

Computabilidad y Complejidad

Tema 3: Funciones μ -Recursivas.

Índice:

1. Introducción.
2. Funciones Recursivas Primitivas .
3. Funciones μ -Recursivas.

Bibliografía Básica Recomendada

- ♦ Elements of the theory of computation (Lewis, Harry R., Papadimitriou, Christos H.)
- ♦ Teoría de la computación (Brookshear, J. Glenn)

Introducción

Aparte del modelo de la máquina de Turing existen otros formalismos para definir las funciones computables numéricas. En particular en este tema veremos el referente a las funciones μ -recursivas.

Antes de abordarlo en su totalidad introduciremos, como paso previo, las funciones recursivas primitivas.

Estas funciones se definen sobre tupas de números naturales, con una signatura genérica

$$g: \mathbb{N}^k \longrightarrow \mathbb{N}$$

Recuérdese que si $f(x)$ es una función, entonces:

- ▶ $f^0(x) = x, \forall x.$
- ▶ $f^{i+1}(x) = f(f^i(x)), \forall x, \forall i \geq 0.$

Funciones recursivas primitivas

Las funciones recursivas primitivas son funciones `totales` que se definen a partir de unas funciones básicas inmediatamente computables, y de unos operadores que permiten definir nuevas funciones a partir de otras ya definidas mediante el uso de un determinado tipo de ecuaciones funcionales; introduciéndose como mayor novedad el operador de recursión primitiva.

Funciones recursivas primitivas básicas

Las funciones recursivas primitivas básicas son las siguientes:

- La función sin argumentos `cero`, $c: c() = 0$
- La función `sucesor`, s : para n ya definido, $s(n)$ es el sucesor de n
($= n+1$)

Así los números naturales quedan definidos inductivamente a partir de la función `cero` y la aplicación sucesiva de la función `sucesor`:

$$n \equiv s^n(c()) = s^n(0).$$

$$p_i = p_{i,k} = \text{proj}^k_i$$

➤ Las funciones de proyección o de selección $p_{i,k}$:

- $\forall k > 0, \forall n_1, \dots, n_k \in \mathbb{N}, \forall i \in \{1, \dots, k\} : p_{i,k}(n_1, \dots, n_k) = n_i$
- $\forall k > 0, \forall n_1, \dots, n_k \in \mathbb{N} : p_{0,k}(n_1, \dots, n_k) = c() = 0$

Siempre que no exista posibilidad de ambigüedad o de confusión en lugar de

$$p_{i,k}(n_1, \dots, n_k)$$

escribiremos (sobrecargando las funciones) simplemente

$$p_i(n_1, \dots, n_k)$$

En lo que sigue utilizaremos un pseudocódigo (autoexplicado) de programación cercano a los lenguajes de programación actuales que utilizan clausuras.

En nuestro pseudocódigo existen dos tipos de entidades: el tipo **natural(es)** y el tipo **función(es)**.

Programáticamente, las funciones primitivas básicas pueden expresarse mediante los códigos siguientes:

```
natural:función c() {return 0;}
```

```
natural:función s(natural n) {return n+1;}
```

$\forall k > 0$

◆ Para $i = 1, \dots, k$

```
natural:función p_i_k(naturales n1, ..., nk) {  
    return ni;
```

◆ Para $i = 0$

```
natural:función p_0_k(naturales n1, ..., nk) {  
    return 0;}
```

Operadores de composición

Los operadores de composición son los siguientes:

▷ **Operador de sustitución:** Sea $f(n_1, \dots, n_k)$ una función recursiva primitiva y sean también las siguientes funciones recursivas primitivas: $g_1(n_1, \dots, n_m), \dots, g_k(n_1, \dots, n_m)$; entonces la función

$$h(n_1, \dots, n_m) = f(g_1(n_1, \dots, n_m), \dots, g_k(n_1, \dots, n_m))$$

también es recursiva primitiva.

