.

Consider \( L' = \{ ow \mid w \in L \} \cup \{ ow \mid w \not\in L \} \).

What do we know about \( L' \) and \( \overline{L}' \)?

We show that \( L' \) is non-R.E.

Suppose by contradiction that we have a procedure \( M_{L'} \) for \( L' \).

Then we can construct a procedure \( M_L \) for \( L \) as follows:

- on input \( w \), \( M_L \) changes the input to \( 1w \) and simulates \( M_{L'} \).

- if \( M_{L'} \) accepts \( 1w \), then \( w \in L \), and \( M_L \) accepts

- if \( M_{L'} \) does not terminate or terminates and answers no, then \( w \not\in L \), and \( M_L \) does not terminate or terminates and answers no.

\[
\Rightarrow M_L \text{ would accept exactly } L. \text{ Contradiction}
\]

\( \overline{L}' = \{ ow \mid w \in L \} \cup \{ 1w \mid w \in L \} \cup \{ 1 \} \)

Reasoning as for \( L' \), we get that \( \overline{L}' \) is non-R.E.
The complement of the halting problem, i.e., the set of pairs \( \langle M, w \rangle \) such that \( M \) on input \( w \) does not halt, is non-R.E.

Proof: By reduction from \( \overline{T_{m}} \), which is non-R.E.

Idea: we show how to convert any TM \( M \) into another TM \( M_{k} \) such \( M_{k} \) halts on \( w \) iff \( M \) accepts \( w \).

Construction:

1) Ensure that \( M_{k} \) does not halt unless \( M \) accepts.

   - add to the states of \( M \) a new loop state \( q_{L} \), with \( \delta(q_{L}, x) = (q_{L}, x, l) \) for all \( x \in \Gamma \)

   - for each \( \delta(q, s) \) that is undefined and \( q \in \text{F} \),
     add \( \delta(q, s) = (q, s, s_{L}) \)

2) Ensure that, if \( M \) accepts, then \( M_{k} \) halts.

   - make \( \delta(q, s) \) undefined for all \( q \in \text{F} \) and \( s \in \Gamma \)

3) The other moves of \( M_{k} \) are as those of \( M \)