Enumerable and Recursive Sets

Unfortunately, it seems that very few of the general problems concerning the nature of computation are solvable. Now is the time to take a closer look at some of these problems and in classify them with regard to a finer metric. To do this we need more precision and formality. So, we shall bring forth a little basic mathematical logic.

Examining several of the sets with unsolvable membership problems, we find that while we cannot decide their membership problems, we are often able to determine when an element is a member of some set. In other words, we know if a Turing machine accepts a particular set and halts for some input, then that input is a member of a set. Thus the Turing machine halts for members of the set and provides no information about inputs that are not members. An example is K, the set of Turing machines that halt when given their own indices as input. Recalling that

\[ K = \{ i \mid M_i(i) \text{ halts} \} = \{ i \mid i \in W_i \} \]

consider the machine \( M \) that can be constructed from the universal Turing machine \( M_u \) as follows.

\[ M(i) = M_u(i, i) \]

Another way to describe \( M \) (possibly more intuitively) is:

\[ M(i) = \begin{cases} \text{halt if } M_i(i) \text{ halts} \\ \text{diverge otherwise} \end{cases} \]

This is a Turing machine. And, since it was just \( M_u(i, i) \) we know exactly how to build it and even find its index in our standard enumeration. Furthermore, if we examine it carefully, we discover that it accepts the set \( K \). That is, \( M \) will halt for all inputs which are members of \( K \) but diverge for nonmembers. There is an important point about this that needs to be stressed.

If some integer \( x \) is a member of \( K \) then \( M(x) \) will always tell us so. Otherwise, \( M(x) \) provides us with absolutely no information.

This is because we can detect halting but cannot always detect divergence. After all, if we knew when a machine did not halt, we would be able to solve the
halting problem. In fact, there are three cases of final or terminal behavior in the operation of Turing machines:

a) halting,
b) non-halting which we might detect, and
c) non-detectable divergence.

The latter is the troublesome kind that provides us with unsolvability.

Some of the computable sets have solvable membership problems (for example, the sets of even integers or prime numbers) but many such as K do not. In traditional mathematical logic or recursion theory we name our collection of computable sets the class of recursively enumerable sets. There is a reason for this exotic sounding name that will be completely revealed below. The formal definition for members of the class follows.

**Definition.** A set is recursively enumerable (abbreviated r.e.) if and only if it can be accepted by a Turing machine.

We call this family of sets the class of r.e. sets and earlier we discovered an enumeration of all of them which we denoted \( W_1, W_2, \ldots \) to correspond to our standard enumeration of Turing machines. Noting that any set with a solvable membership problem is also an r.e. set (as we shall state in a theorem soon) we now present a definition of an important subset of the r.e. sets and an immediate theorem.

**Definition.** A set is recursive if and only if it has a solvable membership problem.

**Theorem 1.** The class of recursively enumerable (r.e.) sets properly contains the class of recursive sets.

**Proof.** Two things need to be accomplished. First, we state that every recursive set is r.e. because if we can decide if an input is a member of a set, we can certainly accept the set. Next we present a set that does not have a solvable membership problem, but is r.e. That of course, is our old friend, the diagonal set K.

That is fine. But, what else do we know about the relationship between the r.e. sets and the recursive sets? If we note that since we have total information about recursive sets and only partial information about membership in r.e. sets, the following characterization of the recursive sets follows very quickly.

**Theorem 2.** A set is recursive if and only if both the set and its complement are recursively enumerable.
Proof. Let $A$ be a recursive set. Then its complement $\overline{A}$ must be recursive as well. After all, if we can tell whether or not some integer $x$ is a member of $A$, then we can also decide if $x$ is not a member of $A$. Thus both are r.e. via the last theorem.

Suppose that $A$ and $\overline{A}$ are r.e. sets. Then, due to the definition of r.e. sets, there are Turing machines that accept them. Let $M_a$ accept the set $A$ and $M_{\overline{A}}$ accept its complement $\overline{A}$. Now, let us consider the following construction.

$$M(x) = \begin{cases} 
1 & \text{if } M_a(x) \text{ halts} \\
0 & \text{if } M_{\overline{A}}(x) \text{ halts} 
\end{cases}$$

If we can build $M$ as a Turing machine, we have the answer because $M$ does solve membership for the set $A$. But, it is not clear that $M$ is indeed a Turing machine. We must explain exactly how $M$ operates. What $M$ must do is to run $M_a(x)$ and $M_{\overline{A}}(x)$ at the same time. This is not hard to do if $M$ has four tapes. It uses two of them for the computation of $M_u(a, x)$ and two for the computation of $M_u(\overline{a}, x)$ and runs them in time-sharing mode (a step of $M_a$, then a step of $M_{\overline{A}}$). We now note that one of $M_a(x)$ and $M_{\overline{A}}(x)$ must halt. Thus $M$ is indeed a Turing machine that decides membership for the set $A$.