Programáticamente, este operador puede especificarse mediante el código siguiente:

```
función:función sustitución(funciones f, g1, ..., gk) {  
  return natural:función(naturales n1, ..., nm) {  
    naturales r, r1, ..., rk;  
    r1 = g1(n1, ..., nm);  
    . . . . .  
    rk = gk(n1, ..., nm);  
    r = f(r1, ..., rk);  
    return r;  
  }  
}
```

h \equiv sustitución(f, g₁, ..., g_k)

h(n₁, ..., n_k) \equiv sustitucion(f, g₁, ..., g_k)(n₁, ..., n_k)

f es el caso base, g es la función recursiva y h es la función resultante

▷ Operador de recursión primitiva: Sean las siguientes funciones recursivas primitivas:

- ▶ $f(n_1, \dots, n_k), y$
- ▶ $g(i, j, n_1, \dots, n_k)$

Correction:
 $g: \mathbb{N}^3 \longrightarrow \mathbb{N}$

$f: \mathbb{N} \rightarrow \mathbb{N}$

entonces la función:

- $h(0, n_1, \dots, n_k) = f(n_1, \dots, n_k)$
- $h(s(n), n_1, \dots, n_k) = g(h(n, n_1, \dots, n_k), n, n_1, \dots, n_k)$

es también recursiva primitiva.

El esquema de cálculo introducido por el operador de recursión primitiva consiste en un despliegue en el que se decrementa, en cada etapa del desarrollo, en una unidad el valor del contador, n , asociado hasta llegar al valor 0.

Recuérdese que $n = s^n(0)$.

Así, para un $n > 0$ dado:

$$h(s^n(0), n_1, \dots, n_k) = g(h(s^{n-1}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k)$$

Primitive Recursion

$$f: \mathbb{N}^n \rightarrow \mathbb{N} \quad g: \mathbb{N}^{n+2} \rightarrow \mathbb{N}$$

$$h = \rho^n(f, g): \mathbb{N}^{n+1} \rightarrow \mathbb{N}$$

BASE CASE

$$h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n)$$

RECURSIVE CASE

$$h(x_1, \dots, x_n, y+1) = g(x_1, \dots, x_n, y, h(x_1, \dots, x_n, y))$$

El esquema de cálculo introducido por el operador de recursión primitiva consiste en un despliegue en el que se decrementa, en cada etapa del desarrollo, en una unidad el valor del contador, n , asociado hasta llegar al valor 0.

Recuérdese que $n = s^n(0)$.

Así, para un $n > 0$ dado:

$$h(s^n(0), n_1, \dots, n_k) = g(h(s^{n-1}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k)$$

$$= g(g(h(s^{n-2}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k)$$

El esquema de cálculo introducido por el operador de recursión primitiva consiste en un despliegue en el que se decrementa, en cada etapa del desarrollo, en una unidad el valor del contador, n , asociado hasta llegar al valor 0.

Recuérdese que $n = s^n(0)$.

Así, para un $n > 0$ dado:

$$h(s^n(0), n_1, \dots, n_k) = g(h(s^{n-1}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k) =$$

$$g(g(h(s^{n-2}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), n, n_1, \dots, n_k) =$$

$$g(g(g(h(s^{n-3}(0), n_1, \dots, n_k), s^{n-3}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k)$$

El esquema de cálculo introducido por el operador de recursión primitiva consiste en un despliegue en el que se decrementa, en cada etapa del desarrollo, en una unidad el valor del contador, n , asociado hasta llegar al valor 0.

Recuérdese que $n = s^n(0)$.

Así, para un $n > 0$ dado:

$$\begin{aligned}
 h(s^n(0), n_1, \dots, n_k) &= g(h(s^{n-1}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k) = \\
 g(g(h(s^{n-2}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), n, n_1, \dots, n_k) &= \\
 g(g(g(h(s^{n-3}(0), n_1, \dots, n_k), s^{n-3}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), & \\
 s^{n-1}(0), n_1, \dots, n_k) = \dots &= \\
 g(g(g(\dots h(0, n_1, \dots, n_k) \dots, s^{n-3}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, & \\
 n_k), s^{n-1}(0), n_1, \dots, n_k) &
 \end{aligned}$$

El esquema de cálculo introducido por el operador de recursión primitiva consiste en un despliegue en el que se decrementa, en cada etapa del desarrollo, en una unidad el valor del contador (n) asociado hasta llegar al valor 0.

Recuérdese que $n = s^n(0)$.

Así, para un $n > 0$ dado:

$$h(s^n(0), n_1, \dots, n_k) = g(g(g($$

...

