A Functional Proof Pearl: Inverting the Ackermann Hierarchy

Linh Tran, Anshuman Mohan, Aquinas Hobor

The 9th ACM SIGPLAN International Conference on Certified Programs and Proofs
January 20, 2020

The Ackermann Function

The Ackermann-Péter function is defined as:

$$A(n, m) \triangleq \begin{cases} m+1 & \text{when } n = 0 \\ A(n-1, 1) & \text{when } n > 0, m = 0 \\ A\big(n-1, A(n, m-1)\big) & \text{otherwise} \end{cases}$$

The *diagonal* Ackermann function is $\mathcal{A}(n) \triangleq A(n, n)$.

First few values of $\mathcal{A}(n)$:

n	0	1	2	3	4
$\mathcal{A}(n)$	1	3	7	61	$2^{2^{2^{65536}}} - 3$

Explosive growth!

The Inverse Ackermann Function

The *inverse Ackermann function* $\alpha(n) \triangleq \min\{k \in \mathbb{N} : n \leq \mathcal{A}(k)\}$.

$\alpha(n)$ grows slowly but is hard to *compute* for large n
because it is entangled with the explosively-growing $\mathcal{A}(k)$.

Naive Approach: Compute $\mathcal{A}(0), \mathcal{A}(1), \ldots$ until $n \leq \mathcal{A}(k)$. Return k.

Time complexity: $\Omega(\mathcal{A}(\alpha(n)))$.

Computing $\alpha(100) \mapsto^* 4$ requires at least $\mathcal{A}(4) = 2^{2^{2^{65536}}} - 3$ steps!

Engineering hack: Hardcode with lookup tables.
$n > 61 \Rightarrow \text{ans} = 4$. Wrong for large enough inputs.

Our Goal. Compute α for *all inputs* without computing $\mathcal{A}$.

Our Solution

```
Require Import Omega Program.Basics.

Fixpoint cdn_wkr (a : nat) (f : nat -> nat) (n b : nat) :=
 match b with 0 => 0 | S b' =>
  if (n <=? a) then 0 else S (cdn_wkr f a (f n) k')
 end.

Definition countdown_to a f n := cdn_wkr a f n n.

Fixpoint inv_ack_wkr (f : nat -> nat) (n k b : nat) :=
 match b with 0 => 0 | S b' =>
  if (n <=? k) then k else let g := (countdown_to f 1) in
                        inv_ack_wkr (compose g f) (g n) (S k) b
 end.

Definition inv_ack_linear (n : nat) : nat :=
 match n with 0 | 1 => 0 | _ =>
  let f := (fun x => x - 2) in inv_ack_wkr f (f n) 1 (n - 1)
 end.
```

Introduction: Ackermann *vs* Hyperoperation

Treating b as the main argument reveals similarities between $A(n,b)$ and the *hyperoperations* $a[n]b$, while allows the notion of *inverse functions*.

n	$a[n]b$	$A(n,b)$	$a\langle n\rangle b$	$\alpha_n(b)$
0	$1+b$	$1+b$	$b-1$	$b-1$
1	$a+b$	$2+b$	$b-a$	$b-2$
2	$a\cdot b$	$2b+3$	$\left\lceil \frac{b}{a} \right\rceil$	$\left\lceil \frac{b-3}{2} \right\rceil$
3	a^b	$2^{b+3}-3$	$\lceil \log_a b \rceil$	$\lceil \log_2 (b+3) \rceil - 3$
4	$\underbrace{a^{\cdot^{\cdot^{\cdot^{a}}}}}_{b}$	$\underbrace{2^{\cdot^{\cdot^{\cdot^{2}}}}}_{b+3}-3$	$\log_a^* b$	$\log_2^* (b+3) - 3$

Connection? $A(n,b) = 2[n](b+3)-3$ and $\alpha_n(b) = 2\langle n\rangle(b+3)-3$.

Introduction: Inverse Hierarchies to Inverse Ackermann

We explore the upper inverse relation:

$$\begin{cases} \forall b.\forall c. \quad b \le \mathcal{A}_n(c) & \Leftrightarrow \quad \alpha_n(b) \le c \\ \forall b.\forall c. \quad b \le a[n]c & \Leftrightarrow \quad a\langle n\rangle b \le c \end{cases}$$

Redefine α: $\alpha(n) = \min\{k : n \le \mathcal{A}_k(k)\} = \min\{k : \alpha_k(n) \le k\}$.

Computing α through α_i! No need to go through $\mathcal{A}$.

Goal. Build the inverse towers independent from the original towers.