From this theorem come several interesting facts about computation. First, we gain a new characterization of the recursive sets and solvable decision problems, namely, both the problem and its complement are computable. We are also soon be able to present our first uncomputable or non-r.e. set. In addition, another closure property for the r.e. sets falls out of this examination. Here are these results.

**Theorem 3.** The complement of $K$ is not a recursively enumerable set.

Proof. The last theorem states that if $\overline{K}$ were r.e. then both it and its complement must be recursive. Since $K$ is not a recursive set, $\overline{K}$ cannot be an r.e. set.

**Corollary.** The class of r.e. sets is not closed under complement.

Remember the halting problem? Another one of the ways to state it is as a membership problem for the set of pairs:

$$H = \{ <i, x> \mid M_i(x) \text{ halts} \} = \{ <i, x> \mid x \in W_i \}.$$
We have shown that it does not have a solvable membership problem. A little bit of contemplation should be enough to convince anyone that is an r.e. set just like $K$. But, what about its complement? The last two theorems provide the machinery to show that it also is not r.e.

**Theorem 4.** The complement of the halting problem is not r.e.

Proof. We now know two ways to show that the complement of the halting problem, namely the set $\{<i, x> | M_i(x) \text{ diverges}\}$ is not an r.e. set. The first is to use theorem 2 that states that a set is recursive if and only if both it and its complement are r.e. If the complement of the halting problem were r.e. then the halting problem would be recursive (or solvable). This is not so and thus the complement of the halting problem must not be r.e.

Another method is to note that if $\{<i, x> | M_i(x) \text{ diverges}\}$ were r.e. then $\overline{K}$ would have to be r.e. also. This is true because we could use the machine that accepts the complement of the halting problem in order to accept $\overline{K}$. Since $\overline{K}$ is not r.e. then the complement of the halting problem is not either.

The second method of the last proof brings up another fact about reducibilities. It is actually the r.e. version of a corollary to the theorem stating that if a nonrecursive set is reducible to another then it cannot be recursive either.

**Theorem 5.** If $A$ is reducible to $B$ and $A$ is not r.e. then neither is $B$.

Proof. Let $A$ be reducible to $B$ via the function $f$. That is, for all $x$:

$$x \in A \iff f(x) \in B.$$  

Let us assume that $B$ is an r.e. set. That means that there is a Turing machine $M_b$ that accepts $B$. Now construct $M$ in the following manner:

$$M(x) = M_b(f(x))$$

and examine the following sequence of events.

$$x \in A \iff f(x) \in B \text{ [since } A \leq B]$$

$$\iff M_b(f(x)) \text{ halts [since } M_b \text{ accepts } B]$$

$$\iff M(x) \text{ halts [due to definition of } M]$$
This means that if \( M \) is a Turing machine (and we know it is because we know exactly how to build it), then \( M \) accepts \( A \). Thus \( A \) must also be an r.e. set since the r.e. sets are those accepted by Turing machines.

Well, there is a contradiction! \( A \) is not r.e. So, some assumption made above must be wrong. By examination we find that the only one that could be wrong was when we assumed that \( B \) was r.e.

Now the time has come to turn our discussion to functions instead of sets. Actually, we shall really discuss functions and some of the things that they can do to and with sets. We know something about this since we have seen reducibilities and they are functions that perform operations upon sets.

Returning to our original definitions, we recall that Turing machines compute the computable functions and some compute total functions (those that always halt and present an answer) while others compute partial functions which are defined on some inputs and not on others. We shall now provide names for this behavior.

**Definition.** A function is (total) **recursive** if and only if it is computed by a Turing machine that halts for every input.

**Definition.** A function is partial recursive (denoted prf) if and only if it can be computed by a Turing machine.

This is very official sounding and also quite precise. But we need to specify exactly what we are talking about. Recursive functions are the counterpart of recursive sets. We can compute them totally, that is, for all inputs. Some intuitive examples are:

\[
f(x) = 3x^2 + 5x + 2
\]

\[
f(x, y) = \begin{cases} x \text{ if } y \text{ is prime} \\ 0 \text{ otherwise} \end{cases}
\]

