

Ackermann's Function Is Not Primitive Recursive

Lawrence C. Paulson

February 6, 2026

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	5
1.3 Main Result: Ackermann's Function is not Primitive Recursive	6

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*]: $j < \text{ack } i \ j$
by (*induct* *i* *j* *rule*: *ack.induct*) *simp-all*

PROPERTY A 5-, the single-step lemma

lemma *ack-less-ack-Suc2* [*iff*]: $\text{ack } i \ j < \text{ack } i \ (\text{Suc } j)$
by (*induct* *i* *j* *rule*: *ack.induct*) *simp-all*

PROPERTY A 5, monotonicity for <

lemma *ack-less-mono2*: $j < k \implies \text{ack } i \ j < \text{ack } i \ k$
by (*simp* *add*: *lift-Suc-mono-less*)

PROPERTY A 5', monotonicity for \leq

lemma *ack-le-mono2*: $j \leq k \implies \text{ack } i \ j \leq \text{ack } i \ k$
by (*simp* *add*: *ack-less-mono2 less-mono-imp-le-mono*)

PROPERTY A 6

lemma *ack2-le-ack1* [*iff*]: $\text{ack } i \ (\text{Suc } j) \leq \text{ack } (\text{Suc } i) \ j$
proof (*induct* *j*)
 case 0 **show** ?*case* **by** *simp*
next
 case (*Suc* *j*) **show** ?*case*
 by (*metis* *Suc* *ack.simps*(3) *ack-le-mono2 le-trans less-ack2 less-eq-Suc-le*)
qed

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

lemma *ack-less-ack-Suc1* [*iff*]: $\text{ack } i \ j < \text{ack } (\text{Suc } i) \ j$
by (*blast* *intro*: *ack-less-mono2 less-le-trans*)

lemma *less-ack1* [*iff*]: $i < \text{ack } i \ j$
by (*induct* *i*) (*auto* *intro*: *less-trans-Suc*)

PROPERTY A 8

lemma *ack-1* [*simp*]: $\text{ack } (\text{Suc } 0) \ j = j + 2$
by (*induct* *j*) *simp-all*

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

lemma *ack-2* [*simp*]: $ack (Suc (Suc 0)) j = 2 * j + 3$
by (*induct j*) *simp-all*

Added in 2022 just for fun

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

proof (*induct j*)

case 0

then show *?case* **by** *simp*

next

case (*Suc j*)

with *less-le-trans* **show** *?case*

by (*fastforce simp add: power-add algebra-simps*)

qed

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 (*induct i k* *rule: ack.induct*)

case ($1 n$) **show** *?case*

using *less-le-trans* **by** *auto*

next

case ($2 m$) **thus** *?case* **by** *simp*

next

case ($3 m n$) **thus** *?case*

using *ack-less-mono2 less-trans* **by** *fastforce*

qed

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

using *ack-less-mono1-aux less-iff-Suc-add* **by** *auto*

PROPERTY A 7', monotonicity for \leq

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

using *ack-less-mono1 le-eq-less-or-eq* **by** *auto*

PROPERTY A 10

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

proof –

have $ack i1 (ack i2 j) < ack (i1 + i2) (ack (Suc (i1 + i2)) j)$

by (*meson ack-le-mono1 ack-less-mono1 ack-less-mono2 le-add1 le-trans less-add-Suc2 not-less*)

also have $\dots = ack (Suc (i1 + i2)) (Suc j)$

by *simp*

also have $\dots \leq ack (2 + (i1 + i2)) j$

using *ack2-le-ack1 add-2-eq-Suc* **by** *presburger*

finally show *?thesis* .

qed

PROPERTY A 11

lemma *ack-add-bound*: $ack\ i1\ j + ack\ i2\ j < ack\ (4 + (i1 + i2))\ j$
proof –
 have $ack\ i1\ j \leq ack\ (i1 + i2)\ j$ $ack\ i2\ j \leq ack\ (i1 + i2)\ j$
 by (*simp-all add: ack-le-mono1*)
 then have $ack\ i1\ j + ack\ i2\ j < ack\ (Suc\ (Suc\ 0))\ (ack\ (i1 + i2)\ j)$
 by *simp*
 also have $\dots < ack\ (4 + (i1 + i2))\ j$
 by (*metis ack-nest-bound add.assoc numeral-2-eq-2 numeral-Bit0*)
 finally show *?thesis* .
qed

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 –
 have $i + j < ack\ k\ j + ack\ 0\ j$
 using *assms* **by** *auto*
 also have $\dots < ack\ (4 + k)\ j$
 by (*metis ack-add-bound add.right-neutral*)
 finally show *?thesis* .
qed

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  $\Longrightarrow$  listsp PRIMREC fs  $\Longrightarrow$  PRIMREC (COMP g fs)
| PREC: PRIMREC f  $\Longrightarrow$  PRIMREC g  $\Longrightarrow$  PRIMREC (PREC f g)
monos listsp-mono

```

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

```

lemma SC-case: SC l < ack 1 (sum-list l)
unfolding SC-def
by (induct l) (simp-all add: le-add1 le-imp-less-Suc)

```

```

lemma CONSTANT-case: CONSTANT n l < ack n (sum-list l)
by (simp add: CONSTANT-def)

```

