# 9. $\mu$-Primitive Recursive Functions

We inductively define the class $\boldsymbol{\mu}$**PR** of ***$\boldsymbol{\mu}$-primitive recursive functions***. This is the least class of functions which

(1) contains the ***initial functions*** $\boldsymbol{S}$, $\boldsymbol{Z}$ and $\boldsymbol{U}_i^n$;

(2) is closed under ***composition*** and ***primitive recursion***; and

(3) is closed under the (unbounded) ***$\boldsymbol{\mu}$-operator***, i.e., if $g \in \mu\mathrm{PR}^{(n+1)}$ and

$$f(\vec{x}) \simeq \mu y[g(\vec{x}, y) \downarrow 0], \tag{1}$$

then $f \in \mu\mathrm{PR}^{(n)}$;

where $\mu\mathrm{PR}^{(n)}$ is the class of $\mu$PR functions of arity $n$.
(The $\mu$-operator was introduced on p. 5-10.)

NOTES:

1. Without clause 3, the definition yields the class PR.
   The effect of clause 3 is to include ***partial functions***.
   E.g., if $g = \lambda\vec{x}, y \cdot 1$, then $f$ is the *totally undefined function* $\lambda\vec{x}\cdot\uparrow$ .

2. Note the ***constructive*** or ***computational meaning*** of $\boldsymbol{\mu}$:
   Suppose, e.g., that in (1), for some given $\vec{x}$,

   $$g(\vec{x}, 0) \downarrow 1, \quad g(\vec{x}, 1) \downarrow 1, \quad g(\vec{x}, 2) \uparrow, \quad g(\vec{x}, 3) \downarrow 0.$$

   Then $f(\vec{x})\uparrow$, since in the computation of $g(\vec{x}, y)$ for $y = 0, 1, 2, \ldots,$ we never reach $y = 3$. (This is ***local divergence*** at $y = 2$. We have ***global divergence*** if for all $y$, $g(\vec{x}, y) \downarrow\neq 0$.)

3. Each $\mu$PR function has an associated ***$\boldsymbol{\mu}$PR-derivation***, which is similar to a PR-derivation, but with the extra possibility of obtaining a function from a previous function in the derivation by applying the $\mu$-operator.

**Lemma 9.1**. *In (1), $f \in \mathcal{G}$-COMP$(g)$. Hence if $g \in \mathcal{G}$-COMP, so is $f$. In other words, $\mathcal{G}$-COMP* ***is closed under the $\mu$-operator***.

**Proof:** The following $\mathcal{G}$-program with an oracle for $g$, computes $f$:

```
[A]  Z ← g(X⃗, Y)
     if Z = 0 goto E
     Y++
     goto A
```

□

Next we give two celebrated results, essentially due to Kleene (using a different formalism and terminology—see [Kle52], Part III).

**Theorem 9.2 (Normal Form Theorem for $\mathcal{G}$-COMP)**. *For all $n$, there exists a PR $(n+2)$-ary predicate $\boldsymbol{T}^{(n)}$, and a PR function $\boldsymbol{U}$, such that for all $e$ and $\vec{x}$,*

$$\varphi_e^{(n)}(\vec{x}) \simeq \boldsymbol{U}(\mu y\, \boldsymbol{T}^{(n)}(e, \vec{x}, y)). \tag{2}$$

**Proof:** A ***computation history number*** (gn of a terminating computation) has the form

$$y \;=\; p_1^{e_1} p_2^{e_2} \ldots p_\ell^{e_\ell} \quad \big( = [e_1, \ldots, e_\ell] + 1 \big)$$

where for $1 \leq t \leq \ell$, $e_t$ is a ***snapshot*** at time $t$, i.e.

$$e_t = \langle k_t, s_t \rangle$$

$$\begin{aligned} \text{where} \quad k_t &= \boldsymbol{K}^{(n)}(e, \vec{x}, t) \\ \text{and} \quad s_t &= \boldsymbol{S}^{(n)}(e, \vec{x}, t) \end{aligned}$$

(as defined in pages 7-8/9). Now define the predicate $\boldsymbol{T}^{(n)}(e, \vec{x}, y)$:

"$y$ is the comp. history no. when $\mathcal{P}_e$ has input $\vec{x}$."

