0.1 Axt, Loop Program, and Grzegorczyk Hierarchies

Computable functions can have some quite complex definitions. For example, a loop programmable function might be given via a loop program that has depth of nesting of the loop-end pair, say, equal to 200. Now this *is* complex! Or a function might be given via an arbitrarily complex sequence of primitive recursions, with the restriction that the computed function is *majorized* by some known function, for all values of the input (for the concept of majorization see Subsection on the Ackermann function.).

But does such *definitional*—and therefore, "static"—complexity have any bearing on the *computational*—dynamic—complexity of the function? We will see that it does, and we will connect definitional and computational complexities quantitatively.

Our study will be restricted to the class $\mathscr{PR}$ that we will subdivide into an infinite sequence of increasingly more inclusive subclasses, S_i. A so-called *hierarchy* of classes of functions.

0.1.0.1 Definition. A sequence $(S_i)_{i\geq 0}$ of subsets of $\mathscr{PR}$ is a *primitive recursive hierarchy* provided all of the following hold:

(1) $S_i \subseteq S_{i+1}$, for all $i \geq 0$

(2) $\mathscr{PR} = \bigcup_{i\geq 0} S_i$.

The hierarchy is *proper* or *nontrivial* iff $S_i \neq S_{i+1}$, for all but finitely many i.

If $f \in S_i$ then we say that its *level in the hierarchy* is $\leq i$. If $f \in S_{i+1} - S_i$, then its level is equal to $i+1$. □

The first hierarchy that we will define is due to Axt and Heinermann [[5] and [1]].

0.1.0.2 Definition. (The Axt-Heinermann Hierarchy) We define the class $\mathscr{K}_n$ for each $n \geq 0$ by recursion on n. We let $\mathscr{K}_0$ stand for the closure of $\{\lambda x.x, \lambda x.x+1\}$ under substitution.

For $n \geq 0$, $\mathscr{K}_{n+1}$ is the closure under substitution of $\mathscr{K}_n \cup \{prim(h,g) : h \in \mathscr{K}_n \wedge g \in \mathscr{K}_n\}$, where $prim(h,g)$ is the function defined by primitive recursion from the basis function h and the iterator function g. □

Thus, primitive recursion is the "expensive" operation, an application of which takes us out of a given $\mathscr{K}_n$. On the other hand, as the classes are defined (the $n+1$ case), it follows that any finite number of substitution operations keeps us in the same class; all $\mathscr{K}_n$, that is, are closed under substitution.

We list a number of straightforward properties.

0.1.0.3 Proposition. *$(\mathscr{K}_n)_{n\geq 0}$ is a hierarchy, that is,*

(1) *$\mathscr{K}_n \subseteq \mathscr{K}_{n+1}$, for $n \geq 0$,*

and

(2) $\mathscr{PR} = \bigcup_{i\geq 0} \mathscr{K}_i$.

Proof.

(1) Immediate from the definition of $\mathscr{K}_{n+1}$ in 0.1.0.2.

(2) This is straightforward, from 0.1.0.2 and the inductive definition of$\mathscr{PR}$ —where we replace $\mathscr{I}$ by $\{\lambda x.x, \lambda x.x+1\}$ in the original definition, and replacing Comp by Grzegorczyk substitution. The part $\supseteq$ is rather trivial, while the $\subseteq$ part can be done by induction on $\mathscr{PR}$, showing that $\bigcup_{i\geq 0} \mathscr{K}_i$ contains the same initial functions as $\mathscr{PR}$ and is closed under Substitution and Prim. Recursion. □

0.1.0.4 Proposition. *$\lambda x.A_n(x) \in \mathscr{K}_n$, for all $n \geq 0$, where $\lambda nx.A_n(x)$ is the Ackermann function.*

Proof. Induction on n. For $n = 0$, we note that $A_0 = \lambda x.x+2 \in \mathscr{K}_0$. By 0.1.0.2, if $\lambda x.A_n(x) \in \mathscr{K}_n$, then $\lambda x.A_{n+1}(x) \in \mathscr{K}_{n+1}$—since $\lambda x.2 \in \mathscr{K}_0$ by substitution, and $\mathscr{K}_0 \subseteq \mathscr{K}_n$— and this concludes the induction. □

0.1.0.5 Proposition. *For every $f \in \mathscr{K}_n$ there is a $k \in \mathbb{N}$ such that $f(\vec{x}) \leq A_n^k\big(\max(\vec{x})\big)$, for all $\vec{x}$.*

Proof. We have proved that the Ackermann function majorises every primitive recursive function. The induction proof over $\mathscr{PR}$ demonstrated that composing finitely many functions f_i—each majorised by $A_n^{k_i}$ using the same fixed n—produces a function that is majorised by $A_n^{\sum_i k_i}$. That is, the index n does not increase through substitution.

Thus, in the present context, and to settle the proposition by induction on n, we will only need to show that every *initial* function of $\mathscr{K}_0$ is majorised by some A_0^r and each initial function of $\mathscr{K}_{n+1}$, namely,

$$\text{any } f \in \mathscr{K}_n \cup \{prim(h, g) : h \in \mathscr{K}_n \wedge g \in \mathscr{K}_n\} \tag{1}$$

is majorised by some appropriate A_{n+1}^r.

Well, each of x and $x + 1$ are less than $x + 2 = A_0(x)$ and this settles the basis. Assume the claim (I.H.) for $\mathscr{K}_n$—fixed $n \geq 0$—and tackle that for $\mathscr{K}_{n+1}$. By our plan, we need to show the initial function are majorised by some A_{n+1}^r.

For those $f \in \mathscr{K}_n$ [cf. (1)] this is the result of the I.H. on n and $A_n(x) \leq A_{n+1}(x)$ for all x. If now, $f = prim(h, g)$, then, by the I.H. on n, we have, for all x, z and $\vec{y}$,

$$h(\vec{y}) \leq A_n^{r_1}\big(\max(\vec{y})\big) \tag{1}$$

and

$$g(x, \vec{y}, z) \leq A_n^{r_2}\big(\max(x, \vec{y}, z)\big) \tag{2}$$

In our old proof —that any $f \in \mathscr{PR}$ is majorised by some A^l_m— recall that we relied on an intermediate result, namely, that (1) and (2) imply

$$f(x, \vec{y}) \leq A_n^{r_2 x + r_1}\Big(\max(x, \vec{y})\Big) < A_{n+1}\Big(r_2 x + r_1 + \max(x, \vec{y})\Big)$$

from which we concluded easily that we have some r such that $f(x, \vec{y}) \leq A^r_{n+1}\Big(\max(x, \vec{y})\Big)$, for all x and $\vec{y}$. □

0.1.0.6 Corollary. *The Axt-Heinermann hierarchy is proper.*

Proof. Indeed, $\lambda x.A_{n+1} \in \mathscr{K}_{n+1} - \mathscr{K}_n$, for all $n \geq 0$. By 0.1.0.4, we only need to see that $\lambda x.A_{n+1} \notin \mathscr{K}_n$. Indeed, otherwise, we would have, for all x, and some r, $A_{n+1}(x) \leq A^r_n(x)$ which contradicts $A^r_n(x) < A_{n+1}(x)$ a.e. with respect to x. □

We can also base the definition of classes similar to $\mathscr{K}_n$ on simultaneous recursion:

0.1.0.7 Definition. We define the class $\mathscr{K}^{sim}_n$ for each $n \geq 0$ by recursion on n. We let $\mathscr{K}^{sim}_0 = \mathscr{K}_0$.

For $n \geq 0$, $\mathscr{K}^{sim}_{n+1}$ is the *closure under substitution* of $\mathscr{K}^{sim}_n \cup \{f : f$ is obtained by simultaneous primitive recursion from functions in $\mathscr{K}^{sim}_n\}$. □

The following are straightforward.

0.1.0.8 Proposition. *For $n \geq 0$, we have $\mathscr{K}_n \subseteq \mathscr{K}^{sim}_n$.*

Thus, $\mathscr{PR} = \bigcup_{n\geq 0} \mathscr{K}_n \subseteq \bigcup_{n \geq 0} \mathscr{K}^{sim}_n \subseteq \mathscr{PR}$.

Thus, by 0.1.0.4,

0.1.0.9 Corollary. *For $n \geq 0$, we have $\lambda x.A_n(x) \in \mathscr{K}^{sim}_n$.*

0.1.0.10 Proposition. *For every $f \in \mathscr{K}^{sim}_n$ there is a $k \in \mathbb{N}$ such that $f(\vec{x}) \leq A^k_n\big(\max(\vec{x})\big)$, for all $\vec{x}$.*

Proof. A straightforward modification of the proof of 0.1.0.5. □

0.1.0.11 Corollary. *The $(\mathscr{K}^{sim}_n)_{n\geq 0}$ hierarchy is proper.*

Proof. Exactly as in the proof of 0.1.0.6. □

A closely related hierarchy—that is once again defined in terms of how complex a function's definition is—is based on loop programs [7].

