PARTIAL RECURSIVE FUNCTIONS.

CONTENTS

1. PRELIMINARY DEFINITIONS.

N = {0, 1, 2, …} is the set of *natural numbers*. In what follows we will feel free to say 'number', instead of 'natural number'.

Given a set A and a number $k \geq 0$, let A^k denote the set of all k-tuples of k elements of A. By definition we have that:

i) A^0 is the singleton made out of the 0-tuple <> (also called *empty tuple* or *empty string* and denoted by ε), that is, $A^0 = \{<>\}$,

ii) A^1 is equal to A (that is, we identify <x> with x for each $x \in A$), and

iii) $A^k = \{<x_1, \ldots, x_k> \mid x_1, \ldots, x_k \in A\}$ for any $k>1$.

Let A* denote the set of all finite k-tuples of elements of A, with $k \geq 0$, that is,

$$A^* = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^k \cup \ldots$$

The *length* of an element of A* is k iff it belongs to A^k, for $k \geq 0$.

Recall that if A is denumerable then A* is denumerable [Pettorossi 91, Chapter 1].

Let D be a (not necessarily proper) subset of N^k for some $k \geq 0$.

A *partial function* f of arity k, that is, of k arguments, from N^k to N is a subset of $D \times N$, with $D \subseteq N^k$, such that $\forall x \in D$ there exists a unique $m \in N$ such that $<x,m> \in f$.

The *domain* of a partial function f which is a subset of $D \times N$, is D. We say that D is the subset of N^k where f is *defined*. For every $x \in D$ we will also say that f is *convergent*.

We will say that a partial function f is *divergent* for an argument x (or f is *undefined* for an argument x) iff x does not belong to the domain D of f.

A partial function of arity k is said to be *total* iff $D = N^k$.

The *codomain* of a partial function from N^k to N is N. The *range* of a partial function f from N^k to N is the set $\{m \mid <x,m> \in f\}$, that is, the subset of N whose elements are the *values* of f.

Partial functions of arity 0, which are total, will be identified with elements of N. The domain of any partial function f which is total and has arity 0, is the singleton $A^0 = \{<>\}$. Thus, recalling that $N^0 = \{<>\}$, we will identify the function $\{<<>,m>\}$ with m, for each $m \in N$. There exists a unique partial function of arity 0 which is not total. It is the function which is undefined for the argument <>.

Unless otherwise specified, when we will say 'function' we will mean 'partial function'.

We write the application of the function f of arity k to the argument x as 'f(x)' or 'f x'. If x denotes the k-tuple $<x_1, \ldots, x_k>$ we also write that application as '$f(x_1,\ldots, x_k)$' or '$f\, x_1 \ldots x_k$' or '$f(z, x_k)$', where z denotes the (k–1)-tuple $<x_1, \ldots, x_{k-1}>$.

Instead of $<x,m> \in f$, we will write $m = f(x)$.

We sometimes use extra parentheses, and we write an expression e as (e).

Below we will define the set PRF_k of partial recursive functions of arity k, for $k \geq 0$. It is a subset of the set of all partial functions of arity k.

The set PRF of all partial recursive functions is given by $PRF_0 \cup PRF_1 \cup PRF_2 \cup \ldots$ By a total partial recursive function we mean a function in PRF which is total. Often in the literature a total partial recursive function is also called a total recursive function, or simply, a recursive function.

There exists a bijection, call it τ, between N* and N (see below). Thus, there is a bijection between the set PRF of all partial recursive functions of zero or more arguments and the set PRF_1 of all partial recursive functions of one argument (that is, from N to N).

We will see below that: i) this bijection τ and its inverse τ^{-1} are total partial recursive functions (actually, they are primitive recursive functions), and ii) the class of the partial recursive functions is closed under composition.

Thus, for the study of a class C of properties of partial recursive functions such that a partial recursive function g(n) from N to N has a property P in C iff the corresponding partial recursive function, say $g(\tau(x))$ from N^k to N, with $k \geq 0$ and $\tau(x) = n$, has property P, we may assume, without loss of generality, that the generic partial recursive function is from N to N.

For denoting functions we will also use the *lambda notation*. This means that a function which given the argument n, returns the value of the expression e, is written as the *lambda term* $\lambda n.e$.

Often, in order to make it explicit that the argument n may occur in the expression e, we will write $\lambda n.e[n]$, instead of $\lambda n.e$. We will allow ourselves to say 'the function e', instead of 'the function $\lambda n.e$', when the argument n is understood from the context.

Let us now introduce some elementary notions concerning the lambda notation which will be useful below.

Given a set K of constants (with k ranging over K) and a set V of variables (with v ranging over V), a lambda term e is either a constant c or a variable v or an abstraction $\lambda v.e$ or an application $(e_1\ e_2)$ (also denoted by $e_1(e_2)$).

We define the notion of 'p *occurs in* e' (or 'p is a *subterm of* e' or 'p *contains* e') as follows: i) p occurs in p, ii) if p occurs in e_1 or e_2 then p occurs in $(e_1\ e_2)$, and iii) if p occurs in e or p is v then p occurs in $\lambda v.e$.

An occurrence of a variable v in a term e is said to be *bound* iff it occurs in a subterm of e of the form $\lambda v.e_1$. An occurrence of a variable v in a term e is said to be *free* iff it is not bound. For instance, the leftmost occurrence of v in $v(\lambda v.(vx))$ is free, while the other occurrences are bound.

Given a lambda term e the set FV(e) of its free variables can be inductively defined as follows:

$FV(c) = \{\}$, $FV(v) = \{v\}$, $FV(\lambda v.e) = FV(e) - \{v\}$, $FV((e_1\ e_2)) = FV(e_1) \cup FV(e_2)$.

The result of the application of the function $\lambda n.e[n]$ to the argument m is equal to $e[m/n]$, where by $e[m/n]$ we mean the expression e where all free occurrences of n in e have been replaced by m.

For instance, i) $(\lambda n.n+1)\ 5 = 5+1$, and ii) $(\lambda n.n+((\lambda n.n+1)\ 3))\ 5 = 5+((\lambda n.n+1)\ 3) = 5+(3+1) = 9$.

Functions of zero arguments returning the value of e are written as $\lambda().e$ or simply e. For $k\geq 2$ we will write $\lambda x_1 \ldots x_k.e$ as an abbreviation for $\lambda x_1.(\ldots\ (\lambda x_k.e)\ldots)$.

2. PRIMITIVE RECURSIVE FUNCTIONS.

DEFINITION 1. The set of *primitive recursive functions* is the smallest set, call it PR, of functions, each function being from N^k to N for some $k\geq 0$,
which includes:

(i) the *zero* functions z_k of $k\geq 0$ arguments, that is, $\lambda x_1 \ldots x_k.0$,

(ii) the *successor* function s of one argument, that is, $\lambda x.x+1$, and

(iii) the *projection* functions π_{ki} of $k\geq 1$ arguments, that is, $\lambda x_1 \ldots x_k.x_i$, for $1\leq i\leq k$,

and it is closed under *composition* and *primitive recursion*, that is,

(iv) (*composition*) if the function h of $k\geq 1$ arguments is in PR and the k functions $g_1,\ldots, g_k$, each of $m\geq 0$ arguments, are in PR, then also the following function f of m arguments

$$f(x_1,\ldots,x_m) = h(g_1(x_1,\ldots,x_m),\ldots,\ g_k(x_1,\ldots,x_m))$$

is in PR, and

(v) (*primitive recursion*) if a function h of k+1 arguments and a function g of k–1 arguments are in PR for some $k\geq 1$, then also the function f of k arguments such that:

$$f(0,x_2,\ldots,x_k) = g(x_2,\ldots,x_k)$$
$$f(x_1+1,x_2,\ldots,x_k) = h(x_1,x_2,\ldots,x_k,\ f(x_1,x_2,\ldots,x_k))$$

is in PR. ■

The *identity function* $\lambda x.x$ is primitive recursive, because it is a particular projection function.

LEMMA 2. All partial functions of arity k, for $k\geq 0$, which are constant functions, are primitive recursive, that is, for all $n\geq 0$ and $k\geq 0$, $\lambda x_1 \ldots x_k.n$ is primitive recursive.

PROOF. By induction on the number k of arguments.

