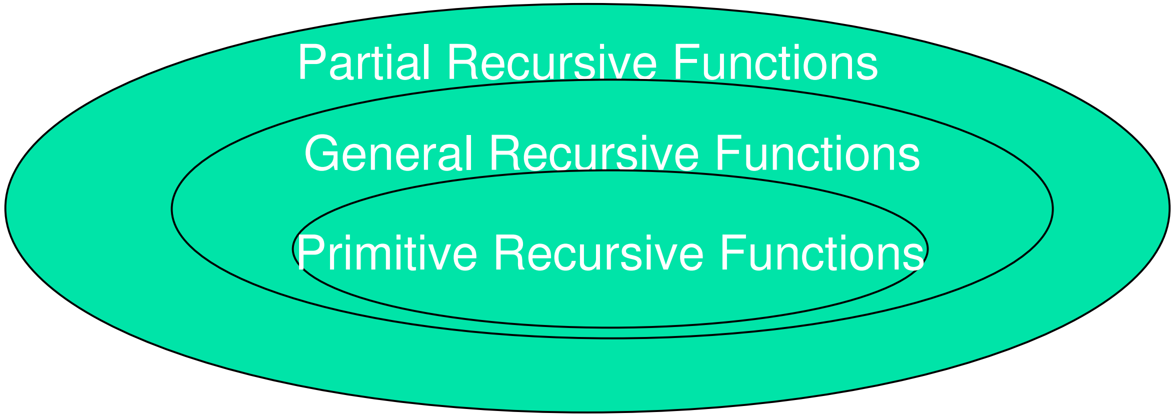
CAS 701 Presentation

Ackermann's Function

Qinglei Zhang, Nov. 20, 2008.

History

- The belief in the early 1900s: every computable function was also primitive recursive
- A **strict** subset of the recursive functions: every primitive recursive function is total recursive, but not all total recursive functions are primitive recursive.
- Well known Counterexample: David Hibert(*On the Infinite*), Gabriel Sudan, Wihelm Ackermann(1928)



Basic conceptions

- **Recursive function theory**: one way to make formal and precise the intuitive, informal, and imprecise notion of an effective method.
- *Church*'s *thesis*: every function that is effectively computable in the intuitive sense is computable in these formal ways.

Inductive Definition of Primitive Recursive Functions

- The initial functions: The **zero** function, the **successor** function, and all **projection** functions.
- Functions which arise by **composition** and **primitive recursion** from primitive recursive functions.
- In the programming language: it has **FOR-loops** as the only iterative control structure.

Recursive Function

- Algorithms can be written in the form of WHILE-Loop.
- Technically, add a construct operation called **minimization** which does something equivalent.

Unbounded search: If we say that g(x) is a function that computes the least x such that f(x) = 0, then we know that g is computable. We will say that g is produced from f by

minimization.

Ackermann's Function

- Ackermann originally considered a function of three variables A(m,n,p) =m→ n→p (Conway chained arrow notation).
- Hyper operators: a variant of Ackermann function For the successive operators beyond exponentiation.

  hyper(a,n,b)= $a\uparrow^{(n-2)} b$ (*knuth's up-arrownotation*)

  =a→b→(n-2).
- Ackermann proved that A is computable and not a primitive recursive function.

Different Versions of Ackermann's function

- Van Heijenoort(1928)
- $$\text{ack}(x,y,z)=\begin{cases} y+z & \text{for } x=0, \\ 0 & \text{for } x=1, z=0, \\ 1 & \text{for } x=2, z=0, \\ y & \text{for } x>2,\ z=0, \\ \text{ack}(x-1,y,\text{ack}(x,y,z-1)) & \text{for } x,z>0. \end{cases}$$
- Analysis: ack(0,y,z)=y+z;
  ack(1,y,z)=y×z;
  ack(2,y,z)=y^z;
  ack(3,y,z)=hyper(y,4,z+1).

