

Ackermann's Function Is Not Primitive Recursive

Lawrence C. Paulson

March 17, 2025

Abstract

Ackermann's function is defined in the usual way and a number of its elementary properties are proved. Then, the primitive recursive functions are defined inductively: as a predicate on the functions that map lists of numbers to numbers. It is shown that every primitive recursive function is strictly dominated by Ackermann's function. The formalisation follows an earlier one by Nora Szasz [1].

Contents

1 Ackermann's Function and the PR Functions	3
1.1 Ackermann's Function	3
1.2 Primitive Recursive Functions	4
1.3 Main Result: Ackermann's Function is not Primitive Recursive	5

Remark. This development was part of the Isabelle distribution from 1997 to 2022. It has been transferred to the AFP, where it may be more useful.

1 Ackermann's Function and the PR Functions

This proof has been adopted from a development by Nora Szasz [1].

theory *Primrec* **imports** *Main* **begin**

1.1 Ackermann's Function

fun *ack* :: [*nat*,*nat*] \Rightarrow *nat* **where**
 ack 0 *n* = *Suc* *n*
 | *ack* (*Suc* *m*) 0 = *ack* *m* 1
 | *ack* (*Suc* *m*) (*Suc* *n*) = *ack* *m* (*ack* (*Suc* *m*) *n*)

PROPERTY A 4

lemma *less-ack2* [*iff*]: $j < \text{ack } i \ j$
<proof>

PROPERTY A 5-, the single-step lemma

lemma *ack-less-ack-Suc2* [*iff*]: $\text{ack } i \ j < \text{ack } i \ (\text{Suc } j)$
<proof>

PROPERTY A 5, monotonicity for $<$

lemma *ack-less-mono2*: $j < k \implies \text{ack } i \ j < \text{ack } i \ k$
<proof>

PROPERTY A 5', monotonicity for \leq

lemma *ack-le-mono2*: $j \leq k \implies \text{ack } i \ j \leq \text{ack } i \ k$
<proof>

PROPERTY A 6

lemma *ack2-le-ack1* [*iff*]: $\text{ack } i \ (\text{Suc } j) \leq \text{ack } (\text{Suc } i) \ j$
<proof>

PROPERTY A 4'? Extra lemma needed for *CONSTANT* case, constant functions

lemma *ack-less-ack-Suc1* [*iff*]: $\text{ack } i \ j < \text{ack } (\text{Suc } i) \ j$
<proof>

lemma *less-ack1* [*iff*]: $i < \text{ack } i \ j$
<proof>

PROPERTY A 8

lemma *ack-1* [*simp*]: $\text{ack } (\text{Suc } 0) \ j = j + 2$
<proof>

PROPERTY A 9. The unary *1* and *2* in *ack* is essential for the rewriting.

lemma *ack-2* [*simp*]: $\text{ack } (\text{Suc } (\text{Suc } 0)) \ j = 2 * j + 3$
<proof>

Added in 2022 just for fun

lemma *ack-3*: $ack (Suc (Suc (Suc 0))) j = 2^{j+3} - 3$
(*proof*)

PROPERTY A 7, monotonicity for $<$ [not clear why *ack-1* is now needed first!]

lemma *ack-less-mono1-aux*: $ack i k < ack (Suc (i+j)) k$
(*proof*)

lemma *ack-less-mono1*: $i < j \implies ack i k < ack j k$
(*proof*)

PROPERTY A 7', monotonicity for \leq

lemma *ack-le-mono1*: $i \leq j \implies ack i k \leq ack j k$
(*proof*)

PROPERTY A 10

lemma *ack-nest-bound*: $ack i1 (ack i2 j) < ack (2 + (i1 + i2)) j$
(*proof*)

PROPERTY A 11

lemma *ack-add-bound*: $ack i1 j + ack i2 j < ack (4 + (i1 + i2)) j$
(*proof*)

PROPERTY A 12. Article uses existential quantifier but the ALF proof used $k + 4$. Quantified version must be nested $\exists k'. \forall i j. \dots$

lemma *ack-add-bound2*:
assumes $i < ack k j$ **shows** $i + j < ack (4 + k) j$
(*proof*)

1.2 Primitive Recursive Functions

primrec *hd0* :: $nat\ list \Rightarrow nat$ **where**
 $hd0 [] = 0$
 $hd0 (m \# ms) = m$

Inductive definition of the set of primitive recursive functions of type $nat\ list \Rightarrow nat$.

