Feb 26

0.0.1 The Ackermann Function

The "Ackermann function" was proposed, of course, by Ackermann. The version here is a simplification offered by Robert Ritchie.

What the function does is to provide us with an example of a number-theoretic *intuitively computable, total* function that is not in $\mathcal{PR}$. But this function is more than just *intuitively* computable! It *is* computable—no hedging—as we will show by showing it to be a member of $\mathcal{R}$.

Another thing it does is that it provides us with an example of a function $\lambda\vec{x}.f(\vec{x})$ that is "hard to compute" ($f \notin \mathcal{PR}$) but whose *graph*—that is, the predicate $\lambda y\vec{x}.y = f(\vec{x})$—is nevertheless "easy to compute" ($\in \mathcal{PR}_*$).*

0.0.1 Definition. The Ackermann function, $\lambda nx.A_n(x)$, is given, for all $n \geq 0, x \geq 0$ by the equations

$$
\begin{aligned}
A_0(x) &= \; x + 2 \\
A_{n+1}(x) &= \; A_n^x(2)
\end{aligned}
$$

where h^x is function iteration. □

For any $\lambda y.h(y)$, the function $\lambda xy.h^x(y)$ is given by the primitive recursion

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

It is obvious then that if $h \in \mathcal{PR}$ then so is $\lambda xy.h^x(y)$.

The λ-notation makes it clear that both n and x are *arguments* of the Ackermann function. While we could have written $A(n, x)$ instead, it is notationally less challenging to use the chosen notation. We refer to the n as the *subscript argument*, and to x as the *inner argument*.

0.0.2 Remark. An alternative way to define the Ackermann function, extracted directly from Definition 0.0.1, is as follows:

$$
\begin{aligned}
A_0(x) &= \; x + 2 \\
A_{n+1}(0) &= \; 2 \\
A_{n+1}(x+1) &= \; A_n(A_{n+1}(x))
\end{aligned}
$$

□

*Here the colloquialisms "easy to compute" and "hard to compute" are aliases for "primitive recursive" and "not primitive recursive", respectively. This is a hopelessly coarse rendering of *easy*/*hard* and a much better gauge for the runtime complexity of a problem is on which side of $O(2^n)$ it lies. However, our gauge will have to do for now: All I want to leave you with is that for some functions it is *easier* to compute the graph—to the quantifiable extent that it is in $\mathcal{PR}_*$—than the function itself, to the extent that it fails being primitive recursive.

0.0.2 Properties of the Ackermann Function

We present a sequence of less than earth-shattering—but useful—theorems. So we will just call them lemmata.

0.0.3 Lemma. *For each $n \geq 0$, $\lambda x.A_n(x) \in \mathcal{PR}$.*

Proof. Induction on n: For the basis, clearly $A_0 = \lambda x.x + 2 \in \mathcal{PR}$. Assume now the case for (arbitrary, fixed) n—i.e., $A_n \in \mathcal{PR}$—and go to that for $n+1$. Immediate from Definition 0.0.2, last two equations. □

It turns out that the function blows up in size far too fast with respect to the argument n. We now quantify this remark.

The following unassuming lemma is the key to proving the growth properties of the Ackermann function. It is also the least straightforward to prove, as it requires a double induction—at once on n and x—as dictated by the fact that the "recursion" of Definition 0.0.2 does not leave any argument fixed.

The above shows in particular that, for all n and x, $A_n(x) \downarrow$. That is, $\lambda nx.A_n(x)$ is total.

0.0.4 Lemma. *For each $n \geq 0$ and $x \geq 0$, $A_n(x) > x + 1$.*

Proof. We start an induction on n:

n-Basis. $n = 0$: $A_0(x) = x + 2 > x + 1$; true.

n-I.H.[†] For all x and a fixed (but unspecified) n, *assume* $A_n(x) > x + 1$.

n-I.S.[‡] For all x and the above fixed (but unspecified) n, we must *prove* $A_{n+1}(x) > x + 1$.

We do the n-I.S. by induction on x:

x-Basis. $x = 0$: $A_{n+1}(0) = 2 > 1$; true.

x-I.H. For the above fixed n, we now fix an x (but leave it unspecified) for which we *assume* $A_{n+1}(x) > x + 1$.

x-I.S. For the above fixed (but unspecified) n and x, *prove* $A_{n+1}(x+1) > x + 2$.

