2. Geração de Funções Computáveis

2.1 As funções básicas

2.2 Concatenação de programas

2.3 Substituição

2.4 Recursão

2.5 Minimização

2.1 As funções básicas

Métodos que permitem combinar funções computáveis dão origem a novas funções computáveis.

É possível demonstrar rapidamente que um grande número de funções são computáveis sem ser necessário escrever os repectivos programas URM.

As seguintes **funções básicas** são computáveis**:**

(a) a função zero **0** (**0**(x) = 0, para todo o x); [Z(1)]

(b) a função sucessor **S** : S(x) = x+1; [S(1)]

(c) $\forall\ n \geq 1$ e $\forall\ 1 \leq i \leq n$, a função projecção $\mathbf{U_i^n}$

$$U_i^n(x_1, x_2, \ldots, x_n) = x_i \qquad [T(i,1)]$$

2.2 Concatenação de Programas

Sejam P e Q dois programas.

Queremos escrever o programa R composto por P e Q, que execute primeiro P e a seguir Q.

Definição:

Diz-se que um programa P = $I_1, I_2, \ldots, I_s$ se encontra **normalizado** se, para toda a instrução de salto J(m, n, q) de P, se tem $q \leq s+1$.

2.2 Concatenação de Programas

Lema:

Para todo o programa P, existe um programa normalizado P* tal que para qualquer $a_1, a_2, \ldots, a_n, b$,

$P(a_1, a_2, \ldots, a_n)\downarrow b$ se e só se $P^*(a_1, a_2, \ldots, a_n)\downarrow b$

Prova:

Seja $P = I_1, I_2, \ldots, I_s$ o programa a normalizar. Seja P* o programa obtido a partir de P tal que $P^* = I^*_1, I^*_2, \ldots, I^*_s$, no qual

se I_k não é uma instrução de salto, então $I^*_k = I_k$

se $I_k = J(m,n,q)$, então $I^*_k = \begin{cases} I_k & \text{se } q \leq s+1 \\ J(m,n,s+1) & \text{se } q > s+1 \end{cases}$

2.2 Concatenação de programas

Sejam P e q dois programas normalizados. A junção de P e Q num único programa R impõe que cada instrução J(m,n,q) de Q seja transformada na instrução J(m,n,q+s) de R (considera-se que P tem s instruções).

Definição

Sejam P e Q programas normalizados com *s* e *t* instruções, respectivamente. A **concatenação** ou **junção** de P e Q, designada por **PQ** é o programa $I_1, I_2,\ldots,I_s, I_{s+1}, I_{s+2},\ldots,I_{s+t}$, onde $P = I_1, I_2,\ldots,I_s$, e as instruções $I_{s+1}, I_{s+2},\ldots,I_{s+t}$ são as instruções de Q com cada instrução de salto J(m,n,q) substituída por J(m,n,s+q).

2.2 Concatenação de programas

Como todo o programa P é finito, existe uma ordem *u* tal que nenhum dos registos R_k, para k > *u*, é referido em P. Seja $r\ (P)$ o menor *u* que verifica esta condição, ou seja $r\ (P)$ é o maior registo mencionado em P.

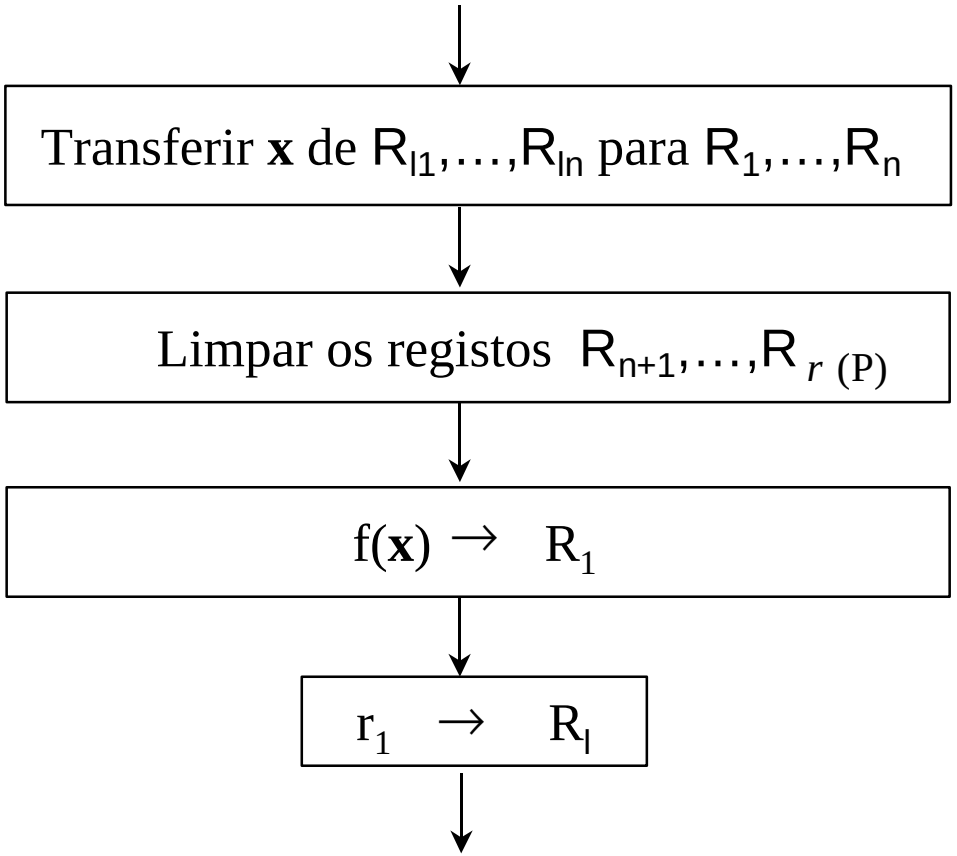
Seja P um programa normalizado para a função $f(x_1,x_2,\ldots,x_n)$.

