We state for the record:

0.0.1 Proposition. *$\mathcal{R}$ is closed under composition.*

Proof. Because $\mathcal{P}$ is, and the operation conserves totalness. We did this in class on Jan. 16 (see Notes #2). $\square$

0.0.2 Example. Is the function $\lambda \vec{x}_n.x_i$, where $1 \le i \le n$, in $\mathcal{P}$? Yes, and here is a program, M, for it:

$$
\begin{array}{ll}
1: & \mathbf{w}_1 \leftarrow 0 \\
\vdots & \\
i: & \mathbf{z} \leftarrow \mathbf{w}_i \ \{\textbf{Comment.}\ \text{Macro}\} \\
\vdots & \\
n: & \mathbf{w}_n \leftarrow 0 \\
n+1: & \textbf{stop}
\end{array}
$$

$\lambda \vec{x}_n.x_i = M_{\mathbf{z}}^{\vec{\mathbf{w}}_n}$. To ensure that M indeed *has* the $\mathbf{w}_i$ as variables we reference them in instructions at least once, in any manner whatsoever.

The function $\lambda \vec{x}_n.x_i$ is denoted by U_i^n and is called "generalised identity" since it is the identity for input x_i while the extra arguments offer nothing towards obtaining the output. $\square$

0.0.1 Primitive Recursive Functions

The successor, zero, and the *generalised identity* functions respectively—which we will often name S, Z and U_i^n respectively—are in $\mathcal{P}$; thus, not only are they "intuitively computable", but they are so **in a precise mathematical sense**: each is computable by a URM.

We have also shown that "computability" of functions is **preserved** by the operations of **composition**, **primitive recursion**, and **unbounded search**. In this subsection we will explore the properties of the important set of functions known as **primitive recursive**. Most people introduce them via **derivations** just as one introduces the theorems of logic via proofs, as in the definition below.

0.0.3 Definition. ($\mathcal{PR}$-derivations; $\mathcal{PR}$-functions) The set

$$\mathcal{I} = \left\{ S, Z, \left(U_i^n\right)_{n \ge i > 0} \right\}$$

is the set of **Initial $\mathcal{PR}$** functions.

A *$\mathcal{PR}$-derivation* is a finite (ordered!) sequence of number-theoretic functions*

$$f_1, f_2, f_3, \ldots, f_i, \ldots, f_n \tag{1}$$

*That is, *left field* is $\mathbb{N}^n$ for some $n > 0$, and *right field* is $\mathbb{N}$.

such that, for **each** i, one of the following holds

1. $f_i \in \mathcal{I}$.
2. $f_i = prim(f_j, f_k)$ and $j < i$ and $k < i$ —that is, f_j, f_k appear **to the left of** f_i.
3. $f_i = \lambda\vec{y}.g\big(r_1(\vec{y}), r_2(\vec{y}), \ldots, r_m(\vec{y})\big)$, and **all** of the $\lambda\vec{y}.r_q(\vec{y})$ and $\lambda\vec{x}_m.g(\vec{x}_m)$ appear **to the left of** f_i in the sequence.

Any f_i in a derivation is called a **derived** function.†

The set of primitive recursive functions, $\mathcal{PR}$, is **all those that are derived**. That is,

$$\mathcal{PR} \stackrel{Def}{=} \{f : f \text{ is derived}\}$$ □

The above definition defines essentially what Dedekind called "recursive" functions. Subsequently they were renamed *primitive recursive* allowing the unqualified term *recursive* to be synonymous with (total) *computable* and apply to the functions of $\mathcal{R}$.

0.0.4 Lemma. *The concatenation of two derivations is a derivation.*

Proof. Let

$$f_1, f_2, f_3, \ldots, f_i, \ldots, f_n \tag{1}$$

and

$$g_1, g_2, g_3, \ldots, g_j, \ldots, g_m \tag{2}$$

be two derivations. Then so is

$$f_1, f_2, f_3, \ldots, f_i, \ldots, f_n, g_1, g_2, g_3, \ldots, g_j, \ldots, g_m$$

because of the fact that each of the f_i and g_j satisfies the three cases of Definition 0.0.3 in the standalone derivations (1) and (2). But this property of the f_i and g_j *is preserved* after concatenation. □

0.0.5 Corollary. *The concatenation of any finite number of derivations is a derivation.*

0.0.6 Lemma. *If*

$$f_1, f_2, f_3, \ldots, f_k, f_{k+1}, \ldots, f_n$$

is a derivation, then so is $f_1, f_2, f_3, \ldots, f_k$.