(Basis) For k=0 we have that: (i) $\lambda().0$ is primitive recursive by Definition 1(i), and (ii) for all $n\geq 0$ $(\lambda().n+1)() = s((\lambda().n)())$, where for any 0-ary function f we denote by f () the application of f to the empty tuple of arguments. Thus, by Definition 1(iv) we have that for all $n\geq 0$ if the function $\lambda().n$ is primitive recursive then also the function $\lambda().n+1$ is primitive recursive.

(Step) Let us consider the constant function $f_m = \lambda x_1 \ldots x_m.n$. Then the function $f_{m+1} = \lambda x_1 \ldots x_m x_{m+1}.n$ is defined as follows:

$$f_{m+1}(0,x_2,\ldots,x_{m+1}) = f_m(x_2,\ldots,x_{m+1})$$
$$f_{m+1}(x+1,x_2,\ldots,x_{m+1}) = \pi_{m+2,m+2}(x,\ x_2,\ldots,x_{m+1},\ f_m(x_2,\ldots,x_{m+1})).$$ ■

LEMMA 3. Primitive recursive functions are closed under: (i) addition of dummy arguments, (ii) substitution of constants for variables, (iii) identification of arguments, and (iv) permutation of arguments.

PROOF. The reader may convince himself by looking at the following examples.

(i) Construction of $\lambda xy.f(x,y)$ from $\lambda x.g(x)$ such that $f(x,y) = g(x)$ for all x and y. By primitive recursion we have:
$f(x,0) = g(x) \quad f(x,y+1) = \pi_{33}(x,y,f(x,y))$.

(ii) Construction of $\lambda x.g(x)$ from $\lambda xy.f(x,y)$ such that $g(x) = f(x,3)$. By composition we have:
$g(x) = f(\pi_{21}(x,y), (\lambda xy.3)(x,y))$.

(iii) Construction of $\lambda x.g(x)$ from $\lambda xy.f(x,y)$ such that $g(x) = f(x,x)$. By composition we have:
$g(x) = f(\pi_{21}(x,y), \pi_{21}(x,y))$.

(iv) Construction of $\lambda xy.g(x,y)$ from $\lambda xy.f(x,y)$ such that $g(x,y) = f(y,x)$. By composition we have: $g(x,y) = f(\pi_{22}(x,y), \pi_{21}(x,y))$. ∎

When showing that some functions are primitive recursive we will allow ourselves to make use of Lemma 3 and to adopt abbreviated notations.

For instance, we will give the proof that the addition is primitive recursive by writing the following equations:

$sum(0,y) = y \qquad sum(x+1,y) = sum(x,y)+1$ (by primitive recursion)

instead of

$sp(x,y,z) = (\lambda x.x+1)(\pi_{33}(x,y,z))$ (by composition)

$sum(0,y) = (\lambda x.x)y \qquad sum(x+1,y) = sp(x,y,sum(x,y))$ (by primitive recursion).

If we apply Lemma 3 the Composition Schema (Definition 1.iv) may be generalized by allowing some of the g_i's to be functions of some, but not all the variables $x_1,\ldots,x_m$. Analogously, the Primitive Recursion Schema (Definition 1.v) may be generalized by allowing the function h to have as arguments only some of the m+1 arguments $x_1,\ldots,x_m$, $f(x_1,x_2,\ldots,x_k)$.

The functions defined by equations like the ones of case v) of Definition 1, which are called *recursive equations*, are assumed to be evaluated in the 'call-by-value' (or 'inside-out and left-to-right' [Roger 67]) mode.

This means that the evaluation of an expression proceeds by

i) considering the *leftmost* subexpression, say u, among all *innermost* subexpressions which match a left-hand-side of a recursive equation, and

ii) replacing u by $(R\ \theta)$, if u is equal to $(L\ \theta)$ for some recursive equation $L = R$ and substitution θ.

It can be shown that if we assume the call-by-value mode of evaluation, the functions defined by composition and primitive recursion are uniquely determined by the corresponding recursive equations (Recursion Theorem, see [Enderton 77]). Thus, using the call-by-value mode of evaluation, for all x in the domain of f we can compute the unique element of N which is the value of f(x).

It follows from Definition 1 that the set of primitive recursive functions is denumerable. Indeed, for the set PR we have the following domain equation [Pettorossi 91, Chapter 4]:

$$PR ::= N \mid \{1\} \mid N \times N \mid PR \times PR^* \mid PR \times PR.$$

Every primitive recursive function is total.
Almost all "algorithmic functions" of ordinary mathematics are primitive recursive. Here is the proof for some of them. The proof is given by exhibiting the recursive equations of those functions. We leave to the reader to fill in the details and to show how these recursive equations can be replaced by suitable applications of the composition and the primitive recursion operators. We have:

1. (addition) $\quad sum(0,y) = y \quad sum(x+1,y) = sum(x,y)+1$
   We will also write sum(x,y) as x+y.

2. (predecessor: $x \neg 1 =$ *if* $x=0$ *then* 0 *else* $x-1$)
   $0 \neg 1 = 0 \quad (x+1) \neg 1 = x$

3. (proper subtraction: $x \neg y =$ *if* $y \geq x$ *then* 0 *else* $x-y$)
   $x \neg 0 = x \quad x \neg (y+1) = (x \neg y) \neg 1$
   If $x \geq y$ we will feel free to write $x-y$, instead of $x \neg y$. As often done in the literature we used for proper subtraction the same symbol as for predecessor. Notice that if $x \geq y$ then $x \neg y = x-y$, where '–' denotes the usual subtraction operation between natural numbers we learnt at school.

4. (multiplication) $\quad 0\, y = 0 \quad (x+1)\, y = (xy) + y$

5. (exponentiation) $\quad x^0 = 1 \quad x^{y+1} = x^y\, x$

6. (factorial) $\quad 0! = 1 \quad (x+1)! = (x+1)\, x!$

7. (sign: $sg(x) =$ *if* $x=0$ *then* 0 *else* 1)
   $sg(0) = 0 \quad sg(x+1) = 1$

8. (negated sign: $nsg(x) =$ *if* $x=0$ *then* 1 *else* 0)
   $nsg(0) = 1 \quad nsg(x+1) = 0$

9. (min) $\quad min(x,y) = x \neg (x \neg y)$

10. (max) $\quad max(x,y) = x + (y \neg x)$

11. (absolute difference: $|x-y| = max(x,y) \neg min(x,y)$)
    $|x-y| = (x \neg y) + (y \neg x)$

    We have that: for any a and b in N,

$$\textit{if } x=0 \textit{ then } a \textit{ else } b \;=\; a\, nsg(x) + b\; sg(x)$$
$$\textit{if } x=y \textit{ then } a \textit{ else } b \;=\; a\, nsg(|x-y|) + b\; sg(|x-y|),$$

    and in particular, for any a in N,

$$\textit{if } x=y \textit{ then } 0 \textit{ else } a \;=\; \textit{if } x\neq y \textit{ then } a \textit{ else } 0 \;=\; a\;\; sg(|x-y|)$$
$$\textit{if } x=y \textit{ then } a \textit{ else } 0 \;=\; \textit{if } x\neq y \textit{ then } 0 \textit{ else } a \;=\; a\, nsg(|x-y|)$$

12. (rem(x,y) = integer remainder when x is divided by y. To get a total function we assume that rem(x,0)=x). Notice that, contrary to what happens in standard arithmetics, rem(x,y) < y iff y>0.

We have that: rem(0,y) = 0

rem(x+1,y) = *if* rem(x,y)+1≠y *then* rem(x,y)+1 *else* 0

Thus, rem(0,y) = 0

rem(x+1,y) = (rem(x,y)+1) sg(|y–(rem(x,y)+1)|)

This last equation can be rewritten as the following two equations:

rem(x+1,y) = g(y, rem(x,y))

g(y,z) = (z+1) sg(|y–(z+1)|)

and the function g is primitive recursive because it can be obtained by composition.

13. (quot(x,y) = integer quotient when x is divided by y. To get a total function we assume that quot(x,0)=0).

We have that: quot(0,y) = 0

quot(x+1,y) = *if* rem(x,y)+1≠y *then* quot(x,y) *else* quot(x,y)+1

= quot(x,y) + *if* rem(x,y)+1≠y *then* 0 *else* 1

Thus, quot(0,y) = 0

quot(x+1,y) = quot(x,y) + nsg(|y–(rem(x,y)+1)|)

14. (divisibility: div(x,y) = *if* x is divided by y *then* 1 *else* 0. To get a total function we assume that x is divided by 0 iff x=0, that is, div(x,0)=1 iff x=0.).

