A Functional Proof Pearl: Inverting the Ackermann Hierarchy

Aquinas Hobor
Anshuman Mohan
Linh Tran

National University of Singapore

July 18, 2019

Abstract

We implement in Gallina a hierarchy of functions that calculate the upper inverses to the hyperoperation/Ackermann hierarchy. Our functions run in $\Theta(b)$ for inputs expressed in unary, and $O(b^2)$ for inputs expressed in binary ($b =$ bitlength). We use our inverses to define linear-time functions—$\Theta(b)$ for both unary- and binary-represented inputs—that compute the upper inverse of the diagonal Ackermann function $\mathcal{A}(n)$ and show that these functions are consistent with the usual definition of the inverse Ackermann function $\alpha(n)$.

The Ackermann and Inverse Ackermann functions

The Ackermann-Péter function (hereafter just "the" Ackermann function) is written $A : \mathbb{N}^2 \to \mathbb{N}$ and defined as follows:

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

The one-variable *diagonal* Ackermann function $\mathcal{A} : \mathbb{N} \to \mathbb{N}$ is defined as $\mathcal{A}(n) \triangleq A(n, n)$.

The inverse Ackermann function $\alpha(n)$ is the smallest k for which $n \leq \mathcal{A}(k)$, *i.e.*

$$\alpha(n) \triangleq \min \{k \in \mathbb{N} : n \leq \mathcal{A}(k)\}$$

Initial values for $\mathcal{A}(n)$ and $\alpha(n)$

TODO: Value table for $\mathcal{A}(n)$
Grows astronomically fast!

TODO: Value table for $\alpha(n)$
Grows astronomically slow!

Computing $\alpha(n)$

Despite growing extremely slow, $\alpha(n)$ is difficult to compute for large n due to the explosive growth of $\mathcal{A}(k)$.

The Naive Approach: start at $k = 0$, calculate $\mathcal{A}(k)$, compare it to n, and increment k until $n \leq \mathcal{A}(k)$.

Time complexity: $\Omega(\mathcal{A}(\alpha(n)))$, so *e.g.* computing $\alpha(100) \mapsto^* 4$ in this way requires $\mathcal{A}(4) = 2^{2^{2^{65536}}} - 3$ steps!

The hierarchy of Ackermann levels

The Ackermann function is easy to define, but hard to understand. We see it as a sequence of n-indexed functions $\mathcal{A}_n \triangleq \lambda b.A(n, b)$, where for each $n > 0$, $\mathcal{A}_n$ is the result of applying the previous $\mathcal{A}_{n-1}$ b times, with a *kludge*.

To better understand the Ackermann function as a hierarchy and this kludge, we explore the closely-related hyperoperations.

The Ackermann hierarchy and hyperoperations

TODO: Polish, add names of levels to this table without overflowing the page

n	$a[n]b$	$\mathcal{A}_n(b)$	$a\langle n\rangle b$	$\alpha_n(b)$
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	$\underbrace{a^{\cdot^{\cdot^{\cdot^{a}}}}}_{b}$	$\underbrace{2^{\cdot^{\cdot^{\cdot^{2}}}}}_{b+3} - 3$	$\log_a^* \ b$	$\log_2^* \ (b+3) - 3$

The kludge: $\mathcal{A}_n(b) = 2[n](b+3) - 3$ and $\alpha_n(b) = 2\langle n\rangle(b+3) - 3$.

Roadmap

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

Step 1. Explore the hyperoperations/Ackermann function hierarchical structure: Connect consecutive levels with **Repeater**.

Step 2. Invert each level in both hierarchies:

- *What is inverse?* Upper inverse and increasing functions.
- *Can Repeater preserve Invertibility?* Repeatable functions.
- *Computing inverse with inverse:* Contractions and **Countdown**.
- Invert each level in hyperoperations/Ackermann hierarchies.

Step 3. Implement the Inverse Ackermann function via the inverse Ackermann hierarchy.

Step 4. Optimize its time complexity.

Roadmap

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

Step 1. Explore the hyperoperations/Ackermann function hierarchical structure: Connect consecutive levels with **Repeater**.

Step 2. Invert each level in both hierarchies:

- *What is inverse?* Upper inverse and increasing functions.
- *Can Repeater preserve Invertibility?* Repeatable functions.
- *Computing inverse with inverse:* Contractions and **Countdown**.
- Invert each level in hyperoperations/Ackermann hierarchies.

Step 3. Implement the Inverse Ackermann function via the inverse Ackermann hierarchy.

Step 4. Optimize its time complexity.

Step 1: Hyperoperations and Ackermann via Repeater

Repeated Application

Let X be any set and $f : X \to X$ be a function on X. Define the notation:

$$f^{(k)}(u) \triangleq (f \circ f \circ \cdots \circ f)(u),$$

which denotes k compositional applications of a function f to an input u.