Well,

$$\begin{aligned} A_{n+1}(x+1) &= A_n(A_{n+1}(x)) \quad \text{by Def. 0.0.2} \\ &> A_{n+1}(x) + 1 \quad \text{by } n\text{-I.H.} \\ &> x + 2 \quad \text{by } x\text{-I.H.} \end{aligned}$$

□

0.0.5 Lemma. $\lambda x.A_n(x) \nearrow$.

"$\lambda x.f(x) \nearrow$" means that the (total) function f is *strictly increasing*, that is, $x < y$ implies $f(x) < f(y)$, for any x and y. Clearly, to establish the property one just needs to check for the arbitrary x that $f(x) < f(x + 1)$.

[†]To be precise, what we are proving is "$(\forall n)(\forall x)A_n(x) > x + 1$". Thus, as we start on an induction on n, its I.H. is "$(\forall x)A_n(x) > x + 1$" for a fixed unspecified n.

[‡]To be precise, the step is to prove—from the basis and I.H.—"$(\forall x)A_{n+1}(x) > x + 1$" for the n that we fixed in the I.H. It turns out that this is best handled by induction on x.

Proof. We handle two cases separately.

A_0: $\lambda x.x+2\nearrow$; immediate.

A_{n+1}: $A_{n+1}(x+1) = A_n(A_{n+1}(x)) > A_{n+1}(x)+1$—the "$>$" by Lemma 0.0.4. $\square$

0.0.6 Lemma. $\lambda n.A_n(x+1)\nearrow$.

Proof. $A_{n+1}(x+1) = A_n(A_{n+1}(x)) > A_n(x+1)$—the "$>$" by Lemmata 0.0.4 (left argument $>$ right argument) and 0.0.5. $\square$

The "$x+1$" in Lemma 0.0.6 is important since $A_n(0) = 2$ for all n. Thus $\lambda n.A_n(0)$ is increasing but *not* strictly (constant).

0.0.7 Lemma. $\lambda y.A_n^y(x)\nearrow$.

Proof. $A_n^{y+1}(x) = A_n(A_n^y(x)) > A_n^y(x)$—the "$>$" by Lemma 0.0.4. $\square$

0.0.8 Lemma. $\lambda x.A_n^y(x)\nearrow$.

Proof. Induction on y: For $y=0$ we want that $\lambda x.A_n^0(x)\nearrow$, that is, $\lambda x.x\nearrow$, which is true. We next take as I.H. that

$$A_n^y(x+1) > A_n^y(x) \tag{1}$$

We want

$$A_n^{y+1}(x+1) > A_n^{y+1}(x) \tag{2}$$

But (2) follows from (1) and Lemma 0.0.5, by applying A_n to both sides of "$>$". $\square$

0.0.9 Lemma. *For all* n, x, y, $A_{n+1}^y(x) \ge A_n^y(x)$.

Proof. Induction on y: For $y=0$ we want that $A_{n+1}^0(x) \ge A_n^0(x)$, that is, $x \ge x$, which is true. We now take as I.H. that

$$A_{n+1}^y(x) \ge A_n^y(x)$$

We want

$$A_{n+1}^{y+1}(x) \ge A_n^{y+1}(x)$$

This is true because

$$\begin{aligned} A_{n+1}^{y+1}(x) &= A_{n+1}\Big(A_{n+1}^y(x)\Big) \\ &\quad \text{by Lemma 0.0.6} \\ &\ge A_n\big(A_{n+1}^y(x)\big) \\ &\quad \text{Lemma 0.0.5 and I.H.} \\ &\ge A_n^{y+1}(x) \end{aligned}$$

$\square$

0.0.10 Definition. Given a predicate $P(\vec{x})$, we say that $P(\vec{x})$ *is true almost everywhere*—in symbols "$P(\vec{x})$ a.e."—iff the set of (vector) inputs that make the predicate *false* is *finite*. That is, the set $\{\vec{x} : \neg P(\vec{x})\}$ is finite.

