MATA51 – Teoria da Computação
Profa. Laís Salvador

TEORIA DAS FUNÇÕES RECURSIVAS – Funções Recursivas Primitivas

→ Vamos definir uma classe de funções de números naturais que representa o conjunto de funçõescomputáveis.

→ Uma definição de números naturais (PEANO – 1858 – 1932)

- Propriedades

1) *0* é um número natural.
2) Se $\underline{x}$ for um número natural, então existe um número natural definido a partir de *x*, denotado como $S(x)$, (Sucessor de x).
3) Para $\forall$ *x* natural, temos que $0 \neq S(x)$.
4) Se $S(x) = S(y)$, então $x = y$.
5) Se uma propriedade P for tal que:
   a. 0 tem a propriedade P;
   b. Se *x* tem a propriedade P, então S(x) também tem a propriedade P para qualquer *x* natural;

Então todos os números naturais têm a propriedade P.

→ Demonstraremos os números naturais com a notação usual, que facilita a nossa leitura. Assim escreveremos 0, 1, 2, 3, ... em vez de 0, S(0), S(S(0)), S(S(S(0))), ...

→ Kleene (1936) propôs um conjunto de funções iniciais e um conjunto de construtores de funções que podem ser encadeados. Uma **função computável**, segundo Kleene, **é uma função inicial ou uma função obtida a partir das funções iniciais por meio de encadeamento dos construtores.**

→Funções Recursivas Primitivas

As funções iniciais são as seguintes:

1) A função anuladora Zero(x) = 0 para qualquer número natural x.
2) A função sucessor Suc(x) = S(x) para qualquer número natural x.
3) As funções de projeção (seleção):

   $Pr_i^n(x_{1,} \dots, x_n) = x_i$ para qualquer número natural x_i, qualquer $n>0$ e qualquer $1 \leq i \leq n$.

O conjunto de **<u>Funções Recursivas Primitivas (F.R.P)</u>** é o menor conjunto, composto por:

→Funções iniciais;

→Função resultante de um número finito de aplicações dos seguintes construtores sobre FRP:

- **Substituição** (composição):

Se $g(x_1, \ldots, x_m)$, $h_1(x_1, \ldots, x_n)$, ... $h_m(x_1, \ldots, x_n)$, m,n> 0, são funções de números naturais, então construímos por substituição a função:

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m((x_1, \ldots, x_n))$$

- **Recursão**:

Se $g(x_1, \ldots, x_n)$, $h(x_1, \ldots, x_n, y_1, y_2)$, n≥ 0, são funções de números naturais, então construímos recursivamente a função:

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n);$$

$$f\big(x_1, \ldots, x_n, S(y)\big) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$$

Se n =0:

$$f(0) = k, uma\ constante\ arbitrária;$$

$$f\big(S(y)\big) = h\,(y, f(y))$$

Obs. **h é uma função pré-definida.**

EXEMPLOS de FUNÇÕES RECURSIVAS PRIMITIVAS

<u>Ex. (1): Função constante:</u>

$f\ (x) = \ k$

k aplicações da função sucessor sobre Zero(x).

$$f\ (x) = Suc\ \Big(Suc\big(\ \dots, Suc(Zero(x))\big)\Big)$$

$$f\ (x) = Suc^k(Zero(x))$$

<u>Ex. (2): Função Constante com recursão</u>

$f\ (x) = \ k$

por recursão, temos:

(i) $f\ (0) = \ k$

$(ii)\ f\ (S(y)) = h(y, f(y)) = \ Pr_2^2\ (y, f(y)) = f\ (y)$

Ex. Para y=4

$$f\big(S(4)\big) = f(5) = f\ (4) = f\ (3) = f\ (2) = f\ (1) = f\ (0) = k$$

Obs.: Desenvolvimento diferente do apresentado na referência 1.