É frequente recorrer a registos auxiliares $R_{l1},\ldots,R_{ln}$, para guardar os valores iniciais de $x_1,x_2,\ldots,x_n$ e a um registo R_l para guardar o resultado de $f(x_1,x_2,\ldots,x_n)$.

Escreve-se $\mathbf{P[I_1, I_2,\ldots,I_n \rightarrow I]}$ para designar o programa que calcula $f(r_{l1},\ldots,r_{ln})$ e guarda o resultado em R_l .

Os únicos registos afectados por P são os que se encontram entre R_1 e $R_{r\ (P)}$ e ainda R_l .

2.2 Concatenação de programas



$$\begin{cases} T\,(l_1,\ 1) \\ \vdots \\ T\,(l_n,\ n) \end{cases}$$

$$\begin{cases} Z(n+1) \\ \vdots \\ Z(r\ \ (P)) \end{cases}$$

P

$T\,(1,\ l)$

2.3 Substituição

Uma forma comum de construir novas funções é utilizar a chamada composição de funções.

A classe $\mathbb{C}$ é fechada para a composição de funções:

Teorema

Sejam $f(y_1, \ldots, y_k)$ e $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ funções computáveis,

onde $\mathbf{x} = (x_1, \ldots, x_n)$. Então a função $h(\mathbf{x})$ dada por

$h(\mathbf{x}) = f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$ é computável.

Nota: h(x) está definida se e só se $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ estão todas definidas e $(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})) \in \text{Dom}(f)$; logo, se f e $g_1,\ldots,g_k$ são totais, então h é total.

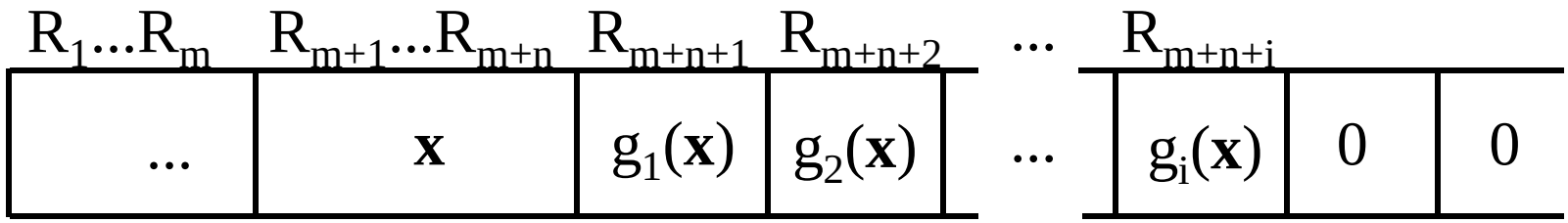
2.3 Substituição

Prova:

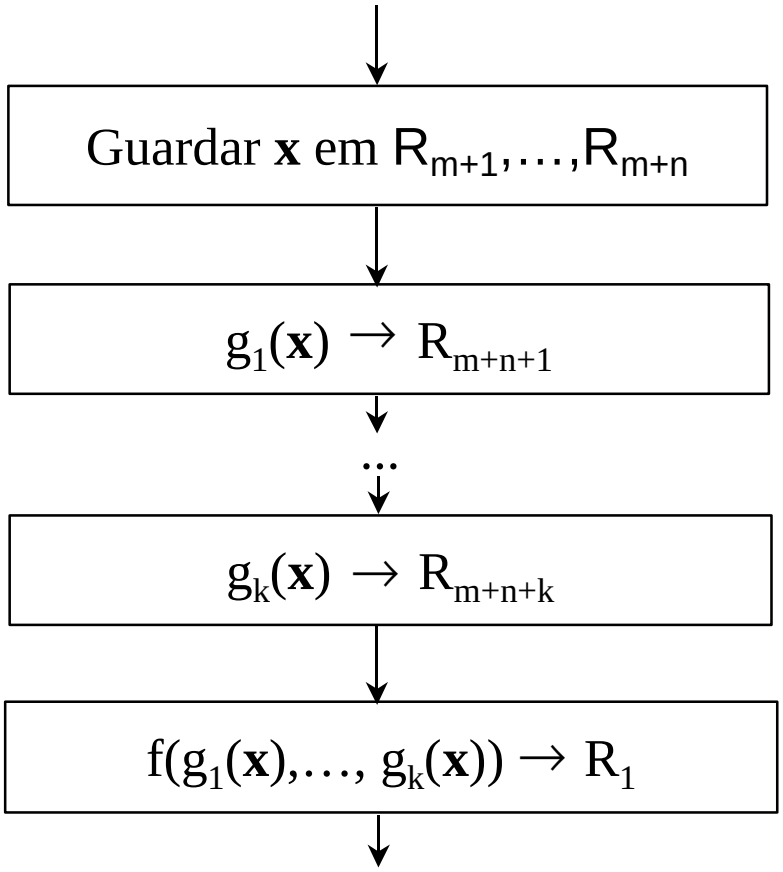
Sejam F, $G_1,\ldots,G_k$ programas normalizados para as funções f, $g_1,\ldots,g_k$, respectivamente. Construiremos o programa **H** para a função **h** da seguinte forma: dado **x**, usar os programas $G_1,\ldots,G_k$ para calcular $g_1(\mathbf{x}),\ldots,g_k(\mathbf{x})$. Usar então o programa F para calcular $f(g_1(\mathbf{x}),\ldots,g_k(\mathbf{x}))$.

Seja $m = \max(n, k, r(F), r(G_1),\ldots, r(G_k))$

Guardamos **x** nos registos $R_{m+1},\ldots,R_{m+n}$ e $g_1(\mathbf{x}),\ldots,g_k(\mathbf{x})$ nos registos $R_{m+n+1},\ldots,R_{m+n+k}$.



2.3 Substituição



$$\begin{cases} T(1,\ m+1) \\ \quad\vdots \\ T(n,\ m+n) \end{cases}$$

$$G_1[m+1, m+2, \ldots, m+n \rightarrow m+n+1]$$

...

$$G_k[m+1, m+2, \ldots, m+n \rightarrow m+n+k]$$

$$F[m+n+1, \ldots, m+n+k \rightarrow 1]$$

2.3 Substituição

Corolário

Seja $f(y_1,\ldots,y_k)$ uma função computável e $x_{i1}, x_{i2},\ldots,x_{ik}$ uma sequência de k das variáveis $x_1,\ldots,x_n$ (possivelmente com repetições).

Então a função $h(x_1,\ldots,x_n) \approx f(x_{i1},\ldots,x_{ik})$ é computável.

Prova:

Seja $\mathbf{x} = (x_1,\ldots,x_n)$. Seja $h(\mathbf{x}) = f\,(\,U^n_{i_1}(\mathbf{x}),\ U^n_{i_2}(\mathbf{x}),\ldots,\ U^n_{i_k}(\mathbf{x}))$

e h é uma função computável.

2.3 Substituição

Podem obter-se novas funções computáveis a partir de uma dada função computável, por rearranjo, identificação ou adição de var

A partir da função $f(y_1, y_2)$ podemos obter:

Rearranjo: $h_1(x_1,x_2) \approx f(x_2, x_1)$

[*f(x,y)=x-y;* *$h_1(x_1,x_2)=x_2-x_1$*]

Identificação: $h_2(x) \approx f(x,x)$

[*f(x,y)=x*y;* *$h_2(x) = x_2$*]

Adição: $h_3(x_1,x_2,x_3) \approx f(x_2,x_3)$

[*f(x,y) = x*y;* *$h_3(x_1,x_2,x_3)=x_2*x_3$*]

2.4 Recursão

Recursão é um método para definir uma função, especificando cada um dos seus valores à custa de valores previamente definidos.

Sejam f(**x**) e g(**x**, y, z) funções (não necessariamente computáveis ou totais). Seja h(x,y) a nova função definida por:

(i) $h(\mathbf{x}, 0) \approx f(\mathbf{x})$

(ii) $h(\mathbf{x}, y+1) \approx g(\mathbf{x}, y, h(\mathbf{x}, y))$

A função h diz-se definida por recursão a partir de f e g.

