Phil 320

Chapter 6: Recursive functions

0. Introduction

Recursive functions: this is our third attempt to characterize the set of computable functions. Unlike the first two definitions, we don't introduce any external devices (Turing machines or abacus machines). We simply identify an obviously computable set of *basic* functions, together with some 'mechanical' operations (composition, primitive recursion, minimization) for generating new functions.

Church's Thesis: every effectively computable function is recursive.

Main objectives:

i) Learn how the set of recursive functions (and the special subset of primitive recursive functions is defined).

ii) Become proficient at showing how complicated functions are obtained through composition, primitive recursion and minimization.

New notation:

Use 0, 0′, 0 ′′, 0′′′, ….
In general: 0 followed by ′…′ (*n* primes) stands for *n*.
We'll sometimes use 1, 2, 3, … as shorthand for 0 followed by primes.

1. Primitive recursive functions

The *primitive recursive* (p.r.) functions consist of:

- *Basic* functions: $z(x)$, $s(x)$ and $id_k^n(x_1,\ldots, x_n)$

  $z(x) = 0$
  $s(x) = x'$
  $id_i^n(x_1,\ldots, x_n) = x_i$

- All functions that can be obtained from primitive recursive functions by two operations: *composition* and *primitive recursion*. [Note: they are all ultimately obtained from basic functions.]

Our 'library' of simple functions known to be p.r. will include constant functions, sum, prod, Exp, $\dot{-}$, Pred, sg and $\overline{sg}$. You may use all of these to obtain other p.r. functions.

A. Composition.

Examples.

Example 1: Is $f(x) = x+2$ p.r.? $f(x) = s(s(x))$.
Example 2: Is $f(x) = 12$ p.r.? $f(x) = s(s(\ldots s(z(x))\ldots))$.
Example 3: Is $f(x_1,\ldots,x_n) = 0$ p.r.? $f(x_1,\ldots,x_n) = z(id_n^n(x_1,\ldots, x_n))$.
Example 4: Is $f(x_1, x_2, x_3) = x_2 + 1$ p.r.? $f(x_1, x_2, x_3) = s(id_2^3(x_1,x_2,x_3))$.

Definition.

Suppose f is a function of *m* arguments and each of $g_1,\ldots, g_m$ is a function of *n* arguments. Then the composite function

$$h(x_1,\ldots, x_n) = f(g_1(x_1,\ldots, x_n), \ldots, g_m(x_1,\ldots, x_n))$$

is primitive recursive, provided that f and each of $g_1,\ldots,g_m$ is primitive recursive.

Method.

If you can write a new function h as a composition of functions known to be p.r., then you know that h is p.r.

Ex: $h(x, y, z) = x^{y+z} \cdot z^{y \dot{-} x} = prod(Exp(x, y+z), Exp(z, y \dot{-} x))$

Ex: $h(x) = \begin{cases} x', & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{cases} = s(x) \cdot sg(x)$

Three types of notation.

$h(x_1, x_2, x_3) = x_2 + 1.$ **Informal** – just provide the formula.

$h(x_1, x_2, x_3) = s(id_2^3(x_1, x_2, x_3))$ **Moderately formal** – use functions known to be p.r.

$h = Cn[s, id_2^3]$ **Super-formal** – use $Cn[f, g_1,\ldots,g_m]$.

The informal or moderately formal notation is usually just fine.

B. Primitive recursion.

Examples.

Example 1. sum(x, 0) = x [base clause]
sum(x, y′) = [sum(x, y)]′ [recursion clause]

Example 2. prod(x, 0) = 0
prod(x, y′) = sum(prod(x, y), x)

Example 3. factorial(0) = 1
factorial(y′) = prod(y′, factorial(y))

Example 4. sg(0) = 0
sg(y ′) = 1

To compute sum(x, y), you work your way through sum(x, 0), then sum(x, 1), …, sum(x, y). The same point applies to each of these examples.

Definition.

Case 1: h has two arguments. h is defined by primitive recursion from f and g if

h(x, 0) = f(x)
h(x, y′) = g(x, y, h(x, y)).