Ex. (3): Função Soma Primitiva

Por recursão, temos:

$$(1)\ g(x,0) = Pr_1^1(x) = x$$

$$(2)\ g(x,S(y)) = h\,(x,y,g(\,x,y)),$$

Por substituição, se definirmos que

$$h\,(x,y,z) = Suc\left(Pr_3^3(x,y,z)\right) = Suc\,(z), temos\ que$$

$$por\ (2)\ temos,$$

$$g\,(x,S(y)) = h\,(x,y,g(\,x,y)) = Suc\big(g\,(x,y)\big)$$

$$logo,$$

$$(1)\ g(x,0) = x$$

$$(2) g\,\big(x,S(y)\big) = Suc\big(\,g(x,y)\big)$$

$Ex.$ para x=3 e y=1

$$g(\,3,S(1)) = Suc(\,g(\,3,1)) = Suc(\,Suc(\,g(\,3,0)))$$

$$= Suc\,\big(\,Suc(3)\big) = Suc\,(4) = 5$$

Ex. (4)

Seja k um número natural qualquer.

A função $f\ (x) = k * x$ é recursiva primitiva, onde k é uma constante.

Por substituição, obtemos $f\ (x) = Pr_1^1(\rho\ (k, x))$, para alguma função $\rho(\ k, x)$. Se $\rho(\ k, x)$ for primitiva recursiva, então *f(x)* também o será.

Por recursão, seja

$$(1)\rho(\ k, 0) = Zero(k) = 0$$
$$(2)\rho(\ k, S(y)) = h\ (k, y, \rho(\ k, y))$$

Por substituição,

$$h(x, y, z) = g\left(Pr_1^3(x, y, z), Pr_3^3(x, y, z)\right) = g(x, z)$$

onde **g** é a função soma definida no Ex. 3, então

$$\rho(\ k, S(y)) = h\ (k, y, \rho(\ k, y)) =$$
$$g\ \left(k, \rho(k, y)\right)$$

Assim, temos:

$$(1)\rho(\ k, 0) = Zero(k) = 0$$
$$(2)\rho(\ k, S(y)) = g\ \left(k, \rho(k, y)\right)$$

Exemplos:

$f(x) = 2 * x$

$f(1) = \rho(2,S(0)) = g(2,\rho(2,0))=g(2,0)=2$

$f(4) = \rho(2,S(3)) = ...$

Em outras palavras,

$$mult(k,0) = 0$$

$$mult(k,y+1) = k + \big(mult(k,y)\big)$$

<u>Ex. (5)</u>

A função $\mu\ (x, y) = \ x\ .\ y$ é recursiva primitiva

Por recursão, seja

$$(1)\mu\ (x, 0) = \ Zero(x) = \ 0$$

$$(2)\mu\ \big(x, S\ (y)\big) = h\ \big(\ x, y, \mu\ (\ x, y\ )\big)$$

Por substituição, seja

$$h\ (x, y, z) = \ g\ \left(Pr_1^3(\ x, y, z), Pr_3^3(\ x, y, z)\right) = g(x, z)$$

onde **g** é a função soma definida no Ex. 3, então

$$\mu\ \big(x, S(y)\big) = h\ \big(\ x, y, \mu\ (\ x, y\ )\big) = \ g\ (x, \mu(x, y))$$

Logo,

$$(1)\mu\ (x, 0) = \ 0$$

$$(2)\mu(x, S\ (y)) = g\ (\ x, \mu(x, y))$$

em outras palavras,

mult (x,0) = 0

mult (x, y + 1) = x+mult (x, y)

Ex. (6)

A função *fat(n) = n!* é recursiva primitiva por recursão, seja

$$\begin{cases} fat(0) = 1 \\ fat(\,S(y)) = h\,(y, fat\,(y)) \end{cases}$$

*//Lembrete: fat(y+1) = (y+1) * fat(y)*

Seja $h(x, y) = \mu\left(Suc\,\left(Proj_1^2(x, y)\right), Proj_2^2(x, y)\right)$, por substituição.

$= \mu(Suc\,(x), y)$

Onde μ é a função de multiplicação definida no Ex. 5 e provada ser FRP.

