An alternative definition for the Inverse Ackermann function, and a linear time computation in Gallina

Abstract

We build a hierarchy of functions that are upper inverse of the usual Ackermann hierarchy, then use this inverse hierarchy to compute the inverse of the diagonal Ackermann function $\mathrm{A}n, n$. We show that this computation is consistent with the usual definition of the inverse Ackermann function $\alpha(n)$ and implement this computation in Gallina, where we show that it runs in linear time.

Keywords: Inverse Ackermann, Automata, Union-Find, Division

1. Overview

1.1. Ackermann function and its inverse

The time complexity of the union-find data structure has traditionally been hard to estimate, especially when it is implemented with the heuristic rules of *path compression* and *weighted union*. Tarjan [**?**] showed that for a sequence of m FINDs intermixed with $n-1$ UNIONs such that $m \geq n$, the time required $t(m, n)$ is bounded as: $k_1 m\alpha(m, n) \leq t(m, n) \leq k_2 m\alpha(m, n)$.

Here k_1 and k_2 are positive constants and $\alpha(m, n)$ is the inverse of the Ackermann function.

The Ackermann function, commonly denoted $\mathrm{A}(m, n)$, was first defined by Ackermann [**?**], but this definition was not as widely used as the below variant, given by Peter and Robinson [**?**]:

Definition 1.1. The Peter-Ackermann function is a recursive two-variable function $\mathrm{A} : \mathbb{N}^2 \to \mathbb{N}$ such that:

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

The diagonal Ackermann function is then denoted simply as:

$$\mathrm{A}n = \mathrm{A}(n, n) \tag{2}$$

Definition 1.2. The inverse Ackermann function $\alpha(n)$, as defined by many authors, is the minimum k for which $n \leq \mathrm{A}(k, k)$:

$$\alpha(n) = \min\{k \in \mathbb{N} : n \leq \mathrm{A}(k, k)\} \tag{3}$$

As many texts have suggested, $\mathrm{A}(m, n)$ increases extremely fast on both inputs, hence does $\mathrm{A}(n, n)$. This implies $\alpha(n)$ increases extremely slow, although it still tends to infinity. However, it does not mean computing $\alpha(n)$ for each n is an easy task. In fact, the naive method would iteratively check $\mathrm{A}(k, k)$ for $k = 0, 1, \ldots,$ until $n \leq \mathrm{A}(k, k)$, which could lead to unimaginably large computation time. For instance, suppose $n > 1$, and $\alpha(n) = k + 1$. This is equivalent to

$$\mathrm{A}(k, k) < n \leq \mathrm{A}(k + 1, k + 1) \tag{4}$$

The naive algorithm would need to compute $\mathrm{A}t, t$ for $t = 0, 1, \ldots, k, k+1$ before terminating. Although one could argue that the total time to compute $\mathrm{A}(t, t)$ for $t \leq k$ is still $O(n)$, as they are all less than n, the time to compute $\mathrm{A}(k+1, k+1)$ could be astronomically larger than n. This situation motivates the need for an alternative, more efficient approach to compute the inverse Ackermann function.

1.2. The hierarchy of Ackermann functions

If one denotes $\mathrm{A}_m(n) = \mathrm{A}(m, n)$, one can think of the Ackermann function as a hierarchy of functions, each level A_m is a recursive function built with the previous level A_{m-1}:

Definition 1.3. The Ackermann hierarchy is a sequence of functions $\mathrm{A}_0, \mathrm{A}_1, \ldots$ defined as:

1. $A_0(n) = n + 1 \quad \forall n \in \mathbb{N}$.
2. $A_m(0) = A_{m-1}(1) \quad \forall m \in \mathbb{N}_{>0}$.
3. $A_m(n) = A_{m-1}^{(n)}(0) \quad \forall m \in \mathbb{N}_{>0}$,

where $f^{(n)}(x)$ denotes the result of applying n times the function f to the input x. This hierarchy satisfies $\mathrm{A}_m(n) = \mathrm{A}(m, n) \;\; \forall m, n \in \mathbb{N}$.