Roadmap

$\boxed{\textbf{Goal. Inverting } \mathcal{A} \textbf{ - without computing } \mathcal{A}\textbf{.}}$

Step 1. Build the hyperoperations/Ackermann hierarchies via **Repeater**.

Step 2. Build inverses of the hyperoperations/Ackermann hierarchies via **Countdown**, the inverse of *Repeater*.

Step 3. Use the hierarchy to implement the inverse Ackermann function for inputs encoded in unary. $O(n)$.

Bonus. Improve efficiency for inputs encoded in binary. $O(\log_2 n)$.

Hierarchical Relation from Recursive Rule

Let's look at the recursive rules for the Ackermann function and the hyperoperations.

Hyperoperations: $a[n+1](b+1) \triangleq a[n]\big(a[n+1]b\big)$

Ackermann function: $A(n+1, b+1) \triangleq A\big(n, A(n+1, b)\big)$

Indexing $\mathcal{A}_n \triangleq \lambda b.A(n, b)$: $\mathcal{A}_{n+1}(b+1) = \mathcal{A}_n\big(\mathcal{A}_{n+1}(b)\big)$.

The next level can be built from the previous level!

Building the Next Level

$$
\begin{aligned}
& a[n+1]b \\
= \;& a[n]\big(a[n+1](b-1)\big) \\
= \;& \big(a[n] \circ a[n]\big)\big(a[n+1](b-2)\big) \\
= \;& \dots \\
= \;& \underbrace{\big(a[n] \circ \cdots \circ a[n]\big)}_{b \text{ times}} \big(a[n+1]0\big) \\
= \;& \underbrace{\big(a[n]\big)^{(b)}}_{\substack{\text{repeated} \\ \text{applications}}} \underbrace{\big(a[n+1]0\big)}_{\text{initial value}}
\end{aligned}
$$

$$
\begin{aligned}
& \mathcal{A}_{n+1}(b) \\
& \mathcal{A}_n\big(\mathcal{A}_{n+1}(b-1)\big) \\
& \mathcal{A}_n\big(\mathcal{A}_n\big(\mathcal{A}_{n+1}(b-2)\big)\big) \\
& \dots \\
& \underbrace{\big(\mathcal{A}_n \circ \cdots \circ \mathcal{A}_n\big)}_{b \text{ times}} \big(\mathcal{A}_{n+1}(0)\big) \\
& \underbrace{\big(\mathcal{A}_n\big)^{(b)}}_{\substack{\text{repeated} \\ \text{applications}}} \underbrace{\big(\mathcal{A}_{n+1}(0)\big)}_{\text{initial value}}
\end{aligned}
$$

Mechanizing Our Observations

Repeater. $f_u^{\mathcal{R}}(b) \quad = \quad f^{(b)}(u) \quad \approx \quad \mathrm{iter}(b, f, u)$.
Read $f_a^{\mathcal{R}}$ as "the *repeater from a* of *f*".

```
Fixpoint repeater_from (f : nat -> nat) (a n : nat) : nat :=
match n with 0 => a | S n' => f (repeater_from f a n') end.
```

Drop b to form higher-order recursive rule between levels:

$$\begin{cases} a[n+1] & = \left(a[n]\right)^{\mathcal{R}}_{a[n+1]0} \\ \mathcal{A}_{n+1} & = \left(\mathcal{A}_n\right)^{\mathcal{R}}_{\mathcal{A}_{n+1}(0)} \end{cases}$$

Hyperoperations via Repeater

Without Repeater (via double recursion):

```
Definition hyperop_init (a n : nat) : nat :=
  match n with 0 => a | 1 => 0 | _ => 1 end.

Fixpoint hyperop_original (a n b : nat) : nat :=
  match n with
  | 0    => 1 + b
  | S n' => let fix hyperop' (b : nat) := match b with
                | 0    => hyperop_init a n'
                | S b' => hyperop_original a n' (hyperop' b')
                end in hyperop' b
  end.
```

With Repeater:

```
Fixpoint hyperop (a n b : nat) : nat :=
  match n with
  | 0    => 1 + b
  | S n' => repeater_from (hyperop a n') (hyperop_init a n') b
  end.
```

Ackermann via Repeater

Without Repeater (via double recursion):

```
Fixpoint ackermann_original (m n : nat) : nat :=
  match m with
  | 0    => 1 + n
  | S m' => let fix ackermann' (n : nat) : nat := match n with
                | 0    => ackermann_original m' 1
                | S n' => ackermann_original m' (ackermann' n')
               end in ackermann' n
  end.
```

With Repeater:

```
Fixpoint ackermann (n m : nat) : nat :=
  match n with
  | 0    => S m
  | S n' => repeater_from (ackermann n') (ackermann n' 1) m
  end.
```

Roadmap

$\boxed{\textbf{Goal.}\ \text{Inverting } \mathcal{A} \text{ - without computing } \mathcal{A}.}$

Step 1. Build the hyperoperations/Ackermann hierarchies via **Repeater**.