0.1.0.12 Definition. (A Hierarchy of Loop Programs) We denote by L_0 the class of all loop programs that do not employ the **Loop-end** instruction pair.

Assuming that L_n has been defined, then L_{n+1} is the set of programs that is *the closure under program concatenation* of this initial set:

$$L_n \cup \Big\{\mathbf{Loop}X; P; \mathbf{end} : \text{for any variable } X \text{ and } P \in L_n\Big\} \qquad \square$$

Trivially, $L_n \subseteq L_{n+1}$ and the maximum nesting depth of the **Loop-end** pair increases by one as we pass from L_n to L_{n+1}. Of course, by virtue of $L_n \subseteq L_{n+1}$, not every $P \in L_{n+1}$ nests the **Loop-end** pair as deep as $n+1$. Thus, $R \in L_n$ iff the depth of nesting of the **Loop-end** instruction pair is at most n. Nesting depth equal to 0 means the absence of a **Loop-end** instruction pair.

The following is immediate.

0.1.0.13 Proposition. $(L_n)_{n \geq 0}$ *is a proper* L*-hierarchy. That is,*

(1) $L_n \subset L_{n+1}$, *for* $n \geq 0$

and

(2) $L = \bigcup_{n \geq 0} L_n$

We are more interested in the induced (by the L_n sets) hierarchy of primitive recursive classes:

0.1.0.14 Definition. We denote by $\mathscr{L}_n$, for $n \geq 0$, the class

$$\{P_{x_k}^{\vec{x}_r} : P \in L_n \wedge \text{ the } \vec{x}_r \text{ and } x_k \text{ occur in } P\} \qquad \square$$

0.1.0.15 Proposition. *For* $n \geq 0$*, we have that* $\mathscr{K}_n^{sim} = \mathscr{L}_n$.

Proof. In outline, the instruction pair **Loop-end** implements one simultaneous recursion. On the other hand, by the definition of $\mathscr{K}_n^{sim}$, this class contains functions obtained from those of $\mathscr{K}_0^{sim} = \mathscr{K}_0$ by n nested simultaneous recursions (and possibly some substitutions).

In detail, one can do induction on n and imitate the proofs of $\mathscr{PR} \subseteq \mathscr{L}$ and $\mathscr{L} \subseteq \mathscr{PR}$ that we have done in class. Briefly,

- By induction on n, note first that, trivially, $\mathscr{K}_0^{sim} = \mathscr{L}_0$. Taking the I.H. on n, we turn to the establishing $\mathscr{K}_{n+1}^{sim} \subseteq \mathscr{L}_{n+1}$. Well, assume we can program in L_n all the h_i and g_i, $i = 1, \ldots, n$, that are in $\mathscr{K}_n^{sim}$.

  Consider a simultaneous recursion that produces f_i (same i-range). They are by definition in $\mathscr{K}_{n+1}^{sim}$.

  We see, via pseudo code, that the f_i are in $\mathscr{L}_{n+1}^{sim}$ —establishing $\mathscr{K}_{n+1}^{sim} \subseteq \mathscr{L}_{n+1}$— by programming the latter, adding a single loop around the programs for the g_i: The variables F_i will eventually hold $f_i(a, \vec{y})$, where X

holds the value a initially.

$$
\begin{array}{l}
\quad F_1 = h_1(\vec{y}) \\
\quad\vdots \\
\quad F_n = h_n(\vec{y}) \\
\quad i = 0 \\
\textbf{Loop } X \\
\quad F_1 = g_1(i, \vec{y}, F_1, \ldots, F_n) \\
\quad F_2 = g_2(i, \vec{y}, F_1, \ldots, F_n) \\
\quad\vdots \\
\quad F_n = g_n(i, \vec{y}, F_1, \ldots, F_n) \\
\quad i = i + 1 \\
\textbf{end}
\end{array}
$$

- By induction on n, of the **program** hierarchy L_n. We have $\mathscr{K}_0^{sim} = \mathscr{L}_0$. Taking the I.H. that $\mathscr{L}_n \subseteq \mathscr{K}_n^{sim}$ we next show that $\mathscr{L}_{n+1} \subseteq \mathscr{K}_{n+1}^{sim}$. Assume that for a $P \in L_n$ we have that *all* P_Y are in $\mathscr{L}_n$. **This rephrases the I.H.**

  What about the functions that we compute by the L_{n+1} program, Q, below?

$$
\begin{array}{l}
\textbf{Loop } X \\
\qquad P \\
\textbf{end}
\end{array}
$$

  Well, our work in the Loop Program section showed that the above computes all functions obtained by a single simultaneous recursion on *all* the P_Y. Since by the I.H. all P_Y are in $\mathscr{K}_n^{sim}$, we have that all the Q_Y are in $\mathscr{K}_{n+1}^{sim}$, thus $\mathscr{L}_{n+1} \subseteq \mathscr{K}_{n+1}^{sim}$.

This proof ignored the trivial effects of substitution ($\mathscr{K}_{n+1}^{sim}$) and (equivalently) program concatenation (L_{n+1}). □

Thus, everything we said about the $(\mathscr{K}_n^{sim})_{n \geq 0}$ hierarchy carries over to the $(\mathscr{L}_n)_{n \geq 0}$ hierarchy—after all, it is the same hierarchy under two different definitions.

0.1.0.16 Proposition. *The $\mathscr{PR}$- (or $\mathscr{L}$-)hierarchy, $(\mathscr{L}_n)_{n \geq 0}$, is proper.*

0.1.0.17 Example. Here are some functions and predicates in the "lower" (small n) classes of the $(\mathscr{K}_n^{sim})_{n \geq 0}$ hierarchy.

The following are in $\mathscr{K}_1$ and hence in $\mathscr{K}_1^{sim} = \mathscr{L}_1$.

(1) $\lambda xy.x+y$. Indeed,

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

and $\lambda y.y$ and $\lambda z.z+1$ are in $\mathscr{K}_0 = \mathscr{K}_0^{sim}$.

(2) $\lambda xy.x(1 \dot{-} y)$. Indeed,

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

and $\lambda y.y$ and $\lambda z.0$ are in $\mathscr{K}_0 = \mathscr{K}_0^{sim}$.

(3) $\lambda x.1 \dot{-} x$. By substitution operations from the previous function.

(4) $\lambda x.x \dot{-} 1$. Indeed,

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

and $\lambda y.y$ and $\lambda z.0$ are in $\mathscr{K}_0 = \mathscr{K}_0^{sim}$.

(5) $\lambda x.\lfloor x/2 \rfloor \in \mathscr{K}_1^{sim}$.

This example shows that $\mathscr{K}_1 \neq \mathscr{K}_1^{sim}$, since $\lambda x.\lfloor x/2 \rfloor \notin \mathscr{K}_1$ as follows from results of [7] and [9] that were retold in [8].

(6) $switch = \lambda xyz.\text{if } x=0 \text{ then } y \text{ else } z$. Indeed, we have the recursion

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

where $\lambda y.y$ is in $\mathscr{K}_0 = \mathscr{K}_0^{sim}$.

The following are in $\mathscr{K}_2$ and hence in $\mathscr{K}_2^{sim} = \mathscr{L}_2$.

(a) $\lambda xy.x \dot{-} y$. Indeed,

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

and $\lambda y.y$ and $\lambda z.z \dot{-} 1$ are in $\mathscr{K}_1 \subseteq \mathscr{K}_1^{sim}$.

(b) $\lambda xy.xy$. Indeed,

$$\begin{aligned} x0 &= 0 \\ x(y+1) &= xy+x \end{aligned}$$

and $\lambda y.0$ and $\lambda wz.w+z$ are in $\mathscr{K}_1 \subseteq \mathscr{K}_1^{sim}$.

