Computabilidad y Complejidad (CMC)

Funciones μ-Recursivas

Índice

Bibliografía básica recomendada

- Elements of the theory of computation (Lewis, Harry R., Papadimitriou, Christos H.)
- Teoría de la computación (Brookshear, J. Glenn)
- An Early History of Recursive Functions & Computability from Gödel to Turing (Adams, Ron)
- Computability Theory. An Introduction to Recursion Theory (Enderton, Herbert B.)

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:

- $(\forall x \in \mathbb{N})\ f^0(x) = x.$
- $(\forall x \in \mathbb{N})\ (\forall i > 0)\ f^{i+1}(x) = f(f^i(x)).$

Funciones Recursivas Primitivas

Las funciones recursivas primitivas son funciones totales que se definen a partir de unas funciones básicas que son computables de forma trivial, y de unos operadores de composición 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.

De esta forma, la definición de las funciones recursivas primitivas es constructiva, introduciendo, en primer lugar, las funciones básicas y, a partir de ellas, definiendo el resto de funciones mediante la aplicación un número finito de veces los operadores de composición.

En todos y cada una de las funciones recursivas primitivas, podemos establecer su computabilidad mediante máquinas de Turing. Por lo tanto, las funciones recursivas primitivas son funciones Turing-computables.

Funciones Recursivas Primitivas

Funciones básicas (I)

La primera función básica que definiremos es la función cero $c(\,) = 0$

Proposición. La función cero es Turing-computable

De forma trivial, una máquina de Turing totalmente bloqueada, es decir, sin ningún movimiento definido, calcularía la función cero. Obsérvese que dado que la función carece de argumentos, en su configuración inicial la cinta está totalmente en blanco y, al finalizar su computación también. De acuerdo con las codificaciones de funciones en máquinas de Turing, la cinta en blanco codificaría el valor 0 que es el valor de salida de la función.

Funciones Recursivas Primitivas

Funciones básicas (II)

La función sucesor $s: \mathbb{N} \longrightarrow \mathbb{N}$ definida de forma que $s(n) = n + 1$ es también una función básica recursiva primitiva

Proposición. La función sucesor es Turing-computable

Podemos definir una función Turing-computable que añada un cero a su entrada. Esto lo haríamos con la definición de los siguientes movimientos:

- $f(q_0, B) = (q_1, 0, R)$
- $f(q_0, 0) = (q_0, 0, R)$

De esta forma, los números naturales quedan definidos inductivamente a partir de la función cero y la aplicación sucesiva de la función sucesor:

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

Funciones Recursivas Primitivas

Funciones básicas (III)

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

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

Siempre que no exista posibilidad de ambigüedad o de confusión en lugar de $p_{i,k}(n_1, \ldots, n_k)$ escribiremos (sobrecargando las funciones) simplemente $p_i(n_1, \ldots, n_k)$.

Proposición. La funciones de proyección son Turing-computables

Para cada función de proyección $p_{i,k}(n_1, \ldots, n_k)$ es fácil diseñar una máquina de Turing que elimine todos los argumentos de entrada salvo n_i, sobreescribiendo en ellos un símbolo B (blanco).

Funciones Recursivas Primitivas

Operadores de composición (I)

El operador de sustitución se define como sigue: 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

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

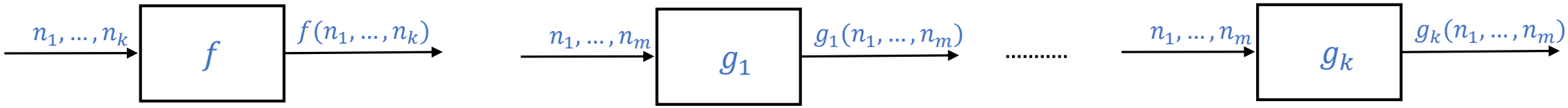
también es recursiva primitiva.

Proposición. Sean las funciones recursivas primitivas $f(n_1, \dots, n_k), g_1(n_1, \dots, n_m), \dots, g_k(n_1, \dots, n_m)$ Turing-computables. 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 Turing-computable.

