The Recursion Theorem

One of the problems in the previous approach to undecidability (i.e., proving that certain sets are not recursive or recursively enumerable) is that the major tool - Rice’s Theorem - requires the sets to be index-sets.

In order to extend both the known family of undecidable sets, and to understand what recursion means in terms of Turing Machines, we take a slightly different approach. The main result we will see is due to Kleene, originally in the context of the \( \lambda \)-calculus. Students who have taken the Design of Programming Languages course have already met it under the guise of fixed points of functions and the \( \text{Y} \)-combinator.

In the implementation of recursion we deal with a function that, besides receiving parameter values, receives (and can execute) its own code as an input.

The Recursion Theorem

- The **enumeration theorem** gives: for any partial recursive \( f : (\{0, 1\}^*)^m \rightarrow \{0, 1\}^* \) there exists a \( z \in \{0, 1\}^* \) such that \( f(x_1, \ldots, x_m) = \phi^z_{j+1}(x_1, \ldots, x_m) \).
- The **s-m-n Theorem** gives: for each pair of integers \( m, n > 0 \), there is a primitive recursive function \( x_n : (\{0, 1\}^*)^m \rightarrow \{0, 1\}^* \) such that
  \[
  \phi^z_{j+1}(x_1, \ldots, x_m, y) = \phi^z_{s(m+1,n+1)}(x_1, \ldots, x_m, y).
  \]
- Applying the two theorems sequentially gives:
  \[
  f(x_1, \ldots, x_m, y) = \phi^z_{j+1}(x_1, \ldots, x_m, y) = \phi^z_{s(m+1,n+1)}(x_1, \ldots, x_m).\]

**Note:** \( z \) depends only on \( f \) and not on \( (x_1, \ldots, x_m) \). Multiple \( z \)'s can correspond to TMs computing \( f \).

The Recursion Theorem

- In the discussion that follows, we put together a few results from previous sections - primarily the **Enumeration Theorem** and the s-m-n **Theorem** - and show how they can be “glued together” to provide us with the desired conclusion and existence proof.
- We also get a second proof of Rice’s Theorem, more or less for free.
- The **Busy Beaver Theorem** produces a non-recursive set - using the Recursion Theorem - without the use of index-set techniques: a new approach to such proofs is at least hinted at.

The Recursion Theorem

One of the questions we could ask is: is there, for at least one of the \( z \)'s computing \( f \), a \( y \) s.t. \( y = s_j(z) \)? If that were the case, we would have
\[
\phi^z_j(x_1, \ldots, x_m, y) = \phi^y_j(x_1, \ldots, x_m, y) = \phi^y_j(x_1, \ldots, x_m).
\]

We would also have a machine that - in some sense - can access its own code as an input. If we chose \( y = z \), we would have
\[
\phi^z_j(x_1, \ldots, x_m, z) = \phi^z_j(x_1, \ldots, x_m, z) = \phi^z_j(x_1, \ldots, x_m),
\]
and corresponding machines \( M = M_j \) and \( M_{s_j(z)} \). Starting from \( f \), the machine simulates \( f \) on its code \( z \). Unfortunately, the final machine’s code is not \( z \), but \( s_j(z) \); the last input \( z \) is no longer the machine code we want. How do we ensure that the code of the final machine is the correct last parameter to \( f \)? Recall that the code \( z \) depends on \( f \) and not on the values of its parameters. So we might be able to use the information we have to pick the right value of that last parameter.
The Recursion Theorem

Theorem 5.30 (Rice’s Theorem - again). All non-trivial function-index sets are non-recursive.

Note: recall that set-index sets are function-index sets, so we could leave out the modifier.

Proof. Assume \( A \) is a non-trivial recursive function-index set. Let \( n_1 \in A \) and \( n_2 \not\in A \). Define

\[
 f(x, y) = \begin{cases} 
 \phi_{n_1}(x) & \text{if } y \in A \\
 \phi_{n_2}(x) & \text{if } y \not\in A. 
\end{cases}
\]

Since \( A \) is recursive, \( f \) is partial recursive. By the recursion theorem 3\( \exists \)\( r \) s.t. \( \phi_r(x) = f(x, e) \).