(c) $\lambda x.2^x$. Indeed,

$$2^0 = 1$$
$$2^{y+1} = 2^y + 2^y$$

and $\lambda y.1$ and $\lambda wz.w + z$ are in $\mathscr{K}_1 \subseteq \mathscr{K}_1^{sim}$. □

0.1.0.18 Definition. As is usual, the predicate classes $\mathscr{K}_{n,*}$ and $\mathscr{K}_{n,*}^{sim}$—the latter being the same as $\mathscr{L}_{n,*}$—are defined for all $n \geq 0$ as $\{f(\vec{x}) = 0 : f \in \mathscr{K}_n\}$ and $\{f(\vec{x}) = 0 : f \in \mathscr{K}_n^{sim}\}$, respectively. □

0.1.0.19 Proposition. *For $n \geq 1$, we have that $\mathscr{K}_{n,*}$ and $\mathscr{K}_{n,*}^{sim}$ are closed under $\neg$ and $\vee$—and hence under $\wedge, \rightarrow$, and $\equiv$ as well.*

Proof. Let $Q(\vec{x}) \in \mathscr{K}_{n,*}$. Then, for some $q \in \mathscr{K}_n$, $Q(\vec{x}) \equiv q(\vec{x}) = 0$. Since $r = \lambda\vec{x}.1 \dot{-} q(\vec{x}) \in \mathscr{K}_n$ if $n \geq 1$ by 0.1.0.17, we are done, noting $\neg Q(\vec{x}) \equiv r(\vec{x}) = 0$. Next, let also $S(\vec{y}) \equiv s(\vec{y}) = 0$ with $s \in \mathscr{K}_n$. Then $Q(\vec{x}) \vee S(\vec{y}) \equiv switch(q(\vec{x}), 0, r(\vec{y})) = 0$; but $switch \in \mathscr{K}_n$, for $n \geq 1$ (cf. 0.1.0.17).

The cases for $\mathscr{K}_{n,*}^{sim}$ are argued identically with the preceding two. □

0.1.0.20 Corollary. *The relations $\lambda x.x \leq a$, $\lambda x.x < a$ and $\lambda x.x = a$ are in $\mathscr{K}_{1,*}$ and hence in $\mathscr{K}_{1,*}^{sim}$.*

Proof. By 0.1.0.17(4) and substitution, we have that $\lambda x.x \dot{-} a \in \mathscr{K}_1$. But $x \leq a \equiv x \dot{-} a = 0$. On the other hand, $x < a \equiv x + 1 \dot{-} a = 0$. Thus the claim about $\lambda x.x < a$ is true. Noting that $\lambda x.a \leq x$ is in $\mathscr{K}_{1,*}$ due to

$$a \leq x \equiv \neg x < a$$

and 0.1.0.19, we have that $\lambda x.x = a$ is in $\mathscr{K}_{1,*}$ by 0.1.0.19 and the observation $x = a \equiv x \leq a \wedge a \leq x$. □

0.1.0.21 Proposition. *For $n \geq 1$, we have that $\mathscr{K}_n$ and $\mathscr{K}_n^{sim}$ are closed under definition by cases.*

Proof. This is immediate from either of the suggested proofs for definition-by-cases, noting 0.1.0.17, (1), (2) and (6). □

The three hierarchies that we introduced include increasingly complex classes, using as a yardstick of complexity the nesting depth of primitive recursion. The next hierarchy, due to [2], gauges *complexity of definition* by the (numerical) size of the function it produces—and, correspondingly, the class complexity at level n by the size of the functions it contains. As the definition does *not necessarily* force a function such as $prim(h, g)$ to exit from a given level, the Grzegorczyk hierarchy is much more amenable to mathematical analysis.

0.1.0.22 Definition. (The Grzegorczyk Hierarchy) We are given a fixed sequence of functions, $(g_n)_{n\ge 0}$ by

$$g_0 = \lambda x.x+1$$
$$g_1 = \lambda xy.x+y$$
$$g_2 = \lambda xy.xy$$

and, for $n \ge 2$,

$$g_{n+1} = \lambda xy.A_n\big(\max(x,y)\big)$$

where $\lambda ny.A_n(x)$ is the Ackermann function that we studied earlier.

The hierarchy $(\mathscr{E}^n)_{n\ge 0}$ is defined as follows: $\mathscr{E}^n$ is the closure of

$$\{\lambda x.x+1, \lambda x.x, g_n\}$$

under *substitution* and *bounded primitive recursion*, the latter being the schema below

$$f(0,\vec{y}) = h(\vec{y})$$
$$f(x+1,\vec{y}) = q\big(x,\vec{y}, f(x,\vec{y})\big)$$
$$f(x,\vec{y}) \le B(x,\vec{y})$$

where h, q and B are given functions. □

A class $\mathscr{C}$ is closed under bounded primitive recursion iff whenever h, q, and B are in $\mathscr{C}$, then so is the f produced as above.

We note that the bounded recursion is an ordinary number-theoretic primitive recursion along with a condition that the function f has actually been "produced" *only if* its values are bounded everywhere by those of the *given* B.

The g_n-function included among the initial functions at each level, which gauges the (numerical) size of functions included in each $\mathscr{E}^n$ is (a version of) the Ackermann function. Grzegorczyk used a different version than we do here. Our choice to use the function due to Robert Ritchie was partly dictated by ease-of-use considerations, but mostly because we know quite a bit about the A_n already. The reader may consult [8] to read a proof that the version we use here produces the same $\mathscr{E}^n$ classes as in [2].

The class of relations at level n of the Grzegorczyk hierarchy is defined as usual.

0.1.0.23 Definition. $\mathscr{E}^n_*$, for $n \ge 0$, denotes the class of relations $\{f(\vec{x}) = 0 : f \in \mathscr{E}^n\}$. □

0.1.0.24 Example. Here are some examples of functions and relations in $\mathscr{E}^0$ and $\mathscr{E}^0_*$:

(1) $\lambda xy.x(1 \dot{-} y)$.

$$\begin{cases} x(1 \dot{-} 0) = x \\ x(1 \dot{-} (y+1)) = 0 \\ x(1 \dot{-} y) \le x \end{cases}$$

(2) $\lambda x.1 \dot{-} x$. By (1) and substitution.

(3) $\lambda x.x \dot{-} 1$.

$$\begin{cases} 0 \dot{-} 1 = 0 \\ (x+1) \dot{-} 1 = x \\ x \dot{-} 1 \le x \end{cases}$$

(4) $\lambda xy.x \dot{-} y$.

$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 \\ x \dot{-} y \le x \end{cases}$$

(5) $\lambda xy.x \le y$ and $\lambda xy.x < y$ are in $\mathscr{E}^0_*$. Indeed, $x \le y \equiv x \dot{-} y = 0$ and $x < y \equiv (x+1) \dot{-} y = 0$. □

0.1.0.25 Lemma. *For all $n \ge 0$, $\mathscr{E}^0 \subseteq \mathscr{E}^n$.*

Proof. $\mathscr{E}^n$ contains the initial functions of $\mathscr{E}^0$ and is closed under the same operations. □

0.1.0.26 Theorem. *For $n \ge 0$, $\mathscr{E}^n_*$ is closed under Boolean operations and also under bounded quantification, namely, $(\exists y)_{<z}$, $(\exists y)_{\le z}$, $(\forall y)_{<z}$, $(\forall y)_{\le z}$.*

Proof. We implicitly use 0.1.0.25. For Boolean operations it suffices to consider $\neg$ and $\vee$ only. So, let $R(\vec{x}) \equiv r(\vec{x}) = 0$ and $Q(\vec{y}) \equiv q(\vec{y}) = 0$, where r and q are in $\mathscr{E}^n$. Now, $\neg R(\vec{x}) \equiv 1 \dot{-} r(\vec{x}) = 0$ and we are done by 0.1.0.24(2). On the other hand, $R(\vec{x}) \vee Q(\vec{y}) \equiv r(\vec{x})\big(1 \dot{-} (1 \dot{-} q(\vec{y}))\big) = 0$ and we are done by 0.1.0.24(1).

For closure under bounded quantification, let $P(y, \vec{x}) \equiv p(y, \vec{x}) = 0$, where $p \in \mathscr{E}^n$. Let $\chi_\exists$ be the characteristic function of $(\exists y)_{<z} P(y, \vec{x})$. Noting that

$$(\exists y)_{<0} P(y, \vec{x}) \text{ is false, and } (\exists y)_{<z+1} P(y, \vec{x}) \equiv P(z, \vec{x}) \vee (\exists y)_{<z} P(y, \vec{x})$$

we have that $\chi_\exists$ satisfies the bounded recursion below:

$$\begin{cases} \chi_\exists(0, \vec{x}) = 1 \\ \chi_\exists(z+1, \vec{x}) = \chi_\exists(z, \vec{x})\Big(1 \dot{-} \big(1 \dot{-} p(z, \vec{x})\big)\Big) \\ \chi_\exists(z, \vec{x}) \le 1 \end{cases}$$

and we are done. The "1" in the inequality above is the output of $\lambda x.1$ which is in $\mathscr{E}^0$. Clearly $\chi_\exists$ belongs where p does, and $(\exists y)_{<z} P(y, \vec{x}) \equiv \chi_\exists(z, \vec{x}) = 0$.