Step 2. Build inverses of the hyperoperations/Ackermann hierarchies via **Countdown**, the inverse of *Repeater*.

Step 3. Use the hierarchy to implement the inverse Ackermann function for inputs encoded in unary. $\mathrm{O}(n)$.

Bonus. Improve efficiency for inputs encoded in binary. $\mathrm{O}(\log_2 n)$.

Repeatable Functions

Functions in the Ackermann and hyperoperation (when $a \geq 2$) hierarchies are all *repeatable function*.

Repeatable functions: A F is *repeatable* from a if
F is strictly increasing and F is an *expansion* that is *strict from* a, i.e. $\forall n \geq a.\ F(n) > n$.

We extend our scope of study from functions in the hyperoperations and Ackermann hierarchies to repeatable functions.

Advantage. If F is repeatable from a, F^{-1} makes sense and is total, and $F_a^{\mathcal{R}}$ is repeatable from 1.

Inverting Repeater: The Idea of Countdown

The *upper inverse* of F, written F^{-1}, is $\lambda n. \min\{m : F(m) \geq n\}$.

Logical equivalence (more useful): If $F : \mathbb{N} \to \mathbb{N}$ is increasing, then $f = F^{-1}$ iff $\forall n, m.\ f(n) \leq m \Leftrightarrow n \leq F(m)$.

Idea. Build $\left(F_a^{\mathcal{R}}\right)^{-1}$ from $f \triangleq F^{-1}$.

$$\begin{array}{rlll} & \left(F_a^{\mathcal{R}}\right)^{-1}(n) \leq m & \Leftrightarrow\ n \leq F_a^{\mathcal{R}}(m) & \Leftrightarrow n \leq F^{(m)}(a) \\ \Leftrightarrow & f(n) \leq F^{(m-1)}(a) & \Leftrightarrow\ \cdots & \Leftrightarrow f^{(m)}(n) \leq a \end{array}$$

$\boxed{\left(F_a^{\mathcal{R}}\right)^{-1}(n) \text{ is } \textit{the least } m \text{ for which } f^{(m)}(n) \leq a.}$

Formalizing countdown

Does such *m* exists?
Yes because f is a **contraction** when F is repeatable!

Contraction. A function $f : \mathbb{N} \to \mathbb{N}$ is a *contraction* if $\forall n.\ f(n) \leq n$.
Given an $a \geq 1$, a contraction f is *strict above* a if $\forall n > a.\ n > f(n)$.

Countdown. Let $f \in \text{CONT}_a$. The *countdown to a* of f, written $f_a^{\mathcal{C}}(n)$, is the smallest number of times f needs to be compositionally applied to n for the answer to equal or go below a. *i.e.*,

$$f_a^{\mathcal{C}}(n) \triangleq \min\{m : f^{(m)}(n) \leq a\}.$$

Theorem. If F is repeatable from a, then $\left(F_a^{\mathcal{R}}\right)^{-1} = \left(F^{-1}\right)_a^{\mathcal{C}}$.

A Coq Computation of Countdown

Idea. Compute $f^k(n)$ for $k = 0, 1, \ldots$ until $f^{(k)}(n) \leq a$. Return k.

Primary issue. Termination in Coq.

Secondary issue. It's hard to restrict f to contractions only.

The worker function. $\begin{cases} \text{Budget } b: & \text{Maximum } b \text{ steps.} \\ \text{Step } i: & \text{Compute } f^{(i)}(n). \\ \text{Stops when:} & \text{budget reaches 0 or } f^{(i)}(n) \leq a. \end{cases}$

Primary issue. How to determine a sufficient budget?

A Coq Computation of Countdown

The worker. The *countdown worker* to a of f is a function $f_a^{\mathcal{CW}}$:

$$f_a^{\mathcal{CW}}(n, b) = \begin{cases} 0 & \text{if } b = 0 \lor n \leq a \\ 1 + f_a^{\mathcal{CW}}(f(n), b-1) & \text{if } b \geq 1 \land n > a \end{cases}$$

```
Fixpoint cdn_wkr (a : nat) (f : nat -> nat) (n b : nat) : nat :=
  match b with 0 => 0 | S b' =>
    if (n <=? a) then 0 else S (cdn_wkr f a (f n) k')
end.
```

The Countdown. Budget $b = n$ is sufficient.

