Lecture I
COMPUTABILITY (I)

§1. COMPUTABILITY

1 Universal Turing Machines

So far, each TM is thought of solving one particular problem. We now introduce the concept of a Universal TM. This is supposed to be a 2-tape TM that simulates all simple TMs. Let U denote the universal TM.

To be more specific, we want U to have this behavior: assume that we encode any TM M by a string $\overline{M}$ in the alphabet of U. Also, any input string x of M is encoded as $\overline{x}$. If U is given the input $\overline{M}\underline{;}\overline{x}$ (here, "$\underline{;}$" is another special symbol of U), the we want the following equality to hold:

$$U(\overline{M}\underline{;}\overline{x}) = M(x).$$

In this sense, U simulates all other TM's.

Problem: U has a finite set of symbols and finitely many states; but the set of ALL Turing machines involves infinitely many symbols and infinitely many states.

Idea: we will basically use a 0/1 string to encode each symbol and each state of M. The particular choice of encoding is unimportant. Given a TM $M = (K, \Sigma, \delta, s)$, we only need to encode δ since K, Σ are (essentially) implicit in δ (the choice of s will be established by another convention we will state shortly). We represent δ as the encoding of a sequence of 5-tuples of the form

$$(q, a, q', a', D)$$

where D is the direction, q, q' are states and a, a' are symbols. We assume that each of these are $0, 1$ strings. Since $D \in \{\leftarrow, \rightarrow, -\}$, we can also assume that D is encoded by an element of $\{01, 10, 00\}$. Thus, the symbol set of U is

$$\{\underline{0}, \underline{1}, \underline{(}, \underline{)}, \underline{,}, \underline{;}\}$$

Similarly, the input string x needs to be encoded by some $\overline{x}$, which will be a 0/1 string. Here is how we obtain $\overline{x}$: first, each symbol in x is replaced by a 0/1-string (by a convention consistent with that for encoding $\overline{M}$). but each 1 is next replaced by the digram $\underline{10}$, 1 by $\underline{11}$. The separators between encodings of successive 0/1 strings will be the diagram $\underline{00}$. Sometimes, we need to encode a sequence of input words: in this case we can use the digram $\underline{01}$ to be the word separator.

It is not hard to construct a 3-tape Turing machine U that simulates $M(x)$ for any input $\overline{M}\underline{;}\overline{x}$. See text.

DISCUSSION: the book simply writes "M" instead of "$\overline{M}$". We prefer to keep the notational distinction between M and its representation. A point of confusion is that the book does not discuss the problem of when the input does not really encode a machine where one is expected. There are various conventions to get around this (e.g., such an encoding represents the null machine). But we need to be able to detect null machines, which is not hard. Further, the book often wants to discuss the computation of "M on input $\overline{M}$". What if $\overline{M}$ does not represent a valid input for M? E.g., the tape symbols of M has only one symbol "0" but $\overline{M}$ involves other symbols as well. Check that all these issues can be resolved!

2 HALTING PROBLEM

Let $HALT$ (or H) be the following language

$$H = \{\overline{M}\underline{;}\overline{x} : M(x) \downarrow\}.$$

Here $M(x) \downarrow$ means that M on input x does not go into any infinite loop (which has the notation $M(x) \uparrow$).

Proposition: $HALT$ is r.e..

Theorem: $HALT$ is not recursive.

The proof is very basic, and students should learn this by hard. It illustrates the concept of "reduction".

3 RICE'S THEOREM

This theorem is a fairly general one, and basically shows that any non-trivial questions we want to ask about the languages accepted by Turing machines are undecidable.

Let RE denote the class of r.e. languages. Let $C \subseteq RE$. We say C is **non-trivial** if $C \neq \emptyset$ and $C \neq RE$. Now define $L(C) := \{\overline{M} : L(M) \in C\}$.

THEOREM (RICE): If C is non-trivial then $L(C)$ is undecidable.