... $f(n_1, \dots, n_k)$...

...

$$s^{n-3}(0), n_1, \dots, n_k), s^{n-2}(0), n_1, \dots, n_k), s^{n-1}(0), n_1, \dots, n_k)$$

Programáticamente, este operador puede especificarse mediante el siguiente código de índole iterativa:

```
función:función recursiónPrimitiva(funciones g,f) {  
    return natural:función(naturales n,n1,...,nk) {  
        natural r = f(n1,...,nk);  
        for(i = 1; i ≤ n; i++) r = g(r,i-1,n1,...,nk);  
        // Nótese que el valor de n no se modifica durante la ejecución del bucle.  
        return r;  
    }  
}
```

$h \equiv \text{recursiónPrimitiva}(g,f)$

$h(s^n(0), n_1, \dots, n_k) \equiv \text{recursiónPrimitiva}(g,f)(n, n_1, \dots, n_k)$

Donde se ha supuesto que las funciones f y g son computadas por otros módulos similares (a los presentados) en los que las únicas estructuras de repetición utilizadas son bucles `for` con una cabecera que presenta exactamente el formato

```
for(i = 1; i ≤ n; i++).
```

Nótese que el valor de n no se modifica manteniéndose invariable.

Funciones recursivas primitivas

f es el caso base, g es la función recursiva
y h es la función resultante

Primitive Recursion

$$f: \mathbb{N}^n \rightarrow \mathbb{N} \quad g: \mathbb{N}^{n+2} \rightarrow \mathbb{N}$$

$$h = \rho^n(f, g): \mathbb{N}^{n+1} \rightarrow \mathbb{N}$$

BASE CASE

$$h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n)$$

RECURSIVE CASE

$$h(x_1, \dots, x_n, y+1) = g(x_1, \dots, x_n, y, h(x_1, \dots, x_n, y))$$

Las funciones recursivas primitivas son aquellas que pueden definirse a partir de las funciones recursivas primitivas básicas, los operadores de sustitución y de recursión primitiva en un número finito de pasos.

Proposición: Cada función recursiva primitiva es una función total Turing computable.

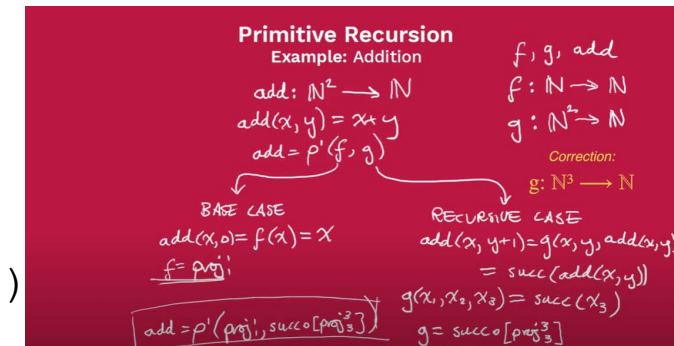
Algunas funciones recursivas primitivas especialmente útiles

Ejemplos: Algunas de las funciones recursivas primitivas más utilizadas son las siguientes:

1) función suma:

$$\text{suma}(0, m) = p_1(m) = m$$

$$\begin{aligned} \text{suma}(s(n), m) &= s(p_1(\text{suma}(n, m), n, m)) \\ &= s(\text{suma}(n, m)) \end{aligned}$$



En lo que sigue, para aligerar la exposición, utilizaremos el símbolo operacional $+$ para operacionalmente denotar esta función, así como para expresar sus propiedades algebraicas usuales.

$$\text{suma}(0, m) = p_1(m) = m$$

$$\text{suma}(1, m) = s(\text{suma}(0, m)) = s(m) = m+1$$

$$\text{suma}(2, m) = s(\text{suma}(1, m)) = s(s(m)) = s(m+1) = m+2$$

.....

2) función predecesor:

$$\text{pred}(0) = c() = 0$$

$$\text{pred}(s(n)) = p_2(\text{pred}(n), n) = n$$

$$\text{pred}(0) = c() = 0$$

$$\text{pred}(1) = p_2(\text{pred}(0), 0) = 0$$

$$\text{pred}(2) = p_2(\text{pred}(1), 1) = 1$$

$$\text{pred}(3) = p_2(\text{pred}(2), 2) = 2$$

.....

