# LECTURE #5 (Sept. 23; Continued)

Before we get more immersed into *partial functions* let us **redefine equality for function calls**.

**0.0.1 Definition.** Let $\lambda\vec{x}.f(\vec{x}_n)$ and $\lambda\vec{y}.g(\vec{y}_m)$.

We extend the notion of equality $f(\vec{a}_n) = g(\vec{b}_m)$ to include the case of *undefined calls*:

For any $\vec{a}_n$ and $\vec{b}_m$, $f(\vec{a}_n) = g(\vec{b}_m)$ means *precisely one of*

- For some $k \in \mathbb{N}$, $f(\vec{a}_n) = k$ and $g(\vec{b}_m) = k$
- $f(\vec{a}_n)\uparrow$ and $g(\vec{b}_m)\uparrow$

*For short*,

$$f(\vec{a}_n) = g(\vec{b}_m) \equiv (\exists z)\Big(f(\vec{a}_n) = z \wedge g(\vec{b}_m) = z \vee f(\vec{a}_n)\uparrow \wedge g(\vec{b}_m)\uparrow\Big)$$

□

The definition is due to Kleene and he preferred, as I do in the text, to use a new symbol for the extended equality, namely $\simeq$.

Regardless, by way of this note we agree to use the same symbol for equality for **both** total and nontotal calls, namely, "=" (this convention is common in the literature, e.g., [Rog67]).

**0.0.2 Lemma.** *If $f = prim(h, g)$ and $h$ and $g$ are* ***total****, then so is $f$.*

*Proof.* Let $f$ be given by:

$$f(0, \vec{y}) = h(\vec{y})$$
$$f(x + 1, \vec{y}) = g(x, \vec{y}, f(x, \vec{y}))$$

*We do induction on $x$* to prove

$$\text{“For all } x, \vec{y},\ f(x, \vec{y}) \downarrow\text{”} \tag{$*$}$$

*Basis.* $x = 0$: Well, $f(0, \vec{y}) = h(\vec{y})$, but $h(\vec{y}) \downarrow$ for all $\vec{y}$, so

$$f(0, \vec{y}) \downarrow \text{ for all } \vec{y} \tag{$**$}$$

As I.H. (Induction *Hypothesis*) take that

$$f(x, \vec{y}) \downarrow \text{ for all } \vec{y} \text{ and } \textit{fixed } x \tag{$\dagger$}$$

Do the Induction *Step* (I.S.) to show

$$f(x + 1, \vec{y}) \downarrow \text{ for all } \vec{y} \text{ and } \underline{\text{the fixed } x \text{ of } (\dagger)} \tag{$\ddagger$}$$

Well, by ($\dagger$) and the assumption on $g$,

$$g\big(x, \vec{y}, f(x, \vec{y})\big) \downarrow, \text{ for all } \vec{y} \text{ and the fixed } x \text{ of } (\dagger)$$

which says the same thing as ($\ddagger$). □

**0.0.3 Corollary.** *$\mathcal{R}$ is closed under primitive recursion.*

*Proof.* Let $h$ and $g$ be in $\mathcal{R}$. Then they are in $\mathcal{P}$. But then $prim(h, g) \in \mathcal{P}$ as we showed in class/text and Notes #2.

By 0.0.2, $prim(h, g)$ is total.

By definition of $\mathcal{R}$, as **the subset of** $\mathcal{P}$ **that contains all total functions of** $\mathcal{P}$, we have $prim(h, g) \in \mathcal{R}$. □

Why all this dance **in colour** above? Because to prove $f \in \mathcal{R}$ you need **TWO** things: That

1. $f \in \mathcal{P}$

   AND

2. $f$ is total

But aren't all the *total* functions in $\mathcal{R}$ anyway?

NO! They *need to be computable too!*

*We will see in this course soon that NOT all total functions are computable*!

### 0.0.1 Primitive Recursive Functions

We saw that

1. The successor —$S$

2. zero —$Z$

3. and the *generalised identity* functions —$U_i^n = \lambda \vec{x}_n . x_i$

   are all 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.

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

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

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

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

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

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 ([Ded88]) called "*recursive*" functions.

Subsequently they were renamed to *primitive recursive* allowing the unqualified term *recursive* to be synonymous with (total) *computable* and apply to the functions of $\mathcal{R}$.

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

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

**0.0.5 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 0.0.4 in the standalone derivations (1) and (2). But this property of the $f_i$ and $g_j$ *is preserved* after concatenation. □

# *Lecture #6 (Sept. 28)*

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

**0.0.7 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 0.0.4 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 $1 \le m \le k$. □

**0.0.8 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 0.0.4.

(b) The *Only If*. Say $f \in \mathcal{PR}$. Then, by 0.0.4,

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

for some derivation like the (1) above.

By 0.0.7, $g_1, \ldots, g_m, \boxed{f}$ is also a derivation. □

**0.0.9 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 0.0.5.

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

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 0.0.4.

*This implies $prim(h,g) \in \mathcal{PR}$ by 0.0.4.*

- 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 0.0.4 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 0.0.5,

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

is a derivation, and by 0.0.4, case 3, so is

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

*This implies $\lambda\vec{y}.h(g_1(\vec{y}), \ldots, g_n(\vec{y})) \in \mathcal{PR}$ by 0.0.4.* □

**0.0.10 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 this 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 0.0.9**.

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

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

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

**0.0.11 Example. (1)** To demonstrate the first Answer above (0.0.10), 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 a *variable* $y$ —the input!