To conclude the proof for the remaining cases of quantification, note that $(\exists y)_{\le z} R \equiv R \vee (\exists y)_{<z} R$; moreover, the universal quantifier cases follow from the closure of $\mathscr{E}_*^n$ under negation. □

The following result is, modulo choice of Ackermann function, from [2].

0.1.0.27 Lemma. (Bounding Lemma) (1) *For each $f \in \mathscr{E}^0$, there are i and k such that $f(\vec{x}) \le x_i + k$ everywhere.*

(2) *For each $f \in \mathscr{E}^1$, there are C and k such that $f(\vec{x}) \le C \max(\vec{x}) + k$ everywhere.*

(3) *For each $f \in \mathscr{E}^2$, there are C, n, and k such that $f(\vec{x}) \le C \max(\vec{x})^n + k$ everywhere.*

(4) *For each $f \in \mathscr{E}^{n+1}$, $n \ge 2$, there is a k such that $f(\vec{x}) \le A_n^k\big(\max(\vec{x})\big)$ everywhere.*

Proof.

All proofs are by induction over the appropriate $\mathscr{E}^n$.

(1) The claim trivially holds for the initial functions and propagates with bounded recursion since the I.H. applies to whichever bounding function B was employed. Consider the substitution, using g and h in $\mathscr{E}^0$.

$$\begin{array}{c} g(\vec{w},\quad x \quad, \vec{z}) \\ \uparrow \\ h(\vec{y}) \end{array}$$

By I.H. on h we have $h(\vec{y}) \le y_i + k$, for all $\vec{y}$.

By I.H. on g we have one of

- $g(\vec{w}, x, \vec{z}) \le x + l$, for all $\vec{w}, x, \vec{z}$, thus, $g(\vec{w}, h(\vec{y}), \vec{z}) \le y_i + k + l$, for all $\vec{w}, \vec{y}, \vec{z}$.
- $g(\vec{w}, x, \vec{z}) \le w_j + l'$, for all $\vec{w}, x, \vec{z}$, thus, $g(\vec{w}, h(\vec{y}), \vec{z}) \le w_j + l'$, for all $\vec{w}, \vec{y}, \vec{z}$.
- $g(\vec{w}, x, \vec{z}) \le z_m + l''$, for all $\vec{w}, x, \vec{z}$, thus, $g(\vec{w}, h(\vec{y}), \vec{z}) \le z_m + l''$, for all $\vec{w}, \vec{y}, \vec{z}$.

(2) The basis and the propagation of the claim with bounded recursion are as above [note, incidentally, that $x + y \le 2\max(x, y)$]. Let us now look at a substitution $h(\vec{y}, g(\vec{x}), \vec{z})$. We have, by the I.H. applied to h,

$$\begin{aligned} h(\vec{y}, g(\vec{x}), \vec{z}) &\le C \max(\vec{y}, g(\vec{x}), \vec{z}) + k \\ &\overset{\text{I.H. for } g}{\le} C \max(\vec{y}, C' \max(\vec{x}) + k', \vec{z}) + k \\ &\le CC' \max(\vec{y}, \vec{x}, \vec{z}) + Ck' + k \end{aligned}$$

(3) Left as an exercise.

(4) The claim is true for the initial functions and propagates with bounded recursion for the reason named earlier. As for substitution, we know that the subscript n will not change and thus if $A_n^{k_i}$ majorize the component-functions of the substitution, then $A_n^{\sum k_i}$ majorizes the result (to say this briefly we overkilled the exponent). □

We can now prove that $\mathscr{E}^n \subset \mathscr{E}^{n+1}$ for all n.

0.1.0.28 Theorem. $(\mathscr{E}^n)_{n\geq 0}$ *is a proper primitive recursive hierarchy.*

Proof. First, $\mathscr{E}^n \subseteq \mathscr{E}^{n+1}$, for all n, since every bounded recursion in $\mathscr{E}^n$ can use as bounding functions the bounds from $\mathscr{E}^{n+1}$ and thus is a bounded recursion in $\mathscr{E}^{n+1}$ too. Thus, for $\mathscr{E}^0 \subseteq \mathscr{E}^1$ use $C\max(\vec{x})+k$, for $\mathscr{E}^1 \subseteq \mathscr{E}^2$ use $C\max(\vec{x})^r+k$, and for $\mathscr{E}^n \subseteq \mathscr{E}^{n+1}$, for $n \geq 2$, use use A_n^k and the facts that $A_n^k \in \mathscr{E}^{n+1}$ and

$$A_0(x) \leq A_1(x) \leq A_2(x) \leq \ldots A_{n-1}(x) \leq A_n(x) \leq \ldots$$

I am implying an induction over $\mathscr{E}^n$ in the above argument, that shows $\mathscr{E}^n \subseteq \mathscr{E}^{n+1}$. But this requires the initial A_{n-1} of $\mathscr{E}^n$ to be in $\mathscr{E}^{n+1}$. Is it? Yes, if we assume that A_{n-2} is: Induction on n!

Reverting to the unified notation "g_n" and noting that $g_{n+1} \in \mathscr{E}^{n+1} - \mathscr{E}^n$ by 0.1.0.27, we promote $\subseteq$ above to $\subset$:

$$\mathscr{E}^n \subset \mathscr{E}^{n+1}, \text{ for all } n.$$

Now, trivially, $\mathscr{E}^n \subseteq \mathscr{PR}$, for all n. On the other hand, every primitive recursion is a bounded recursion with bounding function A_n^k for some k, so $\mathscr{PR} \subseteq \bigcup_{n\geq 0} \mathscr{E}^n$ as well. □

0.1.0.29 Exercise. In view of 0.1.0.27, prove that *switch* (the "full" if-then-else) and max are *not* in $\mathscr{E}^0$. □

We defined bounded summation and multiplication and saw that, as operations, they do not take us out of $\mathscr{PR}$. More interesting is this:

0.1.0.30 Proposition. *For* $n \geq 2$, $\mathscr{E}^n$ *is closed under bounded summation.*

Proof. We only need a bounding function for $\sum_{i<z} f(i,\vec{x})$ in $\mathscr{E}^n$.

For $n = 2$, $f(i,\vec{x}) = O(\max(i,\vec{x})^r)$, for some r, due to 0.1.0.27. But then,

$$\sum_{i<z} f(i,\vec{x}) = \sum_{i<z} O(\max(i,\vec{x})^r) = O(z\max(z,\vec{x})^r)$$

Since, for any constants C and D, $\lambda z\vec{x}.Cz\max(z,\vec{x})^r + D$ is in $\mathscr{E}^2$, our bounding function is obtained by choosing the right C and D.

For $n > 2$, let, by 0.1.0.27, r be such that $f(i,\vec{x}) \le A^r_{n-1}(\max(i,\vec{x}))$, for all $i,\vec{x}$. Then

$$\sum_{i<z} f(i,\vec{x}) \le \sum_{i<z} A^r_{n-1}\big(\max(i,\vec{x})\big) \le zA^r_{n-1}\big(\max(z,\vec{x})\big) \tag{1}$$

But $\lambda xy.xy$ and $\lambda z\vec{x}.A^k_{n-1}\big(\max(z,\vec{x})\big)$ are in $\mathscr{E}^n$ for $n > 2$. We have obtained the required bounding function in (1). □

A definition of *bounded search* that is used in [2] [cf. also [6]] is the following:

0.1.0.31 Definition. (Alternative Bounded Search) For any total number-theoretic function $\lambda y\vec{x}.f(y,\vec{x})$ we define

$$(\overset{\circ}{\mu} y)_{<z} f(y,\vec{x}) \overset{Def}{=} \begin{cases} \min\{y : y < z \wedge f(y,\vec{x}) = 0\} & \text{if } (\exists y)_{<z} f(y,\vec{x}) = 0 \\ 0 & \text{otherwise} \end{cases}$$

$(\overset{\circ}{\mu} y)_{\le z} f(y,\vec{x})$ means $(\overset{\circ}{\mu} y)_{<z+1} f(y,\vec{x})$, and $(\overset{\circ}{\mu} y)_{<z} R(y,\vec{x})$ means $(\overset{\circ}{\mu} y)_{<z} \chi_R(y,\vec{x})$, where χ_R is the characteristic function of R. □

0.1.0.32 Theorem. *For $n \ge 0$, $\mathscr{E}^n$ is closed under $(\overset{\circ}{\mu} y)_{<z}$.*

Proof. Let $f \in \mathscr{E}^n$. We set $g(z,\vec{x}) = (\overset{\circ}{\mu} y)_{<z} f(y,\vec{x})$. Notice that

$$\begin{cases} g(0,\vec{x}) = & 0 \\ g(z+1,\vec{x}) = & \text{if } (\exists y)_{<z} f(y,\vec{x}) = 0 \text{ then } g(z,\vec{x}) \\ & \text{else if } f(z,\vec{x}) = 0 \text{ then } z \text{ else } 0 \\ g(z,\vec{x}) & \le z \end{cases}$$

The above bounded recursion works for $n \ge 1$, but will not work for $n = 0$ due to 0.1.0.29; some acrobatics will be necessary:

We note that the right hand side of the second equation is obtained by substituting $g(z,\vec{x})$ into the "recursive call slot" w, making the iterator function of the recursion be

$$\begin{cases} It(x,w,z) = & \text{if } x = 0 \text{ then } w \\ & \text{else } \Big(1 \dot{-} f(z,\vec{x})\Big)z \end{cases}$$

where $\chi(z,\vec{x})$—the value at $(z,\vec{x})$ of the characteristic function of $(\exists y)_{<z} f(y,\vec{x}) = 0$—goes into x in It, while the recursive call goes in w.