Proof. In $f_1, f_2, f_3, \ldots, f_k$ every f_m, for $1 \le m \le k$, satisfies 1.–3. of Definition 0.0.3 since all conditions are in terms of what f_m is, or what lies **to the left of** f_m. Chopping the "tail" $f_{k+1}, \ldots, f_n$ in no way affects what lies to the left of f_m, for $1 \le m \le k$. □

†Strictly speaking, *primitive recursively derived*, but we will not considered other sets of derived functions, so we omit the qualification.

0.0.7 Corollary. *$f \in \mathcal{PR}$ iff f appears at the **end** of some derivation.*

Proof.

(a) The *If.* Say $g_1, \ldots, g_n, \boxed{f}$ is a derivation. Since f occurs in it, $f \in \mathcal{PR}$ by 0.0.3.

(b) The *Only If.* Say $f \in \mathcal{PR}$. Then, by 0.0.3,

$$g_1, \ldots, g_m, \boxed{f}, g_{m+2}, \ldots, g_r \tag{1}$$

for some derivation like the (1) above.

By 0.0.6, $g_1, \ldots, g_m, \boxed{f}$ is also a derivation. □

0.0.8 Theorem. *$\mathcal{PR}$ is* closed under *composition and primitive recursion.*

Proof.

- Closure under **primitive recursion**. So let $\lambda \vec{y}.h(\vec{y})$ and $\lambda x\vec{y}z.g(x, \vec{y}, z)$ be in $\mathcal{PR}$. Thus we have derivations

$$h_1, h_2, h_3, \ldots, h_n, \boxed{h} \tag{1}$$

and

$$g_1, g_2, g_3, \ldots, g_m, \boxed{g} \tag{2}$$

Then the following is a derivation by 0.0.4.

$$h_1, h_2, h_3, \ldots, h_n, \boxed{h}, g_1, g_2, g_3, \ldots, g_m, \boxed{g}$$

Therefore so is

$$h_1, h_2, h_3, \ldots, h_n, \boxed{h}, g_1, g_2, g_3, \ldots, g_m, \boxed{g}, prim(h, g)$$

by applying step 2 of Definition 0.0.3.

This implies $prim(h, g) \in \mathcal{PR}$ by 0.0.3.

- Closure under **composition**. So let $\lambda \vec{y}.h(\vec{x}_n)$ and $\lambda \vec{y}.g_i(\vec{y})$, for $1 \le i \le n$, be in $\mathcal{PR}$. By 0.0.3 we have derivations

$$\boxed{\ldots, \boxed{h}} \tag{3}$$

and

$$\boxed{\ldots, \boxed{g_i}}, \text{ for } 1 \le i \le n \tag{4}$$

By 0.0.4,

$$\boxed{\ldots, \boxed{\;h\;}}, \boxed{\ldots, \boxed{\;g_1\;}}, \ldots, \boxed{\ldots, \boxed{\;g_n\;}}$$

is a derivation, and by 0.0.3, case 3, so is

$$\boxed{\ldots, \boxed{\;h\;}}, \boxed{\ldots, \boxed{\;g_1\;}}, \ldots, \boxed{\ldots, \boxed{\;g_n\;}}, \lambda \vec{y}.h(g_1(\vec{y}), \ldots, g_n(\vec{y}))$$

$\square$

0.0.9 Remark. How do you prove that some $f \in \mathcal{PR}$?

Answer. By building a derivation

$$g_1, \ldots, g_m, \boxed{f}$$

After a while it becomes *easier* because you might **know** an h and g in $\mathcal{PR}$ such that $f = prim(h, g)$, or you might know some $g, h_1, \ldots, h_m$ in $\mathcal{PR}$, such that $f = \lambda\vec{y}.g\big(h_1(\vec{y}), \ldots, h_m(\vec{y})\big)$. **If so, just apply 0.0.8**.

