Turing Machines and Diagonalization

1 Turing Machines

Turing machines are a model of computation. It is believed that anything that can be computed can be computed by a Turing Machine. The definition won't look like much, and won't be used much (except in the proof of the Cook-Levin theorem which will be in a different set of notes); however, it is good to have a rigorous definition to refer to.

Def 1.1 A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

- Q is a finite set of states
- Σ is a finite alphabet. It contains the symbol #.
- $\delta : Q - \{h\} \times \Sigma \to Q \times \Sigma \cup \{R, L\}$
- $s \in Q$ is the start state
- $h \in Q$ is the halt state.

We use the following convention:

1. On input $x \in \Sigma^*$, $x = x_1 \cdots x_n$, the machine starts with tape

$$\#x_1x_2 \cdots x_n\#\#\#\# \cdots$$

that is one way infinite.

2. The head is initially looking at the x_n.
3. If $\delta(q, \sigma) = (p, \tau)$ then the state changes from q to p and the symbol σ is overwritten with τ. The head does not move.
4. If $\delta(q, \sigma) = (p, L)$ then the state changes from q to p and the head moves Left one square. overwritten with τ. The head does not move. (Similar for $\delta(q, \sigma) = (p, R)$.
5. If the machine is in state h then it is DONE.
6. If the machine halts and outputs 1 then we say M ACCEPTS x. If the machine halts and outputs 0 then we say M REJECTS x.

Important Note: We can code Turing machines into numbers in many ways. The important think is that when we do this we can, given a number i, extract out which Turing Machine it corresponds to (if it does not correspond to one then we just say its the machine that halts in one step on any input). Hence we can (and will) say things like

- Let $M_1, M_2, M_3, \ldots$ be a list of all Turing Machines.
- Run $M_i(x)$. This is easy- given i, we can find M_i, — that is, find the code for it, and then run it on x.

Def 1.2 A set A is DECIDABLE if there is a Turing Machine M such that

$$x \in A \rightarrow M(x) = 1$$

$$x \notin A \rightarrow M(x) = 0$$

Def 1.3 Let $T(n)$ be a computable function (think of it as increasing). A is in $DTIME(T(n))$ if there is a TM M that decides A and also, for all x, $M(x)$ halts in time $\leq T(|x|)$.

Def 1.4 A is in Polynomial time (henceforth P) if there is a polynomial $p(n)$ such that A is in $DTIME(p(n))$.

2 There is a set that is NOT in P

Important Note: Imagine doing the following: Take a list of TMs $M_1, M_2, \ldots$ and then bound M_i by n^i. That is, when you run M_i if it has not halted by $|x|^i$ steps then shut it off and declare its answer to be 0. To save on notation we will also call this list

$$M_1, M_2, M_3, \ldots$$

KEY- if a set is in P then there is an i such that M_i decides it in poly time.
KEY- if M_i decides a set then it is in P.
Hence we have a list that represents all of P.

Theorem 2.1 *There exists a decidable set that is NOT in P.*

Proof:

Let $M_1, M_2, \ldots,$ represent all of P as described above.

We construct a set A to NOT be in P. We will want A and to DISAGREE with M_1, to DISAGREE with M_2, etc. Lets state this in terms of REQUIREMENTS

R_i : A and M_i differ on some string.

We want A to satisfy all of these requirements.

Here is our algorithm for A. It will be a subset of 0^*.

1. Input 0^i.
2. Run $M_i(0^i)$. If the results is 1 then output 0. If the results is 0 then output 1.

Note that, for all i, M_i and A DIFFER on 0^i. Hence every R_i is satisfied. Therefore $A \notin P$. ∎