The *apparent* problem is the two possible independent outputs, w and z that make $It \notin \mathscr{E}^0$. Well, "apparent" is the operative word. In this context, whatever gets into w (that is, $g(z,\vec{x})$) is $\le z$ (in fact, $< z$) so the new iterator $\widetilde{It}$ below works equally well with It toward defining g, **and** does **not** have this apparent problem!

$$\begin{cases} \widetilde{It}(x,w,z) = & \text{if } x = 0 \text{ then } \Big(1 \dot{-} (w \dot{-} z)\Big)w \\ & \text{else } \Big(1 \dot{-} f(z,\vec{x})\Big)z \end{cases}$$

Indeed, $\widetilde{It} \in \mathscr{E}^0$, since

$$\begin{cases} \widetilde{It}(0,w,z) = & \Big(1 \dot{-} (w \dot{-} z)\Big)w \\ \widetilde{It}(x+1,w,z) = & \Big(1 \dot{-} f(z,\vec{x})\Big)z \\ \widetilde{It}(x,w,z) & \le z \end{cases}$$

□

The absence of the full switch from $\mathscr{E}^0$ restricts the result about closure under definition by cases:

0.1.0.33 Corollary. *For $n \ge 1$, $\mathscr{E}^n$ is closed under definition by cases.*

$\mathscr{E}^0$ is closed under definition by cases provided the produced function f satisfies $f(\vec{x}) \le x_i + k$ everywhere, for some i and k.

Proof. For $n \ge 1$ the usual proof works. For $\mathscr{E}^0$, if f is given as by-cases from f_i and R_i, where the f_i are in $\mathscr{E}^0$ and the R_i in $\mathscr{E}^0_*$, then

$$f(\vec{x}) = (\mathring{\mu} y)_{\le x_i+k}\Big(y = f_1(\vec{x}) \wedge R_1(\vec{x}) \vee \ldots \vee y = f_{n+1}(\vec{x}) \wedge R_{n+1}(\vec{x})\Big) \qquad (1)$$

where we wrote R_{n+1} for the "otherwise" relation. The reader should carefully identify all the results that we proved so far about the Grzegorczyk classes that make (1) work. □

0.1.0.34 Theorem. *$\mathscr{E}^2$ is closed under* simultaneous bounded recursion*, where, additionally to the standard schema, k bounding functions B_i, for $i = 1, \ldots, k$, are given, and the functions f_i resulting from the schema* must *satisfy $f_i(x,\vec{y}) \le B_i(x,\vec{y})$ everywhere.*

Proof. Consider the schema below, where the h_i, g_i and B_i are in $\mathscr{E}^2$.

$$\begin{cases} f_1(0,\vec{y}_n) & = h_1(\vec{y}_n) \\ \vdots & \\ f_k(0,\vec{y}_n) & = h_k(\vec{y}_n) \\ f_1(x+1,\vec{y}_n) & = g_1(x,\vec{y}_n, f_1(x,\vec{y}_n), \ldots, f_k(x,\vec{y}_n)) \\ \vdots & \\ f_k(x+1,\vec{y}_n) & = g_k(x,\vec{y}_n, f_1(x,\vec{y}_n), \ldots, f_k(x,\vec{y}_n)) \\ f_1(x,\vec{y}_n) & \le B_1(x,\vec{y}_n) \\ \vdots & \\ f_k(x,\vec{y}_n) & \le B_k(x,\vec{y}_n) \end{cases} \qquad (1)$$

The pairing function $J = \lambda xy.(x+y)^2 + x$ is in $\mathscr{E}^2$, and so are its projections $K = \lambda z.(\mathring{\mu} x)_{\le z}(\exists y)_{\le z} J(x,y) = z$ and $L = \lambda z.(\mathring{\mu} y)_{\le z}(\exists x)_{\le z} J(x,y) = z$. Thus, we have the coding-decoding scheme—$\lambda\vec{z}_k.[\![\, z_1, \ldots, z_k \,]\!]^{(k)}$ and Π^k_i—in $\mathscr{E}^2$, where, by recursion on k, we define

$$[\![\, z_1, \ldots, z_k \,]\!]^{(k)} = \begin{cases} z_1 & \text{if } k = 1 \\ J\Big([\![\, z_1, \ldots, z_{k-1} \,]\!]^{(k-1)}, z_k \Big) & \text{if } k > 1 \end{cases} \tag{1}$$

The role of the Π^k_i is to decode numbers of the form $[\![\, z_1, \ldots, z_k \,]\!]^{(k)}$, thus, they must satisfy, for $1 \le i \le k$,

$$\Pi^k_i\Big([\![\, z_1, \ldots, z_k \,]\!]^{(k)} \Big) = z_i$$

In terms of the K and L, the Π^k_i are expressible as follows (Exercise!):

$$\text{For } k \ge 2,\ \Pi^k_i = \begin{cases} LK^{k-i} & \text{if } 2 \le i \le k \\ K^{k-1} & \text{if } i = 1 \end{cases} \tag{2}$$

(1) and (2) confirm the claim "$\lambda\vec{z}_k.[\![\, z_1, \ldots, z_k \,]\!]^{(k)}$ and Π^k_i are in $\mathscr{E}^2$", which we made above. The Hilbert-Bernays proof of how to simulate a simultaneous recursion by a single recursion goes through unchanged if we replace the originally used prime power coding/decoding by the alternative $[\![\ldots]\!]/\Pi^k_i$ adopted here. Noting that

$$[\![\, f_1(x, \vec{y}_n), \ldots, f_k(x, \vec{y}_n) \,]\!]^{(k)} \le [\![\, B_1(x, \vec{y}_n), \ldots, B_k(x, \vec{y}_n) \,]\!]^{(k)}$$

and that the right hand side of the above $\le$ is in $\mathscr{E}^2$ (as a function of $x, \vec{y}_n$) by substitution, we obtain that

$$\lambda x\vec{y}_n.[\![\, f_1(x, \vec{y}_n), \ldots, f_k(x, \vec{y}_n) \,]\!]^{(k)} \in \mathscr{E}^2$$

and therefore, for $i = 1, \ldots, k$, $f_i = \lambda x\vec{y}_n.\Pi^k_i\big([\![\, f_1(x, \vec{y}_n), \ldots, f_k(x, \vec{y}_n) \,]\!]^{(k)} \big)$ is in $\mathscr{E}^2$. □

0.1.0.35 Corollary. *$\mathscr{E}^n$, for $n \ge 2$, is closed under simultaneous bounded recursion.*

We have introduced four primitive recursive hierarchies—of Axt-Hienermann, Dennis Ritchie, and Grzegorczyk—the yardstick of "complexity" of a class at each level n being that of its *definition*, whether the measure was *numerical size* of produced functions (Grzegorczyk) or *nesting depth* of primitive recursion (in all the others).

We conclude this subsection by showing that this *definitional complexity* tracks very accurately the *computational complexity* of the primitive recursive functions. *The URM formalism will be the computing model to which the computational complexity will related.*

The "main lemma" toward connecting the four hierarchies to each other on one hand, and with the computational complexity of their functions on the other, will be the *Ritchie*[*]*-Cobham property* of the Grzegorczyk classes, that

$$\text{for } n \geq 0,\ f \in \mathscr{E}^n \text{ iff } f \text{ is computable by some URM within time } t \in \mathscr{E}^n \qquad (RC)$$

We will need a *simulation tool*, namely, we will show that the *computation* of a URM can be simulated by a very simple simultaneous primitive recursion. The reader should review the yields operation that connects successive IDs in a computation.