How do you prove that *ALL* $f \in \mathcal{PR}$ have a property Q —that is, for all f, $Q(f)$ is true?

Answer. By doing **induction on the derivation length** of f. $\square$

Here are two examples of the above questions and their answers.

0.0.10 Example. (1) To demonstrate the first Answer above (0.0.9), show (prove) that $\lambda xy.x + y \in \mathcal{PR}$. Well, observe that

$$0 + y = y$$
$$(x + 1) + y = (x + y) + 1$$

Does the above look like a primitive recursion? Well, almost. However, the first equation should have a function call "$H(y)$" on the rhs but instead has just y —the input! Also the second equation should have a rhs like "$G(x, y, x + y)$". We can do that! Take $H = \lambda y.U^1_1(y)$ and $G = \lambda xyz.S\Big(U^3_3(x, y, z)\Big)$. Be sure to agree that

- H and G recast the two equations above in the correct **form**:

$$0 + y = U^1_1(y)$$
$$(x + 1) + y = SU^3_3\Big(x, y, (x + y)\Big)$$

- The functions U^1_1 (initial) and SU^3_3 (composition) are in $\mathcal{PR}$ (**NOTE** the "SU^3_3" with no brackets around U^3_3; this is normal practise!) By 0.0.8 so is $\lambda xy.x + y$.

  In terms of derivations, we have produced the derivation:

$$U^1_1, S, U^3_3, SU^3_3, \underbrace{prim\left(U^1_1, SU^3_3\right)}_{\lambda xy.x+y}$$

(2) To demonstrate the second Answer above (0.0.9), show (prove) that every $f \in \mathcal{PR}$ is **total**. Induction on derivation length, n, where f occurs.

Basis. $n = 1$. Then f is the only function in the derivation. Thus it must be one of S, Z, or U_i^m. But all these are total.

I.H. (Induction Hypothesis) Assume that the claim is true for all f that occur in derivations of lengths $n \le l$. That is, *we assume that all such f are total.*

I.S. (Induction Step) Prove that the claim is true for all f that occur in derivations of lengths $n = l + 1$.

$$g_1, \ldots, g_i, \boxed{f}, g_{i+2}, \ldots, g_{l+1} \tag{1}$$

- Case where f is not the last function in the derivation (1). Then dropping the tail $g_{i+2}, \ldots, g_{l+1}$ we have f appear in a derivation of length $\le l$ and thus it is total by the I.H.

  The interesting case is when f is the last function of a derivation of length $l + 1$ as in (2) below:

  $$g_1, \ldots, g_l, \boxed{f} \tag{2}$$

  We have three subcases:

  - $f \in \mathcal{I}$. But we argued this under *Basis*.
  - $f = prim(h, g)$, where h and g are among the $g_1, \ldots, g_l$. By the I.H. h and g are total. But then so is f by a Lemma in the Notes #3.
  - $f = \lambda\vec{y}.h\Big(q_1(\vec{y}), \ldots, q_t(\vec{y})\Big)$, where the functions h and $q_1, \ldots, q_t$ are among the $g_1, \ldots, g_l$. By the I.H. h and $q_1, \ldots, q_t$ are total. But then so is f by a Lemma in the Notes #2, when we proved that $\mathcal{R}$ is closed under composition.

$\square$

0.0.11 Example. If $\lambda xyw.f(x, y, w)$ and $\lambda z.g(z)$ are in $\mathcal{PR}$, how about $\lambda xzw.f(x, g(z), w)$? It is in $\mathcal{PR}$ since

$$\lambda xzw.f(x, g(z), w) = \lambda xzw.f(U_1^3(x, z, w), g(U_2^3(x, z, w)), U_3^3(x, z, w))$$

and the U_i^n are primitive recursive. The reader will see at once that to the right of "=" we have correctly formed compositions as expected by the "rigid" definition of composition given in class.