Also the *second equation* should have a rhs like "$G(x, y, x + y)$".

We can do that!

*Take $H = U_1^1$ and $G = SU_3^3$* —**NOTE** the "$SU_3^3$" with no brackets around $U_3^3$; this is normal practise!

Be sure to agree that we now have

-
$$0 + y = H(y)$$
$$(x+1) + y = G\Big(x, y, x + y\Big)$$

- The functions $H = U_1^1$ (*initial*) and $G = SU_3^3$ (*composition*) are in $\mathcal{PR}$. By 0.0.9 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 (0.0.10), 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) *Fix an $l$.* Assume that the claim is true for all $f$ that occur *at the end of derivations of lengths* $n \leq l$. 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 at the *end of a derivation* —see 0.0.8— of length $n = l + 1$.

$$g_1, \ldots, g_l, \boxed{f} \qquad (1)$$

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. Elaboration: Any such $g_i$ is at the end of a derivation of length $\leq l$. So I.H. kicks in.

  But then so is $f$ by Lemma 0.0.2.

- $f = \lambda \vec{y}.h\Big(q_1(\vec{y}), \ldots, q_t(\vec{y})\Big)$, 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. □

**0.0.12 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, by *COMPOSITION*,

$$f(x,g(z),w) = f(U_1^3(x,z,w), \underline{g(U_2^3(x,z,w))}, U_3^3(x,z,w))$$

and the $U_i^n$ are all 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

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

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

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

In this connection, note that while $\underline{\lambda xy.g(x,y) = \lambda yx.g(y,x)}$, 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

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

(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 four examples are summarised, named, and generalised in the following straightforward exercise:

**0.0.13 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 $\lambda \vec{x} \vec{z}.f(\vec{x})$. □

*By 0.0.13 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.

**0.0.14 Example.** The definition of primitive recursion is also rigid. *However this is also 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,

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

*that is,* $\widetilde{p} = prim(Z, U_1^3)$.

*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* $\mathcal{PR}$*:*

- $\widetilde{p} = prim(Z, U_1^3)$ is in $\mathcal{PR}$ since $Z$ and $U_1^3$ are, then invoking 0.0.9.
- $p = \lambda x.\widetilde{p}(x, 0)$ is in $\mathcal{PR}$ since $\widetilde{p}$ is, then invoking 0.0.13.

# *Lecture # 7 (Sept. 30)*

*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 also an illusion.*

Consider, for example, $sub = \lambda xy.x \dot{-} y$, hence $x \dot{-} 0 = x$ and $x \dot{-} (y+1) = p(x \dot{-} y)$

*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* $\widetilde{sub} = \lambda xy.y \dot{-} x$*:*

So

$$\begin{aligned} y \dot{-} 0 &= y \\ y \dot{-} (x+1) &= p\Big(y \dot{-} x\Big) \end{aligned}$$

That is,

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

*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 $\mathcal{PR}$. $\square$

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

**0.0.16 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 $sw(x + 1, y, z) = z$. □

**0.0.17 Exercise.** $\mathcal{PR} \subseteq \mathcal{R}$. □

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

**0.0.18 Example.** Consider the exponential function $x^y$ given by

$$\begin{aligned} x^0 &= 1 \\ x^{y+1} &= x^y x \end{aligned}$$

Thus,

*if $x = 0$, then $x^y = 1$, but $x^y = 0$ for all $y > 0$.*

*BUT that $x^y$ is "mathematically" undefined when $x = y = 0$.*[‡]

Thus, by Example 0.0.11 *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 *ridiculous* 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 "$x^y$" so that "$0^0 = 1$"*.
*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.

**BUT also examples where we CANNOT do this; and WHY.** □

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

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

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

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,*

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:

**0.0.20 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 $c_R \in \mathcal{PR}$.*

*This is so since $c_R = \lambda \vec{x}.1 \dot{-} (1 \dot{-} f(\vec{x}))$* (using Grzegorczyk substitution and $\lambda xy.x \dot{-} y \in \mathcal{PR}$; cf. 0.0.14).

*For the* only if-*part*, $f = c_R$ will do. □

**0.0.21 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, and 0.0.17. □

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

*Proof.* By the above corollary and 0.0.17. □

**0.0.23 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 (0.0.19), $c_R \in \mathcal{PR}$. But then $c_{\neg R} \in \mathcal{PR}$, since $c_{\neg R} = \lambda \vec{x}.1 \dot{-} c_R(\vec{x})$, 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 $\lambda \vec{x}\vec{y}.c_{R\vee Q}(\vec{x}, \vec{y})$ is given by

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

and therefore is in $\mathcal{PR}$. □

*"if $R(\vec{x})$" above means "if $c_R(\vec{x}) = 0$"*

**0.0.24 Remark.** *Alternatively, for the $\vee$ case above, note that $c_{R\vee Q}(\vec{x}, \vec{y}) = c_R(\vec{x}) \times c_Q(\vec{y})$ and invoke 0.0.15.* □

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

*Proof.* As above, mindful of 0.0.17. □

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

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

Note that $x \le y \equiv x \dot{-} y = 0$ and invoke 0.0.20. Finally invoke Boolean closure and note that $x < y \equiv \neg y \le x$ while $x = y$ is equivalent to $x \le y \wedge y \le x$. □

# Bibliography

[Ded88] R. Dedekind, *Was sind und was sollen die Zahlen?*, Vieweg, Braunschweig, 1888, [In English translation by W.W. Beman; cf. [**?**]].

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

[Rog67] H. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.