O domínio de h satisfaz as condições:

$(\mathbf{x}, 0) \in Dom(h)$ sse $\mathbf{x} \in Dom(f)$

$(\mathbf{x}, y+1) \in Dom(h)$ sse $(\mathbf{x},y) \in Dom(h)$ e $(\mathbf{x}, y, h(\mathbf{x},y)) \in Dom(g)$

2.4 Recursão

Teorema

Seja $\mathbf{x} = (x_1,\ldots,x_n)$ e sejam f ($\mathbf{x}$) e g($\mathbf{x}$, y, z) funções; então, existe uma única função h($\mathbf{x}$, y) satisfazendo as equações de recursão:

(i) h($\mathbf{x}$, 0) $\approx$ f($\mathbf{x}$)

(ii) h($\mathbf{x}$, y+1) $\approx$ g($\mathbf{x}$, y, h($\mathbf{x}$, y))

Nota: Se n = 0 , as equações têm a forma:

(i) h(0) $\approx$ a

(ii) h(y+1) $\approx$ g (y, h(y))

2.4 Recursão

Exemplos

Adição: h(x,y) = x + y

x + 0 = x

x + (y+1) = (x + y) + 1

h é definida por recursão a partir das funções f(x) = x e g(x, y, z) = z + 1:

Factorial: y! Com a convenção de que 0! = 1,

0! = 1

(y+1)! = y! (y+1)

h(y) = y! é definida por recursão a partir de 1 e g(y, z) = z (y+1).

2.4 Recursão

A classe $\mathbb{C}$ é fechada para a definição de funções por recursão

Teorema

Seja **x** = (x_1,…,x_n) e sejam f (**x**) e g(**x**, y, z) funções computáveis; então, a função h(x, y) obtida a partir de f e g por recursão é computável.

Prova:

Sejam F e G programas normalizados para as funções f(x) e g(x,y,z).

Iremos escrever um programa H para a função h(x,y) definida pelas equações de recursão.

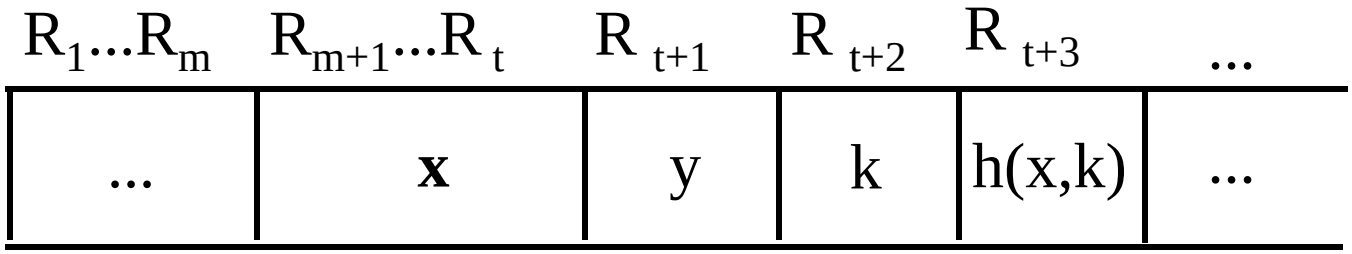
2.4 Recursão

Prova (cont.):

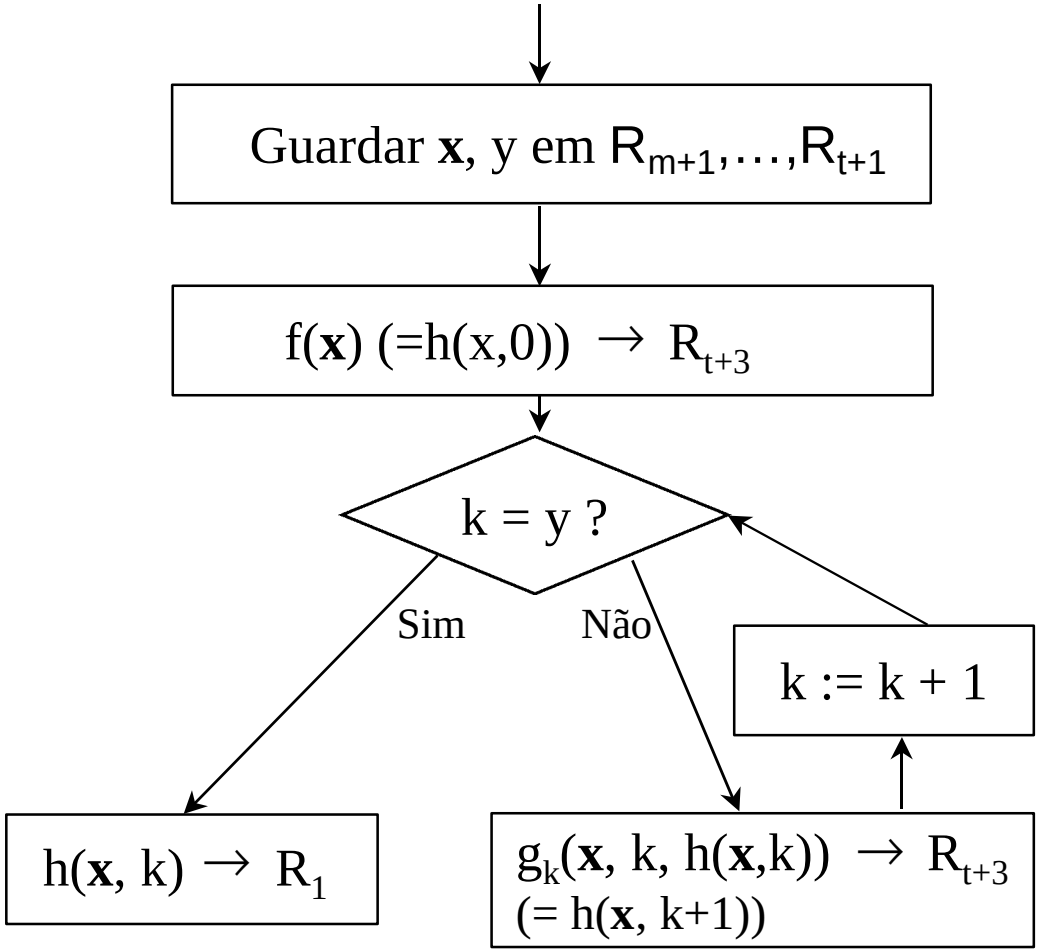
Para uma configuração inicial $x_1,\ldots,x_n$, y, 0,0,… **H** calcula h(**x**,0) usando F; então (se $y \neq 0$), **H** usará G para calcular sucessivamente h(**x**,1), h(**x**,2),…,h(**x**,y).

