1 | Primitive Recursive Functions (§6.1)

(Dfn.1.1) *Informally,*: an **effectively computable function** *f* from natural numbers to natural numbers is the notion of a function for which there is a finite list of definite, explicit instructions (w/o regard for time, expense, etc.) that in principle make it possible to determine the value $f(x1, \ldots, xn)$ for any arguments $x1, \ldots, xn$.

- **Aim of Chapter**: study one kind of effectively computable function i.e. the recursive functions. There are two kinds primitively recursive functions (§6.1) and those that are not (§6.2)
- **Significance**: If Church's Thesis is true, then the primitively recursive functions are all the effectively computable functions i.e. every effectively computable function is primitively recursive.
- Recursive Computability is independent of Turing and Abacus computability, even thought all three are equivalent

(Dfn.1.2) *Informally,* a **primitive recursive functions** consists of i) basic functions: $z(x)$ (zero function), $s(x)$ (successor function) and $\mathrm{id}^n_k(x_1, \ldots, x_n)$ (identity function) or ii) functions generated by composition, primitive recursion

(Dfn.1.3) **Recursive Functions**: all functions that can be obtained from basic recursive functions by three operations: composition, primitive recursionand minimization

- Basic functions are obviously enumerable. Composition and recursion operations clearly yield enumerable functions, if given enumerable functions. So, the functions generated by these operations when supplied with basic functions is going to be enumerable as well.
- Note: the primitive recursive functions are a subset of the recursive functions.
- Every primitive recursive function is total, but some recursive functions are partial.
- All primitively recursive functions are effectively computable.
  For if *f* and *g* are effectively computable functions, then *h* is an effectively computable function.
- The number of steps needed to compute h(x,y) will be the sum of the number of steps needed to compute $z_0 = f(x) = h(x, 0)$, the number needed to compute $z1 = g(x, 0, z0) = h(x, 1)$, the number needed to compute $z_2 = g(x, 1, z1) = h(x, 2)$, and so on up to $z_y = g(x, y1, zy1) = h(x, y)$

2 | Basic Primitively Recursive Functions (§6.1)

- Note: choice of numerals for arguments and values does not affect the class of functions that are effectively computable, but can make proofs easier
- Numerals: '0' is represented as '0', and every natural number n > 0 is represented by 0 followed by a sequence of n accents. (1 = 0'; 2 = 0"; ...)

Examples

- **Zero Function**: z(x) = 0, for any x
  To compute the zero function, given any any argument, we simply ignore the argument and write down the symbol 0.
- **Successor Function**: s(x) = x + 1, for any x
  s(0) = 0'; s(0') = 0"; s(0") = 0'".
  To compute the successor function in our special notation, given a number written in that notation, we just add one more accent at the right.
- **Identity (projection) Functions**:
  One argument: id^1_1(x) = x; assigns to each natural number as argument the same number as value.
  Two arguments: $\mathrm{id}^2_1(x, y) = x$; $\mathrm{id}^2_2(x, y) = y$
  In general, for each positive integern, there arenidentity functions ofnarguments, which pick out the first, second,..., andnth of the arguments: $\mathrm{id}^n_i(x1, \ldots, xi, \ldots, xn) = x_i$

3 | Non-Basic Functions by Composition (§6.1)

(Dfn.3.4) Operation: Composition ('Substitution') (C_n): If f is a function of m arguments and each of $g_1, ..., g_m$ is a function of n arguments, then the function obtained by composition from $f, g1, ..., g_m$ is the function h where we have (C_n) $h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$; Shorthand: h = $C_n[f, g_1, ..., g_m]$

- the set of enumerable functions is enlarged beyond the basic ones by applying composition operations to the basic functions, generating a larger group of functions.
- Clearly, if the functionsgi are all effectively computable and the function f is effectively computable, then so is the functionh. The number of steps needed to computeh(x1,...,xn) will be the sum of the number of steps needed to compute y1=g1(x1,...,xn), the number needed to computey2=g2(x1,...,xn), and so on, plus at the end the number of steps needed to computef(y1,...,ym).