We have that: div(x,y) = nsg(rem(x,y)).

Here are some examples related to the functions quot, rem, and div.

i) For the division 0/4 we have: quot(0,4)=0 and rem(0,4)=0.

ii) For the division 4/0 we have: quot(4,0)=0 and rem(4,0)=4.

iii) div(0,x) = nsg(rem(0,x)) = 1 for any x≥0.

iv) div(x,0) = nsg(rem(x,0)) = 0 for x≠0.

A *predicate* P on N^k for k≥0, is a subset of N^k. If x ∈ P we also write P(x) and we say that P(x) holds. Given a predicate P on N^k, its *characteristic function* f_P is a function of arity k from N^k to N, defined as follows: $f_P(x)$ = *if* x ∈ P *then* 1 *else* 0, for any x ∈ N^k.

A *predicate* is said to be *primitive recursive* iff its characteristic function is primitive recursive. For instance, the predicates x<y, eq(x,y) (also written as x=y), and x>y, for x,y ∈ N are primitive recursive. Indeed, $f_{x<y}$ = sg(y¬x), $f_{eq(x,y)}$ = nsg(|x–y|), and $f_{x>y}$ = sg(x¬y).

Lemma 3 can easily be extended to primitive recursive predicates. We leave that extension to the reader.

LEMMA 4. Primitive recursive predicates are closed under negation, (finite) disjunction, and (finite) conjunction.

PROOF. i) $f_{\text{not } P}(x)$ = 1¬$f_P(x)$, ii) $f_{P \text{ or } Q}(x,y)$ = sg($f_P(x)$+$f_Q(y)$), and iii) $f_{P \text{ and } Q}(x,y)$ = $f_P(x)$ $f_Q(y)$. ■

We have that the predicate x≤y is primitive recursive, because x≤y iff (x<y or eq(x,y)).

Given a predicate P on N^{k+1} for $k \geq 0$, the *bounded existential quantification* of P is the predicate Q on N^{k+1} such that Q(x,n) iff $\exists$ i. [0,n]. P(x,i).

Analogously, for the *bounded universal quantification* by replacing $\forall$ for $\exists$.

LEMMA 5. Primitive recursive predicates are closed under bounded existential quantification and bounded universal quantification.
PROOF. If $f_P(x,n)$ is the characteristic function of the predicate P(x,n) then the characteristic function $f_Q(x,n)$ of the predicate Q(x,n) = $\exists$ i. [0,n]. P(x,i) is defined by primitive recursion as follows:

$f_Q(x,0) = f_P(x,0) \qquad f_Q(x,n+1) = f_Q(x,n) + f_P(x,n+1)$.

For the bounded universal quantification, it is enough to recall that Lemma 4 and the fact that $\forall$ i. [0,n]. P(x,i) is the same as not($\exists$ i. [0,n]. not(P(x,i))). ■

LEMMA 6. (*Definition by cases*) Let us consider the set $\{<g_i(x), P_i(x)> \mid 1 \leq i \leq r \text{ and } r \geq 2\}$, where for $1 \leq i \leq r$, $g_i(x)$ is a primitive recursive function from N^k to N and $P_i(x)$ is a primitive recursive predicate subset of N^k.

Let us assume that for each x in N^k *one and only one* of the predicates $P_i(x)$, for $1 \leq i \leq r$, holds. Then the function f defined by the following equations:

$$\begin{aligned} f(x) &= g_1(x) && \textit{if } P_1(x) \\ &= g_2(x) && \textit{if } P_2(x) \\ &\dots \\ &= g_r(x) && \textit{if } P_r(x) \end{aligned}$$

is primitive recursive.
PROOF. The case where r=1 is obvious, because the definition of f(x) corresponds to the equation $f(x) = g_1(x)$ *if* true. For r>1we have the following. Let $c_i(x)$ be the characteristic function of $P_i(x)$ for $1 \leq i \leq r$. Then $f(x) = (\dots(g_1(x)\, c_1(x) + g_2(x)\, c_2(x)) + \dots + g_r(x)\, c_r(x))$. ■

Notice that the case where r=2 corresponds to the equation:

$f(x) =$ *if* P(x) *then* $g_1(x)$ *else* $g_2(x)$,

which we also write as:

$$\begin{aligned} f(x) &= g_1(x) && \textit{if } P(x) \\ &= g_2(x) && \textit{otherwise}. \end{aligned}$$

REMARK. In a definition by cases the evaluation of the predicates $P_1(x), P_2(x), \dots, P_r(x)$ is done sequentially in that order. Since the corresponding characteristic functions are total, this condition is not important in the context of the primitive recursive functions. However, when we will allow partial characteristic functions, as we will do in the sequel, this sequentiality condition is important, because otherwise, the definition by cases may fail to specify a partial function. ■

Let us now define the *bounded minimalization operator* min. It allows us to define a function, say f, of arity k+1, with $k \geq 0$, starting from a given predicate Q, subset of N^{k+1}, as follows. For any $x \in N^k$ and $n \in N$

$$\begin{aligned} f(x,n) &= \text{the minimum } y \text{ in } [0,n] \text{ such that } Q(x,y) \text{ holds} && \textit{if } \text{there exists } y \text{ in } [0,n] \text{ s.t. } Q(x,y) \text{ holds} \\ &= 0 && \textit{otherwise}. \end{aligned}$$

The function f(x,n) is also denoted by 'min y. [0,n]. Q(x,y)'.

In the '*otherwise* clause' we could have chosen for f(x,n) a value different from 0 without modifying the theory of the partial recursive functions. The choice of 0 is convenient when formulating Lemma 10 and in particular, when establishing Lemma 10 Point iv).

LEMMA 7. If the predicate Q(x,y) subset of N^{k+1}, for k≥0, is primitive recursive then the function f(x,n) = min y. [0,n]. Q(x,y) is primitive recursive.
PROOF. Let us consider the function

g(x,n,m) = m *if* ∃ i. [0,n]. Q(x,i)
= n+1 *if* not(∃ i. [0,n]. Q(x,i)) and Q(x,n+1)
= 0 *if* not(∃ i. [0,n+1]. Q(x,i)).

The function g is primitive recursive because it is defined by cases starting from primitive recursive functions and primitive recursive predicates. We will show that f(x,n) is primitive recursive by showing that it is equal to the following function h defined by primitive recursion, as follows:

h(x,0) = 0
h(x,n+1) = g(x,n,h(x,n)) (recall that g is primitive recursive).

By unfolding the definition of g, we have that the function h is also defined by the equations:

h(x,0) = 0
h(x,n+1) = h(x,n) *if* ∃ i. [0,n]. Q(x,i)
= n+1 *if* not(∃ i. [0,n]. Q(x,i)) and Q(x,n+1)
= 0 *if* not(∃ i. [0,n+1]. Q(x,i)).

Now, recalling the definition of the bounded minimalization operator min, we also have that:
i) f(x,0) = min y. [0,0]. Q(x,y) = *if* Q(x,0) *then* 0 *else* 0 = 0, and
ii) f(x,n+1) = *if* ∃ i. [0,n]. Q(x,i) *then* f(x,n) *else if* Q(x,n+1) *then* n+1 *else* 0.

Thus, we conclude that h(x,n) = f(x,n) for all x ∈ N^k, and n ∈ N, because they are defined by the same recursion equations. ■

By using Lemma 7 we can show that many other mathematical functions are primitive recursive. For instance, we have that the integer-squareroot(x), defined as the largest number whose square does not exceed x, is a primitive recursive function. Indeed,
integer-squareroot(x) = min y. [0,x]. ($y^2 \leq x$ and $(y+1)^2 > x$).

