Ackermann function is total recursive

Canonical name	AckermannFunctionIsTotalRecursive
Date of creation	2013-03-22 19:07:53
Last modified on	2013-03-22 19:07:53
Owner	CWoo (3771)
Last modified by	CWoo (3771)
Numerical id	15
Author	CWoo (3771)
Entry type	Theorem
Classification	msc 03D75
Related topic	AckermannFunctionIsNotPrimitiveRecursive

In this entry, we give a formal proof that the Ackermann function $A(x,y)$, given by

$$A(0,y)=y{+}1,\qquad A(x{+}1,0)=A(x,1),\qquad A(x{+}1,y{+}1)=A(x,A(x{+}1,y))$$

is both a total function and a recursive function. Actually, the fact that A is total is proved in `http://planetmath.org/PropertiesOfAckermannFunction`this entry. It remains to show that A is recursive.

Recall that the computation of $A(x,y)$, given x,y, can be thought of as an iterated operation performed on finite sequences of integers, starting with x,y and ending with $z=A(x,y)$ (see `http://planetmath.org/ComputingTheAckermannFunction`this entry). It is this process we will utilize to prove that A is recursive.

In the proof below, the following notations and definitions are used to simplify matters:

- if s is the sequence $r_1,\dots,r_m$, then $E(s)$ or $\langle r_1,\dots,r_m\rangle$ denote the code number of s given the encoding E;
- $\mathrm{lh}(n)$ is the length of the sequence whose code number is n;
- $(n)_i$ is the i-th number in the sequence whose code number is n;
- $(n)_{-i}$ is the i-th to the last number in the sequence whose code number is n (so that $(n)_{-1}$ is the last number in the sequence whose code number is n);
- $\mathrm{red}(n)$ is the code number of the sequence obtained by deleting the last number of the sequence whose code number is n;
- $\mathrm{ext}(n,a)$ is the code number of the sequence obtained by appending a to the end of the sequence whose code number is n.

If E is a primitive recursive encoding, then each of the above function is primitive recursive. For example, $(n)_{-i}=(n)_{\mathrm{lh}(n)\dot{-}i+1}$.

Theorem 1. *A is recursive.*

Proof. In this proof, the choice of encoding E is the multiplicative encoding, for it is convenient and, more importantly, a primitive recursive encoding. Briefly,

$$E(r_1,\dots,r_m)=p_1^{r_1+1}\cdots p_m^{r_m+1},$$

where p_i is the i-th prime number (so that $p_1 = 2$).

We know that computing $A(x, y) = z$ is basically a sequence of computations on finite sequences:

$$x, y \longrightarrow \cdots \longrightarrow z \longrightarrow z \longrightarrow \cdots$$

Let $s(x, y, i)$ denote the sequence at step i, then the above sequence can be rewritten:

$$s(x, y, 0) \longrightarrow s(x, y, 1) \longrightarrow \cdots \longrightarrow s(x, y, k) \longrightarrow \cdots$$

Define $f(x, y, i) = E(s(x, y, i))$. From this we see that

$$g(x, y) = \mu i[f(x, y, i) = f(x, y, i + 1)].$$

is the function that computes the smallest number of steps needed so that the code number becomes stationary. When the code number is decoded, we get the resulting value of $A(x, y)$:

$$A(x, y) = D(f(x, y, g(x, y))),$$

where $D(m) := (m)_{-1}$, decodes m, and returns the last number in the sequence s whose code number $E(s)$ is m.

Now the remaining task to show that f is primitive recursive. First, note that

$$f(x, y, 0) = \langle x, y \rangle = 2^{x+1}3^{y+1}$$

is primitive recursive. Next, we want to express

$$f(x, y, n + 1) = h(f(x, y, n)),$$

where h is the function that changes the code number of the sequence $s(x, y, n)$ to the code number of the sequence $s(x, y, n+1)$. Once we obtain h and show that h is primitive recursive, then f is primitive recursive, as it is defined by primitive recursion via primitive recursive functions $\langle x, y \rangle$ and h.

To find out what h is, recall the four rules of constructing the next sequence from the current one givne in this `http://planetmath.org/ComputingTheAckermannFunct` entry. Let $n_1 = E(s(x, y, k))$ and $n_2 = E(s(x, y, k + 1))$. We rewrite the four rules using the notations and definitions here:

1. if $\mathrm{lh}(n_1) = 1$, then $n_2 = n_1$;

2. if $\mathrm{lh}(n_1) > 1$, and $(n_1)_{-2} = 0$, then $n_2 = h_1(n_1)$, where

$$h_1(n) := \mathrm{ext}(\mathrm{red}^2(n), (n)_{-1} + 1);$$

3. if $\mathrm{lh}(n_1) > 1$, and $(n_1)_{-2} > 0$ and $(n_1)_{-1} = 0$, then $n_2 = h_2(n_1)$, where

$$h_2(n) := \mathrm{ext}(\mathrm{ext}(\mathrm{red}^2(n), (n)_{-2} - 1), 1);$$

   or

4. if $\mathrm{lh}(n_1) > 1$, and $(n_1)_{-2} > 0$ and $(n_1)_{-1} > 0$, then then $n_2 = h_3(n_1)$, where

$$h_3(n) := \mathrm{ext}(\mathrm{ext}(\mathrm{ext}(\mathrm{red}^2(n), (n)_{-2} - 1), (n)_{-2}), (n)_{-1} - 1).$$

If we define predicates:

1. $\Phi_0(n) := \mathrm{lh}(n) \leq 1$,
2. $\Phi_1(n) := \mathrm{lh}(n) > 1$, and $(n)_{-2} = 0$,
3. $\Phi_2(n) := \mathrm{lh}(n) > 1$, and $(n)_{-2} > 0$ and $(n)_{-1} = 0$,
4. $\Phi_3(n) := \mathrm{lh}(n) > 1$, and $(n)_{-2} > 0$ and $(n)_{-1} > 0$.

Then each Φ_i is primitive recursive, pairwise exclusive, and $\Phi_0 \equiv \neg\Phi_1 \wedge \neg\Phi_2 \wedge \neg\Phi_3$. Now, define h as follows:

$$h(n) := \begin{cases} \mathrm{id}(n) & \text{if } \Phi_0(n), \\ h_1(n) & \text{if } \Phi_1(n), \\ h_2(n) & \text{if } \Phi_2(n), \\ h_3(n) & \text{if } \Phi_3(n). \end{cases}$$

Since h is defined by cases, and each h_i is primitive recursive, h is also primitive recursive. □