definition *SC* :: $nat\ list \Rightarrow nat$
where $SC\ l = Suc (hd0\ l)$

definition *CONSTANT* :: $nat \Rightarrow nat\ list \Rightarrow nat$
where $CONSTANT\ n\ l = n$

definition *PROJ* :: $nat \Rightarrow nat\ list \Rightarrow nat$
where $PROJ\ i\ l = hd0 (drop\ i\ l)$

definition *COMP* :: $[nat\ list \Rightarrow nat, (nat\ list \Rightarrow nat)\ list, nat\ list] \Rightarrow nat$

where $COMP\ g\ fs\ l = g\ (map\ (\lambda f. f\ l)\ fs)$

fun $PREC :: [nat\ list \Rightarrow nat, nat\ list \Rightarrow nat, nat\ list] \Rightarrow nat$
where
 $PREC\ f\ g\ [] = 0$
 $| PREC\ f\ g\ (x\ \#\ l) = rec\ nat\ (f\ l)\ (\lambda y\ r. g\ (r\ \#\ y\ \#\ l))\ x$
— Note that g is applied first to $PREC\ f\ g\ y$ and then to $y!$

inductive $PRIMREC :: (nat\ list \Rightarrow nat) \Rightarrow bool$ **where**
 $SC: PRIMREC\ SC$
 $| CONSTANT: PRIMREC\ (CONSTANT\ k)$
 $| PROJ: PRIMREC\ (PROJ\ i)$
 $| COMP: PRIMREC\ g \Longrightarrow lists\ PRIMREC\ fs \Longrightarrow PRIMREC\ (COMP\ g\ fs)$
 $| PREC: PRIMREC\ f \Longrightarrow PRIMREC\ g \Longrightarrow PRIMREC\ (PREC\ f\ g)$
monos $lists\ mono$

1.3 Main Result: Ackermann's Function is not Primitive Recursive

lemma $SC\ case: SC\ l < ack\ 1\ (sum\ list\ l)$
 $\langle proof \rangle$

lemma $CONSTANT\ case: CONSTANT\ n\ l < ack\ n\ (sum\ list\ l)$
 $\langle proof \rangle$

lemma $PROJ\ case: PROJ\ i\ l < ack\ 0\ (sum\ list\ l)$
 $\langle proof \rangle$

COMP case

lemma $COMP\ map\ aux: \forall f \in set\ fs. \exists kf. \forall l. f\ l < ack\ kf\ (sum\ list\ l)$
 $\Longrightarrow \exists k. \forall l. sum\ list\ (map\ (\lambda f. f\ l)\ fs) < ack\ k\ (sum\ list\ l)$
 $\langle proof \rangle$

lemma $COMP\ case:$

assumes $1: \forall l. g\ l < ack\ kg\ (sum\ list\ l)$
and $2: \forall f \in set\ fs. \exists kf. \forall l. f\ l < ack\ kf\ (sum\ list\ l)$
shows $\exists k. \forall l. COMP\ g\ fs\ l < ack\ k\ (sum\ list\ l)$
 $\langle proof \rangle$

PREC case

lemma $PREC\ case\ aux:$

assumes $f: \bigwedge l. f\ l + sum\ list\ l < ack\ kf\ (sum\ list\ l)$
and $g: \bigwedge l. g\ l + sum\ list\ l < ack\ kg\ (sum\ list\ l)$
shows $PREC\ f\ g\ (m\ \#\ l) + sum\ list\ (m\ \#\ l) < ack\ (Suc\ (kf + kg))\ (sum\ list\ (m\ \#\ l))$
 $\langle proof \rangle$

lemma $PREC\ case\ aux':$

assumes $f: \bigwedge l. f\ l + sum\ list\ l < ack\ kf\ (sum\ list\ l)$

and $g: \bigwedge l. g\ l + \text{sum-list } l < \text{ack } kg\ (\text{sum-list } l)$
shows $\text{PREC } f\ g\ l + \text{sum-list } l < \text{ack } (\text{Suc } (kf + kg))\ (\text{sum-list } l)$
<proof>

proposition *PREC-case*:

$\llbracket \bigwedge l. f\ l < \text{ack } kf\ (\text{sum-list } l); \bigwedge l. g\ l < \text{ack } kg\ (\text{sum-list } l) \rrbracket$
 $\implies \exists k. \forall l. \text{PREC } f\ g\ l < \text{ack } k\ (\text{sum-list } l)$
<proof>

lemma *ack-bounds-PRIMREC*: $\text{PRIMREC } f \implies \exists k. \forall l. f\ l < \text{ack } k\ (\text{sum-list } l)$
<proof>

theorem *ack-not-PRIMREC*:

$\neg \text{PRIMREC } (\lambda l. \text{ack } (\text{hd0 } l)\ (\text{hd0 } l))$
<proof>

end

References

- [1] N. Szasz. A machine checked proof that Ackermann's function is not primitive recursive. In G. Huet and G. Plotkin, editors, *Logical Environments*, pages 317–338. Cambridge University Press, 1993.