1. The Recursion Theorem

2. The Fixed Point Theorem
The Recursion Theorem: Let $g(z, x_1, \ldots, x_m)$ be a partially computable function of $m + 1$ variables. There exists a number $e$ such that

$$g(e, x_1, \ldots, x_m) = \Phi_e(x_1, \ldots, x_m).$$
Discussion: Let \( e = \#(\mathcal{P}) \) so that

\[
\psi^{(m)}_{\mathcal{P}}(x_1, \ldots, x_m) = \Phi^{(m)}_e(x_1, \ldots, x_m).
\]

This means that \( \mathcal{P} \) is a program that gets access to its own number \( e \) and computes \( g(e, x_1, \ldots, x_m) \). This means that \( \mathcal{P} \) must somehow compute \( e \) because \( e \) does not appear among the arguments of \( \mathcal{P} \).
Intuitively, suppose that you write a JAVA program that has at the top the statement “CONST = 0”. Call this program $\mathcal{P}_0$. If I come along and replace that 0 with a 17, we will call that program $\mathcal{P}_{17}$. More generally, if that 0 is replaced by $b$, it is program $\mathcal{P}_b$. Our theorem says is that there is some number $a$ such that if you use $a$ for the parameter then the program produced is actually program $\mathcal{P}_a$. The resulting program can be said to “know its own index”.
Proof.

Consider the partially computable function

\[ g(S^1_m(v, v), x_1, \ldots, x_m), \]

where \( S^1_m(v, v) \) is the function that occurs in the smn Theorem. There is a number \( z_0 \) such that

\[
g(S^1_m(v, v), x_1, \ldots, x_m) = \Phi^{(m+1)}(x_1, \ldots, x_m, v, z_0) = \Phi^{(m)}(x_1, \ldots, x_m, S^1_m(v, z_0))
\]

by the smn Theorem. Setting \( v = z_0 \) and \( e = S^1_m(z_0, z_0) \) we have

\[ g(e, x_1, \ldots, x_m) = \Phi^{(m)}(x_1, \ldots, x_m, e) = \Phi_e^{(m)}(x_1, \ldots, x_m). \]
One of the applications of the recursion theorem is to allow us to write definitions of functions that involve the program used to compute the function as a part of its definition.

**Corollary**

*There is a number* $e$ *such that for all* $x$ *we have*

$$\Phi_e(x) = e.$$

**Proof.**

Consider the computable function $g(z, x) = u_1^2(z, x) = z$. Applying the recursion theorem we obtain the existence of a number $e$ such that $\Phi_e(x) = g(e, x) = e.$
The program with number \( e \) consumes its input \( x \) and outputs a copy of itself. This program can be regarded as a “self-reproducing organism”.
Example

Let \( g(x, y) \) be a computable function. Define the partially computable function \( f \) as

\[
f(x, t) = \begin{cases} 
  k & \text{if } t = 0, \\
  g(t - 1, \Phi_x(t - 1)) & \text{otherwise}
\end{cases}
\]

By the Recursion Theorem there is a program numbered \( e \) such that

\[
\Phi_e(t) = f(e, t) = \begin{cases} 
  k & \text{if } t = 0, \\
  g(t - 1, \Phi_e(t - 1)) & \text{otherwise}
\end{cases}
\]
Example cont’d

Example

The function $\Phi_e$ is a total, and therefore, a computable function that satisfies the equations

$$\Phi_e(0) = k, \text{ and } \Phi_e(t + 1) = g(t, \Phi_e(t)),$$

that is, $\Phi_e$ is obtained from $g$ by primitive recursion. This is another justification of the correctness of definitions by primitive recursion.
The recursion theorem can be used to justify other recursive definition schemes.