LEMMA 8. Let h(n) be a primitive recursive function from N to N and f(x,m) a primitive recursive function from N^{k+1} to N, where x is a k-tuple of numbers and m is a number, of the form: min y. [0,m]. Q(x,y). Then the function g(x,n) = min y. [0,h(n)]. Q(x,y) is primitive recursive.
PROOF. The function g(x,n) is defined by composition as follows:
g(x,n) = f(x,h(n)) = f($\pi_{k+1\ 1}$(x,n), …, $\pi_{k+1\ k}$(x,n), h($\pi_{k+1\ k+1}$(x,n))). ■

LEMMA 9. Let f(x,n) be a primitive recursive function from N^{k+1} to N. Then the *bounded sum*: of f(x,n) below n, denoted by $\Sigma_{n<r}$ f(x,n), defined as follows:
$\Sigma_{n<r}$ f(x,n) = *if* r=0 *then* 0 *else* (f(x,0) + f(x,1) + … + f(x,r–1)),

and the *bounded product*: of f(x,n) below n, denoted by $\prod_{n<r} f(x,n)$, defined as follows:

$\prod_{n<r} f(x,n) =$ *if* r=0 *then* 1 *else* (f(x,0) f(x,1) … f(x,r–1)),

are primitive recursive. The same holds if we replace < by ≤. Often the qualification 'below n' is dropped.

PROOF. We have the following primitive recursive equations:

$$\sum_{n<0} f(x,n) = 0, \qquad \sum_{n<(r+1)} f(x,n) = (\sum_{n<r} f(x,n)) + f(x,r), \quad \text{and}$$
$$\prod_{n<0} f(x,n) = 1, \qquad \prod_{n<(r+1)} f(x,n) = (\prod_{n<r} f(x,n)) \;\; f(x,r).$$

For instance, if we write $\sum_{n<r} f(x,n)$ as S(x,r), we have that:

S(x,0) = 0, S(x,r+1) = S(x,r) + f(x,r).

S(x,r) is a primitive recursive function, because S(x,r) + f(x,r) = sum1(x,r,S(x,r)) and sum1(x,r,m) is a primitive recursive function. Indeed, we have that:

sum1(x,r,m) = sum(π_{33}(x,r,m), f(π_{31}(x,r,m),π_{32}(x,r,m))).

When replacing < by ≤ we have: $\sum_{n\leq r} f(x,n) = \sum_{n<(r+1)} f(x,n)$, and $\prod_{n\leq r} f(x,n) = \prod_{n<(r+1)} f(x,n)$. They are primitive recursive by Lemma 8. ■

LEMMA 10. The following functions are primitive recursive:

i) D(n) = the number of divisors of n, according to the following definition: y is a divisor of x iff div(x,y)=1 and y≤x. Thus, we have: D(0)=1, D(1)=1, D(2)=2, D(3)=2, D(4)=3, etc.

ii) Pr(n) = *if* n is prime *then* 1 *else* 0

iii) p(n) (also written as p_n) = the n-th prime number, defined as follows: p_0=0, p_1=2, p_2=3, p_3=5, p_4=7, p_5=11, etc.)

iv) e(x,n) = *if* n=0 or x=0 *then* 0 *else* the exponent of p_n in the prime factorization of x. (Thus, e(x,1), e(x,2), e(x,3), … are the exponents of 2, 3, 5, … respectively, in the prime factorization of x.)

v) the inverse functions of the pair function C(x,y) = 2^x (2y+1) ¬ 1, which is a bijection from N^2 to N.

PROOF. i) D(x) = D1(x,x) where the function D1 is defined as follows: D1(x,0) = div(x,0), D1(x,n+1) = D1(x,n) + div(x,n+1).

ii) Pr(x) = *if* D(x)=2 *then* 1 *else* 0. (D(x)=2 means that x>1 and the only divisors of x are 1 and x.) Thus, Pr(x) = nsg(|D(x) – 2|).

iii) p_0=0 and p_{n+1} = min z. [0, p_n! +1]. (z > p_n and z is prime). This is a definition by primitive recursion, because i) (z > p_n and z is prime) is a primitive recursive predicate, ii) we may apply Lemma 8 for h(n) = p_n! +1, and iii) the remainders of the divisions of p_n!+1 by p_0, p_1, …, and p_n, respectively, are all different from 0, and thus, in the interval [p_n+1, p_n! +1] there is at least one prime.

iv) e(x,n) = min z. [0, x]. (div(x,p_n^{z+1}) = 0).

v) Let n be C(x,y). Thus, n+1 is 2^x (2y+1). We have: x = e(n+1,1), because 2y+1 is odd, and y = quot(quot(n+1, 2^x)¬1, 2). ■

An element <x1, x2, …, xk> of N^k, for k≥0, can be encoded by an element b of N as follows:

β(x1, x2, …, xk) = *if* k=0 *then* 1 *else* $p_1^{x1+1}\, p_2^{x2+1} \dots p_n^{xk+1}$.

From the value B of β(x1, x2, …, xk) we can get the values of k and xi for i = 1,…, k, as follows:

$k =$ *if* $B=1$ *then* 0 *else* $\min z.\ [0,B].\ (e(B,z+1) = 0)$, and
for $i = 1,\ldots, k$: $xi = e(B,i) \neg 1$.

The encoding β from N^* to N is *not* a bijection. Indeed, no element x exists in N^* such that $\beta(x) = 3$ (the same holds for $\beta(x) = 0$).

The following encoding τ of k-tuples is a bijection from N^* to N.
Given an element $\langle x1, x2, \ldots, xk\rangle$ of N^k, for $k\geq 0$, let τ be the following function with arity k:

$$\tau(x1, x2, \ldots, xk) = \textit{if } k=0 \textit{ then } 0 \textit{ else } 2^{x1} + 2^{x1 + x2 + 1} + 2^{x1 + x2 + x3 + 2} + \ldots + 2^{x1 + x2 + \ldots + xk + k \neg 1}.$$

If $k=0$, $\tau(x1, x2, \ldots, xk) = \tau(\langle\rangle) = 0$. For $k>0$ $\tau(x1, x2, \ldots, xk) > 0$.
We have that τ is primitive recursive.

Now, given the value of $\tau(x1, x2, \ldots, xk)$ we can compute τ^{-1}, that is, we can derive the values of $x1, x2, \ldots, xk$ as follows, by using primitive recursive functions only.
We have $\tau^{-1}(0) = \langle\rangle$.

Given any $x\geq 0$ it has a unique expression of the form:

$$x = \alpha(0,x)\, 2^0 + \ldots + \alpha(i,x)\, 2^i + \ldots \qquad \text{with } \alpha(i,x) \in \{0,1\} \text{ for } i\geq 0.$$

This is the usual binary expansion of the number x.

Given any $x>0$, it has a unique expression of the form:

$$x = 2^{b(1,x)} + 2^{b(2,x)} + \ldots + 2^{b(M(x),x)} \qquad \text{with } 0 \leq b(1,x) < b(2,x) < \ldots < b(M(x),x) \text{ and } 1 \leq M(x)$$

where $M(x)$ is the number of the powers of 2 occurring in the binary expansion of x. The number x has also a unique expression of the form:

$$x = 2^{a(1,x)} + 2^{a(1,x) + a(2,x) + 1} + 2^{a(1,x) + a(2,x) + a(3,x) + 2} + \ldots$$
$$\ldots + 2^{a(1,x) + a(2,x) + \ldots + a(M(x),x) + k\neg 1}$$

$$\text{with } a(j,x) \geq 0 \text{ for } 1\leq j<M(x) \text{ and } 1 \leq M(x).$$

For example, if $x = 2^5 + 2^8 + 2^{15}$ we have that:
$b(1,x)=5$, $b(2,x)=8$, $b(3,x)=15$, $M(x)=3$, $a(1,x)=5$, $a(2,x)=2$, $a(3,x)=6$, and $\tau^{-1}(x)=\langle 5,2,6\rangle$.