Redefine $f_a^{\mathcal{C}}(n) \triangleq f_a^{\mathcal{CW}}(n, n)$.

```
Definition countdown_to a f n := cdn_wkr a f n n.
```

The Inverse Hyperoperation Hierarchy

The inverse hyperoperations, written $a\langle n\rangle b$, are defined as:

$$a\langle n\rangle b \triangleq \begin{cases} b-1 & \text{if } n=0 \\ a\langle n-1\rangle^{\mathcal{C}}_{a_n}(b) & \text{if } n\geq 1 \end{cases} \quad \text{where} \quad a_n = \begin{cases} a & \text{if } n=1 \\ 0 & \text{if } n=2 \\ 1 & \text{if } n\geq 3 \end{cases}$$

```
Fixpoint inv_hyperop (a n b : nat) : nat :=
  match n with 0 => b - 1 | S n' =>
    countdown_to (hyperop_init a n') (inv_hyperop a n') b
  end.
```

Interesting individual levels: $a\langle 2\rangle b = \lceil b/a\rceil$, $a\langle 3\rangle b = \lceil \log_a b\rceil$, and $a\langle 4\rangle b = \log^*_a b$ (not currently in Coq standard library).

The Inverse Ackermann Hierarchy

Naive approach. $\mathcal{A}_{i+1} = (\mathcal{A}_i)^{\mathcal{R}}_{\mathcal{A}_i(1)}$. Thus $\alpha_{i+1} \triangleq (\alpha_i)^{\mathcal{C}}_{\mathcal{A}_i(1)}$?

Flaw! α_{i+1} depends on $\mathcal{A}_i$.

Observation. $\mathcal{A}_{i+1}(n) = \mathcal{A}_i^{(n)}(\mathcal{A}_i(1)) = \mathcal{A}_i^{(n+1)}(1)$. Thus

$$\alpha_{i+1}(n) = \min\left\{m : n \le \mathcal{A}_i^{m+1}(1)\right\} = \min\left\{m : (\alpha_i)^{(m+1)}(n) \le 1\right\}$$
$$= \min\left\{m : (\alpha_i)^{(m)}(\alpha_i(n)) \le 1\right\} = (\alpha_i)^{\mathcal{C}}_1(\alpha_i(n))$$

Redefine: $\alpha_i \triangleq \begin{cases} \lambda n.(n-1) & \text{when } i = 0 \\ \left(\alpha_{i-1}{}^{\mathcal{C}}_1\right) \circ \alpha_{i-1} & \text{when } i \ge 1 \end{cases}$

```
Fixpoint alpha (m x : nat) : nat :=
  match m with 0 => x - 1 | S m' =>
    countdown_to 1 (alpha m') (alpha m' x)
end.
```

Roadmap

$\boxed{\textbf{Goal. } \text{Inverting } \mathcal{A} \text{ - without computing } \mathcal{A}.}$

Step 1. Build the hyperoperations/Ackermann hierarchies via **Repeater**.

Step 2. Build inverses of the hyperoperations/Ackermann hierarchies via **Countdown**, the inverse of *Repeater*.

Step 3. Use the hierarchy to implement the inverse Ackermann function for inputs encoded in unary. $O(n)$.

Bonus. Improve efficiency for inputs encoded in binary. $O(\log_2 n)$.

Implementation Idea

Redefinition. $\alpha(n) = \min\{k : n \leq A(k,k)\} \triangleq \min\{k : \alpha_k(n) \leq k\}$

The worker function. $\begin{cases} \text{Budget } b: & \text{Maximum } b \text{ steps.} \\ \text{Step } i: & \text{Compute } \alpha_i(n). \\ \text{Stops when:} & \text{budget reaches 0 or } \alpha_i(n) \leq i. \end{cases}$

Advantage. $\alpha_{i+1}(n) = (\alpha_i)_1^{\mathcal{C}}(\alpha_i(n))$

So the next step is computable from the previous.