**Example**

We claim that there are partially computable functions $f$ and $g$ that satisfy the equations:

\[
\begin{align*}
  f(0) &= 1, \\
  f(t + 1) &= g(2t) + 1, \\
  g(0) &= 3, \\
  g(2t + 2) &= f(t) + 2.
\end{align*}
\]
Start with the assumption that there exists a program $z$ such that

$$f(x) = \Phi_z(\langle 0, x \rangle) \quad \text{and} \quad g(x) = \Phi_z(\langle 1, x \rangle).$$

Note that $\langle 0, x \rangle = 2x$ and $\langle 1, x \rangle = 4x + 1$, so the sets

$$\{2x \mid x \in \mathbb{N}\} \quad \text{and} \quad \{4x + 1 \mid x \in \mathbb{N}\}$$

are disjoint: the first consists of even numbers, while the second consists of odd numbers.
The definitions of $f$ and $g$

\[
\begin{align*}
    f(0) &= 1, \\
    f(t + 1) &= g(2t) + 1, \\
    g(0) &= 3, \\
    g(2t + 2) &= f(t) + 2
\end{align*}
\]

can be rewritten as:

\[
\begin{align*}
    \Phi_z(\langle 0, 0 \rangle) &= 1 \\
    \Phi_z(\langle 0, t + 1 \rangle) &= \Phi_z(\langle 1, 2t \rangle) + 1 \\
    \Phi_z(\langle 1, 0 \rangle) &= 3, \\
    \Phi_z(\langle 1, 2t + 2 \rangle) &= \Phi_z(\langle 0, t \rangle) + 2.
\end{align*}
\]
Let $x$ be the argument of $\Phi_z$. The previous equalities can be written as:

- $\Phi_z(x) = 1$ if $x = \langle 0, 0 \rangle$.
- In this case $x = \langle 0, t + 1 \rangle$, so $t = r(x) \div 1$. This means that $\Phi_z(x) = \Phi_z(\langle 1, 2(r(x) \div 1) \rangle) + 1$. This holds when $(\exists y)_{\leq x}(x = \langle 0, y + 1 \rangle)$.
The next case is: \( \Phi_z(x) = 3 \) if \( x = \langle 1, 0 \rangle \).

Finally, if \( x = \langle 1, 2t + 2 \rangle \) (which happens when \( (\exists y)_{\leq x}(x = \langle 1, 2y + 2 \rangle) \)) we have \( 2t + 2 = r(x) \), hence \( t = \lfloor (r(x) \div 2)/2 \rfloor \). This allows us to write:

\[
\Phi_z(x) = \Phi_z(\langle 0, \lfloor (r(x) \div 2)/2 \rfloor \rangle) + 2
\]

if \( (\exists y)_{\leq x}(x = \langle 1, 2y + 2 \rangle) \) holds.
Define the function $F(z, x)$ as

$$F(z, x) = \begin{cases} 
1 & \text{if } x = \langle 0, 0 \rangle \\
\Phi_z(\langle 1, 2(r(x) \div 1) \rangle) + 1 & \text{if } (\exists y) \leq_x (x = \langle 0, y + 1 \rangle).
\end{cases}$$

$$= \begin{cases} 
3 & \text{if } x = \langle 1, 0 \rangle \\
\Phi_z(\langle 0, \lfloor (r(x) \div 2) / 2 \rfloor \rangle) + 2 & \text{if } (\exists y) \leq_x (x = \langle 1, 2y + 2 \rangle). 
\end{cases}$$
By the recursion theorem, there exists $e$ such that $\Phi_e(x) = F(e, x)$. This amounts to:

$$F(e, x) = \Phi_e(x)$$

$$= \begin{cases} 
1 & \text{if } x = \langle 0, 0 \rangle \\
\Phi_e(\langle 1, 2(r(x) \div 1) \rangle + 1 & \text{if } (\exists y) \leq x (x = \langle 0, y + 1 \rangle). \\
3 & \text{if } x = \langle 1, 0 \rangle \\
\Phi_e(\langle 0, \lceil (r(x) \div 2)/2 \rceil \rangle) + 2 & \text{if } (\exists y) \leq x (x = \langle 1, 2y + 2 \rangle). 
\end{cases}$$
If $f(x) = \Phi_e(\langle 0, x \rangle)$ and $g(x) = \Phi_e(\langle 1, x \rangle)$ the previous equalities imply:

$$f(0) = \Phi_e(0) = 1.$$ 

In the second case $x = 2(y + 1)$, $r(x) = y + 1$, so $r(x) \div 1 = y$. The equality in this case translates into:

$$\Phi_e(x) = \Phi_e(\langle 1, 2y \rangle) + 1$$
The Fixed Point Theorem:

Let $f(z)$ be a computable function. Then, there is a number $e$ such that $\Phi_{f(e)}(x) = \Phi_e(x)$ for all $x$.

