Turing Machines and Undecidability

Turing Machines
A Turing machine is an automaton with a finite control and a bidirectional read/write head for processing strings on an input tape.
The control unit operates in discrete steps; at each step, it performs two functions dependent on its current state and the tape symbol currently scanned:
(1) put the control unit in a new state;
(2) Either
(a) Write to the tape, or
(b) Move the tape head right or left one square.

Definition. A Turing machine (TM) $M = (K, \Sigma, \delta, s)$ is a four-tuple where
K is a finite set of states, not containing the special state h, the halt state;
Σ is a finite alphabet with the blank symbol $\#$, but Σ does not contain the symbols L nor R.
$s \in K$, the start state;
$\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$, the transition function.

Note: The operation of the TM M is deterministic and will stop only if M enters the halt state or attempts to move off the left end of the tape (that is, the machine "hangs").

A configuration of a TM M represents the current state, the current input symbol scanned, all the contents of the tape to the left of the current input symbol, and the contents of the tape to the right of the current input symbol to the strings of blanks. Note that when a finite string w is placed on the input tape, the blank symbol $\#$ is placed on all spaces to its right. In other words, the configuration is an element of the set:

$$(K \cup \{h\}) \times \Sigma^* \times \Sigma \times (\Sigma^*(\Sigma \setminus \{\#\}) \cup \{e\}).$$

A convenient notation for a configuration is: $(q, w_1\underline{a}w_2)$.

Definition. If M is a TM and $w \in \Sigma^*$, then M is said to *halt on input* w if $(s, \#w\underline{\#})$ yields a halted configuration, while M *hangs on input* w if $(s, \#w\underline{\#})$ yields some hanging configuration.

Definition. Let Σ_0 and Σ_1 be finite alphabets not containing the blank symbol $\#$. Let $f : \Sigma_0^* \to \Sigma_1^*$. A TM $M = (K, \Sigma, \delta, s)$ is said to *compute* f if $\Sigma_0, \Sigma_1 \subseteq \Sigma$ and for any string $w \in \Sigma_0^*$, if $f(w) = u$, then the configuration $(s, \#w\underline{\#})$ will yield the configuration $(h, \#u\underline{\#})$. We say that the function f is *Turing-computable* or *recursive.*

Definition. A Turing machine M *accepts* a string w if M halts on the input w. The TM M *accepts* or *semidecides* the language L if

$$L = \{w \in \Sigma^* : M \text{ accepts } w\}.$$

Call the language L recursively enumerable.

Definition. Let Σ be a finite alphabet without $\#$ and without the symbols Y, N. Then a language $L \subseteq \Sigma^*$ is *recursive* if the function

$$\chi_L : \Sigma^* \to \{Y, N\}$$

is recursive.

There are several extensions of the basic Turing machine. For example,
(1) Allow a two way infinite tape;
(2) Allow a k-tape machine.
(3) Allow a two-dimensional tape.
These extensions have the same power of acceptance as the basic Turing machine. The argument rests on the idea of treating the basic one tape as having several distinct tracks for models (1), (2), and

(3). For example, for the k-tape machine, the corresponding usual TM will have a single tape with $2k$-tracks, where one track is used to encode the contents of one of the tapes of the original machine and another track is used to record the position of the tape head for that track.

A more serious extension of a deterministic Turing machine is to allow non-deterministism. That is, we replace the transition function δ with a transition relation Δ :

$$\Delta \subseteq (K \times \Sigma) \times ((K \cup \{h\}) \times (\Sigma \cup \{L, R\}))$$

where Δ is a finite subset. Note: NTM's will only be used as ACCEPTORS.

By using the idea of "dovetailing" and breath first search, we can simulate a NTM by a 3-tape TM where the roles of the tapes are:
tape 1: contains the original input;
tape 2: contains the simulated computation;
tape 3: directs the computation by keeping track of the computation branch that the NTM is using.

Universal Turing Machine $M_{\mathcal{U}}$
Goal: given any TM M and input w, we want:
$(s_M, \#w\underline{\#})$ will yield $(h, u\underline{a}v)$ if and only if $(s_{M_{\mathcal{U}}}, < w >)$ will yield $(h, < u\underline{a}v >)$,
where $< w >$ denotes the encoding of the string w for the universal TM $M_{\mathcal{U}}$.
Write: $M_{\mathcal{U}}(< M, w >) \Downarrow$ if and only if $M(w) \Downarrow$.