Seja $m = \max(n + 2,\ r(F),\ r(G))$

e $t = m + n.$



2.4 Recursão



I_q

I_p

$\begin{cases} T(1, m+1) \\ \vdots \\ T(n+1, m+n+1) \end{cases}$	Copia **x** Copia y
$F[1,\ldots,n \rightarrow t+3]$	Calcula h(**x**,0)
$Z(t+2)$	Apaga k
$J(t+2, t+1, p)$	Verifica se k = y
$G[m+1,\ldots,m+n, t+2, t+3 \rightarrow t+3]$	Iteração
$S(t+2)$	Incrementa k
$J(1,1,q)$	Ciclo
$T(t+3,1)$	Resultado final

2.4 Recursão

Usando a substituição e recursão, podemos provar o seguinte teorema

Teorema:

As seguintes funções são computáveis

(a)	x + y	(h)	| x - y |
(b)	xy	(i)	x!
(c)	x^y	(j)	min (x,y)
(d)	x - 1	(k)	max (x,y)
(e)	x - y	(l)	rm(x,y) 'resto qdo y é dividido por x' , rm(0,y) = y
(f)		(m)	qt(x,y) 'quociente qdo y é dividido por x', qt(0,y) = 0
(g)		(n)	

2.4 Recursão

Teorema:

As seguintes funções são computáveis

(a) $x + y$

(b) xy

(c) x^y

(d) $x \dot{-} 1$

(e) $x \dot{-} y$

(f) $sg(x) = \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x \neq 0 \end{cases}$

(g) $\overline{sg}(x) = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{se } x \neq 0 \end{cases}$

(h) $| x - y |$

(i) $x!$

(j) min (x,y)

(k) max (x,y)

(l) rm(x,y) 'resto quando y é dividido por x' , rm(0,y) = y

(m) qt(x,y) 'quociente quando y é dividido por x', qt(0,y) = 0

(n) $div(x,y) = \begin{cases} 1 & \text{se } x \mid y \text{ (x divide y)} \\ 0 & \text{sex } \nmid y \end{cases}$

2.4 Recursão

Corolário

Sejam $f_1(\mathbf{x})$, …, $f_k(\mathbf{x})$ funções totais computáveis e $M_1(\mathbf{x})$, …, $M_k(\mathbf{x})$ predicados decidíveis tais que para cada **x**, um e só um dos predicados $M_i(\mathbf{x})$ (i=1,…,k) se verfica. Então, a função g(**x**) dada por:

$$g(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) \text{ , se } M_1(\mathbf{x}) \text{ se verifica} \\ f_2(\mathbf{x}) \text{ , se } M_2(\mathbf{x}) \text{ se verifica} \\ \vdots \qquad\qquad \vdots \\ f_k(\mathbf{x}) \text{ , se } M_k(\mathbf{x}) \text{ se verifica} \end{cases}$$

é computável.

Prova: $g(\mathbf{x}) = c_{M_1}(\mathbf{x})f_1(\mathbf{x}) + \cdots + c_{M_k}(\mathbf{x})f_k(\mathbf{x})$ é computável por substituição, usando adição e multiplicação.

2.4 Recursão

Álgebra da decidibilidade

Sejam M(**x**) e Q(**x**) predicados decidíveis. Então, os seguintes predicados são também decidíveis.

(a) ‘ não M(**x**)’

(b) ‘ M(**x**) e Q(**x**)’

(c) ‘ M(**x**) ou Q(**x**)’

Prova: As funções características destes predicados são:

(a) $1-c_M(\mathbf{x})$

(b) $c_M(\mathbf{x})\, c_Q(\mathbf{x})$

(c) $\max(c_M(\mathbf{x}), c_Q(\mathbf{x}))$

Estas funções são computáveis usando substituição e recursão.

