Theory of Computation
Recursive Functions

Recursive functions

- This was an alternative formalism to Turing Machines that was proposed by Godel.
- Godel defined a set of functions between $\mathbb{N}^k \to \mathbb{N}$ that he thought would represent all computable functions.
- Church later proved this equivalence.

The Functions

- **Successor.** $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ - computable, total recursive.
- **Zero.** $z : \mathbb{N}^0 \to \mathbb{N}$ such that $z() = 0$ - computable, total recursive.
- **Projections.** $\pi^n_k : \mathbb{N}^n \to \mathbb{N}$ such that $\pi^n_k(x_1, x_2, \ldots, x_n) = x_k$, $1 \leq k \leq n$ - computable, total recursive.
- **Composition.** Given computable functions $f : \mathbb{N}^k \to \mathbb{N}$ and $g_1, g_2, \ldots g_k : \mathbb{N}^n \to \mathbb{N}$, $f \cdot (g_1, g_2, \ldots, g_k) : \mathbb{N}^n \to \mathbb{N}$ such that on input $\overline{x} = (x_1, x_2, \ldots x_n)$ gives $f(g_1(\overline{x}), g_2(\overline{x}), \ldots, g_k(\overline{x}))$ - computable, total recursive.

The Functions contd.

- **Primitive Recursion.** Given computable functions $h_i : \mathbb{N}^{n-1} \to \mathbb{N}$ and $g_i : \mathbb{N}^{n+k} \to \mathbb{N}$, $1 \leq i \leq k$ the inductively defined functions
  $f_i(0, \overline{x}) := h_i(\overline{x}$
  $f_i(y+1, \overline{x}) := g_i(y, \overline{x}, f_1(y, \overline{x}, \ldots, f_k(y, \overline{x}))),$
  $\overline{x} = (x_2, x_3, \ldots, x_n)$
  -computable, total recursive

The Functions contd.

- **Unbounded Minimization.** Given computable function $g : \mathbb{N}^{n+1} \to \mathbb{N}$, the function $f : \mathbb{N}^n \to \mathbb{N}$ that takes input $\overline{x} = (x_1, x_2, \ldots, x_n)$ and gives the least y such that $g(z, \overline{x})$ is defined for all $z \leq y$ and $g(y, \overline{x}) = 0$ if such a y exists. The function f is undefined otherwise. Denoted as:
  $f(\overline{x}) = \mu y.(g(y, \overline{x}) = 0)$.
  -computable, partially recursive.

The Functions contd.

- These functions are called μ-*recursive functions* - all computable.
- The first five functions are called *primitive recursive functions* - all these functions are also total recursive.

Examples

- Constant function. $\mathrm{const}_n := s \cdot \ldots \cdot s \cdot z$
- Addition. Using primitive recursion when
  $k = 1, h = \pi_1^1, g = s \cdot \pi_3^3$
  $\mathrm{add}(0, y) = h(y) = y$
  $\mathrm{add}(x + 1, y) = g(x, y, \mathrm{add}(x, y)) = s(\mathrm{add}(x, y))$

Examples contd.

Try using Primitive recursion for:

- multiplication, exponentiation, predecessor,
- subtraction ($x - y$ if $x \geq y$ and 0 otherwise),
- sign function,
- comparisons like $<, \leq, >, \geq, =, \neq$, where the output can be considered to be from $\{0, 1\}$ depending on the truth of the relation.

The Ackermann Function

- $A(0, y) := y + 1$
  $A(x+1, 0) := A(x, 1)$
  $A(x+1, y+1) := A(x, A(x+1, y))$
- Designed by Ackermann in his PhD thesis, this is an extremely fast growing function that can be shown to grow faster than any primitive recursive function.
- But it can be shown to be a total computable function.
- First example of a total computable function that is not primitive recursive.