Recursive and Recursively Enumerable Sets

Partial recursive functions can be used to classify sets of natural numbers.
Turing machines can be used to classify sets of natural numbers and sets of strings.
We’ll focus on sets of natural numbers.

The definitions can be extended to cover sets of strings as well.

Background:

• A number-theoretic function is partial recursive iff it is Turing-computable. (Proven result)

• Church-Turing Thesis: A number-theoretic function is partial recursive (and Turing-computable) iff there is some algorithm for computing the function.
Acceptors, Recognizers, and Generators

An acceptor for set S answers yes if an element is in the set, and has any other behavior if the element is out of the set.

A recognizer for set S answers yes if an element is in the set, and answers no if the element is not in the set.

A generator for set S outputs the elements of S one at a time, without leaving any element out forever. Elements can be printed in any order, and may be printed more than once.

We can use either partial recursive functions or Turing Machines as acceptors, recognizers, and generators.
Important definitions and theorems

Definition: Recursively enumerable set
A set $S$ is recursively enumerable (r.e.) iff either $S$ is the empty set or $S = \text{image}(f)$ for some total recursive function $f$.

Examples: Recursively enumerable sets
• The set of even numbers is recursively enumerable, because it is the image set of $f(x) = 2 \times x$. The function $2 \times x$ is partial recursive, and it is defined for all $x$; hence, it is total recursive.

  
  \begin{align*}
  f(0) &= 0 \\
  f(1) &= 2 \\
  f(2) &= 4 \\
  f(3) &= 6 \\
  & \text{(etc.)}
  \end{align*}

• The set \{1, 2, 3\} is recursively enumerable because it is the image set of $f(x) = \{x \text{ if } x = 1, 2, \text{ or } 3; \text{ and } 1 \text{ otherwise}\}$. $f$ is partial recursive (since defined by cases using constants and $=$), and it is defined for all $x$; hence, it is total recursive.

  
  \begin{align*}
  f(0) &= 1 \\
  f(1) &= 1 \\
  f(2) &= 2 \\
  f(3) &= 3 \\
  f(4) &= 1 \\
  & \text{(etc.)}
  \end{align*}

Note that \{1, 2, 3\} is also the image of other functions (e.g., $g(x) = (x \mod 3) + 1$); there can be many enumerations of a set.
A set is recursively enumerable if there is a computable (algorithmic, “effective”) way to list all of its members without leaving any out.

For any $n \in S$, there is some $x$ such that $n = f(x)$.

Repetition is allowed (as in the 2nd example above): $n$ may be $f(x)$ for many different $x$s.
Definition: Recursive set

A set $S$ is recursive iff $S$ has a partial recursive characteristic function.

*Since all characteristic functions are total, this means that $S$ has a total recursive characteristic function. The central point of the definition is that the characteristic function is computable for all inputs.*

Examples: Recursive sets

The set of all numbers less than 3 is a recursive set:

$$\chi_S(n) = \begin{cases} 
1 & \text{if } n < 3 \\
0 & \text{otherwise}
\end{cases}$$

The set of all numbers greater or equal to 3 is also a recursive set:

$$\chi_S(n) = \begin{cases} 
1 & \text{if } n \geq 3 \\
0 & \text{otherwise}
\end{cases}$$
**Theorem:** S is recursively enumerable iff $S = \text{domain}(f)$ for some unary partial recursive function $f$.

*Note 1:* Here, domain$(f)$ means the set of all natural numbers for which $f$ is defined.

*Note 2:* “$S = \text{image}(g)$” makes $g$ a kind of generator for $S$. “$S = \text{domain}(f)$” makes $f$ a kind of acceptor for $S$.

*Note 3:* We don’t have non-total characteristic functions. $f$ takes the place of a characteristic function for r.e. sets: instead of looking for 1/0, we look for defined/undefined.

**Proof:**

*Case 1: S is empty*

The theorem is an “iff” assertion, so the proof is in 2 parts.

a) **∅ is recursively enumerable:**

∅ is explicitly stated in definition of “recursively enumerable”

b) **∅ is the domain of a unary partial recursive function f:**

If $f$ is the everywhere-undefined function, then ∅ is the domain of $f$.

Mn[$C_1^2$] is one definition of $f$.

