12. The S^n_m Theorem

In the previous sections we defined various notions of ***computability***, and investigated their ***interrelationship***.

In the remaining 3 sections of these notes, we will study some interesting properties of the ***indexing*** (or ***Gödel numbering***) of $\mathcal{G}$-computable functions.

NOTES:

1. We will write "comp" for "$\mathcal{G}$-computable", and "COMP" for the class "$\mathcal{G}$-COMP".

2. Although our indexing of computable functions is induced by our GN of the programming language $\mathcal{G}$, it can be shown that the results below (**$\mathbf{S^n_m}$ *theorem***, ***fixed point*** and ***recursion theorems***, and ***Rice's theorem***) hold under very general assumptions on the indexing of computable functions.

The main result of this section, the **$\mathbf{S^n_m}$ *Theorem*** of Kleene (also known as the ***parameter theorem***), is very useful for manipulating indices of functions, and is one of the main tools in the proof of the recursion theorem (Sec. 13).

Theorem 12.1 (S^n_m Thm). *For all $m, n > 0$, there is an $(n+1)$-ary function $S^n_m \in \mathrm{PR}$ such that for all $u_1, \ldots, u_n$, $x_1, \ldots, x_m$, y,*

$$\varphi^{(m+n)}_y(\vec{x}, \vec{u}) \simeq \varphi^{(m)}_{S^n_m(y,\vec{u})}(\vec{x}).$$

For some intuition on what this theorem says, let $m = n = 1$. Then there exists a binary PR function $S = S^1_1$ such that for all x, u, y,

$$\varphi^{(2)}_y(x, u) = \varphi_{S(y,u)}(x).$$

We may think of $\varphi^{(2)}_y$ for *fixed y and u* as a unary function $\lambda x \cdot \varphi^{(2)}_y(x, u)$. This function is comp, with gn z (say). So for all x,

$$\varphi_z(x) \simeq \varphi^{(2)}_y(x, u).$$

The theorem says that z ***depends primitive recursively*** on y and u, i.e.

$$z = S(y, u) \text{ for } S \in \mathrm{PR}.$$

Proof: By induction on n:

- **Basis:** $n = 1$. We want a PR function S^1_m such that for $\vec{x} \equiv x_1, \ldots, x_m$,

$$\varphi_y^{(m+1)}(\vec{x}, u) \simeq \varphi^{(m)}_{S^1_m(y,u)}(\vec{x}).$$

  Note that $\mathcal{P}_y$ is the progam for $\varphi_y^{(m+1)}$. For fixed y and u we now want a program $\mathcal{Q}$ for computing $\lambda \vec{x} \cdot \varphi_y^{(m+1)}(\vec{x}, u)$. We can think of $\mathcal{Q}$ as consisting of two parts:

$$\begin{aligned} &\mathcal{Q}_1 : \text{initialise } X_{m+1} \text{ to } u, \\ &\mathcal{Q}_2 : \text{then execute } \mathcal{P}_y. \end{aligned}$$

  Clearly, we can take

$$\mathcal{Q}_1 \equiv \boxed{\left.\begin{array}{c} X_{m+1}\text{++} \\ \vdots \\ X_{m+1}\text{++} \end{array}\right\} u \text{ times}}\;.$$

  Now the gn of the instruction 'X_{m+1}++' is (see p. 6-7)

$$\langle 0, \langle 1, 2m+1 \rangle \rangle = 16m + 10.$$

  So

$$\begin{aligned} \#(\mathcal{Q}_1) &= \textstyle\prod_{i=1}^{u} \big(p_i{}^{16m+10}\big) \mathbin{\dot{-}} 1 \\ &= q_1(u) \ \text{(say)} \\ \text{and} \ \ \#(\mathcal{Q}_2) &= y, \end{aligned}$$

  where $q_1 \in \mathrm{PR}$. Therefore

$$\begin{aligned} \#(\mathcal{Q}) &= \boldsymbol{concat}(q_1(u) + 1, y + 1) \mathbin{\dot{-}} 1 \\ &= S^1_m(y, u), \end{aligned}$$

  where $S^1_m \in \mathrm{PR}$ (by Lemma 6.11).

- **Induction step:** Suppose the result holds for $n = k$. Then

$$\begin{aligned}&\varphi_y^{(m+k+1)}(\vec{x}, u_1, \ldots, u_{k+1})\\ \simeq\ &\varphi^{(m+k)}_{S^1_{m+k}(y,u_{k+1})}(\vec{x}, u_1, \ldots, u_k)\\ \simeq\ &\varphi^{(m)}_{S^k_m(S^1_{m+k}(y,u_{k+1}),u_1,\ldots,u_k)}(\vec{x}).\end{aligned}$$

  By defining

$$\begin{aligned}&S^{k+1}_m(y, u_1, \ldots, u_{k+1})\\ =_{df}\ &S^k_m(S^1_{m+k}(y, u_{k+1}), u_1, \ldots, u_k)\end{aligned}$$

  the result follows. $\Box$

NOTE: In the ***UFT*** (Thm 7.5) and the $\mathbf{S^n_m}$ ***Thm*** we have two powerful tools for forming new computable functions from old:

- The ***UFT*** states that $\varphi^{(n)}_y(\vec{x})$ is a computable function of y and $\vec{x}$ *together*, i.e. it provides a way to ***move arguments up*** from the index.

  EXAMPLE: $\varphi^{(2)}_{\varphi_z(y)}(x, \varphi_{\varphi_u(x)}(z))$ is a computable function of u, x, y, z.

- The $\mathbf{S^n_m}$ ***Thm*** makes it possible to ***move arguments down*** to the index while preserving primitive recursiveness.

  EXAMPLE: Suppose f is a 5-ary computable function of x, y, z, u, v. Then the arguments y, u, v (say) can be moved down to the index, i.e.

$$f(x, y, z, u, v) \simeq \varphi_{g(y,u,v)}(x, z)$$

  for some $g \in \mathrm{PR}$.

- These two tools can be used "simultaneously".

  EXAMPLE: We show that there is a function $g \in \mathrm{PR}$ such that for all u and v, $\varphi_u \circ \varphi_v = \varphi_{g(u,v)}$. Indeed, for some computable function f and some PR function g,

$$\begin{aligned}\varphi_u(\varphi_v(x)) &\simeq f(u, v, x), \quad \text{(by UFT)}\\ &\simeq \varphi_{g(u,v)}(x), \quad \text{(by } S^n_m\text{)}.\end{aligned}$$