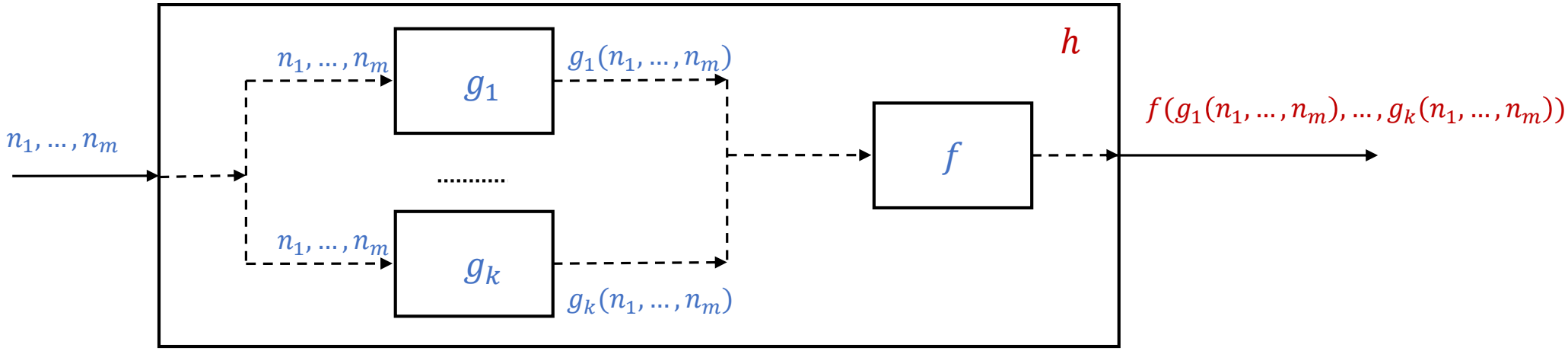
Funciones Recursivas Primitivas

Operadores de composición (II)

Para definir una máquina de Turing que aplique el operador de sustitución, podemos partir de máquinas de Turing que sigan los esquemas



A partir de los anteriores esquemas podemos construir una máquina de Turing que aplique la sustitución de acuerdo a su definición



Funciones Recursivas Primitivas

Operadores de composición (III)

El operador de recursión primitiva se define como sigue: Sean las siguientes funciones recursivas primitivas

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

entonces la función

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

también es recursiva primitiva.

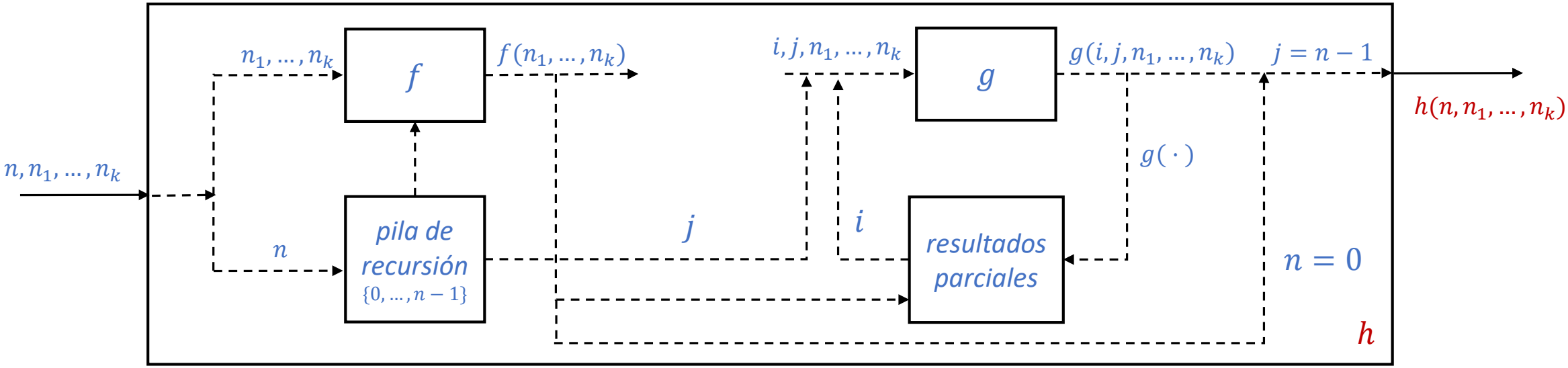
Proposición. Sean las funciones recursivas primitivas $f(n_1, \ldots, n_k)$, y $g(i, j, n_1, \ldots, n_k)$ Turing-computables. Entonces la función $h(n, n_1, \ldots, n_m)$ definida a partir de f y g mediante recursión primitiva también es Turing-computable.

