Hugh Coleman - CSCI 4365 SP 23 - Presentation Outline

1. Title Slide
    a. Introduce

2. Ackermann Function Definition [J. Paul, 2023]
    a. The Ackermann function can be defined in many different ways, with three-variable and one-variable versions existing.
    b. The Most Common is the Ackermann-Peter Version.
    c. Defined for non-negative integers
        i. A(0, j)=j+1 for j ≥ 0
        ii. A(i, 0)=A(i-1, 1) for i > 0
        iii. A(i, j)=A(i-1, A(i, j-1)) for i, j > 0
3. Fast Growing [Bennett, 2017]
    a. The Ackermann grows incredibly fast.
    b. Grows faster than n^2, 2^n, n!,
    c. The number of quarks in the universe is approximately 3.28 * 10^80
    d. This doesn’t put a dent into A(4,2) which is about 2... × 10^19728 let alone A(4,4)
4. Notes [Armando, 2014, Paulson, 2022]
    a. The Ackermann function always halts.
        i. The Ackermann function is total
    b. The Ackermann function is not primitive recursive
        i. Consider the Halting problem.
        ii. The halting problem is not a primitive recursive function.
        iii. Every PR function is total recursive, not every total recursive function is PR
        iv. Any partial function would be not primitive recursive, however the Ackermann fulfills the special case of total primitive recursive function.
    c. Majorization
        i. Present definition of majorization.
        ii. Majorization is the way we show that the Ackermann function “grows” faster than any PR function.
5. Termination [Paulson, 2022]
    a. There are three cases that can occur with the Ackermann function
        i. m or n decreases.
        ii. when n reaches 0, m decreases in the next recursive call
        iii. since m is always either decreasing, or staying the same with n decreasing, m will always reach 0.
6. Iterative Calculation [Grossman and Zeitman, 1988]
    a. The Ackermann function can be computed iteratively.
    b. The simplest method is to use a stack, and emulated the recursion stack.

    c. Stack initially contains two elements, <m,n>
    d. First pop 2 elements off the top of the stack for <m,n>
    e. Then according to the rules, either push 1, 2, or 3 elements back onto the stack.
7. Iterative Computation
    a. Show the Examples
    b. This form of triangle is the pattern that is formed.
    c. Note the plummeting behavior of the stack
    d. ackermann[2,1] = ackermann[1,3]
        i. This is useful for the Inverse later as it shows the bottom up process that is able to be used for a PR Ackermann Graph Algorithm.
8. WHILE Language Implementation
    a. A stack is possible to implement in the WHILE language
        i. Godel Numbering is an option.
        ii. Encode it into a stack
        iii. Massive, and slow, but in terms of the Ackermann function, negligible
    b. n being outside the stack.
        i. It’s just an easier model to look at, and has the same effect as popping n off every time.
    c. Three rules
        i. Go over each rule
    d. Note the singular WHILE loop.
9. Computational Length
    a. PR
        i. Clearly the stack implementation on the left is not PR.
        ii. This is because it has a WHILE loop.
        iii. There is only a singular while loop.
    b. Thus, there is a function COMP(m,n)
        i. COMP computes how many times the WHILE loop will run in the left hand definition.
        ii. The naive way to write the function COMP is simply to add a counter to the WHILE loop definition on the lefthand side.
        iii. Instead of returning the value of A(m,n) return the number of times it loops
10. Ackermann Compute Time
    a. We know that the Ackermann function is not PR, and this is a proven fact.
    b. A definition of COMP that is primitive recursive would be a contradiction to the Ackermann functions primitive recursive nature.