Primitive Recursion
Example: Predecessor

f, g, pred
 $f: \mathbb{N}^0 \rightarrow \mathbb{N}$
 $g: \mathbb{N}^2 \rightarrow \mathbb{N}$

$\text{pred}(x) = x - 1$
 $\text{pred}(0) = 0$
 $\text{pred}: \mathbb{N} \rightarrow \mathbb{N}$

$\text{pred} = \rho^\circ(f, g)$

BASE CASE
 $\text{pred}(0) = f() = 0$
 $f = \text{zero}^\circ$
 $\text{pred} = \rho^\circ(\text{zero}^\circ, \text{proj}_1)$

RECURSIVE CASE
 $\text{pred}(y+1) = g(y, \text{pred}(y))$
 $= y$
 $g(x_1, x_2) = x_1$
 $g = \text{proj}_1$

3) función diferencia propia: dif

$$\text{dif}(0, m) = p_1(m) = m$$

$$\begin{aligned}\text{dif}(s(n), m) &= \text{pred}(p_1(\text{dif}(n, m), n, m)) \\ &= \text{pred}(\text{dif}(n, m))\end{aligned}$$

$$\text{dif}(0, m) = p_1(m) = m = \text{pred}^0(m)$$

$$\text{dif}(1, m) = \text{pred}(\text{dif}(0, m)) = \text{pred}(m) = \text{pred}^1(m)$$

$$\text{dif}(2, m) = \text{pred}(\text{dif}(1, m)) = \text{pred}(\text{pred}(m)) = \text{pred}^2(m)$$

$$\text{dif}(3, m) = \text{pred}(\text{dif}(2, m)) = \text{pred}(\text{pred}^2(m)) = \text{pred}^3(m)$$

.

Operacionalmente denotaremos esta función mediante $\underline{\bullet}$, de modo que

$$\text{dif}(n, m) = m \underline{\bullet} n =$$

➡ $m - n$, si $m \geq n$,

➡ 0 , en otro caso

4) función producto: prod

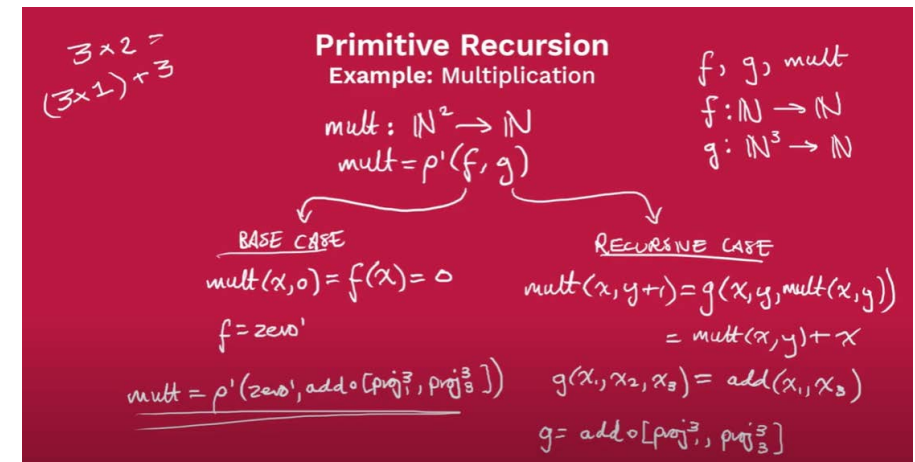
$$\begin{aligned} \text{prod}(0, m) &= p_0(m) = 0 \\ \text{prod}(s(n), m) &= \end{aligned}$$

$$\begin{aligned} &\text{suma}(p_1(\text{prod}(n, m), n, m), p_3(\text{prod}(n, m), n, m)) = \\ &\text{suma}(\text{prod}(n, m), m) = \text{prod}(n, m) + m \end{aligned}$$

En lo que sigue, para aligerar la exposición, utilizaremos el símbolo operacional

- para operacionalmente denotar esta función, así como para expresar sus propiedades algebraicas usuales.

