Universidad de Huelva

Tema 7

Funciones recursivas

- La teoría clásica de la Computabilidad se define como el estudio de modelos que desarrollan el cálculo de funciones sobre el conjunto de los números naturales. (f: $\mathbb{N} \rightarrow \mathbb{N}$)
- Los modelos que hemos estudiado hasta ahora se basan en el concepto de Máquina de Estado y de cinta de almacenamiento:
  - Los Autómatas Finitos son máquinas de estado que tienen un número finito de configuraciones.
  - Los Autómatas de Pila tienen un número infinito de configuraciones, pero su capacidad de memoria está limitada por las características de la pila (al "usar" la información de la cima de la pila, esta información se destruye).
  - Las Máquinas de Turing tienen un número infinito de configuraciones y la capacidad de acceder a cualquier información almacenada en la cinta.

- Los primeros modelos de computación que se plantearon se apoyaban en la definición formal de $\mathbb{N}$, basada en los Axiomas de Peano.
- Los Axiomas de Peano definen los números naturales a partir de dos ideas básicas: el número 0 y la función Sucesor(n). Los axiomas definen las propiedades lógicas que deben cumplir estos dos conceptos para generar el conjunto de números naturales.
- Cualquier teoría lógica que incluya estos axiomas es una representación de los números naturales.

- Todos los números naturales son o el 0 o el sucesor de otro número natural.

$$\forall x \in N \begin{cases} x \equiv 0 \\ x \equiv Sig(y) \mid y \in N \end{cases}$$

- A partir de los axiomas de Peano podemos definir fácilmente el concepto de función Suma:

$$Suma : N \times N \rightarrow N$$
$$Suma(0,b) = b$$
$$Suma(S(a),b) = S(Suma(a,b))$$

- A partir de esta definición se pueden demostrar las propiedades de la suma como la conmutatividad o la asociatividad.

- La definición anterior de la suma es un ejemplo intuitivo de la noción de recursión. La suma se define mediante un caso base, *Suma(0,a)*, y un caso general que se construye mediante la aplicación de la función a valores menores.
- La definición también hace uso de la composición de funciones.
- Estas ideas llevaron en el comienzo del siglo XX al desarrollo de un modelo de computación basado en la construcción de funciones a partir de funciones básicas (primitivas) y operaciones de composición y recursión.
- Los autores fundamentales formaban parte de la "escuela de Göttingen", el grupo de matemáticos liderados por David Hilbert.

- Funciones básicas

  - Función cero:

$$Z_k(n_1, n_2, \ldots, n_k) = 0$$

  - Función sucesor:

$$S(n) = n+1$$

  - Función proyección:

$$U_i^k(n_1, n_2, \ldots, n_k) = n_i$$

- Composición de funciones:

  Dadas f, una función de aridad k, y $g_1,...,g_k$, funciones de aridad n, se define la **composición de f con $g_1,...,g_k$**, como la función

$$\mathbf{C}(f, g_1,...,g_k) = h(x_1,...,x_n) = f(g_1(x_1,...,x_n), ..., g_k(x_1,...,x_n))$$

- Recursión:

  Dadas f, una función de aridad k, y g, una función de aridad k+2, se define la recursión de f y g, $\boldsymbol{R}(f,g)$, como la función h de aridad k+1 dada por

$$h(0,x_1,\ldots,x_k) = f(x_1,\ldots,x_k)$$

$$h(S(n), x_1,\ldots,x_k) = g(h(n, x_1,\ldots,x_k), n, x_1,\ldots,x_k)$$

Ejemplos:

- Suma(a,b)

  Se puede definir mediante recursión: (0+b) = b; (a+1)+b = (a+b)+1;

  Suma(0,b) = $f(b) = b$ …….…….…..... $f \equiv U_1^1$

  Suma(S(a),b) = g(Suma(a,b),a,b) = S(Suma(a,b)) …… $g \equiv C(S, U_1^3)$

$$\text{Suma}(a,b) \equiv R(U_1^1, C(S, U_1^3))$$

Ejemplos:

