4. Problem

Prove, without appealing to Rice's theorem, that the following problem is undecidable:

Given a Turing machine $M$; does $M$ accept the empty string?

Solution:

We can prove that a problem is undecidable by showing that we could use its solution to solve some other problem that we know to be undecidable. In this case we will use the universal language $U$ as our existing undecidable problem and reduce it to the language $L_\varepsilon = \{c_M \mid \varepsilon \in L(M)\}$ where $c_M$ denotes the encoding of a Turing machine $M$ using some suitable binary encoding.

Our proof has these steps:

(a) The universal language $U = \{c_Mc_x \mid x \in L(M)\}$ is known to be undecidable. (We take this as given).

(b) Since $U$ is undecidable, it is not possible to construct a total universal Turing machine $M_U$ where $L(M_U) = U$.\(^1\) A Turing machine is total if it halts for every possible input.

(c) We show that if we have a Turing machine $M_\varepsilon$ where $L(M_\varepsilon) = L_\varepsilon$, we can use it as a building block in constructing an universal Turing machine $M_\varepsilon^U$ in such a way that we can guarantee that the other parts of $M_\varepsilon^U$ are total.

(d) Since we can create a universal Turing machine $M_\varepsilon^U$ out from $M_\varepsilon$ and we know that it is not possible to create a total UTM, we conclude that $M_\varepsilon$ may not be total so $L_\varepsilon$ is undecidable.

Next, we examine the construction phase in detail.

Suppose that we can construct a Turing machine $M_\varepsilon$ for the language $L_\varepsilon$ ($L(M_\varepsilon) = L_\varepsilon$). The machine $M_\varepsilon$ gets as its input a binary encoding $c_M$ of some Turing machine $M$ and then it tells whether $M$ accepts the empty string or not. We treat $M_\varepsilon$ as a black box: it can use any method to determine the answer and we are not concerned of its interior workings.

Next, we want to create a universal Turing machine $M_\varepsilon^U$ in a way that it will use $M_\varepsilon$ to do the hard part of the computation. A UTM gets two inputs, an encoding $c_M$ of a TM $M$ and an encoding $c_x$ of an input string $x$.

The UTM $M_\varepsilon^U$ will work in two phases:

(a) First it uses $M$ and $x$ to create a new Turing machine $M_x$. When this machine is started, it first writes the string $x$ to its tape, rewinds its read/write-head, and then starts to simulate machine $M$.

(b) Next, the UTM will use $M_\varepsilon$ to check whether the new machine $M_x$ accepts the empty string.

In the first phase $M_\varepsilon^U$ will alter the encoding $c_M$ by adding $|x| + 1$ new states for it. In the first $|x|$ states the machine will write one symbol of $x$ to the tape and move the read/write head to right. The last new state rewinds the tape back to the beginning, and then takes

\(^1\)It is possible to construct universal Turing machines but they are not total.
an transition to the original initial state of $M$. We can implement this phase with a total
Turing machine because both $e_M$ and $e_x$ have a finite length by Turing machine definition.
The machine $M_x$ does essentially the same computation with an empty input as $M$ does
with input $x$.
If the language $L_x$ is decidable, then we can make $M_x$ total. However, in this case $M_U$
is also total. Since this is impossible, we know that $M_x$ may not be total and $L_x$ is not
decidable but only semi-decidable.

5. **Problem**: Prove the following connections between recursive functions and languages:

(i) A language $A \subseteq \sum^*$ is recursive (“Turing-decidable”), if and only if its characteristic
function

\[
\chi_A : \sum^* \rightarrow \{0, 1\}, \quad \chi_A(x) = \begin{cases} 
1, & \text{if } x \in A; \\
0, & \text{if } x \notin A 
\end{cases}
\]

is a recursive (“Turing-computable”) function.

(ii) A language $A \subseteq \sum^*$ is recursively enumerable (“semidecidable”, “Turing-recognisable”),
if and only if either $A = \emptyset$ or there exists a recursive function $g : \{0, 1\}^* \rightarrow \sum^*$ such that

\[
A = \{g(x) \mid x \in \{0, 1\}^*\}.
\]

**Solution**: We start by defining five simple helper machines:

- **1** writes ‘1’ to the input tape, moves the read/write head to right and stops.
- **0** writes ‘0’ to the tape and stops.
- **C** empties the input tape, moves the head to the beginning of the tape and stops.
- **NEXT** reads the input $x \in \sum^*$ and replaces it with the lexicographic successor of $x$.
- **Cmp** compares the contents of the input tapes $i$ and $j$ of a multi-tape Turing
machine and accepts if they are identical.