This hierarchical perspective can be reversed, as shown in the next section, to form an inverse Ackermann hierarchy of functions, upon which we can compute the inverse Ackermann function as defined in Definition 1.1.

2. The Inverse Ackermann Hierarchy

In this section we build the inverse Ackermann hierarchy, define and prove the inverse relationship between them and the Ackermann hierarchy.

2.1. The countdown operation

Firstly, to define some sort of "inverse" for the repeat application operation found in Definition 1.3, we define the following *countdown* operation:

Definition 2.1. Given a function $f : \mathbb{N} \to \mathbb{N}$. The *countdown* of f, denoted by f^* is given by:

$$f^*(n) = \begin{cases} 0 & \text{if } n \le \max\{0, 1, f(n)\} \\ 1 + f^*(f(n)) & \text{if } n > \max\{0, 1, f(n)\} \end{cases} \tag{5}$$

The important observation is that, if the sequence $\{n, f(n), f(f(n)), \ldots\}$ strictly decreases to 0, then f^* counts the minimum index where it reaches 1 or below. We give a formal definition for functions with such decreasing sequence:

Definition 2.2. A function $f : \mathbb{N} \to \mathbb{N}$ is a *contraction* if $f(0) = 0$ and $f(n) \le n - 1 \; \forall n > 0$.

Theorem 2.1. *If $f : \mathbb{N} \to \mathbb{N}$ is a contraction, then*

$$A = \left\{k : f^{(k)}(n) \le 1\right\} \ne \varnothing \quad \textit{and} \quad f^*(n) = \min A \tag{6}$$

Proof sketch. By (5) and definition 2.2, we have

$$f^*(n) = \begin{cases} 0 & \text{if } n \le 1 \\ 1 + f^*(f(n)) & \text{if } n > 1 \end{cases}$$

Then by a strong induction on n, the theorem holds. $\square$

2.2. The inverse Ackermann hierarchy

With the above countdown operation, we can build the inverse Ackermann hierarchy.

Definition 2.3. The inverse Ackermann hierarchy is a sequence of functions $\alpha_0, \alpha_1, \ldots$ recursively defined as:

1. $\alpha_0(n) = \max\{n-2, 0\} \quad \forall n \in \mathbb{N}$.

2. $\alpha_m = \alpha^*_{m-1} \quad \forall m \in \mathbb{N}_{>0}$.

We will prove that each function α_m is a contraction, thus the $*$ operations are truly counting down to 1.

Theorem 2.2. *For all $m \in \mathbb{N}$, α_m is a contraction and*

$$\alpha_{m+1}(n) = \min\left\{k : \alpha_m^{(k)}(n) \leq 1\right\} \tag{7}$$

Proof. TODO TODO TODO □

Having sufficiently established the inverse Ackermann hierarchy, we now link it to the Ackermann hierarchy in definition 1.3. However, the relationship is not trivially clear, as the Peter-Ackermann function is in fact not exactly their inverses. We first define a *canonical* variant of the Ackermann hierarchy, then use it as an intermediate to link first two hierarchies.

2.3. The canonical Ackermann hierarchy

The canonical Ackermann hierarchy is in short a somewhat "cleaner" variant of the Ackermann hierarchy, with simpler initial values, but still built on the repeated application operation.

Definition 2.4. The canonical Ackermann hierarchy is a sequence of functions $\mathrm{C}_0, \mathrm{C}_1, \ldots$ defined as:

1. $\mathrm{C}_0(n) = n + 2 \quad \forall n \in \mathbb{N}$.

2. $\mathrm{C}_m(0) = 1 \quad \forall m \in \mathbb{N}_{>0}$.