A statement such as "$\lambda xy.Q(x, y, z, w)$ a.e." can also be stated, less formally, as
"$Q(x, y, z, w)$ a.e. *with respect to x and y*". □

0.0.11 Lemma. $A_{n+1}(x) > x + l$ *a.e. with respect to x.*

Thus, in particular, $A_1(x) > x + 10^{350000}$ a.e.

Proof. In view of Lemma 0.0.6 and the note following it, it suffices to prove

$$A_1(x) > x + l \text{ a.e. with respect to } x$$

Well, since

$$A_1(x) = A_0^x(2) = \overbrace{(\cdots(((y+2)+2)+2)+\cdots+2)}^{x\ 2\text{'s}}\,\|_{\text{evaluated at } y\,=\,2} = 2 + 2x$$

we ask: Is $2 + 2x > x + l$ a.e. with respect to x? It is so for all $x > l - 2$ (only $x = 0, 1, \ldots, l - 2$ fail). □

0.0.12 Lemma. $A_{n+1}(x) > A_n^l(x)$ *a.e. with respect to x.*

Proof. If one (or both) of l and n is 0, then the result is trivial. For example,

$$A_0^l(x) = \overbrace{(\cdots(((x+2)+2)+2)+\cdots+2)}^{l\ 2\text{'s}} = x + 2l$$

We are done by Lemma 0.0.11.

Let us then assume that $l \geq 1$ and $n \geq 1$. We note that (straightforwardly, via Definition 0.0.1)

$$\begin{aligned} A_n^l(x) &= A_n(A_n^{l-1}(x)) \\ &= A_{n-1}^{A_n^{l-1}(x)}(2) = A_{n-1}^{A_{n-1}^{A_n^{l-2}(x)}(2)}(2) = A_{n-1}^{A_{n-1}^{A_{n-1}^{A_n^{l-3}(x)}(2)}(2)}(2) \end{aligned}$$

The straightforward observation that *we have a "ladder" of k A_{n-1}'s precisely when the topmost exponent is $l - k$* can be ratified by induction on k (left to the reader). Thus we state

$$A_n^l(x) = {}^{k\ A_{n-1}}\left\{ A_{n-1}^{\cdot^{\cdot^{A_{n-1}^{A_n^{l-k}(x)}(2)}\cdot^{\cdot}}}\right.(2)$$

In particular, taking $k = l$,

$$A_n^l(x) = {}^{l\ A_{n-1}}\left\{ \begin{array}{l} \quad\quad A_{n-1}^{A_n^{l-l}(x)}(2) \\ \quad \cdot^{\cdot^{\cdot}} \\ A_{n-1} \end{array}\right. \ddots (2) = {}^{l\ A_{n-1}}\left\{ \begin{array}{l} \quad\quad A_{n-1}^{x}(2) \\ \quad \cdot^{\cdot^{\cdot}} \\ A_{n-1} \end{array}\right. \ddots (2) \qquad (*)$$

Let us now take $x > l$.

Thus, by $(*)$,

$$A_{n+1}(x) = A_n^x(2) = {}^{x\ A_{n-1}}\left\{ \begin{array}{l} \quad\quad A_{n-1}^{2}(2) \\ \quad \cdot^{\cdot^{\cdot}} \\ A_{n-1} \end{array}\right. \ddots (2) \qquad (**)$$

By comparing $(*)$ and $(**)$ we see that the first "ladder" is topped (after $l\ A_{n-1}$ "steps") by x and the second is topped by

$${}^{x-l\ A_{n-1}}\left\{ \begin{array}{l} \quad\quad A_{n-1}^{2}(2) \\ \quad \cdot^{\cdot^{\cdot}} \\ A_{n-1} \end{array}\right. \ddots (2)$$

Thus—in view of the fact that $A_n^y(x)$ increases with respect to each of the arguments n, x, y—we conclude by asking ...

$$\text{“Is } {}^{x-l\ A_{n-1}}\left\{ \begin{array}{l} \quad\quad A_{n-1}^{2}(2) \\ \quad \cdot^{\cdot^{\cdot}} \\ A_{n-1} \end{array}\right. \ddots (2) > x \text{ a.e. with respect to } x\text{?”}$$

