Recursive and R.e. sets

Theorem : The complement of a recursive language is recursive.

Theorem : Recursive sets are closed under union and intersection. R.e. sets are closed under union and interesection.

Theroem : If a set and its complement are both r.e., then the set is recursive.

Turing Machine codes : All TM's with a fixed alphabet can be encoded as finite length strings over some finite alphabet. This innocuous statement means that we may treat a TM's description itself as an input string to another TM. Further, the descrition can be "understood" and the described TM simulated by another TM. This is central to modern computing.

Universal Language Theorem $L_u = \{< M, w >: M$ is a legal TM accepting $w\}$ is an r.e. set.

Proof See handout from Hopcroft and Ullmann.

Theorem Undecidability of the Halting Problem L_u is not recrusive.

Proof Suppose L_u were recursive. Then we construct a new TM M^* which will lead to a contradiction. Given as input a binary string $w \in (\mathbf{0} + \mathbf{1})^*$, our machine M^* will check if $< w, w >$ is in L_u, which it can do since L_u is recursive. M^* accepts w if $< w, w > \notin L_u$ and rejects w if $< w, w > \in L_u$. Clearly, under our assumption, M^* is a legal TM (why did we have to assume that L_u is recursive for this assertion ?). The description of M^* can be viewed as a 0-1 string itself. If now $< M^*, M^* > \in L_u$, then our M^* would reject M^*, a contradiction. On the other hand, if $< M^*, M^* > \notin L_u$, then M^* accepts M^*, also a contradiction.

Theorem The properties of emptiness, recursiveness, and finiteness of r.e. sets are all undecidable.

Proof For Emptiness : We will prove this by "reducing" L_u to the set

$$EMPTY = \{M : L(M) = \emptyset\}.$$

For this, given a pair, $< M, w >$, our reduction f constructs the following TM $f(< M, w >)$:

$$f(< M, w >) = \begin{cases} \text{On input } x\text{, run } M \text{ on input } w \\ \text{If } M \text{ stops and accepts } w\text{, then accept } x \\ \text{If } M \text{ stops and rejects } w\text{, then reject } x \end{cases} \tag{1}$$

Note that if M did not terminate on w, then $f(< M, w >)$ also cycles on every input x (and hence accepts the empty set.) This completes the proof.

Definition A **reduction** from a language $L_1 \subseteq \Sigma^*$ to a language $L_2 \subseteq \Gamma^*$ is a **total recursive** function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall x \in \Sigma^*, x \in L_1 \quad \text{iff} \quad f(x) \in L_2.$$

Note that the above reduction has the following properties :

(i) $f(< M, w >)$ accepts the empty set iff M rejects w.

(ii) f is clearly recursive, even though it is not decidable whether M accepts or rejects w (and so also whether $f(< M, w >)$ is empty.) Thus we can write down in finite time the TM $f(< M, w >)$ given M, w, but in general cannot decide in finite time which option in (i) holds.

A non-trivial property of r.e. sets is a property which some r.e. set has and some (other) r.e. set does not have.

Rice's Theorem Any non-trivial property of r.e. sets is undecidable.

Proof Wlg, we may assume that the empty set has the property (or we complement the property). Also, there is an r.e. set, say accepted by M_0 which does not have the property. Then, in the above description of $f(< M, w >)$, we just replace "If M stops and accept w, then accept x" by "If M stops and accept w, then just run M_0 on x".

Post's Correspondance Problem PCP : We are given two lists of strings $A = \{w_1, w_2, \ldots w_k\}$ and $B = \{x_1, x_2, \ldots x_k\}$ over a finite alphabet. We are to decide whether there exist integers $i_1, i_2, \ldots i_m$ with $m \geq 1$ such that

$$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}.$$

Theorem PCP is indecidable.

Proof We do this by reducing L_u to L_{PCP}. Actually, the central part is the reduction of L_u to a modified PCP, where we are required to have $i_1 = 1$. This is done as follows : the final $w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$ will describe a valid computation of M on w (for a string $< M, w >$). The second list will always be one step ahead and each time, whenever we append a string to the first string, we will be forced to append the "next step" of M on w to the second string.