In symbols, putting $L_e = \boldsymbol{Lt}(e+1)$ and $L_y = \boldsymbol{Lt}(y)$:

$$\begin{aligned} &(\forall t \leq L_y)[(y)_t = \langle \boldsymbol{K}^{(n)}(e, \vec{x}, t), \boldsymbol{S}^{(n)}(e, \vec{x}, t) \rangle] \\ \wedge\ &(\forall t < L_y)[(1 \leq \boldsymbol{K}^{(n)}(e, \vec{x}, t) \leq L_e) \\ \wedge\ &(\boldsymbol{K}^{(n)}(e, \vec{x}, L_y) > L_e)]. \end{aligned}$$

Also define $\boldsymbol{U}(y)$ to be the value of the output variable $Y$ at the final state of computation $y$. In symbols:

$$\boldsymbol{U}(y) = (\boldsymbol{r}((y)_{\boldsymbol{Lt}(y)}))_1.$$

Clearly, $\boldsymbol{T}^{(n)}$, $\boldsymbol{U} \in$ PR, and (2) holds. $\square$

**Theorem 9.3**. $\mu$PR $=$ $\mathcal{G}$-COMP.

**Proof:** We will show that

$$f \text{ is } \mu\text{PR} \;\Leftrightarrow\; f \text{ is } \mathcal{G}\text{-computable.}$$

($\Leftarrow$) By Thm 9.2.

($\Rightarrow$) This is obvious from CT. However, we can give a proof without CT, which serves as ***confirmation for CT***. We will effectively associate, with each $\mu$PR-*derivation* of a function $f$, a $\mathcal{G}$-*program* for $f$, by ***CVI on the length of the derivation***. (Cf. proof of Lemma 4.5, p. 4-3.)
If the last step in the derivation is an *initial function*, or formed by *composition* or *primitive recursion*, use Lemma 4.2 (p. 4-1). If the last step is an application of the $\mu$-*operator* (the new case), use Lemma 9.1. $\square$

NOTES:

4. As with PR-derivations (see Ex. 2(a), p. 8-8) we can give an ***effective listing*** of the set $\mu$PR-DERIV of $\mu$PR-derivations, and hence an ***effective listing of $\mu$PR***.

5. The proof of Thm 9.3 actually gives ***effective maps*** between the two programming languages $\mu$PR-DERIV and $\mathcal{G}$-PROG (PR in their gn's, in fact). In other words, it shows how to ***compile*** these two languages (or "models of computation") in each other.

6. The effective listing of $\mu$PR-DERIV given in Note 4, together with the compilation of $\mathcal{G}$-PROG in $\mu$PR-DERIV in Note 5, gives a ***second effective listing*** (and GN) of $\mathcal{G}$-PROG, and hence of $\mathcal{G}$-COMP ($=\mu$PR). (The first was given by the GN in §6.3.)

7. Thms 9.2 and 9.3 together show that any $\mu$PR (or equivalently, $\mathcal{G}$-computable) function has a $\mu$PR-derivation in which the $\mu$-operator is used only once!

8. There is also a ***relativised*** notion of $\mu$-primitive recursiveness, and a relativised version of Thm 9.3:

$$\mu\mathrm{PR}(\vec{g}) \;=\; \mathcal{G}\text{-COMP}(\vec{g}). \tag{3}$$

EXERCISES:

1. Define the class $\mu\mathrm{PR}(\vec{g})$, and outline a proof for (3).

2. Can't we (more simply) prove the '$\Leftarrow$' direction in Thm 9.3, by CVI on the length of the $\mathcal{G}$-program (as with the '$\Rightarrow$' direction), instead of using the NF Thm for $\mathcal{G}$-COMP (Thm 9-2)?