- Producto(a,b)

  Se puede definir mediante recursión: (0*b) = 0; (a+1)*b = (a*b)+b;

  Prod(0,b) = $f(b)$ = 0 ……..…… $f \equiv Z_1$

  Prod(S(a),b) = g(Prod(a,b),a,b)

  = Suma(Prod(a,b),b) …… $g \equiv$ Suma(U_1^3, U_3^3)

$$\text{Prod}(a,b) \equiv R(Z_1, C(\text{ Suma}, U_1^3, U_3^3))$$

Ejemplos:

- Potencia(a,b) = b^a

  Se puede definir mediante recursión: $(b^0) = 1$; $b^{a+1} = (b^a)*b$;

  Potencia(0,b) = $f(b)$ = 1 …….…… $f \equiv S(Z_1)$

  Potencia(S(a),b) = g(Potencia(a,b),a,b)

  = Prod(Potencia(a,b),b) …… $g \equiv Prod(U_1^3, U_3^3)$

Potencia(a,b) ≡ R(C(S, Z_1), C(Prod, U_1^3, U_3^3))

Ejemplos:

- Decremento acotado: Dec(a) = a-1, con Dec(0) = 0

  Se puede definir mediante recursión:

  $\text{Dec}(0) = f() = 0$ ..... $f \equiv Z_0$

  $\text{Dec}(S(a)) = g(\text{Dec}(a), a) = a$ ..... $g \equiv U_2^2$

$$\text{Dec}(a) \equiv R(Z_0, U_2^2)$$

Ejemplos:

- Resta acotada: Resta(a,b) = b-a, con Resta(a,b) = 0 si a>b.

  Se puede definir mediante recursión:

  $\text{Resta}(0,b) = f(b) = b$ …… $f \equiv U_1^1$

  $\text{Resta}(S(a), b) = g(\text{Resta}(a,b), a, b)$

  $= \text{Dec}(\text{Resta}(a,b))$ …… $g \equiv \text{Dec}(U_1^3)$

$$\text{Resta}(a) \equiv R(U_1^1, C(\text{Dec}, U_1^3))$$

Ejemplos:

- Diferencia absoluta, |a-b|

  Se puede definir mediante sustitución:

  $$|a-b| = (a-b) + (b-a) = \text{Resta}(a,b) + \text{Resta}(b,a)$$

$$\text{DiferenciaAbsoluta}(a,b) \equiv C(\text{Suma}, \text{Resta}, C(\text{Resta}, U_2^2, U_1^2))$$

Ejemplos:

- Factorial, a!

  Se puede definir mediante recursión:

  Factorial(0) = $f()$ = 1 …… $f \equiv S(Z_0)$

  Factorial(S(a)) = g(Factorial(a), a)

  = (a+1)*Factorial(a) …… $g \equiv Prod(U_1^2, S(U_2^2))$

$$Factorial(a) \equiv R(C(S,Z_0), C(Prod, U_1^2, C(S,U_2^2)))$$

Ejemplos:

- Mínimo, min(a,b)

  Se puede definir mediante sustitución:

$$\min(a,b) = b - (b-a)$$

$$\text{Mínimo}(a,b) \equiv C(\text{Resta}, U_2^2, \text{Resta})$$

Ejemplos:

- Máximo, max(a,b)

  Se puede definir mediante sustitución:

$$\max(a,b) = a + (b - a)$$

$$\text{Máximo}(a,b) \equiv C(\text{Suma}, U_1^2, \text{Resta})$$

Ejemplos:

- Signo(a): Signo(0) = 0, Signo(a) = 1

  Se puede definir mediante recursión:

  $\text{Signo}(0) = f() = 0$ ..... $f \equiv Z_0$

  $\text{Signo}(S(a)) = g(\text{Signo}(a), a) = 1 = S(Z_2)$

  $= 1 = S(Z_2)$ ..... $g \equiv C(S, Z_2)$

$$\text{Signo}(a) \equiv R(Z_0, C(S, Z_2))$$

Ejemplos:

- SignoInverso(a): SignoInverso(0) = 1, SignoInverso(a) = 0

  Se puede definir mediante recursión:

  SignoInverso(0) = $f()$ = 1 …… $f \equiv S(Z_0)$

  SignoInverso(S(a)) = g(SignoInverso(a), a)

  = 0 = Z_2 …… $g \equiv Z_2$

SignoInverso(a) ≡ R(C(S, Z_0), Z_2)

Ejemplos (Cutlan, página 36):

- If(a, b, c) = Si a es cierto devuelve b, si no devuelve c
- And(a,b)
- Or(a,b)
- Not(a)

Ejemplos (Cutlan, página 36):

- Cociente, a/b
- Resto, a%b
- Divisible, divisible(a,b) = 1 si resto(a,b) = 0

- Sumatorio acotado
- Producto acotado
- Operador de minimización acotado

- Número de divisores
- EsPrimo

- Las funciones que pueden computarse a partir de las funciones recursivas básicas y las operaciones de composición y recursión forman un conjunto cerrado denominado **funciones recursivas primitivas**.
- Existen funciones computables que no son primitivas recursivas, por ejemplo la función de Ackermann:

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

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

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

- Función de Ackermann:

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

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

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

Números de $A(m,n)$

m\n	0	1	2	3	4	n
0	1	2	3	4	5	$n+1$
1	2	3	4	5	6	$n+2$
2	3	5	7	9	11	$2n+3$
3	5	13	29	61	125	$8 \cdot 2^n - 3$
4	13	65533	$2^{65536} - 3 \approx 2 \cdot 10^{19728}$	$A(3,2^{65536}-3)$	$A(3,A(4,3))$	$2^{2^{\cdot^{\cdot^{2}}}} - 3$ ($n+3$ términos)
5	65533	A(4,65533)	A(4,A(5,1))	A(4,A(5,2))	A(4,A(5,3))	
6	A(5,1)	A(5,A(5,1))	A(5,A(6,1))	A(5,A(6,2))	A(5,A(6,3))	

  - **NOTA:** $A(4, 2)$ es mayor que el número de partículas que forman el universo elevado a la potencia 200
  - **NOTA:** $A(5, 2)$ no se puede escribir dado que no cabría en el Universo físico.

- Historia:
  - En 1928, Wilhelm Ackermann consideró una función doblemente recursiva A(m, n, p) de tres variables: m → n → p en la notación de Conway. Ackermann demostró que se trata de una función recursiva que no es primitiva recursiva. Esa definición fue simplificada por Rózsa Péter y Raphael Robinson a la versión de dos variables. Rozsa Peter también demostró que la doble recursión no se puede reducir a recursión primitiva (y que de igual forma la triple recursión no se puede reducir a recursión primitiva y doble recursión, etc).
  - Sin embargo, la primera función doblemente recursiva que no es recursiva primitiva fue descubierta por Gabriel Sudan en 1927:

$$F_0(x, y) = x + y,$$

$$F_{n+1}(x, 0) = x, \quad n \geq 0$$

$$F_{n+1}(x, y+1) = F_n(F_{n+1}(x, y), F_{n+1}(x, y) + y + 1), \quad n \geq 0.$$

  - Tanto Sudan como Ackermann eran alumnos de David Hilbert.

- Teorema:
  - La función de Ackermann no es primitiva recursiva

Demostración

- Se basa en un teorema conocido como "lema de mayoración" que indica que toda función primitiva recursiva está mayorada (acotada superiormente) por una función de Ackermann. Es decir,

$$\text{Si } g(x) \in \text{FRP, existe } k \text{ tal que } g(x) < A(k,x) \text{ para todo } x$$

- Consideremos la función ACK(x) como A(x,x). Si ACK es primitiva recursiva, entonces ACK+1 también debe serlo y, por tanto,

$$\text{existe un } k \text{ tal que } ACK(x)+1 < A(k,x) \text{ para todo } x$$

- Pero entonces, para $x=k$ se obtiene que

$$A(k,k)+1 < A(k,k)$$

- lo que es imposible. Luego ACK no puede ser recursiva primitiva.