Write h = Pr[f, g] [**Examples:** sum, prod, exp]

Case 2: h has 1 argument.

$h(0) = c$ where c = 0 or 0′ or … (some constant)
$h(y') = g(y, h(y))$

Write h = Pr(c, g) [**Examples:** Factorial, Predecessor, sg]

Case 3: h has n+1 arguments. h is defined by primitive recursion from f (n-place) and g (n+2-place) if

$h(x_1,x_2,\ldots,x_n,0) = f(x_1,\ldots,x_n)$
$h(x_1,x_2,\ldots,x_n,y') = g(x_1,\ldots, x_n, y, h(x_1,\ldots, x_n,y))$

Write h = Pr(f, g).

In all three cases: if the functions f and g are primitive recursive, so is h.

Suggested method to show that h is primitive recursive.

Step i: Write out a table of values for $f(x_1,\ldots, x_n, y)$. Fix $x_1,\ldots, x_n$, and let the value of *y* increase by 1 for each row. Use one column for $x_1,\ldots, x_n$, one column for *y*, and one column for the value $f(x_1,\ldots, x_n,y)$. Now inspect the table as indicated in part b).

Step ii: **1-place:** Look for a simple rule for getting from f(x) to f(x+1).
2-place: Look for a simple rule to get from f(x, y) to f(x, y+1).

Step iii: Define f by providing the base clause and the recursion clause.

Example: Exponentiation (exp).

x	y	exp(x, y)
x	0	1
x	1	x
x	2	x · x
x	3	x · x · x
…		

Base: exp(x,0) = 1
Recursion: exp(x, y′) = prod(x, Exp(x,y))

Generalized sums and products.

If f is primitive recursive (p.r.), then

(1) $g(x, y) = f(x, 0) + \ldots + f(x, y) = \sum_{i=0}^{y} f(x,i)$ and

(2) $g(x, y) = f(x, 0) \cdot \ldots \cdot f(x, y) = \prod_{i=0}^{n} f(x,i)$

are primitive recursive functions.

2. Recursive functions

The *recursive* functions consist of:

- *Basic* functions: $z(x)$, $s(x)$ and $id_k^n(x_1,\ldots, x_n)$
- All functions that can be obtained from recursive functions by three operations: *composition*, *primitive recursion* and *minimization*.

Note: The primitive recursive functions are a subset of the recursive functions. One big difference is that every p.r. function is total, but this is not the case for every recursive function.

Minimization.

Examples.

Example 1: Mn[sum].

Mn[sum](x) = least y such that sum(x, y) = 0, or undefined o/w

Clearly, Mn[sum](0) = 0, and Mn[sum](x) is undefined if $x \neq 0$.

Example 2: Mn[prod](x) = 0, all x.

Example 3: $Mn[\dot{-}](x) = x$, all x.

Definition.

If f is a function of n+1 arguments,

$Mn[f](x_1,\ldots, x_n) =$ {y, if $f(x_1,\ldots, x_n, y) = 0$ and $f(x_1,\ldots, x_n, t)$ is defined and > 0 for $t < y$

{undefined, if there is no such y

Usually, n=1. In this case, to compute Mn[f](x), we compute f(x, 0), f(x, 1), … until we find a y such that f(x, y) = 0. If there is such a y, then Mn[f](x) = the least such y; if not, then Mn[f](x) is undefined.

Suggested method to show that a function may be obtained via minimization.

Step i: write out an informal description, using phrase "the least y such that <condition>".

Step ii: Turn <condition> into an expression of the form f(x, y) = 0.

Example: h(x) = round($\sqrt{x}$) = $\sqrt{x}$ rounded up to nearest integer.

h(x) = least y such that $x - y^2 \leq 0$ [first attempt to state the condition]

= least y such that $(x \dot{-} y^2) = 0$ [use a p.r. function of *x* and *y*]

= Mn[f](x),

where $f(x, y) = x \dot{-} y^2$.

A function f is called *regular* if for every $x_1,\ldots, x_n$ there is a y such that $f(x_1,\ldots, x_n, y) = 0$. If f is regular, then Mn[f] will be a total function, i.e., defined everywhere.