3. $\mathrm{C}_m(n) = \mathrm{C}_{m-1}\left(\mathrm{C}_m(n-1)\right) = \mathrm{C}_{m-1}^{(n)}(0) \;\; \forall m, n \in \mathbb{N}_{>0}$.

Firstly, we explore the relationship between C_m and α_m. We use the following theorem:

Theorem 2.3. *For all $m, n \in \mathbb{N}$, we have:*

$$\forall N \in \mathbb{N} : \alpha_m(N) \leq n \iff N \leq C_m(n) \tag{8}$$

In other words

$$C_m(n) = \max \{N : \alpha_m(N) \leq n\} \tag{9}$$

Before proving this theorem, we will need to prove each function in the inverse and canonical Ackermann hierarchy is increasing (non-strictly). Then (8) is sufficient to explain the "inverse" relationship between them.

Lemma 2.4. *For all $m \in \mathbb{N}$, α_m is increasing.*

Proof. TODO TODO TODO □

Lemma 2.5. *For all $m \in \mathbb{N}$, C_m is increasing.*

Proof. TODO TODO TODO □

Now we can proceed with proof for Theorem 2.3

Proof of Theorem 2.3. We prove by induction on m.
Base case: $m = 0$. The goal becomes $N - 2 \leq n \iff N \leq n + 2$, which is trivial.
Inductive step: Let us choose any $m, n_0, N_0 \in \mathbb{N}$. Suppose that we have:

$$\forall n, N \in \mathbb{N}, \alpha_{m-1}(N) \leq n \iff N \leq \mathrm{C}_{m-1}(n) \tag{10}$$

There are two directions:

1. ($\implies$). Suppose $\alpha_m(N_0) \leq n_0$. If $N_0 \leq 1$, we automatically have $N_0 = C_m(0) \leq C_m(n)$. Otherwise we have

   $$\alpha_m(N) = 1 + \alpha_m(\alpha_{m-1}(N)) \leq n \implies \alpha_m(\alpha_{m-1})$$

   TODO TODO TODO TODO

2. $N_0 \leq \mathrm{C}_m(n_0) \implies \alpha_m(N_0) \leq n_0$. TODO TODO TODO

□

Now that we have established our canonical Ackermann hierarchy and its relation with the inverse Ackermann hierarchy, let us connect it to the original Ackermann hierarchy to complete the link.

Theorem 2.6. *For all $m, n \in \mathbb{N}$, we have $C_m(n + 2) = A_{m+1}(n) + 2$.*

Proof. TODO TODO TODO □

2.4. Linking it all together

In this conclusive part of this section, we link everything together by stating and proving the main theorem that connects the inverse Ackermann hierarchy and the Ackermann hierarchy. We then use this theorem to state and prove a relation between the inverse Ackermann hierarchy and the inverse Ackermann function.

Theorem 2.7. *For all $m, n, k \in \mathbb{N}$, we have:*

$$n \leq A(m,k) \iff \begin{cases} n \leq k+1 & \text{if } m = 0 \\ \alpha_{m-1}(n+2) \leq k+2 & \text{if } m > 0 \end{cases} \tag{11}$$

Proof. TODO TODO TODO □

The next theorem is a corollary of the above, which lays the theoretical groundwork for us to compute the inverse Ackermann function in linear time.

Theorem 2.8. *For all $n\mathbb{N}$, we have:*

$$\alpha(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \min\{m : \alpha_m(n+2) \leq m+3\} & \text{if } n \geq 2 \end{cases} \tag{12}$$

Proof. TODO TODO TODO □

In the next section, we will devise an algorithm to compute each of the functions in the inverse Ackermann hierarchy in linear time, and an algorithm to compute the inverse Ackermann function in linear time.

3. A linear time computation in Gallina

The main idea to compute the inverse Ackermann function using what we have established so far is to use Theorem 2.7 to iteratively compute each level in the hierarchy starting from input $n+2$, then stop when the condition $\alpha_m(n+2) \leq m+3$ is met.