... and answering, "Yes", because by $(**)$ this is the same question as "is $A_{n+1}(x-l) > x$ a.e. with respect to x?", which we answered affirmatively in 0.0.11. $\square$

0.0.13 Lemma. *For all n, x, y, $A_{n+1}(x+y) > A_n^x(y)$.*

Proof.

$$\begin{aligned} A_{n+1}(x+y) &= A_n^{x+y}(2) \\ &= A_n^x\Big(A_n^y(2)\Big) \\ &= A_n^x\Big(A_{n+1}(y)\Big) \\ &> A_n^x(y) \quad \text{by Lemmata 0.0.4 and 0.0.8} \end{aligned}$$

$\square$

0.0.3 The Ackermann Function Majorises All the Functions of $\mathcal{PR}$

We say that a function f *majorizes* another function, g, iff $g(\vec{x}) \le f(\vec{x})$ for all $\vec{x}$. The following theorem states precisely in what sense "*the Ackermann function majorizes all the functions of $\mathcal{PR}$*".

0.0.14 Theorem. *For every function $\lambda\vec{x}.f(\vec{x}) \in \mathcal{PR}$ there are numbers n and k, such that for all $\vec{x}$ we have $f(\vec{x}) \le A_n^k(\max(\vec{x}))$.*

Proof. The proof is by induction with respect to $\mathcal{PR}$. Throughout I use the abbreviation $|\vec{x}|$ for $\max(\vec{x})$ as this is notationally friendlier.

For the basis, f is one of:

- *Basis.*

  Basis 1. $\lambda x.0$. Then $A_0(x)$ works $(n = 0, k = 1)$.

  Basis 2. $\lambda x.x + 1$. Again $A_0(x)$ works $(n = 0, k = 1)$.

  Basis 3. $\lambda \vec{x}.x_i$. Once more $A_0(x)$ works $(n = 0, k = 1)$: $x_i \le |\vec{x}| < A_0(|\vec{x}|)$.

- *Propagation with composition.* Assume as I.H. that

$$f(\vec{x}_m) \le A_n^k(|\vec{x}_m|) \tag{1}$$

and

$$\text{for } i = 1, \ldots, m,\ g_i(\vec{y}) \le A_{n_i}^{k_i}(|\vec{y}|) \tag{2}$$

Then

$$\begin{aligned} f(g_1(\vec{y}), \ldots, g_m(\vec{y})) &\le A_n^k(|g_1(\vec{y}), \ldots, g_m(\vec{y})|), \text{ by } (1) \\ &\le A_n^k(|A_{n_1}^{k_1}(|\vec{y}|), \ldots, A_{n_m}^{k_m}(|\vec{y}|)|), \text{ by 0.0.8 and } (2) \\ &\le A_n^k\Big(|A_{\max n_i}^{\max k_i}(|\vec{y}|)\Big), \text{ by 0.0.8 and 0.0.9} \\ &\le A_{\max(n,n_i)}^{k+\max k_i}(|\vec{y}|), \text{ by 0.0.9} \end{aligned}$$

- *Propagation with primitive recursion.* Assume as I.H. that

$$h(\vec{y}) \le A_n^k(|\vec{y}|) \tag{3}$$

and

$$g(x, \vec{y}, z) \le A_m^r(|x, \vec{y}, z|) \tag{4}$$

Let f be such that

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

I claim that

$$f(x, \vec{y}) \le A_m^{rx}\Big(A_n^k(|x, \vec{y}|)\Big) \tag{5}$$

I prove (5) by induction on x:

For $x = 0$, I want $f(0, \vec{y}) = h(\vec{y}) \le A_n^k(|0, \vec{y}|)$. This is true by (3) since $|0, \vec{y}| = |\vec{y}|$.

As an I.H. assume (5) for fixed x.

