Turing Machines

A Turing Machine (*tm*) consists of:

1. *An unbounded tape.* Divided into squares, each square containing a symbol from a finite alphabet with at least 2 symbols: $\{S_0, S_1, S_2, ..., S_n\}$, $n \geq 1$. S_0 represents "blank".
2. *A read/write scanner.* Programmed with a finite list of internal ("memory") states $\{q_0, q_1, ..., q_m\}$, $m \geq 0$.
3. *A program.* Consists of a finite sequence of steps $\{A_0, A_1, ..., A_k\}$, $k \geq 0$. Each step A_i consists of a 4-tuple (*initial state*, *initial symbol*, *action*, *final state*). For initial state and initial symbol (q_i, S_j), there are 3 possible actions, afterwhich the *tm* enters final state q_l:
   (i) Replace initial symbol with S_k. Complete step given by (q_i, S_j, S_k, q_l).
   (ii) Move one square left. Complete step given by (q_i, S_j, L, q_l).
   (iii) Move one square right. Complete step given by (q_i, S_j, R, q_l).

   Comments: For any initial pair (q_i, S_j), there must be at most one A_i containing it. Computation halts when current initial pair doesn't occur in any A_i.

Comments: A *tm* takes in an input tape, and either halts with a modified output tape, or continues indefinitely (doesn't halt).

Props. 7.20, 7.21. The set of Turing machines may be effectively enumerated T_0, T_1, ... in such a way that each suffix determines effectively and completely the instructions for the corresponding machine.

Proof Outline: Any *tm* is fully specified by a string of symbols from $S_0, ..., S_n, q_0, ..., q_m, A_1, ..., A_k$. Just as with first order systems, a Gödel-numbering system can be constructed to encode such strings uniquely as natural numbers. One way:

Symbol	*G*-number
R	3
L	5
S_i	$7 + 4i$
q_i	$9 + 4i$

G-numbers for strings:
For the string $u_1...u_j$ that represents the *tm* T, let

$$g(T) = p_0^{g(u_1)} \times ... \times p_{j-1}^{g(u_j)}$$

where $p_0, ..., p_{j-1}$ are the first j primes 2, 3, 5, ...

Consequence: Every *tm* corresponds to an $n \in \mathbb{N}$. And any $n \in \mathbb{N}$ can be decoded uniquely (*via* prime factorization) to see if it's the *G*-number of a *tm*, and if so, which one.

Def. 7.23. A partial function on $\mathbb{N}$ is **Turing computable** if there's a *tm* that computes its value.

Comment: This definition depends on the conventions dictating how input/output tapes are to represent the domain and range of functions. One way to do this is the following:

Note: A *tm* symbol set must contain at least two symbols S_0, S_1.

So: Can represent any $n_1, ..., n_k \in \mathbb{N}^k$ by the input tape $S_1^{n_1}S_0S_1^{n_2}S_0...S_0S_1^{n_k}$, where S_1^m abbreviates m S_1's. Call this tape $\sigma[(n_1, ..., n_k)]$.

Then: Those *tms* that represent functions are those that take such σ's as input and output tapes of the form $\sigma[m]$ (*i.e.*, just a single $m \in \mathbb{N}$).

Thus: A *tm* T determines a function $f_T^k : A^k \to \mathbb{N}$, $A^k \subseteq \mathbb{N}^k$, by:
If $T(\sigma[(n_1, ..., n_k)]) = \sigma[m]$, $k > 0$, then $f_T^k(n_1, ..., n_k) = m$.

Comments: Technically, a *tm* T determines a map T : {*input tapes for which T halts*} $\to$ {*all possible (appropriately formated) input tapes*}. So the function f_T^k it determines is a *partial* function, defined only on a subset A^k of $\mathbb{N}^k$ (namely, that subset that corresponds to the input tapes for which T halts).

Prop. 7.29. The Halting Problem for Turing machines is unsolvable; *i.e.*, there is no algorithm which provides answers to questions from the set {does machine T_m halt with input n? | $m, n \in \mathbb{N}$}.

Proof: We need to show that the set $A = \{m, n \in \mathbb{N} : T_m \text{ halts on input } n\}$ is not recursive.
Suppose A is recursive.

Then: It's characteristic function c_A is recursive, where

$$c_A(m,n) = \begin{cases} 0 & \text{if } (m,n) \in A \\ 1 & \text{if } (m,n) \notin A \end{cases}$$

Thus: By Prop. 7.25, c_A is Turing computable. So there's a *tm* that computes it.

So: We can use c_A's *tm* to construct a *tm* H such that its output $H(n)$ on input n is given by

$$H(n) = \begin{cases} T_n(n) + 1 & \text{if } (n,n) \in A \\ 0 & \text{if } (n,n) \notin A \end{cases}$$

Note1: H corresponds to T_m for some $m \in \mathbb{N}$. (The Enumeration Theorem.)

Note2: H halts on *every* input. (For *any* input n, if $(n, n) \in A$, then, since T_n halts, so does H. If $(n, n) \notin A$, then H halts with output 0.)

So: For input m, we have:

$$\begin{aligned} H(m) &= T_m(m) \\ &= T_m(m) + 1 \qquad \text{(Definition of } H\text{)} \end{aligned}$$

This is a contradiction, so A cannot be recursive.

Prop. 7.30. The set $K = \{n \in \mathbb{N} : T_n \text{ halts with input } n\}$ is recursively enumerable but not recursive.

Proof: Note that we can construct the *tm* H in the proof of Prop. 7.29 directly from the characteristic function c_K of K. This shows that K cannot be recursive. To show K is recursively enumerable, we appeal to Church's Thesis and the following algorithm:

1. Compute first step of T_0 with input 0.
2. Compute first step of T_1 with input 1, and second step of T_0 with input 0.
3. Compute first step of T_2 with input 2, second step of T_1 with input 1, and third step of T_0 with input 0.
4. *Etc...*

If and when T_i halts, put i in the enumeration of K.

Comment: The set A in Prop. 7.29 is not recursively enumerable.