Since the machines are simple, they are not presented here.

(i) $\Rightarrow$ Let $A \subseteq \sum^*$ be a recursive language. Then there exists a Turing machine $M_A$:

\[
M_A = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle
\]

such that

\[
\forall w \in \sum^*: w \in L \iff (q_0, w) \xrightarrow{\ast} M_A (q_{acc}, \alpha) \quad \text{ja}
\]

\[
w \notin L \iff (q_0, w) \xrightarrow{\ast} M_A (q_{rej}, \alpha)
\]

We construct a machine $M$ by combining $M_A$ with machines **1**, **0**, **C** as follows:

![Diagram of machine M]

If $w \in L$, then $M_A$ accepts $w$. After that $M$ clears the tape and writes 1 to the tape.
Otherwise 0 is written. Since $A$ is recursive, $M_A$ halts always so also $M$ halts and it
computes the function $\chi(w) = \begin{cases} 
1, & w \in A \\
0, & w \notin A 
\end{cases}$ that is the characteristic function of $A$. 

2
Suppose that the function $\chi(w)$ is recursive. Then there exists a Turing machine $M_\chi$ that computes it. We can now construct a machine $M$ as follows:

Now $M$ accepts $w$ whenever $\chi(w) = 1$ and rejects it when $\chi(w) = 0$, so $M$ decides the language $A$ and $A$ is recursive.

(ii) If $A = \emptyset$, then trivially $A \in \mathbf{RE}$ and $g(x) = 0$ is its characteristic function.

If there exists a function $g$ that fulfills the conditions, then there exists a Turing machine $M_g$ that computes $g$. We can trivially modify it so that it becomes a 2-tape machine $M_{g,2}$ that computes $g$ but stores the result in the second tape instead of the first. We now construct a 3-tape machine as follows:

![Diagram of a 3-tape machine](image)

The machine gets its input from its first tape and it stays untouched for the whole computation. In each iteration $M_A$ replaces the bit string $x$ on the second tape by its lexicographic successor $y$, computes $g(y)$ and writes the output on the third tape. Finally, the contents of tapes 1 and 3 are compared and if they match, the word is accepted, otherwise the iteration proceeds into the next round.

Consider the word $w \in A$. Suppose that a recursive function $g$ that fulfills the conditions exists. Then $w = g(x)$ for some $x = x_1x_2 \cdots x_n$ where $n$ is finite. Since each finite string has a finite number of predecessors in the lexicographic order, NEXT eventually generates $x$. $M_{g,2}$ generates $w$ on the third tape and $M_A$ accepts the word. Thus, $M_A$ recognizes the language $A$ so $A \in \mathbf{RE}$.

Next, suppose that $A \in \mathbf{RE} \setminus \{\emptyset\}$. Then there exists a Turing machine $M_A$ that recognizes it. We now define a helper machine $M_{A,i}$ that simulates $M_A$ for $i$ steps. The machine $M_{A,i}$ accepts $x$ if $M_A$ accepts it using at most $i$ steps, and rejects it otherwise. We note that $M_{A,i}$ always halts.

We construct the function $g$ with the help of $M_{A,i}$. Every input $x$ and bound $i$ is encoded into bit strings using the function $c(x,y) = 0^i10^y$. We define that $g(c(x,y)) = x$, if $M_{A,y}$ accepts $x$. We define that $g' : \{0,1\}^* \to \{0,1\}^*$ is the function:

$$g'(w) = \begin{cases} x, & w = 0^i10^y \text{ and } M_{A,y}(x) \text{ accepts} \\ x_0, & \text{otherwise} \end{cases}$$

where $x_0 \in A$. Finally, $g(x) = d(g'(x))$ where $d$ is a function that maps a bit string $0^i$ into the $x$th element of $\Sigma^*$ in the lexicographic order. The value of $g'$ may be computed in a finite time since $M_{A,y}(x)$ always halts. Thus, $g'$ is recursive and so also $g$ is.

Note that while $g$ always exists, it is not always possible to find it since in the general case it is an undecidable problem to find an element $x_0 \in A$ that is needed for the definition.