Recursively Enumerable Sets

As long as the Gödel numbering functions \([x_1, \ldots, x_n]\) and \((x)\) are available, we only need to consider subsets of \(N\) instead of subsets of \(N^n\).

Then we have:

**Theorem 4.2:** Let \(\mathcal{C}\) be a PRC class, and let \(B\) be a subset of \(N^n\), \(m \geq 1\).

Then \(B\) belongs to \(\mathcal{C}\) if and only if \(B' = \{[x_1, \ldots, x_m] \in N^m \mid (x_1, \ldots, x_m) \in B\}\) belongs to \(\mathcal{C}\).

Proof: If \(P_B(x_1, \ldots, x_m)\) is the characteristic function of \(B\), then \(P_B(x) \iff P_B((x)_1, \ldots, (x)_m) \& Lt(x) \leq m \& x > 0\) is the characteristic function of \(B'\), and \(P_B\) clearly belongs to \(\mathcal{C}\) if \(P_B\) belongs to \(\mathcal{C}\).

On the other hand, if \(P_B'(x)\) is the characteristic function of \(B'\), then \(P_B(x_1, \ldots, x_m) \iff P_B'([x_1, \ldots, x_m])\) is the characteristic function of \(B\), and \(P_B\) clearly belongs to \(\mathcal{C}\) if \(P_B'\) belongs to \(\mathcal{C}\).

For example, \(\{[x, y] \in N \mid \text{HALT}(x, y)\}\) is not a computable set.

Recursively Enumerable Sets

**Definition:** The set \(B \subseteq N\) is called **recursively enumerable** if there is a partially computable function \(g(x)\) such that \(B = \{x \in N \mid g(x) \downarrow\}\).

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 \(\varphi\):

\[
\text{[A]} \quad \text{IF \(\neg (x \in B)\) 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, \(B = \{x \in N \mid h(x) \downarrow\}\).

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

If \(B\) and \(\neg B\) are both r.e., then we can devise two algorithms:

- one algorithm that terminates if a given input is in \(B\), and
- another algorithm that terminates if a given input is not in \(B\).

Can we find a way to **combine** these two algorithms to obtain a single algorithm that **always terminates** and tells us whether a given input is in \(B\)?

The trick is to let the two algorithms run for more and more steps until one of them terminates (**dovetailing**).
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 $B = \{x \in \mathbb{N} \mid g(x) \downarrow\}$ and $\neg B = \{x \in \mathbb{N} \mid h(x) \downarrow\}$, where $g$ and $h$ are both partially computable.

Now let $g$ be computed by program $P$ and $h$ be computed by program $Q$, and let $p = \#(P)$ and $q = \#(Q)$.

Then the following program computes $B$:

```plaintext
[A] IF STP(1)(X, p, T) GOTO C
    IF STP(1)(X, q, T) GOTO E
    T \leftarrow T+1
    GOTO A
[C] Y \leftarrow 1
```