Note that $e$ is not quite a fixed point in a mathematical sense. A number $t$ would be a fixed point of $t$ if we would have $f(t) = t$. This theorem says that for every computable function $f$ there is a number of a program $e$ that computes the same function as the program with number $f(e)$. 
Proof.

Let \( g(z, x) = \Phi_{f(z)}(x) \) be a partially computable function. By the recursion theorem, there is a number \( e \) such that

\[
\Phi_e(x) = g(e, x) = \Phi_{f(e)}(x).
\]
Example

Let $P(x)$ be a computable predicate, let $g(x)$ be a computable function and let $\text{while}(n) = \#(Q_n)$, where $Q_n$ is the program:

\[
\begin{align*}
X_2 & \leftarrow n \\
Y & \leftarrow X \\
[A] & \quad \text{IF} \sim P(Y) \text{ GOTO } E \\
Y & \leftarrow \Phi_{X_2}(g(Y))
\end{align*}
\]

The function $\text{while}$ is clearly computable (and, in fact is primitive recursive). By the fixed point theorem, there is a number $e$ such that $\Phi_e(x) = \Phi_{\text{while}(e)}(x)$. 

Example cont’d

Example

The construction of while(\(e\)) implies

\[
\phi_e(x) = \phi_{\text{while}(e)}(x) = \begin{cases} 
  x & \text{if } \sim P(x), \\
  \phi_e(g(x)) & \text{otherwise}.
\end{cases}
\]

Moreover,

\[
\phi_e(g(x)) = \phi_{\text{while}(e)}(g(x)) = \begin{cases} 
  g(x) & \text{if } \sim P(g(x)), \\
  \phi_e(g(g(x))) & \text{otherwise}.
\end{cases}
\]
Example cont’d

Thus, we have:

\[ \Phi_e(x) = \Phi_{\text{while}(e)}(x) = \begin{cases} 
  x & \text{if } \sim P(x) \\
  g(x) & \text{if } P(x) \& \sim P(g(x)), \\
  \Phi_e(g(g(x))) & \text{otherwise.}
\end{cases} \]
Example cont’d

Example

Continuing in this fashion we get

\[ \Phi_e(x) = \Phi_{\text{while}(e)}(x) = \begin{cases} 
  x & \text{if } \sim P(x) \\
  g(x) & \text{if } P(x) & \sim P(g(x)), \\
  g(g(x)) & \text{if } P(x) & P(g(x)) & \sim P(g(g(x))) \\
  \vdots & \& 
\end{cases} \]

In other words, the program whose number is \( e \) behaves like the program

\[
Y \leftarrow X \\
\text{while } P(Y) \text{ do} \\
\quad Y \leftarrow g(Y) \\
\text{end}
\]
Yet another proof of Rice’s Theorem

Suppose that $R_{\Gamma}$ were recursive. Let $P_{\Gamma}$ be the characteristic function of $R_{\Gamma}$, that is,

$$P_{\Gamma}(t) = \begin{cases} 1 & \text{if } t \in R_{\Gamma} \\ 0 & \text{otherwise.} \end{cases}$$

Define $h(t, x)$ as

$$h(t, x) = \begin{cases} g(x) & \text{if } t \in R_{\Gamma}, \\ f(x) & \text{otherwise}, \end{cases}$$

where, as before, $f \in \Gamma$ and $g \not\in \Gamma$. 
Then, since

\[ h(t, x) = g(x) \cdot P_\Gamma(t) + f(x) \cdot \alpha(P_\Gamma(t)) \]

it follows that \( h(t, x) \) is partially computable. By the recursion theorem, there is a number \( e \) such that

\[ \Phi_e(x) = h(e, x) = \begin{cases} 
  g(x) & \text{if } \Phi_e \in \Gamma \\
  f(x) & \text{otherwise.}
\end{cases} \]
Since $f \in \Gamma$ and $g \not\in \Gamma$ we have:

$$e \in R_{\Gamma} \implies \Phi_e(x) = g(x)$$

$$\implies \Phi_e \not\in \Gamma$$

$$\implies e \not\in R_{\Gamma}.$$
Likewise,

\[ e \not\in R_{\Gamma} \implies \Phi_e(x) = f(x) \]
\[ \implies \Phi_e \in \Gamma \]
\[ \implies e \in R_{\Gamma}, \]

so either case leads to a contradiction.