Recursively Enumerable Sets

Definition: The set \( B \subseteq \mathbb{N} \) is called recursively enumerable if there is a partially computable function \( g(x) \) such that
\[
B = \{ x \in \mathbb{N} \mid g(x) \downarrow \}.
\]
The term recursively enumerable is abbreviated r.e.

A set is r.e. just when it is the domain of a partially computable function.

If \( \varphi \) is a program that computes the function \( g \) (see above), then \( B \) is simply the set of all inputs to \( \varphi \) for which \( \varphi \) eventually halts.

Recursively Enumerable Sets

Theorem 4.3: If \( B \) is a recursive set, then \( B \) is r.e.
Proof: Consider the following program \( \varphi \):
\[
[A] \quad \text{IF } \neg (X \in B) \text{ GOTO A}
\]
Since \( B \) is recursive, the predicate \( X \in B \) is computable and \( \varphi \) can be expanded to a program of \( \mathcal{L} \).

Let \( \varphi \) compute the function \( h(x) \). Then, clearly,
\[
B = \{ x \in \mathbb{N} \mid h(x) \downarrow \}.
\]

Recursively Enumerable Sets

Theorem 4.4: The set \( B \) is recursive if and only if \( B \) and \( \neg B \) are both r.e.
Proof: If \( B \) is recursive, then by Theorem 4.1 so is \( \neg B \), and hence by Theorem 4.3, they are both r.e.

Conversely, if \( B \) and \( \neg B \) are both r.e., we may write
\[
B = \{ x \in \mathbb{N} \mid g(x) \downarrow \},
\]
\[
\neg B = \{ x \in \mathbb{N} \mid h(x) \downarrow \},
\]
where \( g \) and \( h \) are both partially computable.

Recursively Enumerable Sets

If we think of \( \varphi \) as providing an algorithm for testing for membership in \( B \), we see that
- if a number belongs to \( B \), the algorithm will provide a positive answer,
- if a number does not belong to \( B \), the algorithm will never terminate.

Such algorithms are called semi-decision procedures.
They can be considered an “approximation” to solving the problem of testing membership in \( B \).

Recursively Enumerable Sets

Now let \( g \) be computed by program \( \varphi \) and \( h \) be computed by program \( \psi \), and let \( p = \#(\varphi) \) and \( q = \#(\psi) \).

Then the following program computes \( B \):
\[
[A] \quad \text{IF STP}(p)(X, p, T) \text{ GOTO C}
\]
\[
\text{IF STP}(q)(X, q, T) \text{ GOTO E}
\]
\[
T \leftarrow T+1
\]
\[
\text{GOTO A}
\]
\[
[C] \quad Y \leftarrow 1
\]

Recursively Enumerable Sets

Definition: We write:
\[ W_n = \{ x \in \mathbb{N} \mid \Phi(x, n) \downarrow \} \]

Theorem 4.6 (Enumeration Theorem):
A set \( B \) is r.e. if and only if there is an \( n \) for which
\[ B = W_n. \]
This is an immediate consequence of the definition of \( \Phi(x, n) \).
The theorem gets its name from the fact that the sequence \( W_0, W_1, W_2, \ldots \) is an enumeration of all r.e. sets.

Recursively Enumerable Sets

We further define:
\[ K = \{ n \in \mathbb{N} \mid n \in W_n \} \]
Then
\[ n \in W_n \iff \Phi(n, n) \downarrow \iff \text{HALT}(n, n). \]
\( K \) is the set of all numbers \( n \) such that program number \( n \) eventually halts on input \( n \).

Theorem 4.7:
\( K \) is r.e. but not recursive.

Proof:
Since \( K = \{ n \in \mathbb{N} \mid \Phi(n, n) \downarrow \} \), and by the universality theorem (Theorem 3.1), \( \Phi(n, n) \) is partially computable, \( K \) is obviously r.e.
If \( K \) were recursive, then \( \neg K \) would be r.e.
If that were the case, then by the enumeration theorem there would have to be some number \( i \) so that \( \neg K = W_i \).
But then:
\[ i \in K \]
\[ \iff i \in W_i \]
\[ \iff i \in \neg K. \]
Contradiction!

Recursively Enumerable Sets

Theorem 4.8:
Let \( B \) be an r.e. set. Then there is a primitive recursive predicate \( R(x, t) \) such that \( B = \{ x \in \mathbb{N} \mid (\exists t) R(x, t) \} \).
Proof:
Let \( B = W_n \). Then
\[ B = \{ x \in \mathbb{N} \mid (\exists t) \text{STP}^{(1)}(x, n, t) \}, \]
and \( \text{STP}^{(1)} \) is primitive recursive by Theorem 3.2.

Recursively Enumerable Sets

Theorem 4.9:
Let \( S \) be a nonempty r.e. set. Then there is a primitive recursive function \( f(u) \) such that \( S = \{ f(n) \mid n \in \mathbb{N} \} \)
\[ = \{ f(0), f(1), f(2), \ldots \}. \] In other words, \( S \) is the range of \( f \).

Theorem 4.10:
Let \( f(x) \) be a partially computable function, and let
\[ S = \{ f(x) \mid f(x) \downarrow \} \] (so \( S \) is the range of \( f \)).
Then \( S \) is r.e.
If we combine Theorems 4.9 and 4.10, we get:

Recursively Enumerable Sets

Theorem 4.11:
Consider a set \( S \neq \emptyset \). The following statements are all equivalent:
1. \( S \) is r.e.;
2. \( S \) is the range of a primitive recursive function;
3. \( S \) is the range of a recursive function;
4. \( S \) is the range of a partial recursive function.
This theorem motivates the term \textit{recursively enumerable}.
A nonempty r.e. set is enumerated by a recursive function.
The Parameter Theorem

The parameter theorem is also called iteration theorem and s-m-n theorem.

It is important to the theory of computation as it relates the functions $\Phi^{(n)}(x_1, \ldots, x_n, y)$ for different values of $n$.

Theorem 5.1 (Parameter Theorem):
For each $n$, $m > 0$ there is a primitive recursive function $S_m^n(u_1, \ldots, u_n, y)$ such that

$$\Phi^{(m + n)}(x_1, \ldots, x_m, u_1, \ldots, u_n, y) = \Phi^{(m)}(x_1, \ldots, x_m, S_m^n(u_1, \ldots, u_n, y)).$$

Suppose that the values for $u_1, \ldots, u_n$, and $y$ are fixed. Then the left side of the equation is a partially computable function of the $m$ arguments $x_1, \ldots, x_m$.

Let the number of the program that computes this function be $q$. Then we have:

$$\Phi^{(m + n)}(x_1, \ldots, x_m, u_1, \ldots, u_n, y) = \Phi^{(m)}(x_1, \ldots, x_m, q).$$

The parameter theorem tells us that there exists such a $q$ that can be obtained from $u_1, \ldots, u_n$, and $y$ by a primitive recursive function.

Let us take a look at the case $n = 1$:

$$\Phi^{(m + 1)}(x_1, \ldots, x_m, u, y) = \Phi^{(m)}(x_1, \ldots, x_m, S_m^1(u, y)).$$

Here, $S_m^1(u, y)$ is the number of a program that receives inputs $x_1, \ldots, x_m$ and computes the same value as program number $y$ does on inputs $x_1, \ldots, x_m$, and $u$.

We can easily obtain $S_m^1(u, y)$ by writing the instruction $X_{m+1} \leftarrow u$ and then appending the program with number $y$.

This works similarly for any given $n$, which can be proven by mathematical induction (see page 86 in the textbook).