Case 1. \( e \in A \). Then \( \phi_r(x) = f(x, e) = \phi_{n_1}(x) \). Since \( A \) is a function-index set, and since we chose \( n_1 \not\in A \), we must have \( e \not\in A \). Contradiction.

Case 2. \( e \not\in A \). Then \( \phi_r(x) = f(x, e) = \phi_{n_2}(x) \). Since \( A \) is a function-index set, and since we chose \( n_2 \in A \), we must have \( e \in A \). Contradiction. Thus \( A \) cannot be recursive.

The Recursion Theorem

Theorem 5.36 (Recursion Theorem). For any partial recursive \( f: \{0, 1\}^{*} \to \{0, 1\} \) there exists a constant \( c \geq 0 \), such that

\[
\phi_c(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, e).
\]

Proof. Let \( f(x_1, \ldots, x_k) \) be given, let \( s_j(w_i, x_k) \) be the primitive recursive function of the \( s-m-n \) theorem. Consider a function

\[
f(x_1, \ldots, x_k, t_k) \equiv \phi_{s_j(w_i, x_k)}(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, t_k) \in \{0, 1\}.
\]

Let \( e \) be the code of the interpreting TM for \( f \), as provided by the enumeration theorem. Then

\[
\phi_{s_j(w_i, x_k)}(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, t_k) \in \{0, 1\}.
\]

Applying the \( s-m-n \) theorem to the leftmost expression, we have:

\[
\phi_{s_j(w_i, x_k)}(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, t_k) \in \{0, 1\}.
\]

The choice \( s_{j+1} \equiv e \) provides the correct value \( y = s_j(x_1, e_1) \) for the last parameter of \( f \).

Note: see Fixed Points in DPL: Y-combinator, Tarski’s Thm.
The Recursion Theorem

By the recursion theorem, \( \exists n \) s.t. \( \phi_n(m) = g(m, n) \). From this, now fixed, \( n \), we have \( \text{domain}(\phi_n) = W_n = \{ m \mid m \leq f(n) \text{ and } m \in W_m \} \), where the last membership follows from \( \phi_n(m) \downarrow \). This implies that \( W_n \) is a finite set and thus recursive. Let \( A = \bar{W}_n = \{ m \mid m > f(n) \text{ or } m \in \bar{W}_m \} \), which is also recursive and therefore equal to \( W_n \) for some \( k \). Observe that, for each \( m \leq f(n) \), \( m \in W_m \iff m \in W_n \), which implies that \( W_n \neq \bar{W}_n \). We conclude that, if \( \bar{W}_n = W_m \), then \( m > f(n) \).

**The Recursion Theorem**

**Theorem 5.40** (The busy beaver theorem). If \( f(x) = \min \{ n \mid \phi_n(x) = x \} \), then \( f \) is not a recursive function.

**Comment.** \( f(x) \) is the smallest amount of information necessary for a Turing Machine to output \( x \). Furthermore, if \( U \) denotes the Universal Turing Machine, by definition \( U(f(x), \epsilon) = x \).

**Proof.** Assume \( f \) recursive. Define \( g(y, m) = (\min x)(f(x) > m) \).

**Claim:** \( f \) is one-to-one, i.e., \( f(x) = f(y) \Rightarrow x = y \).

**Pf. of Claim.** If not, the same code \( n \) must provide \( \phi_n(\epsilon) = x \) and \( \phi_n(\epsilon) = y \).

But then \( \phi_n \) is not a function. Since \( f: \mathbb{N} \rightarrow \mathbb{N} \) is one-to-one, it must be unbounded. Thus \( g \) is a recursive function (a \( (\min x) \) always exists). By the recursion theorem, there exists a constant \( e \) s.t.

\[ \phi_e(\epsilon) = g(\epsilon, e) = (\min x)(f(x) > e) \]

Since \( g \) is recursive, \( \phi_e \) is also recursive. Suppose \( \phi_e(x) = x_0 \). By the definition of \( g \), we have \( f(x_0) > e \). By the definition of \( f \), we have \( f(x_0) \leq e \). Contradiction.