$$\begin{aligned} \text{prod}(0, m) &= p_0(m) = 0 \\ \text{prod}(1, m) &= \text{suma}(\text{prod}(0, m), m) = \text{suma}(0, m) = 0 + m = m \\ \text{prod}(2, m) &= \text{suma}(\text{prod}(1, m), m) = \text{suma}(m, m) = m + m = 2 \bullet m \\ \text{prod}(3, m) &= \text{suma}(\text{prod}(2, m), m) = \text{suma}(2m, m) = 2 \bullet m + m = 3 \bullet m \\ &\dots \end{aligned}$$



5) función signo: sg

$sg(n) =$
 ➡ 0, si $n = 0$,
 ➡ 1, en otro caso

$$sg(0) = c() = 0$$

$$sg(s(n)) = s(p_0(sg(n), n)) = 1$$

Equivalentemente

$$\begin{aligned} \text{sg}(n) &= s(c()) \bullet (s(c()) \bullet n) \\ &= 1 \bullet (1 \bullet n) = \text{dif}(\text{dif}(n, s(0)), s(0)) \end{aligned}$$

Esta última expresión puede definirse con toda formalidad como sigue:

$$\begin{aligned} \text{Uno}(n) &= s(p_0(n)), && \text{aplicando composición} \\ D(n) &= \text{dif}(p_1(n), \text{Uno}(n)), && \text{aplicando composición} \\ \text{sg}(n) &= \text{dif}(D(n), \text{Uno}(n)), && \text{aplicando composición} \end{aligned}$$

6) función cosigno: `cosg`

`cosg(n) =`
 ➡ `1, si n = 0,`
 ➡ `0, en otro caso`

`cosg(0) = s(c()) = 1`

`cosg(s(n)) = p0(cosg(n), n) = 0`

Equivalentemente

$$\begin{aligned}\cos g(n) &= s(c()) \bullet n = 1 \bullet n \\ &= 1 \bullet sg(n)\end{aligned}$$

También

$$sg(n) = 1 \bullet \cos g(n) = \cos g(\cos g(n))$$

7) función menor n que m:

```
menor(n,m) =  
    ➡ 1, si n < m,  
    ➡ 0, en otro caso
```

```
menor(n,m) = sg(dif(n,m))
```

8) función mayor n que m:

```
mayor(n,m) =  
    ➡ 1, si n > m,  
    ➡ 0, en otro caso
```

```
mayor(n,m) = sg(dif(m,n))
```

9) Función de igualdad: `igual`

```
igual(n,m) =  
    ➡ 1, si n = m,  
    ➡ 0, en otro caso
```

```
igual(n,m) = cosg(suma(menor(n,m), mayor(n,m)))
```

10) función desigualdad: `desigual`

```
desigual(n,m) =  
    ➡ 1, si  $n \neq m$ ,  
    ➡ 0, en otro caso
```

```
desigual(n,m) = cosg(igual(n,m))
```

11) función identidad:

```
identidad(n) = n = p1(n)
```


Funciones recursivas primitivas definidas por casos

Seguidamente veremos el modo de definir funciones recursivas primitivas por casos; lo haremos para funciones de una variable pero el proceso fácilmente se generaliza a funciones de varias variables.

Si $g(n)$ es una función recursiva primitiva, entonces la función $f(n)$ definida como sigue también es recursiva primitiva.

$$\forall n \in \mathbb{N}: f(n) =$$

- m_1 , **si $n = n_1$**
- m_2 , **si $n = n_2$**
- \dots
- m_k , **si $n = n_k$**
- $g(n)$, **en otro caso**

Basta ver que

$$\forall n \in \mathbb{N}: f(n) = \text{igual}(n, n_1) \bullet n_1 +$$

$$\text{igual}(n, n_2) \bullet n_2 +$$

$$\dots$$

$$\text{igual}(n, n_k) \bullet n_k +$$

$$\text{desigual}(n, n_1) \bullet \text{desigual}(n, n_2) \bullet \dots \bullet \text{desigual}(n, n_k) \bullet g(n)$$

Funciones recursivas primitivas definidas mediante sumatorios y productorios finitos

Si $g(n, n_1, \dots, n_k)$ es una función recursiva primitiva, entonces las siguientes funciones también lo son.

$$\Rightarrow f(n, n_1, \dots, n_k) = \sum_{0 \leq i \leq n} g(i, n_1, \dots, n_k)$$

$$\Rightarrow h(n, n_1, \dots, n_k) = \prod_{0 \leq i \leq n} g(i, n_1, \dots, n_k)$$