Notation: Write $M(w)$ for the TM M with input string w. Write $< M, w >$ for the encoding of $M(w)$ for the universal TM $M_{\mathcal{U}}$. $M(w) \Downarrow$ means that the TM M converges on the input w.

Outline of the Construction of UTM $M_{\mathcal{U}}$.

(I) We fix countably infinite sets: $K_\infty = \{q_1, q_2, \ldots\}$, $\Sigma_\infty = \{a_1, a_2, \ldots\}$.
Without loss of generality, we assume that every TM M has its state space $K \subseteq K_\infty$ and its alphabet $\Sigma \subseteq \Sigma_\infty$.

(II) We give a formal correspondence λ between the component symbols of a TM M (states and input symbols) and strings over a one symbol alphabet $\{I\}$. In particular, elements from $K \cup \{h\}$ or $\Sigma_\infty \cup \{L, R\}$ will have distinct representations.

$\lambda(q_i) = I^{i+1}$,
$\lambda(h) = I$,
$\lambda(L) = I$,
$\lambda(R) = II$,
$\lambda(a_i) = I^{i+2}$.
(III) We encode the TM $M = (K, \Sigma, \delta, s)$ over a 2-symbol alphabet $\{c, I\}$.
Assume:
$K = \{q_{i_1}, q_{i_2} \ldots, q_{i_k}\}$ and $\Sigma = \{a_{j_1}, a_{j_2} \ldots, a_{j_l}\}$,
where $i_1 < i_2 < \ldots < i_k$ and $j_1 < j_2 < \ldots j_l$.
We encode the transition:
$\delta(q_{i_p}, a_{i_r}) = (q', b)$,
where $q' \in K_\infty \cup \{h\}$ and $b \in \Sigma_\infty \cup \{L, R\}$, as a string $S_{p,r} \in \{c, I\}^*$:

$$S_{p,r} = c\, w_1\, c\, w_2\, c\, w_3\, w_4,$$

where $w_1 = \lambda(q_{i_p})$, $w_2 = \lambda(a_{j_r})$, $w_3 = \lambda(q')$, $w_4 = \lambda(b)$.
We encode the TM M as $< M > \in \{c, I\}^*$:

$$< M >= cS_0cS_{11}S_{12}\cdots S_{1\ell}S_{21}S_{22}\cdots S_{k1}S_{k2}\cdots S_{k\ell}c.$$

Note: S_0 is the encoding of the start state.

We conclude: $< M_1 >=< M_2 > \iff M_1 = M_2$.

(IV) If $w \in \Sigma^*$, so $w = b_1b_2\dots b_n$, with $b_i \in \Sigma_\infty$, we set
$< w >= c\lambda(b_1)c\lambda(b_2)c\dots c\lambda(b_n)c$,
and
$< M, w >=< M >< w >$.

(V) We construct $M_{\mathcal{U}}$ via a 4-tape TM, where
tape 1: encodes the tape of M,
tape 2: encodes the TM M itself;
tape 3: encodes the current state;
tape 4: encodes the current input symbol.

Church-Turing Thesis: The formalization of an algorithm corresponds to the existence of a Turing machine that decides the problem (so it halts on all inputs).

Undecidable Problems

Definition: We let $L_{\mathcal{U}} = \{< M, w >: \ M(w) \Downarrow\}$.

Proposition. $L_{\mathcal{U}}$ is recursively enumerable.

To see this, recall that a language L_0 is recursively enumerable if there is a TM M_0 such that
$L_0 = \{w \in \Sigma^* : M_0(w) \Downarrow\}$.
That is, M_0 halts on the input string w.
Now, by construction, we know that
$M_{\mathcal{U}}(< M, w >) \Downarrow$ if and only if $M(w) \Downarrow$.
The desired TM is found by using a preprocessor to $M_{\mathcal{U}}$ to guarentee that only inputs of the form $< M, w >$ are used.

Recall we are still working under the assumption that for any TM $M = (K, \Sigma, \delta, s)$ that $K \subseteq K_\infty$ and $\Sigma \subseteq \Sigma_\infty$.
We study the "diagonal" language $\mathcal{K}$, where
$\mathcal{K} = \{< M >: \ M(w) \Downarrow, \text{ where } < w >=< M >\}$.
That is, $\mathcal{K}$ consists of all TM's that halt on their own input. Further, note that:
$L_{\mathcal{U}} = \{< M, w >: M(w) \Downarrow\} = \{< M >< w >: M(w) \Downarrow\}$,
so $\mathcal{K}$ is diagonal in this sense.

