Recursive and recursively enumerable sets

So far we have concentrated on the aspect of algorithms to do with computing functions from inputs to outputs. Another important use of algorithms is to generate, or enumerate, the elements of some set of data.

One says that a set $S$ is effectively enumerable if there is some algorithm $A$ which lists the elements of $S$:

$$S = \{ A(0), A(1), A(2), \ldots \}$$

(It may well be that an element of $S$ occurs many times in the list, but no matter.)
Example:

The set \( PR \) of partial recursive functions is effectively enumerated by the algorithm \( A \) which, given input \( x \), decodes \( x \) as a pair \( x = \langle n, e \rangle \), then decodes \( e \) as a register machine program \( \text{Pro}_e \), and returns the \( n \)-ary computable (hence partial recursive) function \( \varphi_e^{(n)} \), where

\[
\varphi_e^{(n)}(x_1, \ldots, x_n) = y \iff \text{computation of \( \text{Pro}_e \) started with } R_1, \ldots, R_n \text{ set to } x_1, \ldots, x_n \text{ halts with } R_0 = y
\]

(because every element of \( PR \) is of the form \( \varphi_e^{(n)} \) for some \( n \) \& \( e \)).

Clearly, \( S \) has to be a countable set if it is effectively enumeratable.

[Recall:

\( S \) is countably infinite if there is some bijection (= one-one and onto function) between \( \mathbb{N} \) and \( S \).

\( S \) is countable if it is either finite or countably infinite.

\( S \) is uncountable if it is not countable.

E.g. \( \text{Fun}(\mathbb{N}, \mathbb{N}) \) is uncountable, by Cantor’s Diagonal Argument.]

The notion of “effective enumerability” is an informal one, because it refers to the informal notion of “algorithm”. We can formalize it using the notion of computable (= partial recursive) function provided we identify the set \( S \) to be enumerated with a subset of \( \mathbb{N} \).... (Since \( S \) is necessarily countable, we can always do this some way).
DEFINITION:

A subset $S \subseteq \mathbb{N}$ of numbers is recursively enumerable (or r.e., for short) if and only if either it is empty ($S = \emptyset$) or there is a (total) recursive function $f \in \text{Fun}(\mathbb{N}, \mathbb{N})$ so that

$$S = \{ f(n) \mid n \in \mathbb{N} \}$$

Recall: $S \subseteq \mathbb{N}$ is decidable if and only if the characteristic function of $S$

$$\chi_S(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

is computable. (cf. p55)

Such sets are also called recursive (since $\chi_S$ is computable if and only if it is recursive, being a total function).

PROPOSITION:

Every recursive set is recursively enumerable.
Proof
Suppose \( S \) is recursive. If \( S = \emptyset \), then \( S \) is r.e. by definition; otherwise we can find some \( x_0 \in S \). Then since \( \chi_S \) is recursive, so is

\[
f(x) \overset{\text{def}}{=} \text{ifzero}(\chi_S(x), x_0, x)
\]

and \( S = \{ f(x) \mid x \in \mathbb{N} \} \), so \( S \) is r.e.

In the section on the Halting Problem we saw that the set

\[
\{ e \in \mathbb{N} \mid \varphi_e \text{ is a total function} \}
\]

is undecidable. In fact it is not even recursively enumerable...

---

**Example of a non-r.e. set**

\[
\text{TOT} \overset{\text{def}}{=} \{ e \in \mathbb{N} \mid \varphi_e \text{ is a total function} \}
\]

is not recursively enumerable.

**Proof**

If \( \text{TOT} \) were r.e., then (since \( \text{TOT} \neq \emptyset \)) \( \text{TOT} = \{ f(x) \mid x \in \mathbb{N} \} \) for some recursive function \( f \in \text{Fun}(\mathbb{N}, \mathbb{N}) \).

Let \( u \in \text{Pfn}(\mathbb{N}^2, \mathbb{N}) \) be the partial function \( u(e, x) \overset{\text{def}}{=} \varphi_e(x) \)

**Claim**

1. \( u \) is partial recursive; hence so is \( g(x) \overset{\text{def}}{=} u(f(x), x) + 1 \)
2. \( g \) is total; hence \( g = \varphi_e \) for some \( e \in \text{TOT} \), but
3. \( e \neq f(x) \) for any \( x \in \mathbb{N} \) — contradiction!
Proof of the Claims:
(1) follows from the work we did in the section on a universal register machine $U$, since $u(e,x)$ is the result (if any) of running $U$ starting with $P = e$ and $A = [x]$. Thus $u$ is computable, and hence is partial recursive.