The case for $x+1$:

$$\begin{aligned}
f(x+1,\vec{y}) &= g(x,\vec{y},f(x,\vec{y}))\\
&\le A^r_m(|x,\vec{y},f(x,\vec{y})|), \text{ by (4)}\\
&\le A^r_m\Big(\Big|x,\vec{y},A^{rx}_m\Big(A^k_n(|x,\vec{y}|)\Big)\Big|\Big), \text{ by the I.H. (5), and 0.0.8}\\
&= A^r_m\Big(A^{rx}_m\Big(A^k_n(|x,\vec{y}|)\Big)\Big), \text{ by 0.0.8 and } A^{rx}_m\big(A^k_n(|x,\vec{y}|)\big) \ge |x,\vec{y}|\\
&= A^{r(x+1)}_m\Big(A^k_n(|x,\vec{y}|)\Big)\\
&\le A^{r(x+1)}_m\Big(A^k_n(|x+1,\vec{y}|)\Big), \text{ by 0.0.8}
\end{aligned}$$

With (5) proved, let me set $l=\max(m,n)$. By Lemma 0.0.9 I now get

$$f(x,\vec{y}) \le A^{rx+k}_l(|x,\vec{y}|) \underset{\text{Lemma 0.0.13}}{<} A_{l+1}(|x,\vec{y}|+rx+k) \tag{6}$$

Now, $|x,\vec{y}|+rx+k \le (r+1)|x,\vec{y}|+k$ thus, (6) and 0.0.5 yield

$$f(x,\vec{y}) < A_{l+1}((r+1)|x,\vec{y}|+k) \tag{7}$$

To simplify (7) note that there is a number q such that

$$(r+1)x+k \le A^q_1(x) \tag{8}$$

for all x. Indeed, this is so since (easy induction on y) $A^y_1(x) = 2^y x + 2^y + 2^{y-1} + \cdots + 2$. Thus, to satisfy (8), just take $y=q$ large enough to satisfy $r+1 \le 2^q$ and $k \le 2^q + 2^{q-1} + \cdots + 2$.

By (8), the inequality (7) yields, via 0.0.5,

$$f(x,\vec{y}) < A_{l+1}(A^q_1(|x,\vec{y}|)) \le A^{1+q}_{l+1}(|x,\vec{y}|)$$

(by Lemma 0.0.9) which is all we want. □

0.0.15 Remark. Reading the proof carefully we note that the *subscript argument* of the *majorant*[§] is precisely the maximum depth of nesting of primitive recursion that occurs in a derivation of f.

Pause. In which derivation? There are infinitely many.◀

Indeed, the initial functions have a majorant with subscript 0; composition has a majorant with subscript no more than the maximum subscript of the component parts—*no increase*; primitive recursion has a majorant with a subscript that is bigger than the *maximum* subscript of the h- and g-majorants by precisely 1. □

0.0.16 Corollary. $\lambda nx.A_n(x) \notin \mathcal{PR}$.

[§] The function that does the majorizing.

Proof. By contradiction: If $\lambda nx.A_n(x) \in \mathcal{PR}$ then also $\lambda x.A_x(x) \in \mathcal{PR}$ (identification of variables). By the theorem above, for some n, k, $A_x(x) \le A_n^k(x)$, for all x, hence, by 0.0.12

$$A_x(x) < A_{n+1}(x), \text{ a.e. with respect to } x \tag{1}$$

On the other hand, $A_{n+1}(x) < A_x(x)$ a.e. with respect to x—indeed for all $x > n+1$ by 0.0.6—which contradicts (1). □

0.0.4 The Graph of the Ackermann Function is in $\mathcal{PR}_*$

How does one compute a yes/no answer to the question

$$\text{“}A_n(x) = z?\text{”} \tag{1}$$

Thinking “recursively” (in the programming sense of the word), we will look at the question by considering three cases, according to the definition in the Remark 0.0.2:

(a) If $n = 0$, then we will directly check (1) as “is $x + 2 = z$?”.

(b) If $x = 0$, then we will directly check (1) as “is $2 = z$?”.

(c) In all other cases, i.e., $n > 0$ and $x > 0$, we may naturally ask *two* questions [both *must* be answerable “yes” for (1) to be true]:¶ “Is there a w such that $A_{n-1}(w) = z$ **and** also $A_n(x-1) = w$?”

Steps (a)–(c) are entirely analogous to steps in a proof. Just as in a proof we verify the truth of a statement via syntactic means, here we are verifying the truth of $A_n(x) = z$ by such means.