The Worker Function

$$\alpha^{\mathcal{W}}(f, n, k, b) = \begin{cases} k & \text{if } b = 0 \vee n \leq k \\ \alpha^{\mathcal{W}}(f_1^{\mathcal{C}} \circ f, f_1^{\mathcal{C}}(n), k+1, b-1) & \text{if } b \geq 1 \wedge n \geq k+1 \end{cases}$$

```
Fixpoint inv_ack_wkr (f : nat -> nat) (n k b : nat) : nat :=
match b with 0 => 0 | S b' =>
  if (n <=? k) then k else let g := (countdown_to f 1) in
                           inv_ack_wkr (compose g f) (g n) (S k) b
end.
```

The Worker Function

Observation.

$$\begin{array}{llllll}
\textit{Initial Arguments} & & (\alpha_i, & \alpha_i(n), & i, & b-i) \\
b-i>0,\ \alpha_i(n)>i & \rightarrow & (\alpha_{i1}^{\mathcal{C}} \circ \alpha_i, & \alpha_{i1}^{\mathcal{C}}(\alpha_i(n)), & i+1, & b-i-1) \\
b>i,\quad \alpha_i(n)>i & \rightarrow & (\alpha_{i+1}, & \alpha_{i+1}(n), & i+1, & b-(i+1))
\end{array}$$

Execution.

$$\begin{array}{l|llllll}
\textit{Step} & \textit{Initial Arguments} & & (\alpha_0, & \alpha_0(n), & 0, & b-0) \\
0 & b>0,\ \alpha_0(n)>0 & \rightarrow & (\alpha_1, & \alpha_1(n), & 1, & b-1) \\
1 & b>1,\ \alpha_1(n)>1 & \rightarrow & (\alpha_2, & \alpha_2(n), & 2, & b-2) \\
\vdots & \vdots & \vdots & & \vdots & & \\
k & b>k,\ \alpha_k(n)\leq k & \rightarrow & \boxed{k=\alpha(n)} & & &
\end{array}$$

The Inverse Ackermann Function

Any budget $b \geq \alpha(n)$ suffices, so we can choose $b = n$.

First version: $\mathrm{O}(n^2)$.

$$\alpha(n) = \alpha^{\mathcal{W}}(\alpha_0, \alpha_0(n), 0, n) = \alpha^{\mathcal{W}}(\lambda n(n-1), n-1, 0, n)$$

Simple improvement: $\mathrm{O}(n)$.

$$\alpha(n) = \begin{cases} \alpha^{\mathcal{W}}(\alpha_1, \alpha_1(n), 1, n-1) & \text{when } n > \mathcal{A}(0) \\ 0 & \text{when } n \leq \mathcal{A}(0) \end{cases}$$

$$= \begin{cases} \alpha^{\mathcal{W}}(\lambda n(n-2), n-2, 1, n-1) & \text{when } n > 1 \\ 0 & \text{when } n \leq 1 \end{cases}$$

Time Complexity of α: A Sketch

Let $\mathcal{T}_f(n)$ denotes the running time of computing $f(n)$ given f and n.

$$\mathcal{T}_{\alpha_{i+1}}(n) \approx \mathcal{T}_{\alpha_i}(n) + \mathcal{T}_{\alpha_i}\big(\alpha_i(n)\big) + \mathcal{T}_{\alpha_i}\big(\alpha_i^{(2)}(n)\big) + \ldots (\text{until } 1)$$

Manual computation: $\mathcal{T}_{\alpha_3}(n) \leq 4n + 4$.
Suppose $\mathcal{T}_{\alpha_i}(n) \leq 4n + \mathrm{O}\big(\log_2 n\big)$. Then

$$\begin{aligned}\mathcal{T}_{\alpha_{i+1}}(n) &\lessapprox 4n + \mathrm{O}\big(\log_2 n\big) + 4\log_2 n + \mathrm{O}\big(\log_2^{(2)} n\big) + \ldots \\ &= 4n + \mathrm{O}\left(\log_2 n + \log_2^{(2)} n + \log_2^{(3)} n + \ldots\right) = 4n + \mathrm{O}\big(\log_2 n\big)\end{aligned}$$

Therefore, $\boxed{T_\alpha(n) \approx \mathcal{T}_{\alpha_{\alpha(n)}}(n) \lessapprox 4n + \mathrm{O}\big(\log_2 n\big) \lessapprox \Theta(n)}$ by induction.

Roadmap

$\boxed{\textbf{Goal.}\ \text{Inverting } \mathcal{A} \text{ - without computing } \mathcal{A}.}$

Step 1. Build the hyperoperations/Ackermann hierarchies via **Repeater**.

Step 2. Build inverses of the hyperoperations/Ackermann hierarchies via **Countdown**, the inverse of *Repeater*.

Step 3. Use the hierarchy to implement the inverse Ackermann function for inputs encoded in unary. $O(n)$.

Bonus. Improve efficiency for inputs encoded in binary. $O(\log_2 n)$.

Countdown in Binary

Our inverse Ackermann implementation runs in $O(n)$ time for unary n.
What about binary input? $\boxed{\textbf{Goal: } \mathrm{O}\left(\log_2 n\right)}$.