The following recursive rule applies:

1. $f^{(0)}(u) = u$ (*i.e.* applying 0 times yields the identity).
2. $f^{(k+1)}(u) = f\left(f^{(k)}(u)\right)$.

Repeated application plays a vital role in both hyperoperations and Ackermann hierarchy.

The hyperoperation formal definition

$$
\begin{array}{lrcl}
\textit{1. } 0^{th} \textit{ level:} & a[0]b & \triangleq & b+1 \\
\textit{2. Initial values:} & a[n+1]0 & \triangleq & \begin{cases} a & \text{when } n=0 \\ 0 & \text{when } n=1 \\ 1 & \text{otherwise} \end{cases} \\
\textit{3. Recursive rule:} & a[n+1](b+1) & \triangleq & a[n]\big(a[n+1]b\big)
\end{array}
$$

Via the recursive rule:

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

The Ackermann hierarchy formal definition

$$\begin{array}{llcl} \textit{1. } 0^{th} \textit{ level:} & \mathcal{A}_0(b) & \triangleq & b+1 \\ \textit{2. Initial values:} & \mathcal{A}_{n+1}(0) & \triangleq & \mathcal{A}_n(1) \\ \textit{3. Recursive rule:} & \mathcal{A}_{n+1}(b+1) & \triangleq & \mathcal{A}_n\big(\mathcal{A}_{n+1}(b)\big) \end{array}$$

Via the recursive rule:

$$\begin{aligned} & \mathcal{A}_{n+1}(b) \\ = \;& \mathcal{A}_n\big(\mathcal{A}_{n+1}(b-1)\big) \;=\; \mathcal{A}_n\Big(\mathcal{A}_n\big(\mathcal{A}_{n+1}(b-2)\big)\Big) \\ = \;& \underbrace{\big(\mathcal{A}_n \circ \mathcal{A}_n \circ \cdots \circ \mathcal{A}_n\big)}_{b \text{ times}}\big(\mathcal{A}_{n+1}(0)\big) \;=\; \big(\mathcal{A}_n\big)^{(b)}\underbrace{\big(\mathcal{A}_{n+1}(0)\big)}_{\text{init value}} \end{aligned}$$

From repeated application to Repeater

The next level in the hyperoperations/Ackermann hierarchy is the result of b compositional applications of the current level to an initial value.

We can abstract the concept of repeated application in a higher-order function called *repeater*:

$\forall a \in \mathbb{N}, f : \mathbb{N} \to \mathbb{N}$, the *repeater from* a of f, denoted by $f_a^{\mathcal{R}}$, is a function $\mathbb{N} \to \mathbb{N}$ such that $f_a^{\mathcal{R}}(n) = f^{(n)}(a)$.

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

Functional-to-function recursive rule: $\begin{cases} a[n+1] & = \left(a[n]\right)^{\mathcal{R}}_{a[n+1]0} \\ \mathcal{A}_{n+1} & = \left(\mathcal{A}_n\right)^{\mathcal{R}}_{\mathcal{A}_{n+1}(0)} \end{cases}.$

Hyperoperations Coq definitions

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 hierarchy Coq definitions

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

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

Step 1. Explore the hyperoperations/Ackermann function hierarchical structure: Connect consecutive levels with **Repeater**.

Step 2. Invert each level in both hierarchies:

- *What is inverse?* Upper inverse and increasing functions.
- *Can Repeater preserve Invertibility?* Repeatable functions.
- *Computing inverse with inverse:* Contractions and **Countdown**.
- Invert each level in hyperoperations/Ackermann hierarchies.

Step 3. Implement the Inverse Ackermann function via the inverse Ackermann hierarchy.

Step 4. Optimize its time complexity.

Roadmap

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

Step 1. Explore the hyperoperations/Ackermann function hierarchical structure: Connect consecutive levels with **Repeater**.

Step 2. Invert each level in both hierarchies:

- *What is inverse?* Upper inverse and increasing functions.
- *Can Repeater preserve Invertibility?* Repeatable functions.
- *Computing inverse with inverse:* Contractions and **Countdown**.
- Invert each level in hyperoperations/Ackermann hierarchies.

Step 3. Implement the Inverse Ackermann function via the inverse Ackermann hierarchy.

Step 4. Optimize its time complexity.

Step 2: Inverting the hyperoperations/Ackermman hierarchies via Countdown

Step 2: Inverting the hierarchies via countdown Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Step 2: Inverting the hierarchies via countdown
Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Upper inverses of increasing, unbounded functions

The *upper inverse* of F, written F^{-1}, is $\lambda n. \min\{m : F(m) \geq n\}$.
Note that F^{-1} is a total function when F is unbounded.