```

lemma PROJ-case: PROJ i l < ack 0 (sum-list l)
proof –
  have hd0 (drop i l)  $\leq$  sum-list l
  by (induct l arbitrary: i) (auto simp: drop-Cons' trans-le-add2)
  then show ?thesis
  by (simp add: PROJ-def)
qed

```

COMP case

```

lemma COMP-map-aux:  $\forall f \in \text{set } fs. \exists kf. \forall l. fl < \text{ack } kf \text{ (sum-list } l)$ 
 $\Longrightarrow \exists k. \forall l. \text{sum-list (map } (\lambda f. fl) fs) < \text{ack } k \text{ (sum-list } l)$ 
proof (induct fs)
  case Nil
  then show ?case
  by auto
next
  case (Cons a fs)
  then show ?case
  by simp (blast intro: add-less-mono ack-add-bound less-trans)
qed

```

```

lemma COMP-case:
  assumes 1:  $\forall l. g l < \text{ack } kg \text{ (sum-list } l)$ 
  and 2:  $\forall f \in \text{set } fs. \exists kf. \forall l. fl < \text{ack } kf \text{ (sum-list } l)$ 
  shows  $\exists k. \forall l. \text{COMP } g \text{ fs } l < \text{ack } k \text{ (sum-list } l)$ 
  unfolding COMP-def
  using 1 COMP-map-aux [OF 2] by (meson ack-less-mono2 ack-nest-bound less-trans)

  PREC case

```

```

lemma PREC-case-aux:
  assumes f:  $\bigwedge l. fl + \text{sum-list } l < \text{ack } kf \text{ (sum-list } l)$ 

```

and $g: \bigwedge l. g\ l + \text{sum-list } l < \text{ack } kg\ (\text{sum-list } l)$
shows $\text{PREC } f\ g\ (m\ \#\ l) + \text{sum-list } (m\ \#\ l) < \text{ack } (\text{Suc } (kf + kg))\ (\text{sum-list } (m\ \#\ l))$
proof $(\text{induct } m)$
case 0
then show $?case$
using $\text{ack-less-mono1-aux } f\ \text{less-trans}$ **by** fastforce
next
case $(\text{Suc } m)$
let $?r = \text{PREC } f\ g\ (m\ \#\ l)$
have $\neg\ g\ (?r\ \#\ m\ \#\ l) + \text{sum-list } (?r\ \#\ m\ \#\ l) < g\ (?r\ \#\ m\ \#\ l) + (m + \text{sum-list } l)$
by force
then have $g\ (?r\ \#\ m\ \#\ l) + (m + \text{sum-list } l) < \text{ack } kg\ (\text{sum-list } (?r\ \#\ m\ \#\ l))$
by $(\text{meson } g\ \text{leI } \text{less-le-trans})$
moreover
have $\dots < \text{ack } (kf + kg)\ (\text{ack } (\text{Suc } (kf + kg))\ (m + \text{sum-list } l))$
using Suc.hyps **by** $\text{simp } (\text{meson } \text{ack-le-mono1 } \text{ack-less-mono2 } \text{le-add2 } \text{le-less-trans})$
ultimately show $?case$
by auto
qed

lemma $\text{PREC-case-aux}'$:
assumes $f: \bigwedge l. f\ l + \text{sum-list } l < \text{ack } kf\ (\text{sum-list } l)$
and $g: \bigwedge l. g\ l + \text{sum-list } l < \text{ack } kg\ (\text{sum-list } l)$
shows $\text{PREC } f\ g\ l + \text{sum-list } l < \text{ack } (\text{Suc } (kf + kg))\ (\text{sum-list } l)$
by $(\text{smt } (\text{verit, best})\ \text{PREC.elims } \text{PREC-case-aux } \text{add.commute } \text{add.right-neutral } f\ g\ \text{less-ack2})$

proposition PREC-case :
 $\llbracket \bigwedge l. f\ l < \text{ack } kf\ (\text{sum-list } l); \bigwedge l. g\ l < \text{ack } kg\ (\text{sum-list } l) \rrbracket$
 $\implies \exists k. \forall l. \text{PREC } f\ g\ l < \text{ack } k\ (\text{sum-list } l)$
by $(\text{metis } \text{le-less-trans } [\text{OF } \text{le-add1 } \text{PREC-case-aux}']\ \text{ack-add-bound2})$

lemma $\text{ack-bounds-PRIMREC}$: $\text{PRIMREC } f \implies \exists k. \forall l. f\ l < \text{ack } k\ (\text{sum-list } l)$
by $(\text{erule } \text{PRIMREC.induct})\ (\text{blast } \text{intro: } \text{SC-case } \text{CONSTANT-case } \text{PROJ-case } \text{COMP-case } \text{PREC-case})+$

theorem ack-not-PRIMREC :
 $\neg\ \text{PRIMREC } (\lambda l. \text{ack } (\text{hd0 } l)\ (\text{hd0 } l))$
proof
assume $*$: $\text{PRIMREC } (\lambda l. \text{ack } (\text{hd0 } l)\ (\text{hd0 } l))$
then obtain m **where** $m: \bigwedge l. \text{ack } (\text{hd0 } l)\ (\text{hd0 } l) < \text{ack } m\ (\text{sum-list } l)$
using $\text{ack-bounds-PRIMREC}$ **by** blast
show False
using m $[\text{of } [m]]$ **by** simp
qed

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.