The following expressions show that the function τ^{-1} is primitive recursive:

i) $\alpha(0,x) = rem(x,2)$ $\qquad \alpha(i+1,x) = \alpha(i,quot(x,2))$

ii) Let us denote b(M(x),x) by E(x). E(x) is the highest exponent of 2 occurring in the binary expansion of x. We have:
E(x) = *if* x=0 *then* 0 *else* min z. [0,x]. ($2^{z+1} > x$)

iii) M(x) = *if* x=0 *then* 0 *else* K(E(x),x) where K(0,x) = α(0,x)
K(j+1,x) = K(j,x) + α(j+1,x)
K(E(x),x) is the sums of the exponents of the powers of 2 occurring in the binary expansion of x. We also have: M(x) = $\sum_{i<x} \alpha(i,x)$ (recall that since $x < 2^x$ for any x≥0, we have that α(i,x)=0 for i≥x).

iv) b(i,x) = *if* x=0 or i=0 or i>M(x) *then* 0 *else*
if i=1 *then* r *else* b(i–1, x–2^r) where r = min z. [0,x]. (α(z,x) > 0).
Thus, r is the smallest power of 2 occurring in the binary expansion of x.
Notice that if 'x=0 or i=0 or i>M(x)' is false then i is in the interval [1,M(x)].
We also have: b(i,x) = min z. [0,x]. (($\sum_{k\leq z} \alpha(k,x)$) = i).

v) The function a(j,x) can be expressed as follows:
a(j,x) = *if* x=0 or j=0 or j>M(x) *then* 0 *else*
if j=1 *then* b(1,x) *else* ((b(j,x) ∸ b(j–1,x)) ∸ 1).

EXERCISES.

i) Show that primitive recursive functions are closed under mutual recursive definitions.
Without loss of generality let us consider the following functions with 1 argument only:

g(0) = a   g(n+1) = r(n, g(n), h(n))
h(0) = b   h(n+1) = s(n, g(n), h(n)),

where the functions r and s are primitive recursive.

We have that the function: f(n) = $2^{g(n)} 3^{h(n)}$ is primitive recursive, because:

f(0) = $2^a 3^b$
f(n+1) = $2^{g(n+1)} 3^{h(n+1)} = 2^{r(n,\, g(n),\, h(n))} 3^{s(n,\, g(n),\, h(n)))}$ =
= $2^{r(n,u,v)} 3^{s(n,u,v)}$ where (u, v) = (e(f(n),1), e(f(n),2)).
(The definition of e(x,n) is given in Lemma 10 Point (iv).)

Thus, g(n) = e(f(n),1) and h(n) = e(f(n),2).

ii) Show that the Fibonacci function is primitive recursive.
We have that:
fib(0) = 0, fib(1) = 1, fib(n+2) = fib(n+1) + fib(n).

We have that the function f(n) = $2^{fib(n+1)} 3^{fib(n)}$ is primitive recursive, because:

f(0) = $2^1 3^0$
f(n+1) = $2^{fib(n+2)} 3^{fib(n+1)} = 2^{u+v} 3^u$ where (u,v) = (fib(n+1),fib(n))
= $2^{u+v} 3^u$ where (u,v) = (e(f(n),1), e(f(n),2)).
(The definition of e(x,n) is given in Lemma 10 Point (iv).)

Thus, fib(n) = e(f(n),2). ■

REMARK. The *course-of-value recursion* can be reduced to primitive recursion (see the work by R. Péter presented in [Kleene 71, page 273]), in the sense that we get an equivalent definition of the set of the primitive recursive functions if in Definition 1 we replace the second equation of v) by the following one:

$$f(x_1+1,x_2,\ldots,x_k) = h(x_1,x_2,\ldots,x_k, f(x_1,x_2,\ldots,x_k), f(x_1-1,x_2,\ldots,x_k), \ldots, f(0,x_2,\ldots,x_k)).$$

■

3. PARTIAL RECURSIVE FUNCTIONS.

We now introduce the *minimalization operator* μ w.r.t. a class C of functions. Given a function t from N^{k+1} to N for some $k\geq 0$, in the class C, the minimalization operator μ allows us to define a new function f denoted by $\lambda x.(\mu y.[t(x,y) = 0])$ from N^k to N such that for any $x \in N^k$,

f(x) is the minimum y such that $t(x,y) = 0$ *if* such a y exists in N,
is undefined *otherwise*.

In the literature there exists a different *minimalization operator*, call it μ^*, w.r.t. a class C of functions. Given a function t from N^{k+1} to N for some $k\geq 0$, in the class C, the minimalization operator μ^* allows us to define a new function f* denoted by $\lambda x.(\mu^* y.[t(x,y) = 0])$, such that for any $x \in N^k$,

f*(x) is the minimum y such that (i) t(x,d) is defined for all $d\leq y$, and (ii) $t(x,y) = 0$
if such a y exists in N,
is undefined *otherwise*.

REMARKS. i) When applying the operators μ and μ^* we need to know the class C of functions from which the function t is taken.
ii) When introducing the minimalization operators, one could consider, instead of the predicate $\lambda x,y.[t(x,y) = 0]$, the predicate $\lambda x,y.[t(x,y) = r]$ for some fixed number $r \in N$. This different choice of r does not modify the theory of the partial recursive functions. ■

Now we define the set PRF of the partial recursive functions and the sets PRF_k, for any $k\geq 0$, of the partial recursive functions of arity k.
DEFINITION 11. The set PRF of all *partial recursive functions* (p.r.f.'s, for short) is the smallest set of partial functions which includes: (i) the zero functions, (ii) the successor function, and (iii) the projection functions (as defined in Definition 1) and closed w.r.t. the following operations: (i) composition, (ii) primitive recursion, and (iii) minimalization μ w.r.t. the subset of the partial recursive functions themselves which are *total* functions.
The set PRF_k of the partial recursive functions of arity k, for any $k\geq 0$, is the subset of PRF, which includes all partial recursive functions of arity k. ■

One can show that the set PRF_k of partial recursive functions is *not* closed under μ operations w.r.t. the class PRF_{k+1} of functions (Notice that PRF_{k+1} include also partial functions). In particular, for k=1 there exists a partial recursive function f such that $\lambda n.(\mu y.[f(n,y) = 0])$ is *not* a partial recursive function [Roger 67, page 42].
One can also give the following equivalent definition of the set of the partial recursive functions.

DEFINITION 12. The set PRF of the partial recursive functions is the smallest set of partial functions which: (i) includes all primitive recursive functions, and (ii) it is closed w.r.t.: (ii.1) composition, (ii.2) primitive recursion, and (ii.3) minimalization μ^* w.r.t. the same set PRF of the partial recursive functions. ∎

THEOREM 13. Each partial recursive function of one argument can be expressed as $\lambda n.p(\mu y.[t(n,y) = 0])$, where p and t are primitive recursive functions. ∎

As a consequence of Theorem 13, the set of p.r.f.'s is denumerable. Indeed, they are as many as the pairs of primitive recursive functions, that is, $|N \times N|$ (which is equal to $|N|$).

REMARK. A statement similar to the one of Theorem 13 where the function p is omitted, does not hold. In particular, there exists a p.r.f. f such that no total p.r.f. g exists such that:

$f = \lambda n.(\mu y.[g(n,y) = 0])$. ∎

THEOREM 14. (Kleene Normal Form) The set PRF_1 of the p.r.f.'s of arity 1 can be obtained by choosing two fixed primitive recursive functions P (of arity 1) and T (of arity 3). We have that $PRF_1 = \{\lambda n.P(\mu y.[T(i,n,y) = 0]) \mid i \geq 0\}$. ∎

The function T is also called the Kleene predicate. It can be described by saying that $T(i,n,y) = 0$ iff i is the encoding of a Turing Machine M such that M with input n computes a value in fewer than y computation steps. (The chapter on Turing Machines shows how one can encode them into numbers.)

Since the function T is primitive recursive Theorem 14 holds also for μ^*, instead of μ.

For $i \geq 0$ the expression $\lambda n.P(\mu y.[T(i,n,y) = 0])$ defines the i-th partial recursive function of arity 1. Every partial recursive function of arity 1 has infinitely many indices, that is, for every p.r.f. f there exists an infinite subset I of N such that $\forall i \in I.\ f = \lambda n.P(\mu y.[T(i,n,y) = 0])$.

Some partial recursive functions are total. Indeed, this is the case when, with reference to Definition 11, for all $x \in N^k$ there exists a minimum value of y such that $t(x,y) = 0$.

A total p.r.f. is also called a *recursive function*.

One can show that the set of the total partial recursive functions can be defined as in Definition 11 with the extra condition that the total partial recursive function t should satisfy the following: $\forall\ x \in N^k.\ \exists\ y \in N.\ t(x,y) = 0$.

A predicate, subset of N^k, is said to be *recursive* iff its characteristic function from N^k to N is a recursive function.

Unfortunately, in the literature partial recursive functions sometimes are called 'recursive functions' tout court. The reader should be aware of this ambiguous terminology.