Basta ver (omitiendo –para simplificar la notación– las pertinentes funciones de proyección para respetar el formalismo en su totalidad) que:

$$\blacktriangleright f(0, n_1, \dots, n_k) = g(0, n_1, \dots, n_k)$$

$$\blacktriangleright f(s(n), n_1, \dots, n_k) = f(n, n_1, \dots, n_k) + g(s(n), n_1, \dots, n_k)$$

$$\blacktriangleright h(0, n_1, \dots, n_k) = g(0, n_1, \dots, n_k)$$

$$\blacktriangleright h(s(n), n_1, \dots, n_k) = h(n, n_1, \dots, n_k) \bullet g(s(n), n_1, \dots, n_k)$$

$$\text{fact}(0) = s(c()) = 1$$

$$\begin{aligned}\text{fact}(s(n)) &= \text{fact}(n) * s(n) \\ &= \text{prod}(\text{p1}(\text{fact}(n), n), s(\text{p2}(\text{fact}(n), n)))\end{aligned}$$

Ejercicios. Definir las siguientes funciones como recursivas primitivas:

- `factorial: fact(n)`
- `exponencial en base n: exp(m, n) (= n^m)`

$$\text{exp}(0, n) = s(p0(n)) = 1$$

$$\begin{aligned}\text{exp}(s(m), n) &= \text{exp}(m, n) * n \\ &= \text{prod}(\text{p1}(\text{exp}(m, n), m, n), \text{p3}(\text{exp}(m, n), m, n))\end{aligned}$$

Funciones μ -recursivas

Las funciones μ -recursivas, en general funciones parciales, se definen a partir de las funciones recursivas primitivas y el operador (mu) μ (de minimización).

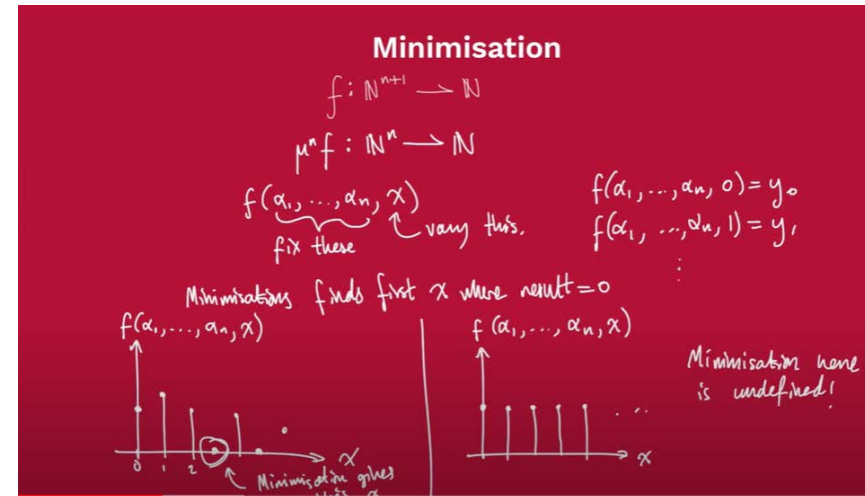
Este operador μ se define como sigue: sea la función

$$g: \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$$

para los valores n_1, \dots, n_k

$$\mu z g(z, n_1, \dots, n_k) = m \Leftrightarrow g(m, n_1, \dots, n_k) = 0 \wedge$$

$$(\forall i \in \{0, \dots, m-1\}) (\exists j \neq 0) (g(i, n_1, \dots, n_k) = j)$$



El término z se utiliza para señalar la posición de la variable asociada al cálculo especificado. Así, por ejemplo, se podrían también definir

$$\mu_z g(n_1, \dots, n_{\tilde{n}}, z, n_{\tilde{n}+1}, \dots, n_k) = m \Leftrightarrow g(n_1, \dots, n_{\tilde{n}}, m, n_{\tilde{n}+1}, \dots, n_k) = 0 \wedge \\ (\forall i \in \{0, \dots, m-1\}) (\exists j \neq 0) (g(n_1, \dots, n_{\tilde{n}}, i, n_{\tilde{n}+1}, \dots, n_k) = j)$$

o

$$\mu_z g(n_1, \dots, n_k, z) = m \Leftrightarrow g(n_1, \dots, n_k, m) = 0 \wedge \\ (\forall i \in \{0, \dots, m-1\}) (\exists j \neq 0) (g(n_1, \dots, n_k, i) = j)$$