That is, $f(n) = \mu m[C_1^2(n, m) = 0 \text{ and } C_1^2(x, m) \text{ is defined for all } x < n]$. 

Copyright © 2014 Nancy Lynn Tinkham
Case 2: $S$ is nonempty

The theorem is an “iff” assertion, so the proof is in 2 parts.

a) If $S = \text{image}(g)$ for some total recursive function $g$, then $S = \text{domain}(f)$ for some unary partial recursive function $f$.

Idea:

We have $g \rightarrow S$ \hspace{1cm} We have a generator
We want $S \rightarrow f$ \hspace{1cm} We want to build an acceptor

Assume that there exists a total recursive function $g$ such that $S = \text{image}(g)$.

Define $f$ thus:

$$f(n) = \begin{cases} 1 & \text{if } \exists i \text{ such that } g(i) = n \\ \text{undefined} & \text{otherwise} \end{cases}$$

Compute $f(n)$ by calculating $g(0), g(1), \text{etc.}$, until $i$ is found.

If there is an $i$, eventually the procedure will halt, and $f$ returns 1.

If there is no such $i$, the procedure will never halt, and $f$ is undefined.
b) If $S = \text{domain}(f)$ for some unary partial recursive function $f$, then $S = \text{image}(g)$ for some total recursive function $g$.

Idea:

We have $S \rightarrow \fbox{f}$  \hspace{1cm} \text{We have an acceptor}

We want $\fbox{g} \rightarrow S$ \hspace{1cm} \text{We want to build a generator}

Since $f$ is a partial recursive function, it can be computed by some Turing Machine $M$.

We will construct a TM $M_1$ that writes elements of $S$ on a tape. From this, we construct a TM $M_2$ that computes a function $g$ such that $S = \text{image}(g)$.

First draft: $M_1$ follows this algorithm:

Compute $f(0)$ by simulating $M$.

If $M$ halts with a number as output, write 0 on the output tape.

If $M$ does not halt (or halts with a non-number), write nothing and proceed to the next step.

Repeat for $f(1)$, $f(2)$, $f(3)$, etc.

Each time that $M$ halts with a number as output when computing $f(i)$, write $i$ on the output tape.

Define $g(n)$ to be the $n^{th}$ number written on the tape in this fashion. $M_2$ computes $g(n)$ by simulating $M_1$ until $n$ numbers have been written on the tape, then reporting that $n^{th}$ number.

This algorithm has a serious flaw. What is it?
Problem: If $f(x)$ is undefined for some $x$, $M$ won’t halt for that $x$, and $M_1$ will get stuck.
Second draft: \( M_1 \) follows this algorithm, which uses dovetailing:

- Simulate \( M \) on input 0 for 1 step.
- Simulate \( M \) on inputs 0 and 1 for 2 steps.
- Simulate \( M \) on inputs 0, 1, and 2 for 3 steps.
- Etc.
- Any time that \( M \) halts with input \( n \), write \( n \) on the output tape.

As before, define \( g(n) \) to be the \( n^{th} \) number written on the tape in this fashion. \( M_2 \) computes \( g(n) \) by simulating \( M_1 \) until \( n \) numbers have been written on the tape, then reporting that \( n^{th} \) number.
Definition: Turing Machine as set generator, version 1 (Sudkamp):

A Turing machine generates a set of numbers $S$ by writing a sequence of numbers on its tape, separated by a delimiter such as #.

The TM is required to write all and only members of $S$. A number may be written on the tape more than once.

If $S$ is infinite, then of course there will be no time at which the TM is done writing; however, for any $n \in S$, there will be a time at which $n$ has been written on the tape.

Definition: Turing Machine as set generator, version 2 (Taylor):

Alternatively, a Turing Machine generator for a set $S$ takes a number $n$ as input and produces as output the $n^{th}$ item in an enumeration of $S$.

---

Sudkamp is Thomas Sudkamp, *Languages and Machines*, Addison-Wesley, 1996.
Check that a Sudkamp-type generator and a Taylor-type generator are equivalent:

Sudkamp: Write elements of set S on a tape
Taylor: Given input n, write the n\textsuperscript{th} element of an enumeration of S on an output tape.

How would you build a Taylor-type generator from a Sudkamp-type generator?