Translating Countdown. Recursive argument for worker is budget b, which should still be in nat.

Issue. b cannot be $O(n)$ due to slow binary $\to$ unary conversion.

Solution. Use $b \approx \log_2 n$ and contractions that contract "fast enough".
i.e. Functions that halve their inputs.

Binary Contractions and Countdown

Binary contractions. f is a *binary contraction* strict above a if $\forall n, f(n) \leq n$ and $f(n) \leq \lfloor \frac{n+a}{2} \rfloor$ when $n > a$.

Observation. f shrinks n past a within $\lfloor \log_2(n - a) \rfloor + 1$ steps.

New Countdown computation:

```
Fixpoint bin_cdn_wkr (f : N -> N) (a n : N) (b : nat) : N :=
  match b with O => 0 | S b' =>
    if (n <=? a) then 0 else 1 + bin_cdn_wkr f a (f n) b'
  end.

Definition bin_countdown_to (f : N -> N) (a n : N) : N :=
bin_cdn_wkr f a n (nat_size (n - a)).
```

where nat_size x computes $\lfloor \log_2 x \rfloor + 1$ as a nat.

Translating the α Hierarchy

Must start with a strict binary contraction.

More hard-coding to skip those that are not fast enough.

```
Fixpoint bin_alpha (m : nat) (x : N) : N :=
match m with
  | 0%nat => x - 1
  | 1%nat => x - 2
  | 2%nat => N.div2 (x - 2)
  | 3%nat => N.log2 (x + 2) - 2
  | S m'  => bin_countdown_to (bin_alpha m') 1 (bin_alpha m' x)
end.
```

Translating the Inverse Ackermann Function

Worker function:

```
Fixpoint bin_inv_ack_wkr (f : N -> N) (n k : N) (b : nat) : N :=
match b with 0%nat => k | S b' => if n <=? k then k else
  let g := (bin_countdown_to f 1) in
    bin_inv_ack_wkr (compose g f) (g n) (N.succ k) b'
  end.
```

Same idea: use logarithmic size budget. More hard-coding.

```
Definition bin_inv_ack (n : N) : N :=
if      (n <=? 1) then 0
else if (n <=? 3) then 1
else if (n <=? 7) then 2
else let f := (fun x => N.log2 (x + 2) - 2)
      in bin_inv_ack_wkr f (f n) 3 (nat_size n).
```

Time Complexity of Binary α: A Sketch

Similar to α on unary inputs, $\mathcal{T}_\alpha(n) \approx \mathcal{T}_{\alpha_k}(n)$ for $k \triangleq \alpha(n)$.

Manual computation: $\mathcal{T}_{\alpha_3}(n) \leq 2\log_2 n + \log_2 \log_2 n + 3$.

Suppose $\mathcal{T}_{\alpha_i}(n) \leq 2\log_2 n + \mathrm{O}\left(\log_2^{(2)} n\right)$. Then

$$\begin{aligned}\mathcal{T}_{\alpha_{i+1}}(n) \ &\lessapprox \ 2\log_2 n + \mathrm{O}\left(\log_2^{(2)} n\right) + 2\log_2^{(2)} n + \mathrm{O}\big(\log_2^{(3)} n\big) + \dots \\ &= 2\log_2 n + \mathrm{O}\left(\log_2^{(2)} n + \log_2^{(3)} n + \log_2^{(4)} n + \dots\right) = 2\log_2 n + \mathrm{O}\left(\log_2^{(2)} n\right)\end{aligned}$$

By induction, $\boxed{\mathcal{T}_\alpha(n) \approx \mathcal{T}_{\alpha_k}(n) \lessapprox 2\log_2 n + \mathrm{O}\left(\log_2^{(2)} n\right) \lessapprox \Theta(\log_2 n)}$.

Roadmap

$\boxed{\textbf{Goal.}\ \text{Inverting } \mathcal{A} \text{ - without computing } \mathcal{A}.}$

Step 1. Build the hyperoperations/Ackermann hierarchies via **Repeater**.

Step 2. Build inverses of the hyperoperations/Ackermann hierarchies via **Countdown**, the inverse of *Repeater*.

Step 3. Use the hierarchy to implement the inverse Ackermann function for inputs encoded in unary. $O(n)$.

Bonus. Improve efficiency for inputs encoded in binary. $O(\log_2 n)$.

Conclusion

We inverted the hyperoperation and Ackermann hierarchies and implemented our inverse hierarchies in Gallina. We used our inverses to compute the upper inverse of the diagonal Ackermann function.