   c. Thus there is no primitive recursive function that can compute how many times the loop will need to run.
11. Stack Model Complexity [Cohen, 1987]
   a. for all m,n it takes no longer than (A(m,n)+1)^m steps.
   b. The maximum length of the stack is A(m, n), as long as m > 0
   c. The plummeting nature of the ackermann function assures that we will have a stack that large.
   d. The function can only use the successor function to increase itself, thus it needs vast amounts of memory and recursion to do this.
12. Grossman & Zeitman [Grossman and Zeitman, 1988]
   a. Their algorithm computes A(m, n) within O(m A(m, n)) time and within O(m) space.
13. Inverse Ackermann [Armando B, 2014]
   a. Single Argument version of Ackermann
      i. based on the Ackermann-Peter version
      ii. just f(n) = A(n,n)
   b. The inverse is known as a(n).
   c. Primitive Recursive.
      i. The function grows incredibly slowly, and is a primitive recursive function.
14. Inverse Ackermann - Examples [Armando B, 2014]
   a. As the Ackermann function is not onto, its inverse is not total.
   b. Examples of the Inverse
      i. The function still grows, just very slowly.
      ii. Past a(61) it becomes effectively constant for any modern computer
   c. Not a total function
      i. The common way to deal with this is to floor the input to the next input in the domain.
      ii. This is important as for a(n) to be primitive recursive, it must be total.
15. Inverse Ackermann Graph [Armando B, 2014]
   a. Note the three rules as they are important
      i. A(m,n) is greater than the sum of its inputs
      ii. A(m, n) < A(m, n + 1)
      iii. A(m, n) < A(m + 1, n)
      iv. The Ackermann function never decreases. This is what lets us build the graph of Ackermann function values.
16. Graph Primitive Recursive [Armando B, 2014]
   a. If A(x, y ) = z, we must have x ≤ z and y ≤ z
   b. f A(m, n) is used as an argument of another computation of A
      i. ackermann[m+1,n] = ackermann[m,ackermann[m,n]]

ii. $A(m, n) < A(m + 1, n)$

17. Ackermann Computation [Armando B, 2014]
    a. Three cases
    b. When $m = 0$, it is a simple immediate computation
    c. When $n = 0$, consult $A(m-1,1)$
    d. When $A(m-1, w)$, $w = A(m, n - 1)$
        i. This depends on the previous index of the rectangle.
18. Function Outline [Armando B, 2014]
    a. Ackermann computed triples
    b. Runs over the x,y,z used in making the triples.
19. Ackermann Graph Computation [Armando B, 2014]
    a. Computation
        i. the (0,n) to z is computed to cover the top edge of the rectangle
        ii. the (m,0) to z is computed to cover the left edge of the rectangle
        iii. For $m = 1, 2, \ldots, z$: Compute and save $A(m, 1), A(m, 2), \ldots, A(m, z)$
            1. This computes the internal rectangle recursively based on the previously computed values.
        iv. NOTE: if $A(m, n) > z$
    b. Search the Graph
        i. if the triple of x,y,z is in the graph, print 1
        ii. search for x,y,w
20. Do Times Program [Armando B, 2014]
    a. We have outlined a function that computes the graph
    b. that program is PR
    c. DO-TIMES program that uses the graph to find if x,y,z triple is in the graph.
    d. That program is PR
    e. Inverse Ackermann has been proved to be PR.
21. Minimum Spanning Trees [Commons, 2023]
    a. A minimum spanning tree of m edges and n vertices
    b. Find the minimum spanning tree.
22. Chazelle's algorithm [Chazelle 2000, Commons, 2023]
    a. Chazelle's algorithm has a complexity of $O(m\alpha(m, n))$
    b. Utilizes a soft heap
    c. This is approximately m, as $a(m,n)$ grows so slowly)

**References**


Armando B, M. (2014).

The inverse of the ackermann function is primitive recursive.

Bennett, J. (2017).

How many particles are in the observable universe?

Chazelle, B. (2000).

A minimum spanning tree algorithm with inverse-ackermann type complexity.

*Journal of the ACM (JACM), 47(6):1028–1047.*

Cohen, D. E. (1987).

Computability and Logic.

Mathematics and its Applications. Ellis Horwood Ltd, Publisher, Harlow, England.

Commons, W. (2023).

File:minimum spanning tree.svg — wikimedia commons, the free media repository.

[Online; accessed 10-April-2023]

Grossman, J. W. and Zeitman, R. (1988).

An inherently iterative computation of ackermann's function.

Theoretical Computer Science, 57(2):327–330.

J. Paul, M. (2023).

Csci 4365 lecture notes.

Paulson, L. C. (2022).

Machine logic.