Então,

$fat\,(s\,(y)) = h\,(y, fat\,(y)) = \mu(Suc(y), fat(y))$

Em outras palavras,

$$\begin{cases} fat\,(0) = 1 \\ fat\,(y+1) = (y+1) * fat\,(y) \end{cases}$$

FUNÇÕES ENVOLVENDO SUBTRAÇÃO

Ex. (7)

A função Predecessor:

$$prd\ (x) = \begin{cases} x - 1, se x > 0 \\ 0, se x = 0 \end{cases}$$

Ela é Recursiva Primitiva pois é definida pela seguinte recursão:

- *prd(0) = 0*
- *prd (S (x)) = x*

Lembrete:

prd (S(x)) = h (x, prd(x))

Seja *h (x, y)* $= Pr_1^2(x, y) = x$ por substituição,

Então *prd (S(x))* = *h (x, prd(x))*$= x$.

Ex. (8)

Subtração não negativa, definida por:

$$mon(x, y) = \begin{cases} x - y, if x > y \\ 0, if x \leq y \end{cases}$$

Usando recursão,

- *mon (x, 0) = x*
- *mon (x, y+1) = prd (mon (x, y))*

Explicação:

mon (x, y+1) = x – (y+1) = x –y -1 = prd (x –y) =

= prd (mon(x, y)).

PREDICADO RECURSIVO PRIMITIVO

É uma função recursiva primitiva que somente assume os valores 0 e 1.

Exemplo (1)

$$iszero\ (x) = \begin{cases} 1, se\ x = 0 \\ 0,\ caso\ contrário \end{cases}$$

É recursiva primitiva pois é definida pela seguinte recursão:

(i) iszero (0) = 1

(ii) iszero (x+1) = 0

OBS.

iszero(Suc(x)) = h(x,iszero(x))=$Pr_1^2(Zero(x), iszero(x)) = 0$

Exemplo (2)

MAIOR_OU_IGUAL (m, n) = iszero (monus (n, m))

Ex.

MAIOR_OU_IGUAL (7,3) = iszero (monus (3, 7)) = iszero (0) = 1

Observamos que:

MENOR (m, n) = monus (1, MAIOR_OU_IGUAL (m, n))).

Como seriam as funções MENOR_OU_IGUAL (m,n) e EQUALS(m,n)?

Exemplo (3)

EQUALS (m, n) = monus (1, iszero(MAIOR_OU_IGUAL (m, n) . MAIOR_OU_IGUAL (n, m))

Onde . é o símbolo para FRP de multiplicação

Observem que todas as funções definidas nos Exemplos (2) e (3) são recursivas primitivas pois foram construídas em cima de outras FRPs através da aplicação dos construtores de recursão e substituição.

FUNÇÕES DEFINIDAS POR CASO

Sejam ***f, g*** funções recursivas primitivas e ***p*** um predicado recursivo primitivo, todos os 3 sobre as mesmas variáveis, então a função definida por casos:

$$f(x_1, \dots, x_k) = \begin{cases} g\ (x_1, \dots, x_k), se\ p\ (x_1, \dots, x_k\ ) \\ h\ (x_1, \dots, x_k), caso\ contrário \end{cases}$$

Também é recursiva primitiva, pois pode ser reescrita como:

$$\begin{aligned} f\ (x_1, \dots, x_k) \\ &= p\ (x_1, \dots, x_k)\ .\ g\ (x_1, \dots, x_k) \\ &+ \big(1\ \sim p\ (x_1, \dots, x_k)\big)\ .\ h\ (x_1, \dots, x_k) \end{aligned}$$

onde $\sim$ é símbolo para FRP monus

Ex. Resto da divisão

(i) REM (0, n) = 0

(ii) REM (m+1, n) = $\begin{cases} 0, se\ equals\ \big(REM\ (\ m, n), prd\ (n)\big); \\ \quad REM\ (m, n) + 1, caso\ contrário. \end{cases}$

Exercícios

Mostrar que as seguintes funções são recursivas primitivas:

$f(x,y) = x^y$

div (x, y) = quociente da divisão inteira de x por y.