Steps (a) and (b) correspond to writing down axioms. Step (c) corresponds to attempting to prove B by applying MP (modus ponens) where we are looking for an A such that we have a proof of both A and $A \to B$. In fact, closer to the situation in (c) above is a proof step where we want to prove $X \to Y$ and are looking for a Z such that both $X \to Z$ and $Z \to Y$ are known to us theorems. Z plays a role entirely analogous to that of w above.

Assuming that we want to pursue the process (a)–(c) by pencil and paper or some other equivalent means, it is clear that the pertinent info that we are juggling are *ordered triples* of numbers such as n, x, z, or $n-1, w, z$, etc. That is, the letter “A”, the brackets, the equals sign, and the position of the arguments (subscript vs. inside brackets) are just *ornamentation*, and the string “$A_i(j) = k$”, *in this section's context*, does not contain any more information than the *ordered* triple “(i, j, k)”.

Thus, to “compute” an answer to (1) we need to write down enough triples, in stages (or steps), as needed to justify (1): At each stage we may write a triple (i, j, k) down just in case *one* of (i)–(iii) holds:

¶Note that $A_n(x) = A_{n-1}(A_n(x-1))$.

(i) $i = 0$ and $k = j + 2$

(ii) $j = 0$ and $k = 2$

(iii) $i > 0$ and $j > 0$, and for some w, we have already written down the two triples $(i-1, w, k)$ and $(i, j-1, w)$.

0.0.17 Remark. Since "(i, j, k)" abbreviates "$A_i(j) = k$", Lemma 0.0.4 implies that $j < k$. □

Our theory is more competent with numbers (than with pairs, triples, etc.) preferring to *code* tuples into single numbers. Thus if we were to carry out the pencil and paper algorithm *within our theory*, then we would be well advised to code all these triples, which we write down step by step, by single numbers: We will use our usual prime-power coding, $\langle i, j, k \rangle$, to do so.

The verification process for $A_n(x) = z$, described in (a)–(c), is a sequence of steps of types (a), (b) or (c) that *ends* with the (coded) triple $\langle n, x, z \rangle$.

We will code such a sequence We note that our computation is "tree-like", since a "complicated" triple such as that of case (iii) above *requires* two similar others to be already written down, each of which in turn will require two *earlier* similar others, etc., until we reach "leaves" [cases (i) or (ii)] that can be dealt with directly without passing the buck.

This "tree", just like the tree of a mathematical proof, can be "linearised" and thus be arranged in a sequence of coded triples $\langle i, j, k \rangle$ so that the presence of a "$\langle i, j, k \rangle$" implies that all its dependencies appear *earlier* (to its left).

We will code the entire proof sequence by a single number, u, using prime-power coding.

The major result in this subsection is the theorem below, that given any number u, we can primitively recursively check whether or not it is a code of an Ackermann function computation:

0.0.18 Theorem. *The predicate*

$$Comp(u) \stackrel{\text{Def}}{=} u \textit{ codes an Ackermann function computation}$$

is in $\mathcal{PR}_*$.

Proof. The auxiliary predicates $\lambda vu.v \in u$ and $\lambda vwu.v <_u w$ mean

$$u = \langle \ldots, v, \ldots \rangle \qquad (v \text{ is member of the coded sequence})$$

and

$$u = \langle \ldots, v, \ldots, w, \ldots \rangle \qquad (v \text{ appears before } w \text{ in the code } u)$$

respectively. Both are in $\mathcal{PR}_*$ since

$$v \in u \equiv Seq(u) \wedge (\exists i)_{<lh(u)} (u)_i = v$$

and

$$v <_u w \equiv Seq(u) \wedge (\exists i, j)_{<lh(u)} \big((u)_i = v \wedge (u)_j = w \wedge i < j \big)$$

The right hand side of "≡" below rests the case of the proof.

$$Comp(u) \equiv Seq(u) \wedge (\forall v)_{\leq u}\Big(v \in u \to Seq(v) \wedge lh(v) = 3 \wedge$$

{**Comment:** Case (i), p. 9} $\Big\{(v)_0 = 0 \wedge (v)_2 = (v)_1 + 2 \vee$

{**Comment:** Case (ii)} $(v)_1 = 0 \wedge (v)_2 = 2 \vee$

{**Comment:** Case (iii)} $\Big((v)_0 > 0 \wedge (v)_1 > 0 \wedge$

$$(\exists w)_{<(v)_2}\big(\langle (v)_0 \dot{-} 1, w, (v)_2\rangle <_u v \wedge \langle (v)_0, (v)_1 \dot{-} 1, w\rangle <_u v\big)\Big)\Big\}\Big)$$