Claim: $\mathcal{K}$ is recursively enumerable. Use the same reasoning as for $L_{\mathcal{U}}$.

Claim: $\overline{\mathcal{K}}$, the complement of $\mathcal{K}$, is NOT recursively enumerable:
$\overline{\mathcal{K}} = \{< M >: M(w) \Uparrow, \text{ where } < w >=< M >\}$.
To see this, suppose $\overline{\mathcal{K}}$ were recursively enumerable. Then there is a Turing machine M^* so that
$(*) \qquad \overline{\mathcal{K}} = L(M^*) = \{< M >: M(w) \Uparrow, \text{ where } < w >=< M >\}$.

Question: Is $< M^* >\in \overline{\mathcal{K}}$?

Case 1: Assume $< M^* >\in \overline{\mathcal{K}}$.
Now, on one hand, $< M^* >\in L(M^*)$ means that $M^*(w) \Uparrow$, where $< w >=< M^* >$, by (*).
On the other hand, $w \in L(M^*)$ always means $M^*(w) \Downarrow$. In particular, $M^*(w) \Downarrow$ for $< w >=< M^* >$.
Contradiction.

Case 2: Assume $< M^* >\notin \overline{\mathcal{K}}$.

In other words, we assume $< M^* > \in \mathcal{K}$. So, $M^*(w) \Downarrow$, since $< w > = < M^* >$.
On the other hand, by definition, $L(M^*) = \{w : M^*(w) \Downarrow\}$. Hence, $< M^* > \in L(M^*)$. But by (*), $< M^* > \in L(M^*)$ means $M^*(w) \Uparrow$, where $< w > = < M^* >$. Contradiction.

We conclude: there is no such string $< M^* >$, so there is no such TM M^*; that is, $\overline{\mathcal{K}}$ is NOT recursively enumerable.

Note: $\mathcal{K}$ is a language which is recursively enumerable but not recursive.

Proposition. The language $L_{\mathcal{U}} = \{< M, w >: M(w) \Downarrow\}$ is not recursive.

Suppose there is a decision procedure for $L_{\mathcal{U}}$. That is, there is a recursive function $f : \{c, I\}^* \rightarrow \{Y, N\}$ so that $f(< M, w >) = Y$ if and only if $M(w) \Downarrow$. Then we can use f to decide $\mathcal{K}$.
Consider $f(< M, w >) = f(< M >< w >) = Y$ if and only if $M(w) \Downarrow$, where $< w > = < M >$.
To finish up, we use a preprocessor to guarentee that the input to f has the format: $< M >< M >$.

Proposition 1. It is undecidable that a given TM M halts on the empty tape.

Suppose that the TM M_0 decides the language $\{< M >: M(e) \Downarrow\}$. We shall use M_0 to decide the language $L_{\mathcal{U}} = \{< M, w >: M(w) \Downarrow\}$.
Given M and w, we construct a TM M_w such that M_w writes the string w on a blank tape, then simulates M.
Now, M_0 yields a decision procedure for $L_{\mathcal{U}}$, since $M(w) \Downarrow$ if and only if $M_w(e)$.

Proposition 2. It is undecidable that a TM M halts on some input string.

We show that if this problem were decidable, then the problem given in Proposition 1 would be decidable.
Given a TM M, modify M so it erases any input string, then it will simulate the original TM M. We call this new TM M'.
Then M' halts on some input string (which is decidable by assumption) if and only if M' halts on all input strings if and only if M halts on the empty string.
We conclude: there exists a string w so $M'(w) \Downarrow$ if and only if $M(e) \Downarrow$.
So, if the problem of Proposition 2 is decidable, so would the problem of Proposition 1. Contradiction.

Corollary. It is undecidable that a given TM M halts on every input.

Proposition 3. It is undecidable that two given TM's M_1 and M_2 halt on the same input string.

We alter the TM M_1 to be the machine that halts immediately after every input. So, if Proposition 3 were decidable, then the problem of the above Corollary would be decidable. Contradiction.

Note: We can re-phrase the above Propositions.
Proposition 2: it is undecidable if a given program can loop forever on some input;
Proposition 3: it is undecidable if two programs produce the same output given the same input.

Rice's Theorem. Suppose $\mathcal{C}$ is a proper non-empty collection of recursively enumerable languages. Then the following problem is undecidable: given a TM M, is $L(M) \in \mathcal{C}$?

Note: $\mathcal{C}$ may be the class of all regular languages, context free languages, or finite languages, for example.