$f(x) = sqr(x) =$ *o maior número natural y tal que* $y \leq \sqrt{x}$

$f(x) = pr(x) =$ a quantidade de números primos menores ou iguais a***x***.

$f(n) = fib(n) =$ o **n**ésimo termo de sequencia de Fibonacci.

Computabilidade das Funções Recursivas Primitivas

Podemos observar que uma função particular f primitiva recursiva é computável e vamos provar esta observação por indução:

Se f é recursiva primitiva, podemos construir uma sequência de funções $f_1, ..., f_n$ tal que $f_n = f$ e cada f_i é uma função inicial ou é obtida de um subconjunto de funções $f_1, ..., f_{i-1}$ primitivas recursivas através de substituição ou recursão. Logo, para provar que f é computável:

Base: As funções iniciais são computáveis.

Hipótese de Indução: Suponha por indução que os algoritmos para avaliar $f_1, ..., f_{i-1}$ foram construídos.

Etapa de Indução: Então com base nas regras de substituição e recursão, podemos construir um algoritmo para avaliar f_i usando os algoritmos $f_1, ..., f_{i-1}$ como subrotinas. Logo f_i também é computável.

O mesmo argumento pode ser usado para demonstrar que as funções recursivas primitivas são totais.

- ⇨ As operações de substituição e recursão produzem funções totais a partir de funções totais.

As funções iniciais são totais

Função de Ackermann

Por outro lado, há funções totais e computáveis que não são primitivas recursivas.

Ex. Função de Ackermann:

$$(1)\ \ H(0,x) = S(x)$$
$$(2)\ \ H(S(n),0) = H(n,1)$$
$$(3)\ H(S(n),S(x)) = H\big(n, H(S(n),x)\big)$$

H (2, 5) = ?

→ Construir um programa (em C/Pascal/Phyton/MatLab/Java) para computar essa função.

Função de Ackermann, de uma outra forma:

$$\begin{cases} H\ (n+1, x+1) = H\ \big(n, H(n+1, x)\big)(1) \\ H\ (\ n+1, 0) = H(n, 1)(2) \\ H\ (0, x) = x\ + 1 \quad (3) \end{cases}$$

Provar que:

$$H\ (1, x) = x + 2 \quad (4)$$

(Provar por indução)

<u>Base</u>:

$$H\ (0, x) = x + 1 \quad (3)$$

<u>Hipótese de Indução:</u>

$$H\ (1, x - 1) = x - 1 + 2 = x + 1 \quad (4)$$

<u>Etapa de Indução</u>:

Por (1) H (1, x) = H (0, H (1, x – 1))

Por (4) H (0, H (1, x – 1))= H (0, x + 1)

Por (3) H (0, x + 1)= x + 2

Provar (5) H (2, x) = 2x + 3 (suponha que (5) seja verdade para x – 1).

$$
\begin{aligned}
H(2, x) &= H(1, H(2, x-1)) && \text{[equação (1)]} \\
&= H(2, x-1) + 2 && \text{[equação (4)]} \\
&= 2(x-1) + 3 + 2 && \text{[passo de indução]} \\
&= 2x \cancel{-2} + 3 \cancel{+2} \\
&= 2x + 3
\end{aligned}
$$

Provar que: $H(3, x) = 2^{x+3} - 3$

Esta função cresce muito rápido:

$H(4, x) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{16}}}}}} - 3$

A demonstração, de que a função de Ackermann não seja uma função recursiva primitiva, está fora do escopo destas notas. Porém, podemos avançar mais sobre funções totais e computáveis que **não são recursivas primitivas**.

Referências

1. **Modelos Clássicos de Computação.** Flavio Soares Correa da Silva e Ana Cristina Vieira de Melo. Cengage, 2010.
2. **Elementos de teoria da computação.**LEWIS, Harry R.; PAPADIMITRIOU, Christos H. Bookman, 2000.
3. **Machines, languages, and computation.** Peter J. Denning, Jack Bonnell Dennis, Joseph E. Qualitz
   Prentice-Hall, 1978.