THEORY OF COMPUTATION
The Recursion Theorem - 15

Prof. Dan A. Simovici

UMB

In proving that HALT(x, y) is not computable we used a program $\mathcal{P}$:

$$[A] \quad \text{IF HALT}(X, X) \text{ GOTO } A$$

Assuming that $\#(\mathcal{P}) = y$, we obtained the contradictory statement:

$$\text{HALT}(y, y) \text{ if and only if } \sim \text{HALT}(y, y).$$

This involved considering the behavior of a program on its own description (something called self-referrence).

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 be the number of a program $\mathcal{P}$, $e = \#(\mathcal{P})$ so that

$$\psi_{\mathcal{P}}^{(m)}(x_1, \ldots, x_m) = \Phi_e^{(m)}(x_1, \ldots, x_m) = g(e, x_1, \ldots, x_m).$$

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 its own number e because e does not appear among the arguments of $\mathcal{P}$.

Proof.

Consider the partially computable function:

$$g(S_m^1(v, v), x_1, \ldots, x_m),$$

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

$$\begin{aligned} g(S_m^1(v, v), x_1, \ldots, x_m) &= \Phi^{(m+1)}(x_1, \ldots, x_m, v, z_0) \\ &= \Phi^{(m)}(x_1, \ldots, x_m, S_m^1(v, z_0)) \end{aligned}$$

by the smn Theorem. Setting $v = z_0$ and $e = S_m^1(z_0, z_0)$ we have

$$g(e, x_1, \ldots, x_m) = \Phi^{(m)}(x_1, \ldots, x_m, e) = \Phi_e^{(m)}(x_1, \ldots, x_m).$$

An alternative, self-referential proof that HALT(x, y) is not computable:

If HALT(x, y) were computable, then the function

$$f(x, y) = \begin{cases} \uparrow & \text{if HALT}(y, x) \\ 0 & \text{otherwise} \end{cases}$$

would be partially computable. Then, the recursion theorem would imply the existence of a number e such that

$$\Phi_e(y) = f(e, y) = \begin{cases} \uparrow & \text{if HALT}(y, e) \\ 0 & \text{otherwise.} \end{cases}$$

The last equality means that

$$\sim \text{HALT}(y, e) \Leftrightarrow \text{HALT}(y, e).$$

The self-reference occurs when Φ_e computes e, tests HALT(y, e) and then does the opposite of what HALT(y, e) says it does.

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

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

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

$$\Phi_e(t) = f(e,t) = \begin{cases} k & \text{if } t = 0, \\ g(t \dot{-} 1, \Phi_e(t \dot{-} 1)) & \text{otherwise.} \end{cases}$$

Example cont'd

Example

The function Φ_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, Φ_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

Do partially computable functions f and g that satisfy the equations

$$\begin{aligned} f(0) &= 1, \\ f(t+1) &= g(2t)+1, \\ g(0) &= 3, \\ g(2t+2) &= f(t)+2. \end{aligned}$$

exist?

Define the partially computable function $F(z,t)$ as

$$F(z,x) = \begin{cases} 1 & \text{if } x = \langle 0,0\rangle \\ \Phi_z(\langle 1, 2(r(x) \dot{-} 1)\rangle) + 1 & \text{if } (\exists y)_{\leqslant x}(x = \langle 0, y+1\rangle). \\ 3 & \text{if } x = \langle 1,0\rangle \\ \Phi_z(\langle 0, \lfloor (r(x) \dot{-} 2)/2 \rfloor\rangle) + 2 & \text{if } (\exists y)_{\leqslant x}(x = \langle 1, 2y+2\rangle). \end{cases}$$

By the Recursion Theorem there exists a number e such that $\Phi_e(x) = F(e, x)$, which means that

$$\Phi_e(x) = \begin{cases} 1 & \text{if } x = \langle 0, 0 \rangle \\ \Phi_e(\langle 1, 2(r(x) \dot{-} 1) \rangle) + 1 & \text{if } (\exists y)_{\leqslant x}(x = \langle 0, y + 1 \rangle). \\ 3 & \text{if } x = \langle 1, 0 \rangle \\ \Phi_e(\langle 0, \lfloor (r(x) \dot{-} 2)/2 \rfloor \rangle) + 2 & \text{if } (\exists y)_{\leqslant x}(x = \langle 1, 2y + 2 \rangle). \end{cases} \tag{1}$$

Example cont'd

Define

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

or

$$f(x) = \Phi_e(2x) \text{ and } g(x) = \Phi_e(4x + 1).$$

Then, Equation 1 amounts to

$$\begin{aligned}
&f(0) = \Phi_e(\langle 0, 0\rangle) = 1\\
&g(0) = \Phi_e(\langle 1, 0\rangle) = 3\\
&f(t+1) = \Phi_e(\langle 0, t+1\rangle) = \Phi_e(\langle 1, 2t\rangle + 1) = g(2t) + 1\\
&g(2t+2) = \Phi_e(\langle 1, 2t+2\rangle) = \Phi_e(\langle 0, t\rangle) + 2 = f(t) + 2.
\end{aligned}$$

Theorem

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 $\text{while}(n) = \#(\mathcal{Q}_n)$, where $\mathcal{Q}_n$ is the program:

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

The function 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

Example

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

$Y \leftarrow X$
while $P(Y)$ do
  $Y \leftarrow g(Y)$
end

Yet another proof of Rice's Theorem

Suppose that R_Γ were recursive. Let P_Γ be the characteristic function of R_Γ, 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 $g \notin \Gamma$.

Then, since

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

if 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 $g \notin \Gamma$ we have:

$$\begin{aligned} e \in R_\Gamma &\Rightarrow \Phi_e(x) = g(x) \\ &\Rightarrow \Phi_e \notin \Gamma \\ &\Rightarrow e \notin R_\Gamma. \end{aligned}$$

Likewise,

$$\begin{aligned} e \notin R_\Gamma &\Rightarrow \Phi_e(x) = f(x) \\ &\Rightarrow \Phi_e \in \Gamma \\ &\Rightarrow e \in R_\Gamma, \end{aligned}$$

so either case leads to a contradiction.