- La demostración de las limitaciones de las funciones primitivas recursivas llevó a varios autores a buscar una nueva operación que permitiera computar recursiones de orden superior (Gödel, Herbrand y Kleene) lo que condujo a la definición del ***operador de minimización.***
- Si $f(x,z_1,z_2,...,z_n)$ es una función parcial sobre los naturales con n+1 argumentos x, $z_1,...,z_n$, la función $\mu x\ f$ es la función parcial con argumentos $z_1,...,z_n$ que retorna el más pequeño x tal que $f(0,z_1,z_2,...,z_n)$, $f(1,z_1,z_2,...,z_n)$, ..., $f(x,z_1,z_2,...,z_n)$ están todas definidas y $f(x,z_1,z_2,...,z_n) = 0$, si un tal x existe; en caso contrario, $\mu x\ f$ no está definida para los valores particulares de los argumentos $z_1,...,z_n$.

- El operador de minimización puede producir funciones parciales, ya que puede que no exista ningún valor de x que cumpla que $f(x,z_1,z_2,...,z_n) = 0$.

- Las funciones que pueden computarse a partir de las funciones recursivas básicas y las operaciones de composición, recursión y minimización forman un conjunto cerrado denominado **funciones recursivas parciales**.

- Las funciones recursivas parciales tienen la misma capacidad de cómputo que las Máquinas de Turing.

- Dada una función recursiva, *f*, es posible calcular un número que la codifique (lo que se conoce como **número de Gödel**).

- Para un par de números (m,n) se puede definir una codificación como

$$\pi(m,n) = 2^m (2n + 1) - 1$$

- La inversa de esta codificación se puede calcular como

$$\pi^{-1}(x) = (\pi_1(x), \pi_2(x))$$

$\pi_1(x)$ = número de veces que (x+1) es divisible entre 2.

$\pi_2(x) = (((x+1) / 2^{\pi 1(x)}) - 1) / 2$

- Una tripleta de números (m, n, q) puede codificarse como

$$\zeta(m, n, q) = \pi(\pi(m,n), q)$$

- La inversa de esta codificación se puede calcular como

$$\zeta^{-1}(x) = (\pi_1(\pi_1(x)), \pi_2(\pi_1(x)), \pi_2(x))$$

- Una lista de números $(a_1, \ldots, a_k)$ puede codificarse como

$$\tau(a_1, \ldots, a_k) = 2^{a1} + 2^{a1+a2+1} + 2^{a1+a2+a3+2} + \ldots + 2^{a1+a2+\ldots+ak+k-1} - 1$$

- Para calcular la inversa se calcula la expresión binaria

$$x+1 = 2^{b1} + 2^{b2} + 2^{b3} \ldots + 2^{bk}$$

y se obtiene a_i en función de b_i

- Para codificar las funciones recursivas hay que definir codificaciones para las funciones básicas y los operadores:

  - Codificación($Z_k\ (n_1, \ldots, n_k)$) = 6 * k
  - Codificación (S(n)) = 1
  - Codificación ($U_i^k\ (n_1, \ldots, n_k)$) = 6 * $\pi(i,k)$ +2
  - Codificación ($\mathbf{C}(f, g_1,\ldots,g_k)$) = 6 * $\tau(f, g_1,\ldots,g_k)$ +3
  - Codificación ($\boldsymbol{R}(f,g)$) = 6 * $\pi(f,g)$ +4
  - Codificación ($\mu x\ (f(x,z_1,z_2,\ldots,z_n)$) = 6 * f +5

- Ejemplo: Suma$(a,b) \equiv R(U_1^1, C(\ S, U_1^3)\ )$

- Dado un número, x, se puede conocer el tipo de función que representa calculando el resto de la división por 6. A partir de ahí se puede calcular los operandos y aplicar la función.