How would you build a Sudkamp-type generator from a Taylor-type generator?
First Turing-machine formulation of recursively enumerable sets:

A set S is recursively enumerable iff it can be generated by a Turing machine.

Why is this equivalent to the earlier definition — S is the image of a total recursive function g?

A generator (in Taylor’s sense) computes a function g whose image set is S.
Second Turing-Machine formulation of recursively enumerable sets:

Set S is recursively enumerable iff S is accepted by some Turing machine.

Why is this an equivalent definition?
"Recursively enumerable" means that there is a TM that takes a number n as input and

- Halts with a number if n is in set S.
- Doesn’t halt if n is not in set S.

Justification, part 1: Suppose that S is recursively enumerable.
Then (by the earlier theorem) there exists a partial recursive function f such that S = domain(f).
Since f is partial recursive, we can construct a TM $M_f$ that computes f.
Construct an acceptor M for S:
Given a number n, run $M_f$ to compute $f(n)$.
If $f(n)$ is defined, $M_f$ will halt and output a number.
When this happens, M should output a 1.
If $f(n)$ is undefined, M will wait forever for $M_f$ to finish.

$M$ accepts S.

Justification, part 2: Suppose that S is accepted by a TM M.
Design a TM $M_f$ that simulates M:
M takes input n.
If M halts and writes 1, $M_f$ halts and writes 1.
If M halts in any other tape configuration, $M_f$ goes into an infinite loop.
If M computes forever, so does $M_f$.

Let $f$ be the function computed by $M_f$.

$S = \text{domain}(f)$. 
Turing-machine formulation of recursive sets:
Set $S$ is recursive iff $S$ is recognized by a Turing machine.

Why is this an equivalent definition?
A set $S$ is recursive iff $S$ has a (total) recursive characteristic function.
The Turing Machine to recognize $S$ is exactly the Turing Machine that computes $S$’s characteristic function.
Theorem: Every recursive set is recursively enumerable.

Think: Why is this true?
Think about recognizers and acceptors.
This one is easy.
**Theorem:**

Every recursive set is recursively enumerable.

**Proof:**

If $S$ is recognized by a TM $M$, then it is accepted by $M$. 
Theorem:

A set $S$ is recursive iff both $S$ and $S^C$ are recursively enumerable.

Think: Why is this true?

If you have an acceptor for $S$
and an acceptor $S^C$,
how would you make a recognizer for $S$?
Theorem:

A set $S$ is recursive iff both $S$ and $S^C$ are recursively enumerable.

Buggy Proof:

If $S$ and $S^C$ are both r.e., then there are TMs $M_1$ which accepts $S$ and $M_2$ that accepts $S^C$.

Build a third TM $M$:

$M$ simulates $M_1$ on input $n$, then simulates $M_2$ on input $n$.
If $M_1$ accepts $n$, then accept $n$.
Else if $M_2$ accepts $n$, then reject $n$.

One of the two machines has to accept $n$, so we’re guaranteed to get an answer.

Why is this buggy?
Theorem:

A set $S$ is recursive iff both $S$ and $S^C$ are recursively enumerable.

Proof:

If $S$ and $S^C$ are both r.e., then there are TMs $M_1$ which accepts $S$ and $M_2$ that accepts $S^C$.

Build a third TM $M$: Given input $n$, $M$ simulates $M_1$ on one work tape and "simultaneously" simulates $M_2$ on a second work tape.

In more detail, $M$ simulates $M_1$ for one step, then $M_2$ for one step, then $M_1$ for one more step, then $M_2$ for one more step, etc.

Eventually, either $M_1$ will halt or $M_2$ will halt.

(Why? — $n$ is in either $S$ or $S^C$. The TM for whichever set $n$ is in is required to halt and answer 1 eventually. We might additionally get lucky and get a "no" answer from the other machine, though we can’t count on that.)

When one machine halts, we halt and report 1 or 0, as appropriate.
**Theorem:**

Any finite set $S$ of natural numbers is recursively enumerable.

**Think: Why is this true?**

List the elements of set $S$ in some order: $n_0, n_1, \ldots, n_k$.

Can you write an algorithm to accept $S$?

Can you write an algorithm to generate $S$?
Theorem:

Any finite set $S$ of natural numbers is recursively enumerable.