Si se asume, por convenio, una posición predefinida para z , dígase la primera, entonces podemos simplemente escribir

$$\mu g(z, n_1, \dots, n_k) = m \Leftrightarrow g(m, n_1, \dots, n_k) = 0 \wedge \\ (\forall i \in \{0, \dots, m-1\}) (\exists j \neq 0) (g(i, n_1, \dots, n_k) = j)$$

Programáticamente, este último operador puede especificarse mediante el siguiente código de índole iterativa:

```
función:función mu(función g) {
  return natural:función(naturales n1,...,nk) {
    natural m = 0;
    while(g(m,n1,...,nk) ≠ 0) m++;
    return m;
  }
}
```

$$\mu g \equiv \text{mu}(g)$$

$$\mu g(z, n_1, \dots, n_k) \equiv \text{mu}(g)(n_1, \dots, n_k)$$

Minimisation
Example: proj

$\text{proj}_z^2 : \mathbb{N}^2 \rightarrow \mathbb{N}$ $(\mu' \text{proj}_z^2) : \mathbb{N} \rightarrow \mathbb{N}$ example: $(\mu' \text{proj}_z^2)(7)$ $\text{proj}_z^2(7, 0) = 0$ $\underline{\mu' \text{proj}_z^2(7) = 0}$	$\text{proj}_i^2 : \mathbb{N}^2 \rightarrow \mathbb{N}$ $\mu' \text{proj}_i^2 : \mathbb{N} \rightarrow \mathbb{N}$ example: $\mu' \text{proj}_i^2(1)$ $\text{proj}_i^2(1, 0) = 1$ $\text{proj}_i^2(1, 1) = 1$ \vdots <u>undefined!</u>
--	---

Las funciones μ -recursivas son aquellas que pueden definirse a partir de las funciones recursivas primitivas básicas, los operadores de composición, recursión primitiva y μ en un número finito de pasos.

Teorema: Una función $g: \mathbb{N}^m \longrightarrow \mathbb{N}$ es Turing computable si y sólo si es μ -recursiva.

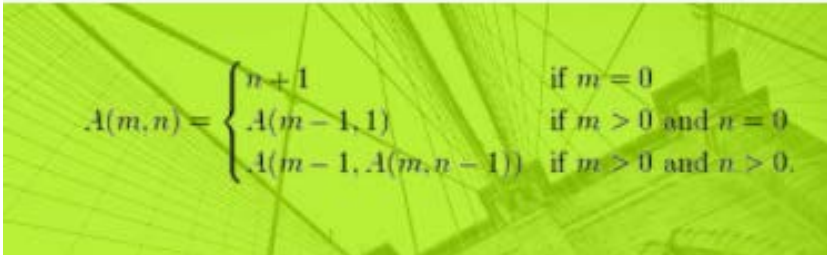
Teorema: Existen funciones totales $g: \mathbb{N}^m \longrightarrow \mathbb{N}$ Turing computables que no son recursivas primitivas.

Para la demostración del último teorema puede utilizarse la función de Ackermann. Esta función, denotada por A , se define como sigue:

$$A: \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

de modo que: $\forall n, m \in \mathbb{N}$:

- $A(0, m) = m + 1$
- $A(n + 1, 0) = A(n, 1)$
- $A(n + 1, m + 1) = A(n, (A(n + 1, m)))$


$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Proposición: La función de Ackermann es una función total Turing computable.

Proposición: Para cada función recursiva primitiva $g: \mathbb{N} \longrightarrow \mathbb{N}$ existe un número natural \tilde{n}_g de modo que

$$\forall m \in \mathbb{N}: g(m) < A(\tilde{n}_g, m)$$

Sea ahora la función $\underline{A}: \mathbb{N} \longrightarrow \mathbb{N}$ definida de modo que

$$\forall n \in \mathbb{N}: \underline{A}(n) = A(n, n)$$

Proposición: La función \underline{A} es una función total Turing computable que no es recursiva primitiva.