Important! Unlike much practice in theory of algorithms, where run time is expressed as a function of input *length*, in the present section we *will gauge run time as function of input (numerical) value.*

Thus, for the record:

0.1.0.36 Definition. Consider the function $f = M_{\mathbf{y}}^{\vec{\mathbf{x}}_n}$, where M is a URM—whether M is normalized or not is immaterial for the purpose of this definition. A function $\lambda \vec{x}_n.t(\vec{x}_n)$ *majorizes* the run time complexity of $M_{\mathbf{y}}^{\vec{\mathbf{x}}_n}$ iff, for all $\vec{a}_n$, if $f(\vec{a}_n)\downarrow$ with an M-computation of length l, then $l \leq t(\vec{a}_n)$; else if $f(\vec{a}_n)\uparrow$, then also $t(\vec{a}_n)\uparrow$.

We say that $\lambda \vec{x}_n.f(\vec{x}_n)$ is *computable within time* $\lambda \vec{x}_n.t(\vec{x}_n)$. □

0.1.0.37 Simulation lemma. *Let M be a normalized URM with variables $V_1, V_2, \ldots V_{n+1}, V_{n+2}, \ldots, V_m$, of which V_1 is the output variable while the V_i, for $i = 2, \ldots, n+1$, are input variables. With reference to the yields operation between IDs, we define $m+1$* simulating functions*—for all $y, \vec{a}_n$—as follows:*

$v_i(y, \vec{a}_n)$ = *the value of variable V_i in the y-th ID of a (<u>possibly non terminating</u>) computation with input $\vec{a}_n$*

$I(y, \vec{a}_n)$ = *instruction number in the y-th ID of a (<u>possibly non terminating</u>) computation with input $\vec{a}_n$*

All the simulating functions are in $\mathscr{K}_2{}^{sim}$.

All the simulating functions are total, since once the instruction **stop** is reached the computation continues forever "trivially", that is, without changing either the V_i or the instruction number.

Proof. We have the following simultaneous recursion that defines the simulating functions:

$$\begin{aligned}
v_1(0, \vec{a}_n) &= 0 \\
v_i(0, \vec{a}_n) &= a_{i-1}, \text{ for } i = 2, \ldots, n+1 \\
v_i(0, \vec{a}_n) &= 0, \text{ for } i = n+2, \ldots, m \\
I(0, \vec{a}_n) &= 1
\end{aligned}$$

[*]Dennis Ritchie.

For $y \geq 0$ and $i = 1, \ldots, m$, we have

$$v_i(y{+}1, \vec{a}_n) = \begin{cases} c & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : V_i \leftarrow c\text{" is in } M \\ v_i(y, \vec{a}_n) + 1 & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : V_i \leftarrow V_i + 1\text{" is in } M \\ v_i(y, \vec{a}_n) \dot{-} 1 & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : V_i \leftarrow V_i \dot{-} 1\text{" is in } M \\ v_i(y, \vec{a}_n) & \text{otherwise} \end{cases}$$

$$I(y+1, \vec{a}_n) = \begin{cases} l_1 & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : \textbf{if } V_i = 0 \textbf{ goto } l_1 \textbf{ else} \\ & \textbf{goto } l_2\text{" is in } M \text{ and } v_i(y, \vec{a}_n) = 0 \\ l_2 & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : \textbf{if } V_i = 0 \textbf{ goto } l_1 \textbf{ else} \\ & \textbf{goto } l_2\text{" is in } M \text{ and } v_i(y, \vec{a}_n) > 0 \\ k & \text{if } I(y, \vec{a}_n) = k \text{ where "}k : \textbf{stop}\text{" is in } M \\ I(y, \vec{a}_n) + 1 & \text{otherwise} \end{cases}$$

Since the iterator functions only utilize the functions $\lambda x.a$, $\lambda x.x + 1$, $\lambda x.x \dot{-} 1$, $\lambda x.x$, and predicates $\lambda x.x = a$, and $\lambda x.x > a$—all in $\mathscr{K}_1^{sim}$ and $\mathscr{K}_{1,*}^{sim}$—it follows that all the simulating functions are in $\mathscr{K}_2^{sim}$, as claimed. □

0.1.0.38 Example. Let M be the program below

$1 : V_1 \leftarrow V_1 + 1$
$2 : V_2 \leftarrow V_2 \dot{-} 1$
$3 :$ **if** $V_2 = 0$ **goto** 4 **else goto** 1
$4 :$ **stop**

Let us assume that V_2 is the input variable and V_1 is the output variable. The simulating equations take the concrete form below, where a denotes the input value:

$$v_1(0, a) = 0$$
$$v_2(0, a) = a$$

For $y \geq 0$ we have

$$v_1(y+1, a) = \begin{cases} v_1(y, a) + 1 & \text{if } I(y, a) = 1 \\ v_1(y, a) & \text{otherwise} \end{cases}$$

$$v_2(y+1, a) = \begin{cases} v_2(y, a) \dot{-} 1 & \text{if } I(y, a) = 2 \\ v_2(y, a) & \text{otherwise} \end{cases}$$

$$I(y+1, a) = \begin{cases} 4 & \text{if } I(y, a) = 3 \wedge v_2(y, a) = 0 \\ 1 & \text{if } I(y, a) = 3 \wedge v_2(y, a) > 0 \\ 4 & \text{if } I(y, a) = 4 \\ I(y, a) + 1 & \text{otherwise} \end{cases}$$

□

0.1.0.39 Corollary. *The simulating functions are in $\mathscr{K}_4$.*

Proof. The above mentioned predicates and functions that are part of the iterator are in $\mathscr{K}_1$ and $\mathscr{K}_{1,*}$. Moreover, $\mathscr{K}_1$ is closed under definition by cases (0.1.0.21). To convert the simultaneous recursion to a single recursion and back, we need pairing functions and their projections.

The quadratic pairing function $J = \lambda xy.(x+y)^2 + x$ is appropriate. Immediately, $J \in \mathscr{K}_2$ by 0.1.0.17. Now, let us place its projections, K and L, in the Axt hierarchy. We know from class/text that $Kz = z \dot{-} \lfloor\sqrt{z}\rfloor^2$ and $Lz = \lfloor\sqrt{z}\rfloor \dot{-} Kz$. By the results of 0.1.0.17 we need only locate $\lambda z.\lfloor\sqrt{z}\rfloor$ in the hierarchy.

We start by noting that if $z+1$ is a perfect square, that is, $z+1 = (k+1)^2$ for some k, then $z+1 = k^2+2k+1$ hence $z = k^2+2k$, thus

$$k^2 \le z < (k+1)^2$$

hence $k = \lfloor\sqrt{z}\rfloor$. This yields

$$\lfloor\sqrt{z+1}\rfloor = k+1 = \lfloor\sqrt{z}\rfloor + 1 \tag{1}$$

Suppose next that $z+1$ is *not* a perfect square. That is,

$$m^2 < z+1 < (m+1)^2 \tag{2}$$

for some m, and hence $m^2 \le z < (m+1)^2$. This entails $m \le \sqrt{z} < m+1$, thus $m = \lfloor\sqrt{z}\rfloor$. But $m = \lfloor\sqrt{z+1}\rfloor$ as well, by (2).

At the end of all this we obtain the following recursion:

$$\begin{cases} \lfloor\sqrt{0}\rfloor & = 0 \\ \lfloor\sqrt{z+1}\rfloor & = \begin{cases} \lfloor\sqrt{z}\rfloor + 1 & \text{if } z+1 = (\lfloor\sqrt{z}\rfloor+1)^2 \\ \lfloor\sqrt{z}\rfloor & \text{otherwise} \end{cases} \end{cases}$$

By reference to 0.1.0.17—and noting that $x = y \equiv (x \dot{-} y) + (y \dot{-} x) = 0$, thus $\lambda xy.x = y \in \mathscr{K}_{2,*}$—we see that $\lambda z.\lfloor\sqrt{z}\rfloor \in \mathscr{K}_3$, and thus so are K and L. But then, the coding/decoding scheme that is based on this J, K, L is in $\mathscr{K}_3$.

Referring back to our proof of the Hilbert-Bernays theorem, you will recall that —translating the technique from $\langle \ldots \rangle$-coding to $[\![\ldots]\!]$-coding— the coded iteration-part of the simultaneous recursion that we be captured in our prime-power coding method as

$$F(y+1,\vec{a}) = \Big\langle \ldots, g_i\Big(y, \vec{a}, \big(F(y,\vec{a})\big)_0, \ldots, \big(F(y,\vec{a})\big)_m\Big), \ldots \Big\rangle$$

where (*in the present context*)

$$\big(F(y,\vec{a})\big)_0 = I(y,\vec{a}), \text{ and, for } i = 1,\ldots,m, \big(F(y,\vec{a})\big)_i = v_i(y,\vec{a})$$

here becomes

$$F(y+1,\vec{a}) = [\![\ldots, g_i\Big(y, \vec{a}, \Pi_1^{m+1}(F(y,\vec{a})), \ldots, \Pi_{m+1}^{m+1}(F(y,\vec{a}))\Big), \ldots]\!]^{(m+1)} \quad (3)$$

where

$$\Pi_1^{m+1}(F(y,\vec{a})) = I(y,\vec{a}), \text{ and, for } i=2,\ldots,m+1, \Pi_i^{m+1}(F(y,\vec{a})) = v_i(y,\vec{a})$$

Thus, the presence of the Π_i^{m+1} in the iterator part (3), causes $F \in \mathscr{K}_4$ since K, L are in $\mathscr{K}_3$, and thus so are the Π_i^{m+1}.