Proof (algorithm to accept $S$):

Let $S = \{n_0, n_1, \ldots, n_k\}$.

To check whether a number $x$ is in $S$, check:

Is $x = n_0$?
Is $x = n_1$?
... 
Is $x = n_k$?

Stop if one of the checks is true, and answer "yes".

Proof (algorithm to generate $S$):

Print $n_0$
Print $n_1$
...
Print $n_k$

(Note that this executes in finite time.)

Proof (using partial recursive functions):

List the elements of set $S$ in some order: $n_0, n_1, \ldots, n_k$.

Define $f(n) = \begin{cases} 
  n_0 & \text{if } n = 0 \\
  n_1 & \text{if } n = 1 \\
  \ldots & \\
  n_k & \text{if } n \geq k 
\end{cases}$

$f$ is partial recursive, since it is defined by cases using constants, =, and $\geq$.

$f$ is total.

$S = \text{image}(f)$; thus $S$ is recursively enumerable.
Theorem:

Any finite set $S$ of natural numbers is recursive.

Proof (algorithm to recognize $S$):

Let $S = \{n_0, n_1, \ldots, n_k\}$.

To check whether a number $x$ is in $S$, check:

- Is $x = n_0$?
- Is $x = n_1$?
- ...
- Is $x = n_k$?

Stop if one of the checks is true, and answer "yes".
If the loop reaches the end and none of the checks is true, answer "no".

Proof (using characteristic functions):

List the elements of set $S$ in some order: $n_0, n_1, \ldots, n_k$.

$$\chi_S = (n = n_0 \lor n = n_1 \lor \ldots \lor n = n_k).$$

This function is partial recursive (and total).
**Summary: Recursive and Recursively Enumerable Sets**

**Definition: Recursively enumerable set**

A set S is recursively enumerable (r.e.) iff either S is the empty set or S = image(f) for some total recursive function f.

*Function f is a generator for S.*

**Definition: Recursive set**

A set S is recursive iff S has a partial recursive characteristic function.

*That is, the characteristic function is computable for all inputs.*

**Theorem:** S is recursively enumerable iff S = domain(f) for some unary partial recursive function f.

*Function f acts like an acceptor for S.*
Definition: Turing Machine as set generator, version 1 (Sudkamp):

A Turing machine generates a set $S$ by writing a sequence of numbers on its tape, separated by a delimiter such as #.

The TM is required to write all and only members of $S$. A number may be written on the tape more than once.

If $S$ is infinite, then of course there will be no time at which the TM is done writing; however, for any $n \in S$, there will be a time at which $n$ has been written on the tape.

Definition: Turing Machine as set generator, version 2 (Taylor):

Alternatively, a Turing Machine generator for a set $S$ takes a number $n$ as input and produces as output the $n^{th}$ item in an enumeration of $S$.

First Turing-machine formulation of recursively enumerable sets:

A set is recursively enumerable iff it can be generated by a Turing machine.

Second Turing-Machine formulation of recursively enumerable sets:

Set $S$ is recursively enumerable iff $S$ is accepted by some Turing machine.

That is, $S$ has a generator exactly if $S$ has an acceptor.

Turing-machine formulation of recursive sets:

Set $S$ is recursive iff $S$ is recognized by a Turing machine.

The Turing Machine computes the characteristic function of $S$. 

Copyright © 2014 Nancy Lynn Tinkham
Theorem:

Every recursive set is recursively enumerable.

Theorem:

A set $S$ is recursive iff both $S$ and $S^C$ are recursively enumerable.

Theorem:

Any finite set of natural numbers is recursively enumerable.

Theorem:

Any finite set of natural numbers is recursive.
Intuition of recursive and r.e. sets

<table>
<thead>
<tr>
<th>Generator</th>
<th>Accept/Recognize</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recursive</strong></td>
<td>Generate algorithmically in increasing order</td>
</tr>
<tr>
<td><strong>Recursively enumerable</strong></td>
<td>Generate algorithmically (perhaps w/ repetitions, in no particular order)</td>
</tr>
</tbody>
</table>

Properties of complements

If $S$ and $S^C$ are both R.E., then $S$ is **recursive**.

**Recursive**: If $S$ is recursive, then $S^C$ is recursive.

**R.E.**: If $S$ is R.E. but not recursive, then $S^C$ is not R.E.