Partial recursive functions are those which do not give us answers for every input. These are exactly the functions we try not to write programs for! This brings up one small thing we have not mentioned explicitly about reducibilities. We need to have answers for every input whenever a function is used to reduce one set to another. This means that the reducibility functions need to be recursive. The proper traditional definition of reducibility follows.
Definition. The set A is reducible to the set B (written $A \leq B$) if and only if there is a recursive function $f$ such that for all $x$: 

$$x \in A \text{ if and only if } f(x) \in B.$$ 

Another thing that functions are useful for doing is set enumeration (or listing). Some examples of set enumeration functions are:

- $e(i) = 2^i = \text{the } i^{\text{th}} \text{ even number}$
- $p(i) = \text{the } i^{\text{th}} \text{ prime number}$
- $m(i) = \text{the } i^{\text{th}} \text{ Turing machine encoding}$

These are recursive functions and we have mentioned them before. But, we have not mentioned any general properties about functions and the enumeration of the recursive and r.e. sets. Let us first define what exactly it means for a function to enumerate a set.

Definition. The function $f$ enumerates the set $A$ (or $A = \text{range of } f$), if and only if for all $y$,

a) If $y \in A$, then there is an $x$ such that $f(x) = y$ and 

b) If $f(x) = y$ then $y \in A$.

Note that partial recursive functions as well as (total) recursive functions can enumerate sets. For example, the function:

$$k(i) = \begin{cases} 
  i & \text{if } M_i \text{ halts} \\
  \text{diverge} & \text{otherwise}
\end{cases}$$

is a partial recursive function that enumerates the set $K$. Here is a general theorem about the enumeration of r.e. sets which explains the reason for their exotic name.

Theorem 6. A set is r.e. if and only if it is empty or the range of a recursive function.

Proof. We shall do away with one part of the theorem immediately. If a set is empty then of course it is r.e. since it is recursive. Now what we need to show is that non-empty r.e. sets can be enumerated by recursive functions and that any set enumerated by a recursive function is r.e.

a) If a set is not empty and is r.e. we must find a recursive function that enumerates it.
Let \( A \) be a non-empty, r.e. set. We know that there is a Turing machine (which we shall call \( M_a \)) which accepts it. Since \( A \) is not empty, we may assume that there is some input (let us specify the integer \( k \)) which is a member of \( A \). Now consider:

\[
M(x, n) = \begin{cases} 
  x & \text{if } M_a(x) \text{ halts in exactly } n \text{ steps} \\
  k & \text{otherwise}
\end{cases}
\]

We claim that \( M \) is indeed a Turing machine and we must demonstrate two things about it. First, the range of \( M \) is part of the set \( A \). This is true because \( M \) either outputs \( k \) (which is a member of \( A \)) or some \( x \) for which \( M_a \) halted in \( n \) steps. Since \( M_a \) halts only for members of \( A \), we know that the range of \( M \subseteq A \).

Next we must show that our enumerating machine \( M \) outputs all of the members of \( A \). For any \( x \in A \), \( M_a(x) \) must halt in some finite number (let us say \( m \)) of steps. Thus \( M(x, m) = x \). So, \( M \) eventually outputs all of the members of \( A \). In other words:

\[
A \subseteq \text{range of } M
\]

and we can assert that \( M \) exactly enumerates the set \( A \).

(N.B. This is not quite fair since enumerating functions are supposed to have one parameter and \( M \) has two. If we define \( M(z) \) to operate the same as the above machine with:

\[
x = \text{number of zeros in } z \\
n = \text{number of ones in } z
\]

then everything defined above works fine after a little extra computation to count zeros and ones. This is because sooner or later every pair of integers shows up. Thus we have a one parameter machine \( M(z) = M(<x, n>) \) which enumerates \( A \).)

We also need to show that \( M \) does compute a recursive function. This is so because \( M \) always halts. Recall that \( M(x, n) \) simulates \( M_a(x) \) for exactly \( n \) steps and then makes a decision of whether to output \( x \) or \( k \).
b) The last part of the proof involves showing that if A is enumerated by some recursive function (let us call it f), then A can be accepted by a Turing machine. So, we shall start with A as the range of the recursive function f and examine the following computing procedure.

```
AcceptA(x)
  n = 0;
  while f(n) ≠ x do n = n + 1;
  halt
```

This procedure is computable and halts for all x which are enumerated by f (and thus members of A). It diverges whenever x is not enumerated by f. Since this computable procedure accepts A, we know that A is an r.e. set.

This last theorem provides the reason for the name recursively enumerable set. In recursion theory, stronger results have been proven about the enumeration of both the recursive and r.e. sets. The following theorems (provided without proof) demonstrate this.

**Theorem 7.** A set is recursive if and only if it is finite or can be enumerated in strictly increasing order.

**Theorem 8.** A set is r.e. if and only if it is finite or can be enumerated in non-repeating fashion.