- Se denomina función universal, $U(x,y)$, a la función que toma como primer argumento la codificación de una función, f, y como segundo argumento la codificación de la entrada a dicha función, $(n_1, \ldots, n_k)$, y calcula dicha función sobre dicha entrada.

$$U(x,y) = U(\text{ codif}(f), \text{codif } (n_1, \ldots, n_k)\ ) = f(n_1, \ldots, n_k)$$

TEOREMA:

- $U(x,y)$ es una función recursiva.

DEMOSTRACIÓN

- El funcionamiento de $U(x,y)$ consiste en decodificar x e y y aplicar la función. Puesto que la decodificación es una función recursiva, la función U también lo es.

- Se dice que una función total es *decidible* si puede expresarse como una función recursiva total.
- Se dice que una función parcial es *parcialmente decidible* si puede expresarse como una función recursiva parcial.
- Se dice que una función es *indecidible* si no puede expresarse como una función recursiva.

- Sea $g(x)$ la función que toma como entrada la codificación de una función ϕ y genera como salida el valor 1 si la función ϕ es total, y el valor 0 si la función ϕ es parcial.

TEOREMA ('ϕ(x) es total' es un problema indecidible)

- La función $g(x)$ no es una función recursiva.

DEMOSTRACIÓN:

- Consideremos la función

$$f(x) = \begin{cases} \phi_x(x)+1 & si \quad \phi_x \quad es \quad total \\ 0 & si \quad \phi_x \quad no \quad es \quad total \end{cases}$$

DEMOSTRACIÓN:

- La función *f* puede reescribirse como

$$f(x)=\begin{cases} U(x,x)+1 & si \quad g(x)=1 \\ 0 & si \quad g(x)=0 \end{cases}$$

- Si $f(x)$ es recursiva, entonces existe la codificación de *f*, (z=codif(f)). En tal caso:

$$U(z,z) = f(z) = U(z,z)+1 \quad \text{¡CONTRADICCIÓN!}$$

- Pero $U(x,y)$ es recursiva, luego $g(x)$ no puede serlo.

Otras funciones no decidibles:

- $\phi_x = 0$
- $\phi_x = \phi_y$
- ...

TEOREMA

El conjunto de funciones recursivas (R) y el conjunto de funciones computables-Turing (TC) es el mismo.

DEMOSTRACIÓN ($R \subseteq TC$):

- Toda función recursiva puede computarse mediante máquinas de Turing.
- La demostración consiste en desarrollar máquinas de Turing que computen las funciones primitivas y los mecanismos de composición, recursión y minimización.

DEMOSTRACIÓN ($TC \subseteq R$):

- Toda función computable-Turing puede calcularse como una función recursiva.
- El comportamiento de una máquina de Turing puede representarse como una transformación entre configuraciones.
- Se puede demostrar que las transformaciones sobre las configuraciones son funciones recursivas.
  - $S_{p,q}(x)$ ≡ configuración sustituyendo el estado p por q.
  - $L(x)$ ≡ configuración moviendo el cabezal a la izquierda
  - $R(x)$ ≡ configuración moviendo el cabezal a la derecha
  - $W_r(x)$ ≡ configuración escribiendo r.
  - $T(x)$ ≡ configuración tras realizar una transición.

DEMOSTRACIÓN ($TC \subseteq R$):

- $c(x,t)$ ≡ configuración tras t iteraciones o configuración final si la máquina se para antes de t iteraciones.
- $j(x,t)$ ≡ código de la transición a realizar tras t iteraciones o código 0 si se alcanza el estado de parada.
- El número de iteraciones de la Máquina de Turing es

$$t_0 = \mu t \ (j(x,t) = 0)$$

- El valor de la función f es

$$f(x) = c(x,t_0)$$