2.4 Recursão

Seja f(x,z) uma função qualquer;

Chama-se **soma limitada** à função (em **x** e y) definida por:

$$\sum_{z<y} f(\mathbf{x},z) = \begin{cases} \sum_{z<0} f(\mathbf{x},z) \;=\; 0 \\ \sum_{z<y+1} f(\mathbf{x},z) = \sum_{z<y} f(\mathbf{x},z) + f(\mathbf{x},y) \end{cases}$$

e **produto limitado** como a função (em **x** e y) definida por:

$$\prod_{z<y} f(\mathbf{x},z) = \begin{cases} \prod_{z<0} f(\mathbf{x},z) \;=\; 1 \\ \prod_{z<y+1} f(\mathbf{x},z) = \prod_{z<y} f(\mathbf{x},z) \cdot f(\mathbf{x},y) \end{cases}$$

2.4 Recursão

Teorema

Seja f(**x**,z) uma função total computável. Então as funções $\sum_{z<y} f(\mathbf{x}, z)$ e $\prod_{z<y} f(\mathbf{x}, z)$ são computáveis.

Prova: Estas funções são definidas por recursão de funções computáveis.

Corolário

Sejam f(**x**,z) e k(**x**, **x'**) funções totais computáveis. Então, as funções (em **x** e **x**') $\sum_{z<k(\mathbf{x},\mathbf{x}')} f(\mathbf{x}, z)$ e $\prod_{z<k(\mathbf{x},\mathbf{x}')} f(\mathbf{x}, z)$ também são computáveis

Prova: Por substituição.

2.4 Recursão

Operador de minimização limitada

$m\ z < y(\ldots)$: " o menor z menor do que y tal que …"

Para que esta função seja total, toma o valor y no caso de não existir z que satisfaça a condição. Dada uma função f(**x**,z), podemos definir a função:

$$g(\mathbf{x}, y) = m\ z < y(f(\mathbf{x}, z) = 0)$$

$$= \begin{cases} \text{o menor } z < y \text{ tal que } f(x, z) = 0 & \text{se um tal z existe} \\ y & \text{se não existe z} \end{cases}$$

2.4 Recursão

Teorema

Seja f(**x**,z) uma função total computável. Então a função $m\ z < y(f(\mathbf{x}, z) = 0)$ também é computável.

Prova: Seja $h(\mathbf{x}, v) = \prod_{u \leq v} sg\ (f(\mathbf{x}, u)) = \begin{cases} 1 & se\ \forall u \leq v\ \ f(\mathbf{x}, u) \neq 0 \\ 0 & se\ \exists u \leq v\ \ f(\mathbf{x}, u) = 0 \end{cases}$

que é computável. Para um dado **x**, y, seja $z_0 = m\ z < y(f(\mathbf{x}, z) = 0)$.

Para $v < z_0$, temos h(**x**,v) = 1. Para $z_0 < v < y$, h(**x**,v) = 0.

Então z_0 = nº de v's menores que y tais que h(**x**,v) = 1.

Logo, $z_0 = \sum_{v<y} h(\mathbf{x}, v)$ e

$m\ z < y(f(\mathbf{x}, z) = 0) = \sum_{v<y} \left(\prod_{u \leq v} sg(f(x, u)) \right)$ que é computável (ver teorema anterior)

2.4 Recursão

Corolário

Sejam f(**x**,z) e k(**x**,**x**') funções totais computáveis. Então a função

$\mu\, z < k(\mathbf{x},\mathbf{x}')(f(\mathbf{x},z) = 0)$ também é computável.

Prova: Por substituição de k(**x**,**x**') por y na função computável $\mu\, z < y(f(\mathbf{x},z) = 0)$

Corolário

Seja R(**x**,y) um predicado decidível. Então

(a) a função $f(\mathbf{x},y) = \mu\, z < y\ R(\mathbf{x},z)$ é computável

(b) os seguintes predicados são decidíveis:

(i) $M_1(\mathbf{x},y) \equiv \forall z < y\ R(\mathbf{x},z)$

(ii) $M_2(\mathbf{x},y) \equiv \exists z < y\ R(\mathbf{x},z)$

2.4 Recursão

Prova

$$c_R(\mathbf{x}, y) = \begin{cases} 1 & \text{se } R(\mathbf{x}, y) \text{ se verifica} \\ 0 & \text{se } R(\mathbf{x}, y) \text{ não se verifica} \end{cases}$$

$$\overline{sg}(x) = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{se } x \neq 0 \end{cases}$$