(2) Since by assumption on $f$, for all $x \in \mathbb{N}$, $f(x) \in \text{TOT}$ so $\varphi_{f(x)}(x) \downarrow$, so $g(x) \downarrow$ (by definition of $g$). Thus $g$ is total, recursive, and hence $g = \varphi_e$ for some $e \in \text{TOT}.

(3) If $e = f(x)$, then
\[ g(x) = \varphi_e(x) \quad \text{since} \quad g = \varphi_e \]
\[ = u(e,x) \quad \text{by definition of } U \]
\[ = u(e,x) + 1 \quad \text{since} \quad u(e,x) = g(x) \downarrow \]
\[ = u(f(x),x) + 1 \quad \text{since} \quad e = f(x) \]
\[ = g(x) \quad \text{by definition of } g \]
contradiction. So $e \neq f(x)$ for any $x$, contradicting the assumption that $f$ enumerates $\text{TOT}$ (since $e \in \text{TOT}$).

---

**Example** of an r.e. set that is not recursive

is provided by the undecidability of the Halting Problem, which in particular implies that
\[ H \overset{\text{def}}{=} \{ e \in \mathbb{N} \mid \varphi_e(0) \downarrow \} \quad [\text{cf. } S_2 \text{ on p. 56}] \]
is undecidable, i.e. is not recursive. But $H$ is r.e. because $H = \text{Dom}(f)$ the domain (of definedness) of the partial recursive function $f(x) \overset{\text{def}}{=} u(x,0)$ (where $u$ is as above) and in general we have...
**Proposition:**
For a subset $S \subseteq \mathbb{N}$, the following are equivalent:

1. $S$ is recursively enumerable
2. $S = \text{Im}(f)$, the image of a (unary) partial recursive function
3. $S = \text{Dom}(f)$, the domain of a unary partial recursive function
4. $S$ is semi-decidable, meaning that the partial function
   \[ \text{in}_S(x) = \begin{cases} 1 & \text{if } x \in S \\ \text{undefined} & \text{if } x \notin S \end{cases} \]
   is partial recursive.

**Notation:**
Given a partial function $f \in \text{Pfn}(X, Y)$
- $\text{Dom}(f) \overset{\text{def}}{=} \{ x \in X \mid f(x) \downarrow \}$ the domain (of definedness) of $f$
- $\text{Im}(f) \overset{\text{def}}{=} \{ y \in Y \mid \text{for some } x \in X, f(x) = y \}$ the image of $f$

**Proof of the Proposition**
We will show $(2) \Rightarrow (1) \Rightarrow (3) \Rightarrow (4) \Rightarrow (2)$.
In all cases the implications are trivial if $S$ is empty (since in $\varnothing$ = completely undefined function, is partial recursive and has domain & image = $\varnothing$). So we can assume $S \neq \varnothing$, say $x_0 \in S$. 


(2) ⇒ (1):

Let $M$ be a register machine computing $f(a)$ in $R_0$ when started with $R_1 = a$.

Construct a new machine $M'$ computing as follows:

- decode $R_1$ as a pair $<a, t>$;
- run $M$ for $t$ steps starting with $R_1 = a$ and if it halts by then, set $RO$ to the value it computes in $R_0$, else set $RO$ to $x_0$.

Let $f'$ be the unary function computed by $M'$ (in $R_0$, starting with input in $R_1$).

By construction $f'$ is total recursive and $f'(x) \in S$ for all $x \in \mathbb{N}$ (since $M$ only computes values $f(a)$ that lie in $S$).

Conversely, if $y \in S = \text{Im}(f)$, then $y = f(a)$ for some $a$.

Now $M$ computes $f(a)$ in a finite number of steps starting from $R_1 = a$, say $t$ steps. Then by construction of $M'$ $f'( <a, t>) = f(a) = y$. Thus every element of $S$ is enumerated by the recursive function $f'$ — so $S$ is r.e.

Remark: Using the techniques of the proof $f$ computable $\Rightarrow f \in \mathbb{P}$ one can show that $S$ is enumerated by a primitive recursive function, since

$$f(x) = \text{if}(x \in \text{Im}(f), \text{val}_0(\text{state}(\pi_1(x), \pi_2(x)), x_0))$$

where $\pi_1, \pi_2$ are primitive recursive projection functions satisfying

$\pi_1(<a, t>) = a$, $\pi_2(<a, t>) = t$, $<\pi_1(x), \pi_2(x)> = x$.  \]
(1) \Rightarrow (3):

Since we are assuming \( S \neq \emptyset \),
\[ S = \{ f(n) \mid n \in \mathbb{N} \} \]
for some recursive function \( f \).

