ENUMERABILITY : TURING MACHINES AS GENERATORS

Let $\Sigma = \{a_1, \ldots, a_n\}$. We define a **total ordering** on $\Sigma^* = \bigcup_{m \geq 0} \Sigma^m$ as follows.

(0) Assume $a_1 < a_2 < \ldots < a_n$.

(1) ε is the least element of Σ^*

(2) For each $m > 0$ there are only a finite number of strings of length m. (i.e. n^m different strings). For each $m > 0$, the strings in Σ^m are ordered lexicographically.

We call the above ordering a **proper ordering**.

When Turing machines are used to generate languages, and especially infinite languages we think of them as generating them in proper order.

Claim. Every string in Σ^* can be generated in proper order

$\vdash$ Starting with an empty tape, simply write a 0 and let the head be on 0. Clearly a representation of the empty string has been generated.

Copy this 0 on to the output tape's leftmost blank cell and move back to working tape. $\dashv$

Lemma. There exists a Turing machine which generates every natural number. $\dashv$

__Theorem__. The set of Turing machines is enumerable. i.e. there exists a Turing machine which can enumerate exactly all the Turing machines.

$\vdash$ We have already seen by the arithmetization that every Turing machine may be encoded as a natural number. Further we know that for each natural $n \geq 0$ there exists a primitive recursive predicate **istm** which determines whether a given natural number is the code of a Turing machine.

We therefore construct the required Turing machine by repeating the following steps.

1. Generate next natural number in succession and
2. Check if it is the code of a Turing machine.
3. If it represents a Turing machine write its code on the output tape

$\dashv$

We may use the same or similar method to generate languages over a given alphabet Σ.

The above enumeration procedure is called an __effective__ or __recursive enumeration__.

Recursive Enumerability and Recursiveness

Definition. A language $L \subseteq \Sigma^*$ is said to be **recursively enumerable (r.e.)** if there exists a Turing machine which accepts L.

Note that such a Turing machine which "accepts L" is guaranteed to terminate only for each $x \in L$. For $y \notin L$, there is no knowing whether or when it will terminate. In our model of the Turing machine we expect it not to terminate for any $y \notin L$.

Definition. A language $L \subseteq \Sigma^*$ is called **recursive** if there is a Turing machine which can determine for each $x \in \Sigma^*$ whether $x \in L$ or $x \notin L$.

In the case of a recursive language we expect the the Turing machine to **decide** the membership problem for L. i.e. **L is a recursive language iff the predicate $x \in L$ is total and decidable.**

Consequently we refer to a **recursively enumerable** language as being **semi-decidable**.

Claim. For any recursive language there exists an easy enumeration procedure.

$\vdash$ Repeat the following steps.

1) Generate the next string $x \in \Sigma^*$ in proper order.

2) Check whether $x \in L$.

3) Write x on output tape if $x \in L$.

Claim 2. There exists an enumeration procedure for each recursively enumerable language.

$\vdash$ Let T be a Turing machine which generates the strings of Σ^* in proper order. Let T_S be the Turing machine which accepts the language $L \subseteq \Sigma^*$.

Since membership in L is only semi-decidable, T_S may not halt on each input of Σ^*.

Let the strings of Σ^* generated in proper order be.

$$w_1, w_2, w_3, \ldots$$

Let T_U be the universal Turing machine which interleaves the computations of T and T_S as follows.

1a Let T generate w_1.

b. Let T_S make a single step move on input w_1.

2a. Let T generate w_2.

2b2. Let T_S make a single move on w_2 (if possible)

2b1. Let T_S make the next move on w_1. (if possible)

3a. Let T generate w_3.

3b3. Let T_S make a move on w_3 (if possible)

3b2. Let T_S make a move on w_2 (if possible)

3b1. Let T_S make a move on w_1 (if possible)

$\vdots$

For each $x \in L$, its membership in L is <u>eventually decided</u> and is output to the output. For each $y \notin L$, either membership is never decided or it is eventually decided that $y \notin L$. In either case y is never written out on the output tape.

Hence even if L is infinite every element of L takes a finite time to appear on the output tape and hence is effectively enumerated $\dashv$

<u>Cor</u>. Every recursive language is also recursively enumerable.

$\dashv$

Cor. If $L \subseteq \Sigma^*$ is recursive then so is $\overline{L} = \Sigma^* - L$

$\vdash$ Since membership in L is (total) decidable it follows that membership is also decidable in $\overline{L}$. Hence $\overline{L}$ is also recursive. $\dashv$

Cor. If $L \subseteq \Sigma^*$ and $\overline{L}$ are both recursively enumerable then both L and $\overline{L}$ are recursive.

$\vdash$ Let T and $\overline{T}$ be respectively the Turing machines which decide $x \in L$ and $x \in \overline{L}$. For each input $w \in \Sigma^*$, let T_U interleave the computations of T and $\overline{T}$. Eventually one of them will accept and output its result on an output tape. The universal Turing machine waits till one of the outputs becomes available and stops running the other machine. $\dashv$

Theorem. A language $L \subseteq \Sigma^*$ is recursive iff both L and $\overline{L}$ are r.e.

$\vdash$ Clearly if L is recursive then both L and $\overline{L}$ are r.e. Further if both L and $\overline{L}$ are r.e. it follows from the previous corollary that L must be recursive $\dashv$

Lemma. For any $\Sigma \neq \varphi$, there exist $L \subseteq \Sigma^*$ that are not recursively enumerable.

$\vdash$ Since there are a countable number of strings in Σ^*, there are an uncountable number of languages $L \subseteq \Sigma^*$. However there are only a countable number of Turing machines. Hence there are languages which are not recursively enumerable $\dashv$

Theorem. There exists a r.e. language whose complement is not r.e.

$\vdash$ Consider an enumeration of all the Turing machines which accept languages $L \subseteq \{a\}^*$. Clearly this is a countable effectively enumerable set (say)

$$T_0, T_1, T_2, \ldots \qquad \text{——— (1)}$$

Now consider the language

$$L = \{a^i \mid i \in \mathbb{N},\ a^i \in \mathcal{L}(T_i)\} \subseteq \{a\}^*$$

Claim. $\overline{L}$ is not r.e.

$\vdash$ Assume $\overline{L}$ is r.e. Then there exists a Turing machine $T = T_k$ in the enumeration which accepts $\overline{L}$. Now consider the string a^k.

Question $a^k \in \overline{L}$?

If $a^k \in \overline{L}$ then $a^k \in \mathcal{L}(T_k)$ which implies $a^k \in L$ and hence $a^k \notin \overline{L}$. But if $a^k \notin \overline{L}$ then $a^k \in L$ which implies $a^k \in \mathcal{L}(T_k)$, which implies $a^k \notin \mathcal{L}(T_k)$

Claim. L is r.e.

$\vdash$ For each a^i, we call T_i and get it to try to decide membership by interleaving the computations of the various Turing machines as was done in the proof for recursive enumerability. Eventually each T_i for which $a^i \in \mathcal{L}(T_i)$ will halt in the final state and each T_j for which $a^j \notin \mathcal{L}(T_j)$ will never halt. Hence every element of L can be generated. $\dashv$