There are p.r.f.'s which are always undefined, that is, p.r.f.'s whose domain is empty. One such function is, for instance, the following: $\lambda n.(\mu y.[t(n,y) = 0])$, where $t = \lambda ny.1$.

Not every total p.r.f. is a primitive recursive function. A p.r.f. which is total and it is not primitive recursive is the Ackermann function from $N \times N$ to N defined as follows:

for any n, m $\in$ N, $ack(0,n) = n+1$
$ack(m+1,0) = ack(m,1)$
$ack(m+1,n+1) = ack(m,ack(m+1,n))$.

We omit the proof that Ackermann function is a p.r.f., and it is *not* primitive recursive. The proof that the Ackermann function is not primitive recursive is based on the fact that it grows 'faster' than any primitive recursive function (see also [Cutland 80, page 194]).

The total p.r.f.'s are as many as $|N|$.

There exists a p.r.f. f from N to N which cannot be extended to a total p.r.f., call it g, from N to N, in the sense that for all $n \in N$ if f(n) is convergent then f(n) = g(n).

Indeed, assume that for each p.r.f. f_i there exists a total p.r.f. f_{i*} which extends f_i. Let us consider the function h such that for all $i \geq 0$,

h(i) = *if* $f_i(i)$ is defined *then* $f_i(i) + 1$ *else* undefined.

Thus, we have that:

$h(i) = \pi_{22}(\mu y.[T(i,i,y) = 0], f_i(i) + 1)$.

The function h is a p.r.f. (Recall that p.r.f.'s are evaluated in a call-by-value mode.) Thus, there exists an index r such that $f_r = h$. By assumption, we have that there exists the p.r.f. f_{r*} which is total. Since f_{r*} extends h and if $f_i(i)$ is defined then h is defined, we have that for all $i \geq 0$ if $f_i(i)$ is defined then $f_{r*}(i)$ is defined and it is equal to $f_i(i) + 1$. If we consider $i=r^*$ we get that:

if $f_{r*}(r^*)$ is defined then $f_{r*}(r^*)$ is defined and it is equal to $f_{r*}(r^*) + 1$,
which is a contradiction because f_{r*} is total. ■

From Definition 11 it follows that given a partial recursive function g of k+1 arguments which is total, then the function $\lambda x_1,\ldots,x_k.(\mu y.[g(x_1,\ldots,x_k,y) = 0])$ is a partial recursive function which is not necessarily total. To see this, it is enough to compute in succession the following expressions: $g(x_1,\ldots,x_k,0)$, $g(x_1,\ldots,x_k,1)$, $g(x_1,\ldots,x_k,2)$, etc., for some given arguments $x_1,\ldots,x_k$. Each of these computations terminates because g is total. The function $\lambda x_1,\ldots,x_k.(\mu y.[g(x_1,\ldots,x_k,y) = 0])$ is always undefined if for the given arguments $x_1,\ldots,x_k$ and all y's the evaluation of $g(x_1,\ldots,x_k,y)$ returns a value different from 0.

4. RECURSIVELY ENUMERABLE SETS AND PARTIAL RECURSIVE FUNCTIONS.

DEFINITION 15. A set A is *recursively enumerable*, r.e. for short, iff *either* $A = \emptyset$ *or* there exists a *total* p.r.f. f from N to N such that A = range(f). ■

This definition explains the name 'recursively enumerable' given to the set A. Indeed, assuming A not empty, we have that the elements of a recursively enumerable set A are 'enumerated, possibly with repetitions,' by a total p.r.f., f in the sense that $A = \{f(i) \mid i \geq 0\}$. The meaning of 'possibly with repetitions' is that in the sequence [f(0), f(1), f(2), ...] we may have identical elements of the set A.

Let us now introduce the notion of dove-tailing evaluation of a partial recursive function.

For this purpose we first need to consider a suitable 'model of computation' and a notion of 'computation step' relative to that model.

Given a partial recursive function f from N to N, we say that the *bounded evaluation* of f(i) for j computation steps is:

- f(i) if performing *at most* j computation steps the evaluation of f(i) gives us a number,
- the distinguished value # (not in N), otherwise.

Given a partial recursive function f from N to N, its *dove-tailing evaluation* is the construction of a two-dimensional matrix where at row i≥0 and column j≥0 we place the bounded evaluation of f(i) for j computation steps. This matrix is constructed element by element, in the increasing order (0), (1), (2), … of the entries, called *position numbers*, of the matrix of Figure 1. For any i, j ∈ N at row i and column j we have the unique position number

$$D(i,j) = ((i+j)^2 + 3i + j) / 2.$$

There are two functions r and c from N to N such that for any position number d≥0 we have that d occurs in the dove-tailing matrix at row r(d) and column c(d). The two functions r and c are primitive recursive, because, as the reader may verify, r(d) = d–S and c(d) = s–r(d), with S = quot($(s^2$+s), 2) and s = quot(integer-squareroot(8d+1)+1, 2) – 1.

Computation steps j :	0	1	2	3	…
i : 0	(0)	(1)	(3)	(6)	…
1	(2)	(4)	(7)	…	
2	(5)	(8)	…		
3	(9)	…			
…	…				

Figure 1. The matrix of the position numbers for the dove-tailing evaluation of a p.r.f. from N to N.

We now state an important result about the dove-tailing evaluation: there exists a partial recursive function, call it I, which given any partial recursive function f and the numbers i and j, behaves as an interpreter and computes the value of the bounded evaluation of f(i) for j computation steps. The function I is a total function as we will see below.

The proof of the existence of the function I (given in [Cutland 80 Chapter 5, Sanchis 88]) relies on the existence of the Universal Turing Machine and on the fact (not proved here) that there exists a bijection between the set PRF of the partial recursive functions and the set of the sequences of instructions for Turing Machines (which are quintuples in the formalism we have introduced in Chapter 2), that is, the set of the Turing Machines themselves. This bijection exists because of i) the Bernstein Theorem [Pettorossi 91], ii) the injection from PRF to the set of the Turing Machines, and iii) the injection from the set of the Turing Machines to PRF.

Since a sequence of instructions can be encoded by a sequence of characters, and therefore, by a sequence of numbers (see, for instance, [Pettorossi 91]), and consequently by a single number (see the encoding function τ above), we have that any partial recursive function, say f, can be encoded by a number, say $\tau^*(f)$, which is called the *number encoding* of f.

Thus, when we said that I takes as input a p.r.f. f we actually meant that it takes the number $\tau^*(f)$ as input.

The function I is a total function from N^3 to $N \cup \{\#\}$. It can also be considered as a function from N to N because of the bijection between N^3 and N, and the one between $N \cup \{\#\}$ and N.

The function I is *total*. It extracts from the value of $\tau^*(f)$ the sequence of instructions corresponding to the function f, and then it interprets that sequence as a program for evaluating f(i) for j computation steps.

AN INFORMAL PROOF. To show in an informal way the existence of the function I, let us now see how it can be constructed by referring to the recursive equations which define a given p.r.f.

Indeed, given a p.r.f., say f, it can be viewed as a set of recursive equations, constructed from the zero, successor, and projection functions, by using composition, primitive recursion, and minimalization. Those recursive equations in turn, can be viewed as a program for the function f. Let $\tau^*(f)$ be the number encoding of that program.

Let us consider the computational model related to the leftmost-innermost rewriting of subexpressions and let us take a computation step to be a rewriting step. The initial expression e_0 (that is, the expression at computation step 0) is (a number encoding of) f(i), for any given p.r.f. f and input number i. Then, given (a number encoding of) an expression e_k at the generic computation step k, the function I produces (a number encoding of) the expression e_{k+1} at the next computation step k+1, by finding in the program a recursive equation that can be applied to the leftmost-innermost subexpression of e_k, and it applies that rewriting.

The only delicate point to examine is the case when the program for the function f requires the application of a minimalization operation. Then the computation proceeds by 'entering a subprogram'. This means that, if we need to compute, for instance, $h(i) = \mu y.[g(i,y)=0]$, the function I first generates from the program for g(i,y) the subprogram for computing the values of g(i,0), g(i,1), g(i,2), … in succession, and then it uses that subprogram until the value 0 is found.