Funciones Recursivas Primitivas

Operadores de composición (IV)

Para definir una máquina de Turing que aplique el operador de recursión primitiva, podemos partir de máquinas de Turing que sigan los esquemas

A partir de los anteriores esquemas podemos construir una máquina de Turing que aplique la recursión primitiva de acuerdo a su definición



Funciones Recursivas Primitivas

A modo de resumen

Funciones básicas + **Operadores**

función cero $c( ) = 0$

función sucesor $s(n) = n + 1$

funciones de proyección $p_i(n_1, \dots, n_k) = n_i$ $p_0(n_1, \dots, n_k) = 0$

Sustitución (composición funcional)

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

Recursión primitiva

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

- Todas las funciones básicas son funciones recursivas primitivas
- Una función es recursiva primitiva si se puede definir aplicando un número finito de veces los operadores de composición (sustitución y recursión primitiva) sobre funciones recursivas primitivas

Funciones Recursivas Primitivas

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

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

Dado que se ha demostrado que las funciones básicas son Turing-computables y la aplicación de los operadores sobre funciones básicas también proporcionan funciones Turing-computables, la proposición queda demostrada.

Funciones Recursivas Primitivas

Ejemplo

Función suma

- $suma(0,m) = p_1(m) = m$
- $suma(s(n),m) = s(p_1(suma(n,m),n,m)) = s(suma(n,m))$

 definicion por recursión primitiva

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.

$suma(0,m) = p_1(m) = m$
$suma(1,m) = s(suma(0,m)) = s(m) = m+1$
$suma(2,m) = s(suma(1,m)) = s(s(m)) = s(m+1) = m+2$
. . .

Funciones Recursivas Primitivas

Ejemplo

Función predecesor

- $pred(0) = c(\ ) = 0$
- $pred(s(n)) = p_2(pred(n),n) = n$

definición por recursión primitiva

$pred(0) = c(\ ) = 0$
$pred(1) = p_2(pred(0),0) = 0$
$pred(2) = p_2(pred(1),1) = 1$
$pred(3) = p_2(pred(2),2) = 2$
. . .

Funciones Recursivas Primitivas

Ejemplo

Función diferencia propia dif

Operacionalmente denotaremos esta función mediante $\dot{-}$, de modo que

$dif(n,m) = m \dot{-} n =$
- $m - n$, si $m \geq n$,
- 0, en otro caso

- $dif(0,m) = p_1(m) = m$
- $dif(s(n),m) = pred(p_1(dif(n,m),n,m)) = pred(dif(n,m))$

definición por recursión primitiva

$dif(0,m) = p_1(m) = m = pred^0(m)$
$dif(1,m) = pred(dif(0,m)) = pred(m) = pred^1(m)$
$dif(2,m) = pred(dif(1,m)) = pred(pred(m)) = pred^2(m)$
. . .

Funciones Recursivas Primitivas

Ejemplo

Función producto prod

- $prod(0,m) = p_0(m) = 0$
- $prod(s(n),m) =$
  $suma(p_1(prod(n,m),n,m),p_3(prod(n,m),n,m)) =$
  $suma(prod(n,m),m) = prod(n,m) + m$

definición por recursión primitiva

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

$prod(0,m) = p_0(m) = 0$
$prod(1,m) = suma(prod(0,m),m) = suma(0,m) = 0+m = m$
$prod(2,m) = suma(prod(1,m),m) = suma(m,m) = m+m = 2\cdot m$
$prod(3,m) = suma(prod(2,m),m) = suma(2m,m) = 2\cdot m+m = 3\cdot m$
. . .

Funciones Recursivas Primitivas

Ejemplo

Función signo sg

$sg(n) =$

- 0, si n = 0,
- 1, en otro caso

- $sg(0) = c(\ ) = 0$
- $sg(s(n)) = s(p_0(sg(n),n)) = 1$

definición por recursión primitiva

Equivalentemente

$$sg(n) = s(c(\ )) \dot{-} (s(c()) \dot{-} n) = 1 \dot{-} (1 \dot{-} n) = dif(dif(n,s(0)),s(0))$$

Esta última expresión puede definirse con toda formalidad como sigue:

- $Uno(n) = s(p_0(n))$
- $D(n) = dif(p_1(n),Uno(n))$
- $sg(n) = dif(D(n),Uno(n))$

definición por sustitución

Funciones Recursivas Primitivas

Ejemplo

Función cosigno cosg

cosg(n) =

- 1, si n = 0,
- 0, en otro caso

- cosg(0) = s(c( )) = 1
- cosg(s(n)) = p_0(cosg(n),n)) = 0

definición por recursión primitiva

Equivalentemente

$$cosg(n) = s(c(\,)) \dot{-} \; n = 1 \dot{-} n = 1 \dot{-} sg(n)$$

También:

$$sg(n) = 1 \dot{-} \; cosg(n) = cosg(cosg(n))$$

Funciones Recursivas Primitivas

Ejemplo

Función menor n que m:

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

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

definición por sustitución

Funciones Recursivas Primitivas

Ejemplo

Función mayor n que m:

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

menor(n,m) = sg(dif(m,n))  definición por sustitución

Obsérvese que el cambio en el orden de los argumentos en dif(m,n) puede definirse como

$$dif(p_2(n,m),p_1(n,m))=dif(m,n)$$

Funciones Recursivas Primitivas

Ejemplo

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)))

definición por sustitución

Funciones Recursivas Primitivas

Ejemplo

Función de desigualdad desigual:

desigual(n,m) =
- 1, si n ≠ m,
- 0, en otro caso

desigual(n,m) = cosg(igual(n,m))  definición por sustitución

Funciones Recursivas Primitivas

Ejemplo

Función identidad:

$$identidad(n) = n = p_1(n)$$

En este caso, la función identidad coincide con una de las funciones básicas recursivas primitivas.

Funciones Recursivas Primitivas

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
- . . .
- m_k, si n = n_k
- g(n), en otro caso

Para comprobar que f(n) es recursiva primitiva basta ver que podemos definir f(n) de la siguiente forma

$$\forall\, n \in \mathbb{N}: \mathrm{f(n)} = \mathrm{igual}(n, n_1) \cdot m_1 + \mathrm{igual}(n, n_2) \cdot m_2 + \cdots + \mathrm{igual}(n, n_k) \cdot m_k +$$
$$+ \mathrm{desigual}(n, n_1) \cdot \mathrm{desigual}(n, n_2) \cdot \cdots \cdot \mathrm{desigual}(n, n_k) \cdot \mathrm{g(n)}$$

Funciones Recursivas Primitivas

Funciones recursivas primitivas definidas mediante sumatorios y productorios finitos

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

(a) Sumatorios finitos

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

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

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

(b) Productorios finitos

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

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

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

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 lo valores $n_1, \dots, n_k$

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

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

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

o

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

Funciones μ-recursivas

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

A las funciones μ-recursivas también se las conoce como funciones recursivas parciales.

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

Para la demostración de la proposición anterior, basta con demostrar que existe una máquina de Turing que puede aplicar el operador μ sobre cualquier función Turing computable. Una máquina que realice ese cálculo debería calcular, de forma incremental, el valor de m para el que $g(m, n_1, \ldots, n_k)$ sea igual a cero, tomando g como una función Turing computable. Bastaría con empezar con un valor de $m = 0$ e ir incrementándolo hasta satisfacer la condición $g(m, n_1, \ldots, n_k) = 0$.

Funciones Turing computables vs funciones recursivas primitivas

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

Para la demostración del 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: $\forall n, m \in \mathbb{N}$:

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

Funciones Turing computables vs funciones recursivas primitivas

Teorema: La función de Ackermann es una función total Turing computable

Para la demostración del teorema puede construirse una máquina de Turing que calcule la función de Ackermann.

- Obsérvese que, de forma intuitiva, basta con implementar una pila de recursividad para que se puedan ir almacenando los sucesivos cálculos de la función.
- Cuando se llega a un valor de argumentos $A(0,k)$, entonces se soluciona con un valor igual a $k+1$ y se pueden resolver el resto de valores de la función donde intervenga este cálculo.
- En cualquier caso, para el valor de función $A(j,i)$ se tiene que resolver hasta un total máximo de $j \times i$ valores de la función que se pueden calcular de forma incremental.

Funciones Turing computables vs funciones recursivas primitivas

Proposición: Para cada función recursiva primitiva $g: \mathbb{N} \longrightarrow \mathbb{N}$ existe un número natural k de modo que $(\forall\, m \in \mathbb{N})\ g(m) < A(k,m)$

Podemos definir la función $\hat{A}: \mathbb{N} \longrightarrow \mathbb{N}$ de modo que $(\forall\, n \in \mathbb{N})\ \hat{A}(n) = A(n,n)$.

Teorema: La función de $\hat{A}$ es una función total Turing computable que no es recursiva primitiva.