(a) $f(\mathbf{x}, y) = m\ z < y\ (\overline{sg}(c_R(\mathbf{x}, z)) = 0)$

(i) $c_{M_1}(\mathbf{x}, y) = \prod_{z<y} c_R(\mathbf{x}, z)$

$$= \begin{cases} 1 & \text{se } \forall z < y\ R(\mathbf{x}, z) \text{ se verifica} \\ 0 & \text{se } \exists z < y \text{ tq } R(\mathbf{x}, z) \text{ não se verifica} \end{cases}$$

(ii) $M_2(\mathbf{x}, y) \equiv \text{não } (\forall z < y\ (\text{não } R(\mathbf{x}, z)))$

$\text{nao } R(\mathbf{x}, y) \equiv \overline{R}(\mathbf{x}, y) : c_{\overline{R}}(\mathbf{x}, y) = 1 \dot{-} c_R(\mathbf{x}, y)$

$\forall z < y\ (\text{não } R(\mathbf{x}, z)) : c(x, z) = \prod_{z<y} (1 - c_R(\mathbf{x}, z))$

$$= \begin{cases} 1 & \text{se } \forall z < y\ R(x, z) \text{ não se verifica} \\ 0 & \text{se } \exists z < y\ R(x, z) \text{ se verifica} \end{cases}$$

$$c_{M_2}(x, y) = \begin{cases} 1 & \text{se } \exists z < y\ R(x, z) \text{ se verifica} \\ 0 & \text{se } \forall z < y\ R(x, z) \text{ não se verifica} \end{cases}$$

$=$

2.4 Recursão

Teorema

As seguintes funções são computáveis:

(a) D(x) = número de divisores de x (D(0) = 1)

(b) $Pr(x) = \begin{cases} 1 & \text{se x é primo} \\ 0 & \text{se x não é primo} \end{cases}$

(c) p_x = x-ésimo número primo ($p_0 = 0$, $p_1 = 2$, $p_2 = 3$, etc.)

(d) $(x)_y = \begin{cases} \text{o expoente de } p_y \text{ na factorizaçao em numeros primos de x} & \text{se x, y} > 0 \\ 0 & \text{se x = 0 ou y = 0} \end{cases}$

2.4 Recursão

Teorema

As seguintes funções são computáveis:

(a) D(x) = número de divisores de x (D(0) = 1)

(b) $Pr(x) = \begin{cases} 1 & \text{se x é primo} \\ 0 & \text{se x não é primo} \end{cases}$

(c) p_x = x-ésimo número primo ($p_0 = 0$, $p_1 = 2$, $p_2 = 3$, etc.)

(d) $(x)_y = \begin{cases} \text{o expoente de } p_y \text{ na factorizaçao em numeros primos de x} & \text{se x, y > 0} \\ 0 & \text{se x = 0 ou y = 0} \end{cases}$

2.4 Recursão

Prova

(a) D(x) = número de divisores de x (D(0) = 1)

$$D(x) = \sum_{y \le x} div(y,x) \qquad \left(div(y,x) = \begin{cases} 1 & \text{se } y \mid x \\ 0 & \text{se } y \nmid x \end{cases} \right)$$

(b) $$Pr(x) = \begin{cases} 1 & \text{se } x \text{ é primo} \\ 0 & \text{se } x \text{ não é primo} \end{cases} = \begin{cases} 1 & \text{se } D(x) = 2 \\ 0 & \text{se } D(x) \neq 2 \end{cases}$$

$$= \overline{sg}(|D(x) - 2|)$$

(c) $p_0 = 0$

$$p_{x+1} = mz < \underbrace{(p_x !+ 1)}_{computavel} \; \underbrace{(z > p_x \text{ e } z \text{ primo})}_{decidivel}$$

(d) $$(x)_y = m\, z < x \; \underbrace{(p_y^{z+1} \nmid x)}_{decidivel}$$

2.5 Minimização

Seja f(**x**,y) uma função (não necessariamente total).

Queremos definir uma função g(**x**) por:

g(**x**) = *menor y tal que f(x,y) = 0*

por forma que, se f é computável, também g é computável.

Mas…

1º Para algum **x** pode não existir y tal que f(**x**,y) = 0;

2º Sendo f computável, considere-se o seguinte algoritmo para computar g(**x**):
Calcular f(**x**,0), f(**x**,1), …, até encontrar y tal que f(**x**,y) =0.

Se f não é total, o algoritmo pode não terminar, mesmo que exista y.