Therefore, the recursion that simulates the simultaneous recursion of the simulation lemma yields the function

$$F = \lambda y \vec{a}_n . [\![\, I(y,\vec{a}_n), v_1(y,\vec{a}_n), \ldots, v_m(y,\vec{a}_n)\,]\!]^{(m+1)}$$

in $\mathscr{K}_4$. This guarantees that

$$\lambda y \vec{a}_n . \Pi_i^{m+1}\Big([\![\, I(y,\vec{a}_n), v_1(y,\vec{a}_n), \ldots, v_m(y,\vec{a}_n)\,]\!]^{(m+1)} \Big)$$

are in $\mathscr{K}_4$, for $i = 1, \ldots, m+1$. □

0.1.0.40 Corollary. *The simulating functions are in $\mathscr{E}^2$.*

Proof. Given that the iterators in the simultaneous recursion employed in 0.1.0.37 are trivially in $\mathscr{E}^2$, we only need to provide $\mathscr{E}^2$-bounds for all the produced functions (0.1.0.34). Well, $I(y,\vec{a}_n) \le k$, where k is the label of the stop instruction of M. On the other hand, since all we do with the iterators can at most add 1 in each step, we also have the bounds $v(y,\vec{a}_n) \le \max \vec{a}_n + y + C$, a bound which is in $\mathscr{E}^2$ as a function of y and $\vec{a}_n$, seeing that $\max(x,y) = x \dot{-} y + y$. The "$+C$" accounts for all the constants that may be assigned to a variable during the computation (instructions of type $V_i \leftarrow a$). □

We can now prove (the nontrivial) half of the Ritchie-Cobham property:

0.1.0.41 Lemma. *If $f = M_{\mathbf{z}}^{\vec{\mathbf{x}}_n}$ runs on M within time $t \in \mathscr{E}^n$, for some $n \ge 2$, then $f \in \mathscr{E}^n$.*

Proof. Let the simulating functions of M be as in 0.1.0.37, where $\mathbf{z}$ is "V_1", the output variable. Then, for all $\vec{a}_n$, we have $f(\vec{a}_n) = v_1\big(t(\vec{a}_n), \vec{a}_n\big)$, and this settles the claim by 0.1.0.40. □

The "easy" half of the Ritchie-Cobham property is proved by doing a bit of programming.

0.1.0.42 Lemma. *For $n \ge 2$, any $\lambda\vec{x}.f(\vec{x}) \in \mathscr{E}^n$ is URM-computable within time $\lambda\vec{x}.t(\vec{x}) \in \mathscr{E}^n$.*

Proof. Induction over $\mathscr{E}^n$.

We settle the case of the initial functions first (cf. 0.1.0.22). $\lambda x.x$ is computable, as $M_{V_1}^{V_2}$, within $O(x)$ steps by the normalized URM M below

$1:$ **if** $V_2 = 0$ **goto** 5 **else goto** 2
$2: V_1 \leftarrow V_1 + 1$
$3: V_2 \leftarrow V_2 \dot{-} 1$
$4:$ **goto** 1
$5:$ **stop**

while $\lambda x.x+1$ is computable, as $N_{V_1}^{V_2}$, also within $O(x)$ steps by the normalized URM N below:

$1:$ **if** $V_2 = 0$ **goto** 5 **else goto** 2
$2: V_1 \leftarrow V_1 + 1$
$3: V_2 \leftarrow V_2 \dot{-} 1$
$4:$ **goto** 1
$5: V_1 \leftarrow V_1 + 1$
$6:$ **stop**

while $\lambda x.x+1$ is computable, as $N_{V_1}^{V_2}$, also within $O(x)$ steps by the normalized URM N below:

The non normalized URM P below

$1: V_1 \leftarrow V_1 + 1$
$2:$ **stop**

computes $\lambda x.x+1$ as $P_{V_1}^{V_1}$ in $O(1)$ steps.

$\lambda xy.xy$ is computable by the following loop-program, R, within time $O(xy)$, as R_Z^{XY}:

Loop X
  Loop Y
    $Z \leftarrow Z + 1$
  end
end

A straightforward URM simulation of the above is

$1:$ **goto** 7 {**Comment**. **Loop** X begins}
$2:$ **goto** 5 {**Comment**. **Loop** Y begins}
$3: \quad Z \leftarrow Z + 1$
$4: \quad Y \leftarrow Y \dot{-} 1$
$5:$ **if** $Y = 0$ **goto** 6 **else goto** 3 {**Comment**. **Loop** Y ends}
$6: \quad X \leftarrow X \dot{-} 1$
$7:$ **if** $X = 0$ **goto** 8 **else goto** 2 {**Comment**. **Loop** X ends}
$8:$ **stop**

This still runs within $O(xy)$ time. With the case of $n = 2$ done, we now turn to the initial functions of $\mathscr{E}^{n+1}$ for $n \geq 2$.

The only new case is A_n. We show that it is computable by some URM M within time A_n^k, for some k.

We know that $A_n \in \mathscr{L}_n$. So let $A_n = P_z^x$, where the program $P \in L_n$ terminates within $O(A_n^k(x))$ steps (Exercise![†])

But how about computing P_z^x on a URM? We can efficiently translate any loop program into a URM program!

To this end, note that loop program instructions, other than those of type $X = Y$ and the **Loop-end** pair, occur also in URM programs and thus can be the translated as themselves. On the other hand, $X = Y$ can be simulated by a URM (as we know).

Recursively, assume that we know how to translate R into a URM $\widetilde{R}$ and consider Q:

$$Q : \begin{cases} \textbf{Loop } X \\ R \\ \textbf{end} \end{cases}$$

This is simulated by the URM

	$B \leftarrow X$	{A *new* B is associated with *each* instruction "**Loop** X"[‡]}
	goto L	{ L labels the "**end**" that matches the simulated "**Loop** X"}
M :		
	$\widetilde{R}$	
	$B \leftarrow B \dot{-} 1$	
L :	**if** $B = 0$	**goto** $L + 1$ **else goto** M
$L + 1$:		

Let next the run time of a loop program be $O(t)$. If an instruction of type "$B \leftarrow X$" were to take 1 step in a URM, then the above described simulating URM would also run within time $O(t)$. But this is not a primitive instruction of a URM! It takes time $O(X)$ to perform it.

Now, for the P above in particular —which computes A_n— and since $t = O(A_n^k(x))$, it follows that for any variable X of P, we have $O(X) = O(A_n^k(x))$,[§] and thus the URM runs within time $O\big((A_n^k(x))^2\big) = O(A_n^{k+1}(x))$ due to $x^2 = O(A_2(x)) = O(A_n(x))$.

We have concluded the basis case for all $n \geq 2$.

To conclude the induction over $\mathscr{E}^n$ ($n \geq 2$) we show that the property *propagates* with *substitution* and *bounded recursion*.

Let then f and g from $\mathscr{E}^n$, $n \geq 2$, be URM-computable (by programs M_f and M_g) with run times bounded by t_f and t_g—both in $\mathscr{E}^n$. Consider

$$\lambda\vec{x}\vec{y}.f(\vec{x}, g(\vec{y})) \qquad (*)$$

[†] *Hint.* Show that, for *any* $P \in \mathscr{L}_n$, P_Y^X runs within time that is also a $\mathscr{L}_n$ function. Then recall that $\mathscr{L}_n = \mathscr{K}_n^{sim}$.

[‡] For a given X the instruction "**Loop** X" may appear several times. Each occurrence is associated with a new "B".

[§] To see this *upper bound* think of X as the output variable!

We can (essentially) concatenate M_g and M_f in that order to compute $(*)$. The run time of this program is bounded by $\lambda\vec{x}\vec{y}.t_g(\vec{y}) + t_f(\vec{x}, g(\vec{y}))$, which is in $\mathscr{E}^n$, just as $\lambda\vec{x}\vec{y}.f(\vec{x}, g(\vec{y}))$ is. The other cases of substitution are trivial and are omitted.