In order for this computation to run in linear time, we need to first make sure *each* level $\alpha_m(n+2)$ is computed in linear time. We will then use a trick to achieve the linear time bound in the total computation using the relation

$$\alpha_m(n+2) = 1 + \alpha_m(\alpha_{m-1}(n+2)) \quad \forall m \in \mathbb{N}_{>0} \tag{13}$$

3.1. Inverse Ackermann hierarchy in linear time

First note that, in the Gallina specification, all natural numbers are represented with a string of S, the successor notation. All recursive functions in Gallina are required to decrease on one of its inputs, one or a few successors per recursive step. Thus all recursive functions, or Fixpoints in Gallina, must run in at least linear time over one of their inputs.

To compute the inverse Ackermann hierarchy for some input n with a Gallina-complied function, we look at its generalization: Given a contraction f over $\mathbb{N}$ and a Gallina function $\widetilde{f}$ computing f, we find a Gallina function $\widetilde{f^*}$ to compute its countdown, f^*.

Definition 3.1. Let $\mathcal{F}_k$ be the set of all Gallina functions $g : \mathbb{N}^k \to \mathbb{N}$. The *countdown recursor helper* is an operator $\mathrm{CRH} : \mathcal{F}_1 \to \mathcal{F}_3$ such that for all $g \in \mathcal{F}_1$ and $n_0, n_1, c \in \mathbb{N}$:

$$\mathrm{CRH}\,(g)\,(n_0, n_1, c) =$$
$$\begin{cases} 0 & \text{if } n_0 \leq 1. \\ 1 & \text{if } n_0 \geq 2, n_1 \leq 1. \\ 1 + \mathrm{CRH}\,(g)\,(n_0 - 1, g(n_1), n_1 - g(n_1) - 1) & \text{if } n_0 \geq 2, n_1 \geq 2, c = 0. \\ \mathrm{CRH}\,(g)\,(n_0 - 1, n_1, c - 1) & \text{if } n_0 \geq 2, n_1 \geq 2, c \geq 1. \end{cases} \tag{14}$$

It is trivial to see that for all $g \in \mathcal{F}$, $\mathrm{CRH}(g)$ is a Gallina Fixpoint (a Gallina-complied recursive function) in $\mathcal{F}_3$ since its second input n_0 decreases by 1 at every recursive step.

Definition 3.2. The *countdown recursor* is an operator $\mathrm{CR} : \mathcal{F}_1 \to \mathcal{F}_1$ such that for all $g \in \mathcal{F}_1$ and $n \in \mathbb{N}$:

$$\mathrm{CR}(g)(n) = \mathrm{CRH}(g)(n, g(n), n - g(n) - 1) \tag{15}$$

Since it is built by a composition of a Gallina Fixpoint CRH and a Gallina function g, $\mathrm{CR}(g)$ is indeed a Gallina function in $\mathcal{F}_1$. We prove that CR is the equivalence of the countdown operation in Gallina for contractions:

Lemma 3.1. *Let $f : \mathbb{N} \to \mathbb{N}$ be a contraction. Suppose a function $\widetilde{f} \in \mathcal{F}_1$ computes f, then $CR\left(\widetilde{f}\right)$ is a function in $\mathcal{F}_1$ computing f^*.*

Proof. TODO TODO TODO □

With this lemma, we can define the equivalence of the inverse Ackermann hierarchy in Gallina:

Definition 3.3. The Gallina inverse Ackermann hierarchy is a sequence of functions $\widetilde{\alpha}_0, \widetilde{\alpha}_1, \ldots$ such that for all $m, n \in \mathbb{N}$:

$$\widetilde{\alpha}_m(n) = \begin{cases} n - 2 & \text{if } m = 0 \\ \mathrm{CR}\left(\widetilde{\alpha}_{m-1}\right)(n) & \text{if } m \geq 1 \end{cases} \tag{16}$$

Note that $x - y$ in Gallina is equivalent to $\max\{x - y, 0\}$ in practice.