Similarly, for the same functions above,

(1) $\lambda yw.f(2,y,w)$ is in $\mathcal{PR}$. Indeed, this function can be obtained by composition, since

$$\lambda yw.f(2,y,w) = \lambda yw.f\Big(SSZ\big(U_1^2(y,w)\big), y, w\Big)$$

where I wrote "$SSZ(\ldots)$" as short for $S(S(Z(\ldots)))$ for visual clarity. Clearly, using $SSZ\big(U_2^2(y,w)\big)$ above works as well.

(2) $\lambda xyw.f(y,x,w)$ is in $\mathcal{PR}$. Indeed, this function can be obtained by composition, since

$$\lambda xyw.f(y,x,w) = \lambda xyw.f\Big(U_2^3(x,y,w), U_1^3(x,y,w), U_3^3(x,y,w)\Big)$$

In this connection, note that while $\lambda xy.g(x,y) = \lambda yx.g(y,x)$, yet $\lambda xy.g(x,y) \neq \lambda xy.g(y,x)$ in general. For example, $\lambda xy.x \dot{-} y$ asks that we subtract the second input (y) from the first (x), but $\lambda xy.y \dot{-} x$ asks that we subtract the first input (x) from the second (y).

(3) $\lambda xy.f(x,y,x)$ is in $\mathcal{PR}$. Indeed, this function can be obtained by composition, since

$$\lambda xy.f(x,y,x) = \lambda xy.f\big(U_1^2(x,y), U_2^2(x,y), U_1^2(x,y)\big)$$

(4) $\lambda xyzwu.f(x,y,w)$ is in $\mathcal{PR}$. Indeed, this function can be obtained by composition, since

$$\lambda xyzwu.f(x,y,w) = \\ \lambda xyzwu.f(U_1^5(x,y,z,w,u), U_2^5(x,y,z,w,u), U_4^5(x,y,z,w,u))$$

□

The above examples are summarised, named, and generalised in the following straightforward exercise:

0.0.12 Exercise. (The [Grz53] Substitution Operations) $\mathcal{PR}$ is closed under the following operations:

(i) *Substitution of a function invocation for a variable*:

From $\lambda \vec{x}y\vec{z}.f(\vec{x},y,\vec{z})$ and $\lambda \vec{w}.g(\vec{w})$ obtain $\lambda \vec{x}\vec{w}\vec{z}.f(\vec{x},g(\vec{w}),\vec{z})$.

(ii) *Substitution of a constant for a variable*:

From $\lambda \vec{x}y\vec{z}.f(\vec{x},y,\vec{z})$ obtain $\lambda \vec{x}\vec{z}.f(\vec{x},k,\vec{z})$.

(iii) *Interchange of two variables*:

From $\lambda \vec{x}y\vec{z}w\vec{u}.f(\vec{x},y,\vec{z},w,\vec{u})$ obtain $\lambda \vec{x}y\vec{z}w\vec{u}.f(\vec{x},w,\vec{z},y,\vec{u})$.

(iv) *Identification of two variables*:

From $\lambda \vec{x} y \vec{z} w \vec{u}.f(\vec{x}, y, \vec{z}, w, \vec{u})$ obtain $\lambda \vec{x} y \vec{z} \vec{u}.f(\vec{x}, y, \vec{z}, y, \vec{u})$.

(v) *Introduction of "don't care" variables*:

From $\lambda \vec{x}.f(\vec{x})$ obtain $\lambda \vec{x} \vec{z}.f(\vec{x})$. $\square$

By 0.0.12 composition can simulate the Grzegorczyk operations if the initial functions $\mathcal{I}$ are present. Of course, (i) alone can in turn simulate composition. With these comments out of the way, we see that the "rigidity" of the definition of composition is gone.

0.0.13 Example. The definition of primitive recursion is also rigid. However this is an illusion.

Take $p(0) = 0$ and $p(x+1) = x$ —this one defining $p = \lambda x.x \dot{-} 1$ —does not fit the schema.

The schema requires the defined function to have one more variable than the basis, so no one-variable function can be directly defined!

We can get around this.