Finally, let $\lambda x\vec{y}.f(x, \vec{y})$ be obtained by a bounded recursion from basis h, iterator g and bound B, all in $\mathscr{E}^n$, and all programmable in respective URMs within time bounds t_h, t_g and t_B, all in $\mathscr{E}^n$. A URM program for f, in "pseudo code", is

$$
\begin{array}{rl}
 & z \leftarrow h(\vec{y}) \\
 & i \leftarrow 0 \\
R: & \textbf{if } x = 0 \textbf{ goto } L \textbf{ else goto } L' \\
L': & z \leftarrow g(i, \vec{y}, z) \\
 & i \leftarrow i + 1 \\
 & x \leftarrow x \dot{-} 1 \\
 & \textbf{goto } R \\
L: & \textbf{stop}
\end{array}
$$

Its run time is

$$t_h(\vec{y}) + O\Big(\sum_{i<x} t_g(i, \vec{y}, f(i, \vec{y}))\Big) \;\P \tag{1}$$

Since t_h, t_g and f are all in $\mathscr{E}^n$, then so is the function given by expression (1), due to 0.1.0.30. □

The simulation of a loop program by a URM given on p. 20 represents the general-purpose, "faithful" simulation that, in particular, is true to the fact that the number of iterations of a loop, **Loop** X, *depend only on the value of* X *upon entry in the loop.* That is the purpose of the new variable B.

The simulation on p. 19 is expedient but acceptable since neither X nor Y are present inside the "scope" of either loop.

By virtue of Lemmata 0.1.0.41 and 0.1.0.42 we have now proved:

0.1.0.43 Theorem. (The Ritchie-Cobham Property of $\mathscr{E}^n$) *For $n \geq 2$, a function f is in $\mathscr{E}^n$ iff it can be computed on some URM within time $t_f \in \mathscr{E}^n$.*

The Ritchie-Cobham property shows the extremely close relationship between static and computational complexity of primitive recursive functions: The *run time* complexity of a function f in $\mathscr{E}^{n+1}$—as it is measured by the amount of time it takes to compute it, namely, A_n^k—is exactly predicted by the *definitional* complexity of the function: its level in the hierarchy. And conversely! The run time predicts the definitional complexity. *Very accurately.*

We can now compare all the hierarchies that we introduced:

0.1.0.44 Corollary. *For $n \geq 2$, we have $\mathscr{K}_n^{sim} = \mathscr{E}^{n+1}$.*

¶ Of course, this denotes, for some C and D, the expression $t_h(\vec{y}) + C\sum_{i<x} t_g(i, \vec{y}, f(i, \vec{y})) + D$.

Proof. The $\supseteq$ is immediate by 0.1.0.43: Let $f \in \mathscr{E}^{n+1}$ and let it run on some M within time $t_f \in \mathscr{E}^{n+1}$. Now $t_f(\vec{x}) \le A_n^r(\max \vec{x})$, everywhere, by 0.1.0.27. If v_1 is, as before (0.1.0.37), the simulating function for the output variable of M, then

$$f = \lambda \vec{x}.v_1(A_n^r(\max \vec{x}), \vec{x})$$

But $A_n^r \in \mathscr{K}_n^{sim}$ (0.1.0.9), thus, $f \in \mathscr{K}_n^{sim}$.

For the $\subseteq$ we do induction on $n \ge 2$. For $n = 2$ note that, trivially, $\mathscr{K}_0^{sim} \subseteq \mathscr{E}^3$. Now—by varying r— we can make A_1^r majorize every function of $\mathscr{K}_1^{sim}$ (0.1.0.10), thus every simultaneous recursion that produces functions in $\mathscr{K}_1^{sim}$ (from functions in $\mathscr{K}_0^{sim}$) is a bounded recursion within $\mathscr{E}^3$ ($A_1 = \lambda x.2x + 2 \in \mathscr{E}^3$). Therefore, $\mathscr{K}_1^{sim} \subseteq \mathscr{E}^3$. Repeating this argument we have that

> *every simultaneous recursion that produces functions in $\mathscr{K}_2^{sim}$ (from functions in $\mathscr{K}_1^{sim}$) is a bounded recursion within $\mathscr{E}^3$ (since $A_2 \in \mathscr{E}^3$).*

thus, $\mathscr{K}_2^{sim} \subseteq \mathscr{E}^3$.

Taking as an I.H. the validity of the claim for some fixed $n \ge 2$, the case for $n+1$ is repeating the idea we employed in the basis: recursions taking us from $\mathscr{K}_n^{sim}$ to $\mathscr{K}_{n+1}^{sim}$ are bounded recursions performed *within* $\mathscr{E}^{n+2}$ ($\supseteq \mathscr{E}^{n+1} \supseteq$, by I.H., $\mathscr{K}_n^{sim}$), with bounding function some A_{n+1}^r—since $A_{n+1}^r \in \mathscr{K}_{n+1}^{sim} \cap \mathscr{E}^{n+2}$. □

By 0.1.0.15 we have at once

0.1.0.45 Corollary. *For $n \ge 2$, we have $\mathscr{L}_n = \mathscr{E}^{n+1}$.*

0.1.0.46 Corollary. *For $n \ge 4$, we have $\mathscr{K}_n = \mathscr{E}^{n+1}$.*

Proof. The proof follows very closely that of 0.1.0.44. The $\subseteq$ goes through unchanged, but the $\supseteq$ "starts" later, $n \ge 4$, due to the fact that the simulating function v_1 is in K_4; cf. 0.1.0.39. □

Schwichtenberg has improved 0.1.0.46 by proving the case for $n = 3$ [4]. This is retold in [8]. [3] gives a proof for the case $n = 2$.

0.1.0.47 Remark. (A Very Hard Problem—Revisited) Corollary 0.1.0.45 adversely impacts a problem of practical significance: That of *program correctness*. The problem "program correctness" is an instance of the *equivalence problem* of programs, since it tasks us to determine whether a program follows faithfully a *specification*, the latter being, of course, given by a *finite description*, just as the program is.

We strengthen here the observation we made earlier in the course, about the *equivalence problem* of primitive recursive functions, that is, the equivalence problem of loop programs:

Given loop programs P and Q, is it the case that $P_Y^{\vec{X}} = Q_Y^{\vec{X}}$?

We saw that the equivalence problem for $\mathscr{PR}$ is unsolvable—indeed, worse: not even c.e.—as a consequence of the fact $\lambda x.1$ and $\lambda y.\chi_T(x, x, y)$ are in $\mathscr{PR}$.

As these functions are also in $\mathscr{E}^3$—a fact that can be readily verified by looking at the proof of the normal form theorem (See Problem Set #3 :-)—it follows that the equivalence problem for $\mathscr{E}^3$ functions is not c.e. either. By virtue of 0.1.0.45, this yields the rather disappointing alternative formulation:

The equivalence problem for programs in L_2—i.e., those that have loop depth equal to two—is not c.e.

Thus the various techniques employed to tackle *loop correctness* can be *successful in all instances of the problem* only when we have un-nested loops—L_1-programs. This holds true even though the loops are "FOTRAN-like", that is, they always terminate and the number of iterations of any such loop is known at the time the loop is entered. It should be noted that Tsichritzis (cf. [9] and [8]) has shown that programs in L_1 have a solvable equivalence problem, but, on the other hand, the corresponding set of functions, $\mathscr{L}_1$ is rather trivial: it is the closure under substitution of $\{\lambda xy.x + y, \lambda x.x \dot{-} 1, \lambda xyz.\text{ if } x = 0 \text{ then } y \text{ else } z, \lambda x, \lfloor x/k \rfloor, \lambda x.rem(x, k)\}$. That is, all "looping" can be eliminated if we adopt this enlarged set of initial functions. □

Bibliography

[1] P. Axt. Iteration of Primitive Recursion. *Zeitschrift für math. Logik*, 11:253–255, 1965.

[2] A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4:1–45, 1953.

[3] H. Müller. Characterization of the Elementary Functions in Terms of Nesting of Primitive Recursions. *Recursive Function Theory: Newsletter*, (5):14–15, April 1973.

[4] H. Schwichtenberg. Rekursionszahlen und die Grzegorczyk-Hierarchie. *Arch. math. Logik*, 12:85–97, 1969.

[5] W. Heinermann. *Untersuchungen über die Rekursionszahlen rekursiven Funktionen*. PhD thesis, Münster, 1961.

[6] Rózsa Péter. *Recursive Functions*. Academic Press, New York, 1967.

[7] D.M. Ritchie. Complexity Classification of Primitive Recursive Functions by their Machine Programs. Term paper for Applied Mathematics 230, Harvard University, 1965.

[8] G. Tourlakis. *Computability*. Reston Publishing, Reston, VA, 1984.

[9] D Tsichritzis. The Equivalence Problem of Simple Programs. *JACM*, 17:729–738, 1970.