Our computations run in linear time – $\Theta(b)$, where b=bitlength – for inputs represented in both unary and binary.

We showed that these functions are consistent with the usual definition of the inverse Ackermann function $\alpha(n)$.

Demo

We present a brief demonstration of our time bounds in Coq.

Correctness and Runtime of Countdown

Sum by component in each recursive step.

Step	*Initial Arguments*			$\left(f^{(0)}(n),\right.$	$\left. n-0\right)$	
0	$n-0>0,$	$f^{(0)}(n)>a$	$\to$	$\left(f^{(1)}(n),\right.$	$\left. n-1\right)$	$+1$
1	$n-1>0,$	$f^{(1)}(n)>a$	$\to$	$\left(f^{(2)}(n),\right.$	$\left. n-2\right)$	$+2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k-1$	$n-(k-1)>0,$	$f^{(k-1)}(n)>a$	$\to$	$\left(f^{(k)}(n),\right.$	$\left. n-k\right)$	$+k$
k	$n-k\ ?\ 0,$	$f^{(k)}(n)\le a$	$\to$	0		$k=f_a^{\mathcal{C}}(n)$
SUM	$\Theta(k)$	$\Theta((a+1)k)$		$\sum_{i=0}^{k-1}\mathcal{T}_f(f^i(n))$		$\Theta(k)$

Substitute $k=f_a^{\mathcal{C}}(n)$: $\boxed{\mathcal{T}_{f_a^{\mathcal{C}}}(n)=\sum_{i=0}^{f_a^{\mathcal{C}}(n)-1}\mathcal{T}_f(f^i(n))+\Theta\left((a+1)f_a^{\mathcal{C}}(n)\right)}$

Runtime of Each α_i (Unary Inputs, Asymptotic Bounds)

$$\alpha_{i+1} = (\alpha_i)_1^{\mathcal{C}} \circ \alpha_i \quad \Rightarrow \quad \mathcal{T}_{\alpha_{i+1}}(n) = \mathcal{T}_{\alpha_i}(n) + \mathcal{T}_{\alpha_{i1}^{\mathcal{C}}}(\alpha_i(n))$$

$$\boxed{\mathcal{T}_{\alpha_{i+1}}(n) = \mathcal{T}_{\alpha_i}(n) + \sum_{i=0}^{\alpha_{i1}^{\mathcal{C}}(\alpha_i(n))-1} \mathcal{T}_{\alpha_i}\left(\alpha_i^{(i+1)}(n)\right) + \Theta\left(\alpha_{i1}^{\mathcal{C}}(\alpha_i(n))\right)}$$

Substitute $\alpha_{i1}^{\mathcal{C}}(\alpha_i(n))$ for $\alpha_{i+1}(n)$:

$$\mathcal{T}_{\alpha_{i+1}}(n) = \mathcal{T}_{\alpha_i}(n) + \sum_{i=0}^{\alpha_{i+1}(n)-1} \mathcal{T}_{\alpha_i}\left(\alpha_i^{(i+1)}(n)\right) + \Theta\left(\alpha_{i+1}(n)\right)$$

$$\boxed{\mathcal{T}_{\alpha_{i+1}}(n) = \sum_{i=0}^{\alpha_{i+1}(n)} \mathcal{T}_{\alpha_i}\left(\alpha_i^{(i)}(n)\right) + \Theta\left(\alpha_{i+1}(n)\right)}$$

Runtime of Each α_i (Unary Inputs, Precise Bounds)

Countdown runtime:

$$\mathcal{T}_{f_a^{\mathcal{C}}}(n) = \sum_{i=0}^{f_a^{\mathcal{C}}(n)-1} \mathcal{T}_f\left(f^{(i)}(n)\right) + (a+2) f_a^{\mathcal{C}}(n) + f^{\left(f_a^{\mathcal{C}}(n)\right)}(n) + 1$$

α_2 and α_3 runtime: $\mathcal{T}_{\alpha_2}(n) \le 2n - 2$ and $\mathcal{T}_{\alpha_3}(n) \le 4n + 4$.

α_i runtime: $\mathcal{T}_{\alpha_{i+1}}(n) \le \sum_{k=0}^{\alpha_{i+1}(n)} \mathcal{T}_{\alpha_i}\left(\alpha_i^{(k)}(n)\right) + 3\alpha_{i+1}(n) + 3$.

Theorem. $\forall i.\ \mathcal{T}_{\alpha_i}(n) \le 4n + \left(19 \cdot 2^{i-3} - 2i - 13\right) \log_2 n + 2i = O(n)$, when using $\alpha_1 \triangleq \lambda n.(n-2)$.