Analogue of inverse of injections: The *upper inverse* makes sense for *strictly increasing* (hereafter referred to simply as *increasing*). Increasingness parallels injectivity.

Note: Increasing functions are trivially unbounded.

Logical equivalence (more useful): If $F : \mathbb{N} \to \mathbb{N}$ is increasing, then f is the upper inverse of F if and only if $\forall n, m.\ f(n) \leq m \Leftrightarrow n \leq F(m)$.

Step 2: Inverting the hierarchies via countdown
Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Step 2: Inverting the hierarchies via countdown
Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Expansions and Repeatable functions 1

Observation: Every function in the hyperoperations (when $a \geq 2$) and the Ackermann hierarchy is increasing. How do they become that?

Generalization: What properties ensure increasing-ness is preserved by Repeater?

Repeatability: a property that encompasses increasing-ness that is preserved through Repeater.

$$\textit{Repeatable} = \textit{Increasing} + \textit{Strict Expanding}$$

Expansions: A function $F : \mathbb{N} \to \mathbb{N}$ is an *expansion* if $\forall n.\ F(n) \geq n$. Further, for $a \in \mathbb{N}$, an expansion F is *strict from a* if $\forall n \geq a.\ F(n) > n$.

Expansion and Repeatable functions 2

Repeatability: An increasing function f is *repeatable* from a if f is also an expansion that is strict from a.

The set of functions repeatable from a is denoted by REPT_a.

Observation. If $a \leq b$, $\mathrm{REPT}_a \subseteq \mathrm{REPT}_b$.

Repeatability Preservation Theorem. $\forall a \geq 1$, if $f \in \mathrm{REPT}_a$, then $f_a^{\mathcal{R}} \in \mathrm{REPT}_0$, meaning $f_a^{\mathcal{R}}$ is repeatable from any b.

Every level in the hyperoperations (when $a \geq 2$) and Ackermann hierarchies are repeatable from their respective initial values.
$\Rightarrow$ All invertible.

Step 2: Inverting the hierarchies via countdown
Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Step 2: Inverting the hierarchies via countdown
Roadmap

1. *What is inverse?* Upper inverse and increasing functions.
2. *Can Repeater preserve Invertibility?* Repeatable functions.
3. *Computing inverse with inverse:* Contractions and **Countdown**.
4. Invert each level in hyperoperations/Ackermann hierarchies.

Definitions

Contractions. 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 $\forall n > a.\ n > f(n)$.

Notations. Set of contractions: $\textsc{cont}$. Set of contractions strict above a: $\textsc{cont}_a$.

Observations. Analogously to Expansions and Repeatable functions, $\forall s \leq t.\ \textsc{cont}_s \subseteq \textsc{cont}_t$.

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

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

The importance of Countdown

Theorem. $\forall a$, $\forall F \in \mathrm{REPT}_a$, define $f \triangleq F^{-1}$.
Then $f \in \mathrm{CONT}_a$ and $f_a^{\mathcal{C}} = \left(F_a^{\mathcal{R}}\right)^{-1}$.

Proof. *Step 1.* $f \in \mathrm{CONT}_a$:
Since F is an expansion, $n \leq F(n) \Rightarrow f(n) \leq n$. Take $n > a$, Since F is strict from a, $n-1 < F(n-1) \Rightarrow n \geq F(n-1)$
$\Rightarrow f(n) \leq n-1 \Rightarrow f(n) < n$.

Step 2. $f_a^{\mathcal{C}} = \left(F_a^{\mathcal{R}}\right)^{-1}$. We have

$$f_a^{\mathcal{C}}(n) \leq m \Leftrightarrow f^{(m)}(n) \leq a \Leftrightarrow f^{(m-1)}(n) \leq F(a) \Leftrightarrow \ldots$$
$$\Leftrightarrow f(n) \leq F^{(m-1)}(a) \Leftrightarrow n \leq F^{(m)}(a) \Leftrightarrow n \leq F_a^{\mathcal{R}}(m)$$

Thus the proof is complete.

A Countdown computation in Coq type `nat`

Idea. To compute $f_a^{\mathcal{C}}(n)$, starting from n, repeatedly apply f to get the chain $\{n, f(n), f^{(2)}(n), \ldots\}$. Stops when $f^{(k)}(n) \leq a$. Returns k.

Key issue. Coq needs a known terminating point, i.e. an explicit decreasing argument. How to know when to terminate beforehand?

The worker function. A worker function takes f, a, n and a budget b and compute the chain $\{n, f(n), \ldots, f^{(b)}(n)\}$. It stops before reaching b if $f^{(k)}(n) \leq a$.

Inverting Hyperoperations and Ackermann

Time Bound of Our Inverses in Unary Encoding

Performance Improvement With Binary Encoding

Further Discussion