Define first $\widetilde{p} = \lambda xy.x \dot{-} 1$ as follows: $\widetilde{p}(0, y) = 0$ and $\widetilde{p}(x+1, y) = x$. Now this can be dressed up according to the syntax of the schema,

$$\begin{aligned} \widetilde{p}(0, y) &= Z(y) \\ \widetilde{p}(x+1, y) &= U^3_1(x, y, \widetilde{p}(x, y)) \end{aligned}$$

that is, $\widetilde{p} = prim(Z, U^3_1)$. Then we can get p by (Grzegorczyk) substitution: $p = \lambda x.\widetilde{p}(x, 0)$. Incidentally, this shows that both p and $\widetilde{p}$ are in $\mathcal{PR}$:

- $\widetilde{p} = prim(Z, U^3_1)$ is in $\mathcal{PR}$ since Z and U^3_1 are, then invoking 0.0.8.
- $p = \lambda x.\widetilde{p}(x, 0)$ is in $\mathcal{PR}$ since $\widetilde{p}$ is, then invoking 0.0.12.

Another rigidity in the definition of primitive recursion is that, *apparently, one can use only the first variable as the iterating variable.*

Not so. This is an illusion.

Consider, for example, $sub = \lambda xy.x \dot{-} y$. Clearly, $sub(x, 0) = x$ and $sub(x, y + 1) = p(sub(x, y))$ is correct semantically, but the **format** is wrong: We are not supposed to iterate along the second variable! Well, define instead $\widetilde{sub} = \lambda xy.y \dot{-} x$:

$$\begin{aligned} \widetilde{sub}(0, y) &= U_1^1(y) \\ \widetilde{sub}(x + 1, y) &= p\big(U_3^3(x, y, \widetilde{sub}(x, y))\big) \end{aligned}$$

Then, using variable swapping [Grzegorczyk operation (iii)], we can get sub: $sub = \lambda xy.\widetilde{sub}(y, x)$. Clearly, both $\widetilde{sub}$ and sub are in $\mathcal{PR}$. □

0.0.14 Exercise. Prove that $\lambda xy.x \times y$ is primitive recursive. Of course, we will usually write multiplication $x \times y$ in "implied notation", xy. □

0.0.15 Example. The very important "*switch*" (or "if-then-else") function $sw = \lambda xyz.\text{if } x = 0 \text{ then } y \text{ else } z$ is primitive recursive. It is directly obtained by primitive recursion on initial functions: $sw(0, y, z) = y$ and $sw(x + 1, y, z) = z$.

□

0.0.16 Proposition. $\mathcal{PR} \subseteq \mathcal{R}$.

Proof. Start with proving $\mathcal{PR} \subseteq \mathcal{P}$ (Problem Set #1) and then use Example 0.0.10. Or, prove directly by induction on derivation length that $\mathcal{PR} \subseteq \mathcal{R}$ □

Indeed, the above inclusion is proper, as we will see later.

0.0.17 Example. Consider the function ex given by

$$\begin{aligned} ex(x,0) &= 1 \\ ex(x,y+1) &= ex(x,y)x \end{aligned}$$

Thus, if $x = 0$, then $ex(x,0) = 1$, but $ex(x,y) = 0$ for all $y > 0$. On the other hand, if $x > 0$, then $ex(x,y) = x^y$ for all y.

Note that x^y is "mathematically" undefined when $x = y = 0$.[‡] Thus, by Example 0.0.10 the exponential cannot be a primitive recursive function!

This is rather silly, since the computational process for the exponential is so straightforward; thus it is a shame to declare the function non-$\mathcal{PR}$. After all, we know *exactly where and how it is undefined* and we can remove this undefinability by *redefining "x^y" to mean $ex(x,y)$ for all inputs.*

Clearly $ex \in \mathcal{PR}$. In computability we do this kind of redefinition a lot in order to remove easily recognisable points of "non definition" of calculable functions. We will see further examples, such as the remainder, quotient, and logarithm functions.

Caution! We cannot always remove points of non definition of a calculable function and still obtain a computable function. That is, there are functions $f \in \mathcal{P}$ that *have no recursive extensions.* This we will show later. $\square$

[‡]In first-year university calculus we learn that "0^0" is an "indeterminate form".