Runtime of Inverse Ackermann for Unary Inputs

Step	Initial Arguments			
				$(\alpha_1,\ \alpha_1(n),\ 1,\ b-1)$
1	$b-1>0,$	$\alpha_1(n)>1$	$\to$	$(\alpha_2,\ \alpha_2(n),\ 2,\ b-2)$
2	$b-2>0,$	$\alpha_2(n)>2$	$\to$	$(\alpha_3,\ \alpha_3(n),\ 3,\ b-3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots\quad\vdots\quad\vdots$
k	$b-k>0,$	$\alpha_k(n)\le k$	$\to$	$k=\alpha(n)$
SUM	$\Theta(k)$	$\Theta\left(\sum_{i=1}^{k} i\right)$		$\sum_{i=1}^{k-1}\mathcal{T}_{\alpha_{i_1}^{c}}(\alpha_i(n))$
$=$	$\Theta(k)$	$\Theta(k^2)$		$\sum_{i=1}^{k-1}\left(\mathcal{T}_{\alpha_{i+1}}(n)-\mathcal{T}_{\alpha_i}(n)\right)$

Therefore,

$$\boxed{\begin{aligned}\mathcal{T}_\alpha(n) &= \mathcal{T}_{\alpha_{\alpha(n)}}(n)-\mathcal{T}_{\alpha_1}(n)+\Theta\left(\alpha(n)^2\right)+\mathcal{T}_{\alpha_1}(n)\\ &= O\left(n+2^{\alpha(n)}\log_2 n+\alpha(n)^2\right)=O(n)\end{aligned}}$$

Runtime of Each α_i (Binary Inputs, Precise Bounds)

Countdown runtime:

$$\mathcal{T}_{f_a^{\mathcal{C}}}(n) \le \sum_{i=0}^{f_a^{\mathcal{C}}(n)-1} \mathcal{T}_f\big(f^{(i)}(n)\big) + (\log_2 a+3)\left(f_a^{\mathcal{C}}(n)+1\right) + 2\log_2 n + \log_2 f_a^{\mathcal{C}}(n)$$

α_3 runtime: $\mathcal{T}_{\alpha_3}(n) \le 2\log_2 n + \log_2\log_2 n + 3$.

α_i runtime: $\mathcal{T}_{\alpha_{i+1}}(n) \le \sum_{k=0}^{\alpha_{i+1}(n)} \mathcal{T}_{\alpha_i}\big(\alpha_i^{(k)}(n)\big) + 6\log_2\log_2 n + 3$.

Theorem.
$\forall i.\ \mathcal{T}_{\alpha_i}(n) \le 2\log_2 n + \left(3\cdot 2^i - 3i - 13\right)\log_2\log_2 n + 3i = O(\log_2 n)$,
when hard-coding up to α_3.

Runtime of Inverse Ackermann for Binary Inputs

Step	*Initial Arguments*			
				$(\alpha_3,\ \alpha_3(n),\ b-0,\ 3)$
1	$b-0>0,$	$\alpha_3(n)>3$	$\rightarrow$	$(\alpha_4,\ \alpha_4(n),\ b-1,\ 4)$
2	$b-1>0,$	$\alpha_4(n)>4$	$\rightarrow$	$(\alpha_5,\ \alpha_5(n),\ b-2,\ 5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots\ \ \vdots\ \ \vdots\ \ \vdots$
k	$b-k>0,$	$\alpha_k(n)\le k$	$\rightarrow$	$k=\alpha(n)$
SUM	$\Theta(k)$	$\Theta\left(\sum_{i=1}^{k}\log_2 i\right)$		$\sum_{i=1}^{k-1}\mathcal{T}_{\alpha_i^c}(\alpha_i(n))$ $\quad\Theta(k)$
$=$	$\Theta(k)$	$\Theta(k\log_2 k)$		$\sum_{i=1}^{k-1}\left(\mathcal{T}_{\alpha_{i+1}}(n)-\mathcal{T}_{\alpha_i}(n)\right)$ $\quad\Theta(k)$

$$\boxed{\begin{aligned}\mathcal{T}_{\alpha}(n) &= \mathcal{T}_{\alpha_{\alpha(n)}}(n)-\mathcal{T}_{\alpha_3}(n)+\Theta\left(\alpha(n)\log_2\alpha(n)\right)+\mathcal{T}_{\alpha_3}(n)\\ &= O\left(\log_2 n+2^{\alpha(n)}\log_2\log_2 n+\alpha(n)\log_2\alpha(n)\right)=O(\log_2 n)\end{aligned}}$$