(Por exemplo: f(**x**,0) é indefinido, mas f(**x**,1) = 0)

2.5 Minimização

Definição

Para qualquer função f(**x**,y),

$$\mu y\,(f(\mathbf{x}, y) = 0) = \begin{cases} \text{o menor y tal que } \begin{cases} \text{(i) } f(\mathbf{x}, z) \text{ esta definido } \forall z \leq y \text{ e} \\ \text{(ii) } f(\mathbf{x}, y) = 0, \text{ se } \exists\, y \end{cases} \\ \text{indefinido, se } \nexists\, y \end{cases}$$

$\mu y\,(\ldots)$ lê-se "*o menor y tal que* …" e designa-se por **operador-μ**

A classe C é fechada para a minimização:

Teorema

Seja $\mathbf{x} = (x_1,\ldots,x_n)$ e seja f (**x**,y) uma função computável. Então, a função

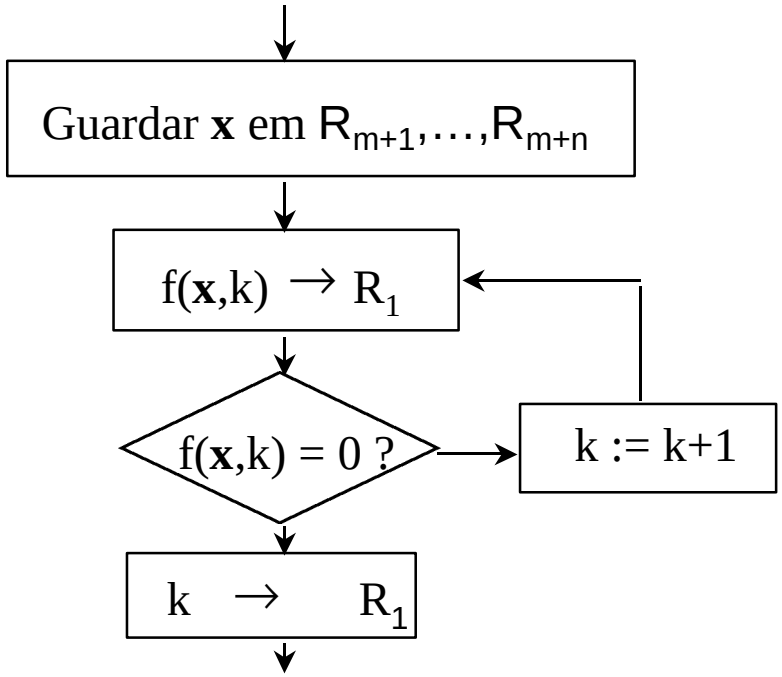
$g(\mathbf{x}) = \mu y\,(f(\mathbf{x}, y) = 0)$ é computável.

2.5 Minimização

Prova:

Seja **x** = (x_1,…,x_n) e **F** um programa normalizado que calcula f(**x**,y). Seja $m = \max(n+1,\ r(F))$. Vamos escrever um programa **G** para computar a função g(**x**).

Para k = 0,1,2,…, calcular f(**x**,k) até um valor de k ser encontrado para o qual f(**x**,k)=0. Este valor de k é o resultado pretendido.



$$\begin{cases} T(1,\ m+1) \\ \quad\vdots \\ T(n,\ m+n) \end{cases}$$

I_p $F[m+1, m+2, \ldots m+n+1 \rightarrow 1]$

$J(1, m+n+2, q)$

$S(m+n+1)$

$J(1,1,p)$

I_q $T(m+n+1,1)$

2.5 Minimização

Corolário

Seja R(**x**,y) um predicado decidível. Então a função $g(\mathbf{x}) = m\ \mathrm{y}\ \mathrm{R}(\mathbf{x}, \mathrm{y})$

$$\mathrm{g}(\mathbf{x}) = m\ \mathrm{y}\ \mathrm{R}(\mathbf{x}, \mathrm{y}) = \begin{cases} \text{o menor y tal que } \mathrm{R}(\mathbf{x}, \mathrm{y}) \text{ se verifica se } \exists\ \mathrm{y} \\ \text{indefinido, se } \nexists\ \mathrm{y} \end{cases}$$

é computável.

Prova: $g(\mathbf{x}) = m\ \mathrm{y}\ \left(\overline{sg}\,(c_R\ (\mathbf{x}, \mathrm{y})) = 0\right)$

O operador-m é também designado por **operador de pesquisa**.

Para um predicado decidível R(**x**,y), a função g(**x**) procura por um y tal que R(**x**,y) se verifica e encontra o menor, no caso de existir algum.

2.5 Minimização

O operador-m permite gerar funções computáveis não totais a partir de funções computáveis totais

Nota: Até agora, só conseguíamos gerar funções totais a partir de funções totais)

Então, o operador-m permite gerar mais funções do que aquelas que são geradas a partir de substituição e recursão.

Exemplo: Seja $f(x, y) = \left|x - y^2\right|$ e $g(x) = m\ y\ (f(x, y) = 0)$

g(x) é uma função não total: $g(x) = \begin{cases} \sqrt{x} & \text{se } x \text{ quadrado perfeito} \\ \text{indefinida} & \text{senão} \end{cases}$

Além disso, existem funções totais para as quais o operador-m é essencial, pois não podem ser geradas a partir de substituição e recursão: **Função de Ackerman**

2.5 Minimização

Função de Ackerman

$$y\,(0, y) = y + 1$$
$$y\,(x+1, 0) \approx y\,(x, 1)$$
$$y\,(x+1, y+1) \approx y\,(x, y\,(x+1, y))$$