Universality

Consider the following predicates:

\[ \text{STP}^{(n)}(x_1, \ldots, x_n, y, t) \]
\[ \iff \text{Program number } y \text{ halts after } t \text{ or fewer steps on inputs } x_1, \ldots, x_n \]
\[ \iff \text{There is a computation of program } y \text{ of length } \leq t + 1, \text{ beginning with inputs } x_1, \ldots, x_n \]

These predicates are computable, which we can easily prove:
We can simply add a counter to our universal programs to determine when we have simulated \( t \) steps.

Universality

We can prove an even stronger theorem:

**Theorem 3.2 (Step-Counter Theorem):**
For each \( n > 0 \), the predicate \( \text{STP}^{(n)}(x_1, \ldots, x_n, y, t) \) is primitive recursive.

**Proof:**
We will provide numerical descriptions of the notions of snapshot and successor snapshot.
This will show that the necessary functions are primitive recursive.
See pages 74 and 75 in the textbook for the proof.

Universality

Now that we know the Step-Counter Theorem, we are ready for yet another theorem.
Proving this theorem will be similar to the last one.

**Theorem 3.3 (Normal Form Theorem):**
Let \( f(x_1, \ldots, x_n) \) be a partially computable function.
Then there is a primitive recursive predicate \( R(x_1, \ldots, x_n, y) \) such that
\[ f(x_1, \ldots, x_n) = l(\min_{z} R(x_1, \ldots, x_n, z)) \]

**Proof:**
Let \( y_0 \) be the number of a program that computes \( f(x_1, \ldots, x_n) \).
Then consider the equation
\[ f(x_1, \ldots, x_n) = l(\min_{z} R(x_1, \ldots, x_n, z)) \]
where \( R(x_1, \ldots, x_n, z) \) is the predicate
\[ \text{STP}^{(n)}(x_1, \ldots, x_n, y_0, r(z)) \]
& \( (r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, r(z))))_1 = l(z) \)
If the right side of the first equation is defined, then there exists a number \( z \) such that
\[ \text{STP}^{(n)}(x_1, \ldots, x_n, y_0, r(z)) \]
and
\[ (r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, r(z))))_1 = l(z) \]

Note:
As introduced in the proof on pages 74 and 75, \( \text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, i) \)
stands for the snapshot of program number \( y_0 \) after executing \( i \) instructions when run with inputs \( x_1, \ldots, x_n \).
The snapshot is represented as a pairing \( <i, s> \),
where \( i \) is the number of the next instruction to be executed and \( s \) is the Gödel number of the current state.

Universality

Consequently, \( \text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, r(z)) \)
is the snapshot after executing \( r(z) \) instructions of program number \( y_0 \).
\( r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, r(z))) \)
is the Gödel number of the state taken from that snapshot.
\( (r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, y_0, r(z))))_1 \)
is the first number in the sequence represented by that Gödel number.
Because \( Y \) is the first variable in the list, this expression equals the value of variable \( Y \) in the program number \( y_0 \) after executing \( r(z) \) instructions.
Universality

For any such $z$,
- the computation by the program with number $y_0$ has reached a terminal snapshot in $r(z)$ or fewer steps,
- $l(z)$ is the value of the output variable $Y$, that is, $l(z) = f(x_1, \ldots, x_n)$.

If the right side of the equation is undefined, it must be true that $\text{STP}^{(n)}(x_1, \ldots, x_n, y_0, t)$ is false for all values of $t$, that is, $f(x_1, \ldots, x_n)$ is undefined.

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 $l$.

When $\min_y R(x_1, \ldots, x_n, y)$ is a total function (that is, when for each $x_1, \ldots, x_n$ there is at least one $y$ for which $R(x_1, \ldots, x_n, y)$ is true), we say that we are applying the operation of proper minimalization to $R$.

Now, if $l(\min_y R(x_1, \ldots, x_n, y))$ is total, then $\min_y R(x_1, \ldots, x_n, y)$ must be total.

This gives us the following theorem:

Theorem 3.5: A function is computable if and only if it can be obtained from the initial functions by a finite number of applications of composition, recursion, and proper minimalization.

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 $\{(x, y) \in \mathbb{N}^2 \mid \text{HALT}(x, y)\}$.

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