We hope that the reader may convince himself that: i) finding the leftmost-innermost subexpression, ii) finding the recursive equation to be applied, iii) rewriting that subexpression, and iv) generating and using the subprograms for the minimalizations, are operations which can be performed starting from the number encoding $\tau^*(f)$ and the number encoding of the expression e at hand, by the application of a suitable total partial recursive function which operates on number encodings. ■

THEOREM 16. A set A is r.e. iff A is the domain of a p.r.f.

PROOF. ($\Rightarrow$) Let us call p the p.r.f. to be found, whose domain is A. If $A = \varnothing$ the function p from N to N, is the everywhere undefined function: $\lambda n.(\mu y.[1 = 0])$.

If $A \neq \varnothing$ by definition A is the range of a *total* p.r.f., say f. For computing p(x), for any $x \in A$, we compute f(0), f(1), f(2), …, in succession. Each computation of f(i), for $i \geq 0$, terminates because f is total. Since $\forall x \in A\ \exists n \in N$ such that f(n) = x (because f is total and A is the range of f), the evaluation of sequence f(0), f(1), …, f(n), one element at a time in succession, gives us after a finite number of computation steps, the value of f(n) = x for some n∈ N. At that point we return n as the value of p(x).

The function p is a p.r.f. because $p(x) = \mu n.[f(n) - x = 0]$.

(⇐) We have that A is the domain of a p.r.f., call it g. If A = ∅ then A is r.e. by definition. If A ≠ ∅ we will now construct a total p.r.f., call it h, such that the range of h is A, and by doing so we will show that A is r.e.

Let us first consider a list of pairs of numbers, call it G, which is assumed to be initially empty, constructed as follows.

We first perform a dove-tailing evaluation of the function g. Let us assume that we get the table depicted in Fig. 2.

Computation steps j :	0	1	2	3	4	5 …
i : 0	#	#	#	#	#	# …
1	#	#	#	#	# …	
…	…					
x	#	#	m	m …		
…	…					

Figure 2. Dove-tailing evaluation of g.

When the value of g(x), say m, different from #, is obtained in row x and column y, we insert at the end of the list G the pair <x, D(x,y)>. (Recall that: $D(i,j) = ((i+j)^2 + 3i + j) / 2$.) By definition of dove-tailing, if <x, D(x,y)> is in G then in G there are also infinitely many pairs whose first component is x (see, for instance, the infinitely many m's in row x in Fig. 2).

Since A is the domain of g, g is convergent for each x ∈ A, and therefore for any x ∈ A there exists y ∈ N (actually, there exist infinitely many y's) such that the pair <x, D(x,y)> is inserted into G.

Since A is not empty, the list G has at least one pair. Let us consider the pair of G with the smallest second component, that is, the one which is first inserted into G during the dove-tailing evaluation of g. Let that pair be $<x_0,d_0>$. Thus, the list G is of the form:

$$G = [<x_0,d_0>, <x_1,d_1>, \ldots, <x_n,d_n>, \ldots],$$

where $d_0 \leq d_1 \leq \ldots \leq d_n \leq \ldots$ is an increasing sequence of numbers.

Notice that the list G is infinite even if A is a finite set (by hypothesis, A is non-empty). We also have that for all x ∈ A and for all d≥0 there exists d_1>d such that if <x,d> is in G then $<x,d_1>$ is also in G.

Let us define the function h from N to N, as follows:

(1) $h(0) = x_0$
(2) h(n+1) = *if* <x,n+1> is in G *then* x *else* x_0

We have the following three points.

(i) The function h is p.r.f. because the function I which performs the bounded evaluation of g(x) for c(n) computation steps for any x≥0 and n≥0, is a total partial recursive function from N^3 to N∪{#}, and so is the function which performs the bounded evaluations of g(r(i)) for c(i) computation steps for i = 0, …, n. Thus, the predicate ‘<x,n+1> is in G’ is recursive.

(ii) By definition of h, we have that h is a total p.r.f.

(iii) Moreover, the range of h is A. Indeed, since A is the domain of g, for any $x \in A$ there exists a position number d such that: i) x = r(d) and ii) the bounded evaluation of g(x) gives us a number in N within c(d) computation steps. In particular, there exists a minimum value of d, call it d_{min}, such that i) and ii) hold. Thus, the pair <x,d_{min}> is in G, and by definition of h (see line (2) above), we have that x is the value of h(d_{min}). ■

REMARK. In order to better understand the (⇐) proof of Theorem 16 and the definition of the function h, let us now consider the following example. Let us assume that the dove-tailing evaluation of the function g, indicated in Fig. 3 below, produces for us the following list G:

G = [<2,8>, <1,11>, <2,12>, <1,16>, <2,17>, <4,19>, …].

<2,8> means that at position number 8, that is, at row r(8)=2 and column c(8)=1, we have computed the value of g(2), which is m_8, with at most 1 computation step,

<1,11> means that at position number 11, that is, at row r(11)=1 and column c(11)=3, we have computed the value of g(1), which is m_{11}, with at most 3 computation steps,

<2,12> means that at position number 12, that is, at row r(12)=2 and column c(12)=2, we have computed (again) the value of g(2), which is $m_{12}=m_8$, with at most 2 computation steps,

… ,

<4,19> means that at position number 19, that is, at row r(19)=4 and column c(19)=1, we have computed the value of g(4), which is m_{19}, with at most 1 computation step, etc.

In this matrix of Fig. 3 we have written, instead of the #’s, the corresponding position numbers between parentheses. The entries of this matrix which are in the same row and are *not* position numbers, are all equal. Indeed, they are the value of the function g for the same input. In particular, in our matrix above:

i) for row 1, we have: $g(1) = m_{11} = m_{16} = \ldots$,
ii) for row 2, we have: $g(2) = m_8 = m_{12} = m_{17} = \ldots$, and
iii) for row 4, we have: $g(4) = m_{19} = \ldots$, etc.

Computation steps j :	0	1	2	3	4	5	…
i : 0	(0)	(1)	(3)	(6)	(10)	(15)	…
1	(2)	(4)	(7)	m_{11}	m_{16}	…	
2	(5)	m_8	m_{12}	m_{17}	…		
3	(9)	(13)	(18)	…			
4	(14)	m_{19}	…				
…	…						

Figure 3. A dove-tailing evaluation of the function g.
The r.e. set A which is the domain of g is {2,1,4,…}. It is also the range of the function h.

Since during the dove-tailing evaluation the first computed value of the function g is g(2) = m_8, we have that for the definition of h, the value of x_0 is 2. Thus,

n:	0	1	2	…	8	9	10	11	12	…	15	16	17	18	19	…
h(n):	2	2	2	…	2	2	2	1	2	…	2	1	2	2	4	…

To sum up, in order to compute h(n) we first perform the dove-tailing evaluation of the function g until the following two conditions are satisfied:

i) the position at row r(n) and column c(n) has been filled with the value of the bounded evaluation of g(r(n)) with at most c(n) computation steps (it may be either a number or #), and

ii) the value of g has been computed for at least one element in A, call it x_0, (recall that A is not empty, by hypothesis) with x_0 different from #.

Then h(n) = r(n) if the bounded evaluation of g(r(n)) returns a number within c(n) computation steps, otherwise h(n) = x_0. ■

THEOREM 17. A set A is r.e. iff A is the range of a p.r.f.
PROOF. (⇒) Let us call p the p.r.f. to be found whose range is A. If A = ∅ the function p is the everywhere undefined function μn.[1=0].

If A ≠ ∅ the p.r.f which has A as range, is the total p.r.f. which exists because A is, by definition, the range of a total p.r.f.
(⇐) We have that A is the range of a p.r.f., call it g. If A = ∅ then A is r.e. by definition. If A ≠ ∅ we will now construct a total p.r.f., call it h, whose range is A, and by doing so we will show that A is r.e.

The proof of this part of the Theorem is like the (⇐) proof of Theorem 16, with the exception that:

i) during the construction of the list G, when the value of g(x), say m, different from #, is obtained in row x and column y, we insert at the end of the list G the pair $\langle m, D(x,y)\rangle$, instead of the pair $\langle x, D(x,y)\rangle$, and

ii) the function h is defined as follows:

$$h(0) = m_0$$
$$h(n+1) = \textit{if } \langle m,n+1\rangle \text{ is in G } \textit{then } m \textit{ else } m_0$$

where m_0 is the first component of the first element of G.

By definition, the function h is total.