Hyper operator

- Buck(1963): recursively define an infix triadic operator
-

$$\text{ack-b}(x,y)=\begin{cases} y+1 & \text{for } x=0, \\ 2 & \text{for } x=1,\ y=0, \\ 0 & \text{for } x=2,\ y=0, \\ 1 & \text{for } x>2,\ y=0, \\ \text{ack-b}(x-1,\ \text{ack-b}(x,y-1)) & \text{for } x,y>0. \end{cases}$$

- Analysis: ack-b(0,y)= hyper(2,0,y)=y+1; (Successor function)
  ack-b(1,y)= hyper(2,1,y)=2+y; (Summation)
  ack-b(2,y)= hyper (2,2,y)=2×y; (multiplication)
  ack-b(3,y)=hyper(2,3,y)=2^y; (exponentiation)
  ack-b(4,y)= hyper(2,4,y)=${}^{y}2$; (superpower, power towers)

- Rósza Péter(1935), Raphael M. Robins(1948)
-

$$\text{ack-p}(a,b)=\begin{cases} b+1 & \text{for } a=0, \\ \text{ack-p}(a-1,1) & \text{for } a>0, b=0, \\ \text{ack}(a-1, \text{ack-p}(a,b-1)) & \text{for } a,b>0. \end{cases}$$

- Analysis:

ack-p(0,b)=b+1;
ack-p(1,b)=2+(b+3)-3;
ack-p(2,b)=2×(b+3)-3;
ack-p(3,b)=2^ (b+3)-3=(2↑ (b+3))-3 =hyper(2,3,b+3)-3;
ack-p(4,b)=(2 ↑↑ (b+3))-3=hyper(2,4,b+3)-3;
ack-p(a,b)=hyper(2,a,b+3)-3.

Values of A(m,n) (Rósza Péter version)

a\b	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	5	7	9	11
3	5	13	29	61	125
4	13	65533	$2^{65536}-3$	$2^{2^{65536}}-3$	A(3,A(4,3))

Computing the value of Ackermann function: Example (Rósza Péter version)

- A(4,3)=A(3,A(4,2))
  =A(3,A(3,A(4,1))) *b* decrease
  =A(3,A(3,A(4,0)))
  =A(3,A(3,A(3,1))) a decrease
  =…
  =A(3,A(3,A(3,13))
  =…
  =A(3, A(3, 65533))
  =…
  =A(3, 2^{65536}-3)
  =…
  =$2^{2^{65536}} \approx 10^{10^{19727.78}}$

Ackermann is total computable functions

- $A(0; y) = y + 1$ (1)
  $A(x + 1; 0) = A(x; 1)$ (2)
  $A(x + 1; y + 1) = A(x; A(x + 1; y))$ (3)
- Define the **lexicographical order** on $N \times N$ as follows:

  $(x; y) > (x0; y0)$ iff $x > x0$ or ($x = x0$ and $y > y0$):

  a well-ordering of order type ω^2
- The clauses (2) and (3) lead to lexicographically smaller arguments; this cannot go on forever, so A must finally halt.

Ackermann is not primitive recursive

- **Theorem**: The Ackermann function dominates every primitive recursive function in the sense that there is a k such that

  $$f(x) < A(k, \sum x),$$

  Where f is a primitive function, $\sum x$ is the sum of all the components of x.

  Sketch of proof:
  One can argue by induction on the buildup of *f*.
  Deal with the atomic functions and then show that the property is preserved during an application of composition and primitive recursion.
- So in particular, A is not primitive recursive.

Applications

- Inverse Ackermann function (extremely slow-growing function)
  $\alpha(m,n)=\min\{i \geq 1, A(i, [m/n]) \geq \log_2 n\}$

  For all practical purposes: $\alpha(m,n)$ can be regarded as being a constant, less than 5.
  In the time complexity of some algorithm:
  Union-Find problem: $O(\alpha(m,n)+n)$),
- Use as Benchmark: Compliers' ability to optimize recursion.

Reference

- Ackermann function: http://en.wikipedia.org/wiki/Ackerman_function
- Hyper operator: http://en.wikipedia.org/wiki/Hyper_operator
- Versions of Ackermann's Function: http://www.mrob.com/pub/math/ln-2deep.html
- A. Garrido. Primitive Recurison and μ-Recursivity. *Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM),* vol.1,no.3,pp.273-280, 2006.
- *B.* Nordstrom**.** Primitive Recursive Functions. *Models of Computation, Chalmers and University of Goteborg,* 2005.
- K. Sutner. Primitive and General Recursive Functions. *Presentation, Carnegie Mellon University*, 2006.
- R.Seidel. Understanding the inverse Ackermann function. *EWCG 2006*, Delphi, March 27–29, 2006.