Examples

1. **Constant**: $\text{const}_n(x) = n$, for all x, for any natural number n
   For each n, const_n can be obtained from the basic functions by finitely many applications of composition
   const_0= is just the zero function z(n) Cn[s,z] is the function h with h(x)=s(z(x))=s(0)=0'=1=const1(x)
   const1=Cn[s,z], for all x const2=Cn[s, const1]
   constn+1=Cn[s, constn]
2. Show that $f(x) = x + 2$ is primitively recursive. $f(x) = s(s(x))$
3. Show that $f(x) = 12$ is p.r.. $f(x) = s(s(\ldots s(z(x))\ldots))$
4. Show that $f(x_1, \ldots, x_n) = 0$ is p.r.. $f(x_1, \ldots, x_n) = z(id^n_n(x1, \ldots, x_n))$
5. Show that $f(x_1, x_2, x_3) = x^2 + 1$ is p.r.. $f(x_1, x_2, x_3) = s(id^3_2(x_1, x_2, x_3))$

4 | Non-Basic Functions By Recursion

(Dfn.4.5) The process for defining new functions from old at work in these cases is called **(primitive)recursion**. h(x,0)=f(x), h(x,y'))=g(x,y,h(x,y)) (Pr). Where the boxed equations — called therecursion equations for the function h — hold, h is said to be definable by (primitive) recursion from the functions f andg. Shorthand, h=Pr[f,g]

Example

1. **Addition**: $\text{sum}(x, 0) = x$ (base); $\text{sum}(x, y) = [\text{sum}(x, y)]$ (recursive)
   More explicitly:
   Rules for computing addition:
   A) x + 0 = x; B)x + y' = (x + y)'
   Example: add 2 and 3
   0"+0"' =(0"+0")' by B), x=0", y=0"
   0"+0" =(0"+0')' by B), x=0", y=0'
   0"+0' =(0"+0)' by B), x=0", y=0
   0"+0 =0" by B), x=0"
   0" + 0"' = (0" + 0")'
   = (0" + 0')"
   = (0" + 0)"'
   = (0"'')
   So the sum is 0"'' = 5.
2. **Multiplication**: mult(x, 0) = 0; prod(x, y') = sum(prod(x, y), x)
3. **Factorial**: factorial(0) = 1; factorial(y') = prod(y', factorial(y))
4. sg(0) = 0; sg(y') = 1

5 | Non-Basic Functions by Minimalization (§6.2)

(Dfn.5.6) Given a function f of $n + 1$ arguments, the operation of **minimization** yields a total or partial function h of n arguments as follows: $Mn[f](x1, ..., xn) = yiff(x1, ..., xn, y) = 0, andforallt <$

$yf(x1, ..., xn, t) is defined and = 0$ undefined if there is no such y.

6 | Church's Thesis

(Dfn.6.7) The functions that can be obtained from the basic functionsz, s,idni by the processes Cn, Pr, and Mn are called **the recursive (total or partial) functions**.

(In the literature, 'recursive function' is often used to mean more specifically 'recursive total function', and 'partial recursive function' is then used to mean 'recursive total or partial function'.)
(Ths.6.8) all recursive functions are effectively computable
(Conjecture/Hyp.6.9) **Church's Thesis**: all effectively computable total functions are recursive
(Conjecture/Hyp.6.10) **Church's Thesis Extended**: all effectively computable partial functions are recursive)

- **Significance of Church's Thesis**: Later chapters will show that some particular functions of great interest in logic and mathematics are nonrecursive.
- In order to infer from such a theoretical result the conclusion that such functions are not effectively computable (from which may be inferred the practical advice that logicians and mathematicians would be wasting their time looking for a set of instructions to compute the function), we need assurance that Church's thesis is correct.
- At present Church's thesis is, for us, simply an hypothesis. It has been made somewhat plausible to the extent that we have shown a significant number of effectively computable functions to be recursive, but one can hardly on the basis of just these few examples be assured of its correctness. More evidence of the correctness of the thesis will accumulate as we consider more examples in the next two chapters. Before turning to examples, it may be well to mention that the thesis that every effectively computable total function isprimitiverecursive would simply be erroneous.
- Examples of recursive total functions that are not primitive recursive are described in the next chapter.