How to Show Non-Recursive-Enumerability?
Written and copyright by Jie Wang

Diagonalization and reduction are two common techniques used to show that a given set is not r.e. These techniques are also commonly used to show that a given set is not recursive (i.e., not decidable). To illustrate how to use these two techniques, we consider a set \( \text{Tot} \) defined as follows:

\[
\text{Tot} = \{ \langle M \rangle \mid M \text{ is a Turing Machine and } M \text{ halts on any binary string} \},
\]

where \( \langle M \rangle \) represents a fixed binary encoding of \( M \)'s description and \( \{0,1\} \) is \( M \)'s input alphabet.

We can show that \( \text{Tot} \) is not r.e., nor is \( \overline{\text{Tot}} \). We first use diagonalization to show that \( \text{Tot} \) is not r.e. We will use the following fact:

**Proposition 1** If \( L \) is r.e., then there is a recursive function \( f \) such that \( L = \text{range}(f) \).

**Proof.** Since \( L \) is r.e., there is a 2-tape Turing enumerator \( M \) that prints all the elements in \( L \) one at a time on its output tape. We construct a 4-tape Turing machine \( M_f \) with one input-tape, two work tapes, and one output tape as follows: On any input \( x \in \{0,1\}^* \) on its input tape, use the two work tapes to simulate \( M \). Whenever an output \( y \) is printed on \( M \)'s output take, check \( M_f \)'s input tape. It its content becomes \( \epsilon \), output \( y \) on \( M_f \)'s output tape and halt. Otherwise, decrease the content of \( M_f \)'s input tape by 1 in the lexicographical order and continue the simulation of \( M \).

Clearly, \( M_f \) computes a function and it always halts on any input. Thus, \( M_f \) computes a recursive function and we denote this function by \( f \). From the construction, we note that \( f(x) \) is an element in \( L \) and for any element \( y \in L \), there must be an \( x \) such that \( f(x) = y \). Thus, \( L = \text{range}(f) \).

**Proposition 2** \( \text{Tot} \) is not r.e.

**Proof.** Suppose for the sake of contradiction that \( \text{Tot} \) is r.e. It follows from Proposition 1 that there is a recursive function \( g \) such that \( \text{Tot} = \text{range}(g) \). This means that for any given binary string \( x \), \( g(x) \) gives a binary encoding of a Turing machine that halts on any
binary string input $y$ (including on input $x$ itself). Denote this Turing machine by $M_{g(x)}$. We construct using diagonalization a recursive function $f$ as follows:

$$f(x) = \begin{cases} 0, & \text{if } M_{g(x)}(x) > 0 \\ 1, & \text{if } M_{g(x)}(x) = 0 \end{cases}$$

The function $f$ is recursive because it is total. It follows from the construction that $f(x) \neq M_{g(x)}(x)$ for any $x$.

Let $M_f$ be a Turing machine that computes $f$. Thus, $M_f$ halts on any input. This implies that $\langle M_f \rangle \in \text{Tot}$ and so there must be an $x$ such that $g(x) = \langle M_f \rangle$. This means that $M_{g(x)}(x) = M_f(x) = f(x)$, a contradiction. Thus, the assumption that $\text{Tot}$ is r.e. is incorrect. This completes the proof.

We now show that $\overline{\text{Tot}}$ is not r.e. using reduction. Recall that the following Turing machine halting problem

$$H = \{ \langle M, x \rangle \mid M \text{ is a Turing machine and } M \text{ on input } x \text{ halts} \}$$

is r.e. but not recursive. That is, $\overline{H}$ is not r.e. Also recall that if $A \leq_m B$ and $A$ is not r.e., then neither is $B$. Thus, it suffices to show that $\overline{H} \leq_m \overline{\text{Tot}}$, which is equivalent to showing that $H \leq_m \text{Tot}$. This is exactly the same proof of the first part in the proof of Rice’s theorem (see my handout on Rice’s theorem).