Universality

Theorem 3.4: A function is partially computable if and only if it can be obtained from the initial functions by a finite number of applications of composition, recursion, and minimalization.

**Proof:** It follows from Theorems 1.1, 2.1, 2.2, 3.1, and 7.2 in Chapter 3 that every function that can be so obtained is partially computable.

Universality

Now let us consider the “opposite direction” of the Normal Form Theorem (Theorem 3.3):

We can use the normal form theorem to write any given partially computable function in the form

\[ l(\min_y R(x_1, \ldots, x_n, y)) \]

where R is a primitive recursive predicate and therefore is obtained from the initial functions by a finite number of applications of composition and recursion.

Finally, our given function is obtained from R by one use of minimalization and then by composition with the primitive recursive function l.

Recursively Enumerable Sets

From previous classes, such as CS 320, you may remember the correspondence between predicates and sets.

We now want to use the set notation in our discussion of solvable and unsolvable problems.

For example, the predicate \( \text{HALT}(x, y) \) is the characteristic function of the set \( \{(x, y) \in \mathbb{N}^2 | \text{HALT}(x, y)\} \).

We say that a set \( B \subseteq \mathbb{N}^m \) belongs to some class of functions if and only if the characteristic function \( P(x_1, \ldots, x_n) \) of \( B \) belongs to the class in question.

Thus, saying that a set \( B \) is computable or recursive is the same as saying that \( P(x_1, \ldots, x_n) \) is a computable function.

Likewise, \( B \) is a primitive recursive set if \( P(x_1, \ldots, x_n) \) is a primitive recursive predicate.

It follows that:

**Theorem 4.1:** Let the sets \( B, C \) belong to some PRC class \( \mathcal{C} \). Then so do the sets \( B \cup C, B \cap C, \neg B \).

**Proof:** This is an immediate consequence of Theorem 5.1, Chapter 3.

Recursively Enumerable Sets

**Example:** Let us consider the case \( B \cup C \).

There must be predicates \( P_B \) and \( P_C \) such that:

\[ B = \{ x \in \mathbb{N} | P_B(x) \} \]
\[ C = \{ x \in \mathbb{N} | P_C(x) \} \]

Then:

\[ B \cup C = \{ x \in \mathbb{N} | P_B(x) \lor P_C(x) \} \]

Since \( \lor \) is a primitive-recursive function, the predicate \( P_B(x) \lor P_C(x) \) is also in class \( \mathcal{C} \). Then the equation above shows that \( B \cup C \) is in \( \mathcal{C} \) as well.
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 \(\mathbb{N}\) instead of subsets of \(\mathbb{N}^n\).

Then we have:

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

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

**Proof:** If \(P_B(x_1, \ldots, x_m)\) is the characteristic function of \(B\), then
\[
P_B(x) \iff P_B((x_1), \ldots, (x_m)) \land Lt(x) \leq m \land x > 0
\]
is the characteristic function of \(B'\), and \(P_B'\) clearly belongs to \(C\) if \(P_B\) belongs to \(C\).

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 \(C\) if \(P_B'\) belongs to \(C\).

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

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
\[
B = \{x \in \mathbb{N} : g(x) \downarrow\}.
\]
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

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 \(B\).