Remark 0.0.17 justifies the bound on $(\exists w)$ above. □

Thus $A_n(x) = z$ iff $\langle n, x, z\rangle \in u$ for some u that satisfies *Comp*. For short

$$A_n(x) = z \equiv (\exists u)(Comp(u) \wedge \langle n, x, z\rangle \in u) \tag{1}$$

If we succeed in finding a bound for u that is a primitive recursive function of n, x, z then we will have succeeded showing:

0.0.19 Theorem. *$\lambda nxz.A_n(x) = z \in \mathcal{PR}_*$.*

Proof. We assume a computation u that as soon as it verifies $A_n(x) = z$ quits, that is, it only includes $\langle n, x, z\rangle$ (at the very end) and all the needed *predecessor* coded triples, but nothing else. How big can u be?

Note that

$$u = \cdots p_r^{\langle i,j,k\rangle+1} \cdots p_l^{\langle n,x,z\rangle+1} \tag{2}$$

for appropriate l ($=lh(u) - 1$). For example, if all we want is to verify $A_0(10) = 12$, then $u = p_0^{\langle 0,10,12\rangle+1}$.

Similarly, if all we want is to verify $A_1(1) = 4$, then—since the "recursive calls" here are to $A_0(2) = 4$ and $A_1(0) = 2$—two possible u-values work: $u = p_0^{\langle 0,2,4\rangle+1} p_1^{\langle 1,0,2\rangle+1} p_2^{\langle 1,1,4\rangle+1}$ or $u = p_0^{\langle 1,0,2\rangle+1} p_1^{\langle 0,2,4\rangle+1} p_2^{\langle 1,1,4\rangle+1}$.

How big need l be? No bigger than needed to provide distinct *positions* ($l+1$ such) in the computation, for all the "needed" triples $\langle i, j, k\rangle$. Since z is the largest possible output (and larger than any input) that is computed, there are no more than $(z+1)^3$ triples possible, so $l + 1 \leq (z+1)^3$. Therefore, (2) yields

$$\begin{aligned}
u &\leq \cdots p_r^{\langle z,z,z\rangle+1} \cdots p_l^{\langle z,z,z\rangle+1} \\
&= \Big(\Pi_{i \leq l} p_i\Big)^{\langle z,z,z\rangle+1} \\
&\leq p_l^{(l+1)(\langle z,z,z\rangle+1)} \\
&\leq p_{(z+1)^3}^{(z+1)^3(\langle z,z,z\rangle+1)}
\end{aligned}$$

Setting $g = \lambda z.p_{(z+1)^3}^{(z+1)^3(\langle z,z,z\rangle+1)}$ we have $g \in \mathcal{PR}$ and we are done by (1):

$$A_n(x) = z \equiv (\exists u)_{\leq g(z)}(Comp(u) \wedge \langle n, x, z\rangle \in u)$$ □

Worth saying: If f is total and $y = f(\vec{x})$ is in $\mathcal{PR}_*$, then it does *not* necessarily follow that $f \in \mathcal{PR}$, as 0.0.19 exemplifies. On the other hand, if f is total and $y = f(\vec{x})$ is in $\mathcal{R}_*$, then, trivially, $f \in \mathcal{R}$ since $f = \lambda x.(\mu y)(y = f(\vec{x}))$.

What is missing from the preceding expression is a *primitive recursive bound* on the search (μy), and this absence does not allow us to conclude that f is primitive recursive even when its graph is. For example, such a bound is impossible in the Ackermann case as we know from its growth properties.

0.0.20 Theorem. *$\lambda nx.A_n(x) \in \mathcal{R}$.*

Proof. $\lambda nx.A_n(x) = \lambda nx.(\mu z)(z = A_n(x))$. But $\lambda nxz.z = A_n(x)$ is in $\mathcal{PR}_*$, thus $\lambda nx.A_n(x) \in \mathcal{P}$. But this function is total! $\square$