0.0.18 Definition. A relation $R(\vec{x})$ is (*primitive*) *recursive* iff its *characteristic function*,

$$\chi_R = \lambda \vec{x}. \begin{cases} 0 & \text{if } R(\vec{x}) \\ 1 & \text{if } \neg R(\vec{x}) \end{cases}$$

is (primitive) recursive. The set of all primitive recursive (respectively, recursive) *relations* is denoted by $\mathcal{PR}_*$ (respectively, $\mathcal{R}_*$). □

Computability theory practitioners often call relations *predicates*.

It is clear that one can go from relation to characteristic function and back in a unique way, since $R(\vec{x}) \equiv \chi_R(\vec{x}) = 0$. Thus, we may think of relations as "0-1 valued" functions. The concept of relation simplifies the further development of the theory of primitive recursive functions.

The following is useful:

0.0.19 Proposition. *$R(\vec{x}) \in \mathcal{PR}_*$ iff some $f \in \mathcal{PR}$ exists such that, for all $\vec{x}$, $R(\vec{x}) \equiv f(\vec{x}) = 0$.*

Proof. For the *if*-part, I want $\chi_R \in \mathcal{PR}$. This is so since $\chi_R = \lambda \vec{x}.1 \dot{-} (1 \dot{-} f(\vec{x}))$ (using Grzegorczyk substitution and $\lambda xy.x \dot{-} y \in \mathcal{PR}$; cf. 0.0.13). For the *only if*-part, $f = \chi_R$ will do. □

0.0.20 Corollary. *$R(\vec{x}) \in \mathcal{R}_*$ iff some $f \in \mathcal{R}$ exists such that, for all $\vec{x}$, $R(\vec{x}) \equiv f(\vec{x}) = 0$.*

Proof. By the above proof, 0.0.16, and 0.0.1. □

0.0.21 Corollary. *$\mathcal{PR}_* \subseteq \mathcal{R}_*$.*

Proof. By the above corollary and 0.0.16. □

0.0.22 Theorem. *$\mathcal{PR}_*$ is closed under the Boolean operations.*

Proof. It suffices to look at the cases of $\neg$ and $\vee$, since $R \to Q \equiv \neg R \vee Q$, $R \wedge Q \equiv \neg(\neg R \vee \neg Q)$ and $R \equiv Q$ is short for $(R \to Q) \wedge (Q \to P)$.

($\neg$) Say, $R(\vec{x}) \in \mathcal{PR}_*$. Thus (0.0.18), $\chi_R \in \mathcal{PR}$. But then $\chi_{\neg R} \in \mathcal{PR}$, since $\chi_{\neg R} = \lambda\vec{x}.1 \dot{-} \chi_R(\vec{x})$, by Grzegorczyk substitution and $\lambda xy.x \dot{-} y \in \mathcal{PR}$.

($\vee$) Let $R(\vec{x}) \in \mathcal{PR}_*$ and $Q(\vec{y}) \in \mathcal{PR}_*$. Then $\lambda\vec{x}\vec{y}.\chi_{R\vee Q}(\vec{x},\vec{y})$ is given by

$$\chi_{R\vee Q}(\vec{x},\vec{y}) = \text{if } R(\vec{x}) \text{ then } 0 \text{ else } \chi_Q(\vec{y})$$

and therefore is in $\mathcal{PR}$. $\square$

0.0.23 Remark. Alternatively, for the $\vee$ case above, note that $\chi_{R\vee Q}(\vec{x},\vec{y}) = \chi_R(\vec{x}) \times \chi_Q(\vec{y})$ and invoke 0.0.14. $\square$

0.0.24 Corollary. *$\mathcal{R}_*$ is closed under the Boolean operations.*

Proof. As above, mindful of 0.0.16, and 0.0.1. $\square$

0.0.25 Example. The relations $x \le y$, $x < y$, $x = y$ are in $\mathcal{PR}_*$.

An addendum to λ notation: Absence of λ is allowed ONLY for relations! Then it means (the absence) that ALL variables are active input!

Note that $x \le y \equiv x \dot{-} y = 0$ and invoke 0.0.19. Finally invoke Boolean closure and note that $x < y \equiv \neg y \le x$ while $x = y$ is equivalent to $x \le y \wedge y \le x$. $\square$

Bibliography

[Grz53] A. Grzegorczyk, *Some classes of recursive functions*, Rozprawy Matematyczne **4** (1953), 1–45.