Then
\[ g(x, y) = \begin{cases} 0 & \text{if } f(y) = x \\ 1 & \text{if } f(y) \neq x \end{cases} \]

is also recursive, since \( g(x, y) = 1 - \text{eq}(f(y), x) \).

Thus \( \mu(g) \) is partial recursive, and

\[ x \in \text{Dom}(\mu(g)) \iff \mu(g)(x) \downarrow \]

\[ \iff g(x, y) = 0 \text{ for some } y \]

\[ \iff f(y) = x \text{ for some } y \]

\[ \iff x \in S \]

Thus \( S = \text{Dom}(\mu(g)) \), as required.

(3) \Rightarrow (4):

If \( S = \text{Dom}(f) \) with \( f \in \text{PR} \), then
\[ \text{ins}_S(x) \equiv \text{if zero}(f(x), 1, 1) \]

is also partial recursive, hence computable: so \( S \) is semi-decidable.

(4) \Rightarrow (2):

Let \( M \) be a register machine computing \( \text{ins}_S(x) \)
in \( R_0 \) when started with \( x \) in \( R_1 \).

Then

\[ \text{START} \rightarrow X := R_1 \rightarrow M \rightarrow R_0 := X \rightarrow \text{HALT} \]

computes the partial recursive function
\[ f(x) \equiv \begin{cases} x & \text{if } \text{ins}_S(x) \downarrow \\ \uparrow & \text{if } \text{ins}_S(x) \uparrow \end{cases} \]

and hence \( \text{Im}(f) = S \).
**Definition:**
A subset $S \subseteq \mathbb{N}$ is called co-r.e. if $\mathbb{N} \setminus S (= \{ x \in \mathbb{N} \mid x \notin S \})$ is r.e.

**Proposition:**
$S$ is recursive if and only if it is both r.e. and co-r.e.

**Proof:**
$$
\chi_S(x) = \begin{cases} 
0 & \text{if } x \notin S \\
1 & \text{if } x \in S 
\end{cases}
$$

$$
\chi_{\mathbb{N} \setminus S}(x) = \begin{cases} 
0 & \text{if } x \in S \\
1 & \text{if } x \notin S 
\end{cases}
$$

$$
= \text{ifzero}(\chi_S(x), 1, 0)
$$

So $S$ recursive $\Rightarrow \mathbb{N} \setminus S$ recursive

So $S$ recursive $\Rightarrow S \& \mathbb{N} \setminus S$ both r.e.

Conversely...
Suppose \( S \) enumerated by recursive function \( \{ f, g \} \).

Let \( M \) be register machine which when started with \( x \) in \( R \):

- Computes successive values of the sequence \( g(0), f(0), g(1), f(1), g(2), f(2), \ldots \)
- Halting (at \( n^{th} \) place in sequence, say) first time get a value \( = x \), and returning \( \{ 0 \} \) in \( R \) if \( n \) is \( \text{even} \).
- Returning \( \{ 1 \} \) in \( R \) if \( n \) is \( \text{odd} \).

Then \( M \) decides membership of \( S \), because...

\[ M \text{ is guaranteed to halt because } f \text{ and } g \text{ are total; and} \]

\[ \text{either } x \in S \text{ in which case } x = f(n), \text{ some } n \]
\[ \text{or } x \notin S \text{ in which case } x = g(n), \text{ some } n. \]

More formally, \( \chi_S(x) = \text{mod}_2(\mu(h)(x)) \), where

\[ h(x, y) \triangleq 1 - \text{eq}(x, \text{ifzero}(\text{mod}_2(y), g(\text{half}(y)), f(\text{half}(y)))) \text{ and mod}_2, \text{half}, \text{eq} \text{ were defined on pages 120 & 124.} \]

Thus \( \chi_S \) is recursive because \( f \) & \( g \) are and because \( \text{eq, ifzero, mod}_2, \text{ half are (primitive) recursive.} \)
SUMMARY

- Formalization of intuitive notion of **ALGORITHM** in several equivalent ways
cf. "Church-Turing Thesis"

- Limitative results: undecidable problems
  uncomputable functions
  "programs as data" + diagonalization