We also have that the range of h is A. Indeed, for every x ∈ A, x is by hypothesis in the range of g, and thus, there exists k such that the bounded evaluation of g(r(k)) gives us the number x within c(k) computation steps. As a consequence, since $\langle x,k\rangle$ is in G we have that for all x ∈ A there exists k ∈ N such that h(k) = x. ■

REMARK. In order to better understand the (⇐) proof of Theorem 17, let us now consider the following example.

Let us assume that the dove-tailing evaluation of the function g, indicated in Fig. 3 above, produces for us the following list G:

$$G = [\langle m_8,8\rangle,\ \langle m_{11},11\rangle,\ \langle m_8,12\rangle,\ \langle m_{11},16\rangle,\ \langle m_8,17\rangle,\ \langle m_{19},19\rangle,\ \ldots].$$

This value of G means that during the dove-tailing evaluation
- at position 8 (and 12, 17, …) we have computed $g(2) = m_8 = m_{12} = m_{17} = \ldots$,
- at position 11 (and 16, …) we have computed $g(1) = m_{11} = m_{16} = \ldots$,
- at position 19 we have computed $g(4) = m_{19}$, etc.

In our case we have that $h(0) = m_8$, and thus, the values of the function h are as follows:

n:	0	1	2	…	8	9	10	11	12	…	15	16	17	18	19	…
h(n):	m_8	m_8	m_8	…	m_8	m_8	m_8	m_{11}	m_8	…	m_8	m_{11}	m_8	m_8	m_{19}	…

■

DEFINITION 18. A set $A \subseteq N$ is *recursive* iff there exists a total p.r.f., say f_A, such that:
$\forall\, x \in N$. if $x \in A$ then $f_A(x) = 1$ and if $x \notin A$ then $f_A(x) = 0$. ■

It follows from the definition that for any given recursive set A the corresponding function f_A (which is a total p.r.f.) can be used for deciding whether or not a number is an element of A, that is, for deciding the membership problem relative to A.

By definition, recursive sets are the largest collection of sets for which it is possible to decide the membership problem via a total p.r.f.

In analogy to Definition 18, we also have the following Theorem.

THEOREM 19. A set A is r.e. iff there exists a partial recursive function, say f_A, such that:

$\forall x \in N$. *if* $x \in A$ *then* $f_A(x) = 1$ *else* undefined. (†)

PROOF. Using the partial recursive function p which we have constructed in the proof of Theorem 16 (part ⇒), we can define the function f_A as follows:

$f_A(x) = nsg(0 \neg p(x))$.

Now we have that: (i) $f_A(x)$ is a p.r.f. because p(x) is a p.r.f. and the class of the partial recursive functions are closed w.r.t. composition, and (ii) $f_A(x)$ satisfies (†) because the domain of p(x) is A and, as we have mentioned in Section 2, expressions are evaluated in the 'call-by-value' mode. ■

There are sets which are r.e. and are not recursive. One such set can be constructed as follows.

Let us consider the following injective encoding, say τ^*, from the set of Pascal programs into the set N of natural numbers. Since every Pascal program is a finite sequence of characters, for each program there exists a unique corresponding finite sequence of numbers. That sequence can be encoded into a single number (via the above mentioned τ encoding function, for instance).

Let us consider the set

$$D = \{\tau^*(p) \mid p \text{ is a Pascal program and p halts when given the input } \tau^*(p)\}.$$

Thus, a number n is in D iff n is the encoding of a Pascal program which halts when its input is n. One can show that D is a r.e. subset of N, but it is not a recursive subset.

5. RECURSIVE ENUMERABLE SETS AND TURING COMPUTABLE FUNCTIONS.

An alternative definition of recursively enumerable sets can be given in terms of Turing Machines, instead of p.r.f.'s. Indeed, it can be shown that a function is a p.r.f. from N to N iff it can be computed by a Turing Machine, having chosen a suitable encoding of numbers by strings of tape symbols.

One can show that r.e. sets are exactly those subsets of N which can be enumerated by Turing Machines in the following sense: for any r.e. set $A \subseteq N$ there exists a Turing Machine T_A such that if A is the empty subset of N then for any input $n \in N$, T_A does not halt, and if A is not empty then

i) for any $n \in N$, when given n as input, the Turing Machine T_A halts and gives us back an element of A, and
ii) for any a if $a \in A$ there exists $n \in N$ such that when given n as input, the Turing Machine T_A halts and gives us back the element a.

Notice that there exists an infinite number of Turing Machines which do not halt for all input $n \in N$.

As already mentioned, there exists a bijection between Turing Machines and p.r.f.'s, and there exists a bijection between Turing Machines which halt for every input value, and total p.r.f.'s.

Analogous statements hold if instead of referring to Turing Machines, we refer to other universal models of computation, like, for instance, Post Systems [Cutland 80, page 61], Post Machines [Manna 74], and Markov Systems [Cutland 80, page 65, and Lewis-Papadimitriou 81, page 263].

The notion of total p.r.f. is also equivalent to the notion of 'always terminating algorithm'. As a consequence, we have that, for instance, there is no p.r.f. which given any p.r.f. f, decides whether or not f is total.

6. COMPUTABILITY IN OTHER DOMAINS

Instead of considering p.r.f.'s from N to N we may consider p.r.f.'s from a different domain M1 to a different domain M2.

In particular, we may choose M1 = M2 = the integers, or $M1 = M2 = \Sigma^* = \{a,b\}^*$. In this last case the set of p.r.f.'s from Σ^* to Σ^* can be defined as the smallest set of functions such that

it includes:

i) the function $f(s) = \varepsilon$ (that is, the empty string),
ii) the successor functions $f_a(s) = s\,a$ and $f_b(s) = s\,b$, and
iii) the projection functions $f(s_1,\ldots, s_n) = s_i$ for all $i = 1,\ldots, n$,

it is closed under

composition (defined as in Definition 1.iv),
primitive recursion defined as follows:

$f(\varepsilon, s_2,\ldots,s_k) = g(s_2,\ldots,s_k)$
$f(s_1a,s_2,\ldots,s_k) = h(s_1,s_2,\ldots,s_k,\ f(s_1,s_2,\ldots,s_k))$
$f(s_1b,s_2,\ldots,s_k) = k(s_1,s_2,\ldots,s_k,\ f(s_1,s_2,\ldots,s_k))$

and minimalization in the sense that the function of arity k of the form:
$\lambda x.(\mu s.[t(x,s) = \varepsilon])$, where $x \in (\Sigma^*)^k$ is in the set if $t(x,s)$ is a total p.r.f. from $(\Sigma^*)^{k+1}$ to Σ^*. To apply the minimalization operator μ we need an ordering among strings and we choose the "canonical" one, that is, ε, a, b, aa, ab, ba, bb, aaa, …

It is not difficult to show that under suitable encodings of strings into numbers, the theory of the p.r.f.'s where each function is from $(\Sigma^*)^k$ to Σ^*, for some $k \geq 0$, is isomorphic to the theory of the p.r.f.'s where each function is from N to N.

REFERENCES

[Cutland 80] Cutland, N. J.: "Computability: An Introduction to Recursive Function Theory", Cambridge University Press, 1980.

[Davis 58] Davis, M.: "Computability and Unsolvability" McGraw-Hill, New York, 1958.

[Enderton 77] Enderton, H. B.: "Elements of Set Theory", Academic Press, New York, 1977.

[Hermes 69] Hermes, H.: "Enumerability, Decidability, Computability", Springer Verlag, 1969.

[Kleene 71] Kleene, S. C.: "Introduction to Metamatematics" North Holland, 1971.

[Lewis-Papadimitriou 81] Lewis, H., and Papadimitriou, C.: "Elements of the Theory of Computation", Prentice Hall, 1981.

[Manna 74] Manna, Z.: "Mathematical Theory of Computation" McGraw Hill, 1974.

[Mendelson 87] Mendelson, E.: "Introduction to Mathematical Logic" 3rd Edition, Wadsworth, Monterey, California, 1987.

[Pettorossi 91] Pettorossi, A.: "Quaderni di Informatica. Parte I", Unitor, 1991.

[Roger 67] Roger, H. Jr.: "Theory of Recursive Functions and Effective Computability", McGraw Hill, 1967.

[Sanchis 88] Sanchis, L.: "Reflexive Structures: An Introduction to Computability Theory" Springer Verlag, 1988.