Lemma 3.1 and Theorem 2.2 trivially implies that $\widetilde{\alpha}_m$ is a Gallina computation of α_m for all $m \in \mathbb{N}$.

The important thing to come up with Gallina computations for the hierarchy is we want to compute them in linear time. We assert the hierarchy $\{\widetilde{\alpha}_m\}$ succeeds in doing so:

Theorem 3.2. *For each $m \in \mathbb{N}$, computing $\widetilde{\alpha}_m(n)$ takes at most $(m+o(1))n$ steps, where n tends to infinity.*

Proof. TODO TODO TODO □

3.2. Inverse Ackermann in linear time

As mentioned, the task is to find the minimum x for which $\alpha_x(n) \leq x+3$ for $n \geq 4$. It is tempting to go for a naive approach, after all the efficiencies we have developed so far: Starting from $x = 0$, we iteratively compute $\alpha_x(n)$ and compare it with $x+3$. However, by our earlier analysis in Theorem 3.2, the total amount of time needed is

$$T(n) = \sum_{m=0}^{\alpha(n)-1} (m + o(1))n = O\left(n\alpha(n)^2\right)$$

which is pretty efficient, given how slow-growing $\alpha(n)$ is. However, our ultimate goal is $O(n)$ time, thus we came up with a better approach. Interestingly, it is also based on Theorem 3.2. Specifically, using (5), Theorem 3.2 implies we can actually compute $\alpha_m(n)$ in $O(\alpha_{m-1}(n))$ time, *if* $\alpha_{m-1}(n)$ is given. It is thus beneficial if we retain the value of $\alpha_{m-1}(n)$ when computing $\alpha_m(n)$. The definition below is our Gallina recursor for this.

Definition 3.4. The inverse Ackermann recursor helper is a function IARH : $\mathcal{F}_1 \times \mathbb{N}^3 \to \mathbb{N}$ such that for all $g \in \mathcal{F}_1, n, c_0, c \in \mathbb{N}$, we have:

$$\mathrm{IARH}(g, n, c_0, c) =$$
$$\begin{cases} c & \text{if } c = 1 \\ 1 + \mathrm{IARH}\left(\mathrm{CR}(g), n, n - \mathrm{CR}(g)(n) - 1, c - 1\right) & \text{if } c \geq 1, c_0 = 0 \\ \mathrm{IARH}\left(g, n - 1, c_0 - 1, c - 1\right) & \text{if } c \geq 1, c_0 \geq 1 \end{cases} \tag{17}$$

Note that $n - 1$ in Gallina means $\max\{n - 1, 0\}$ here.

Definition 3.5. The inverse Ackermann recursor is a function IAR $\in \mathcal{F}_1$ such that for all $n \in \mathbb{N}$, we have

$$\mathrm{IAR}(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ \mathrm{IARH}\left(\alpha_0, n + 2, 2, n\right) & \text{if } n \geq 2 \end{cases} \tag{18}$$

The following theorem is a central result in this paper, which asserts the correctness of IAR.

Theorem 3.3. *IAR is a Gallina function computing $\alpha(n)$.*

It is trivial to see that both IARH and IAR are Gallina functions. It suffices to show that $\mathrm{IAR}(n) = \alpha(n)$ for $n \geq 2$, since their values already match for $n \leq 1$. Furthermore, if we trace the first two recursive steps in IARH, we obtain

$$\mathrm{IARH}(n+2, \alpha_0, 2, n) = 1 + \mathrm{IARH}(\alpha_1, n-2, n-\alpha_1(n)-1, n-3) \;\; \forall n \geq 2 \tag{19}$$

It then suffices to show the following:

Lemma 3.4. *For all $n \geq 2$, IARH$(\alpha_1, n, n-\alpha_1(n)-1, n-3) = \min\{m : \alpha_m(n+2) \leq m + 3\}$.*