Notação

- $P(a_1, \ldots, a_n) \uparrow$ denota que $P(a_1, \ldots, a_n)$ não termina (i.e. a execução de P a partir da configuração inicial $a_1, \ldots, a_n$ **diverge**);
- $P(a_1, \ldots, a_n) \downarrow$ denota que $P(a_1, \ldots, a_n)$ termina em algum momento (i.e. a execução de P a partir da configuração inicial $a_1, \ldots, a_n$ **converge**);
- $P(a_1, \ldots, a_n) \downarrow b, b \in \mathbb{N}_0$ denota que $P(a_1, \ldots, a_n)$ termina em algum momento e na configuração o registo R_1 contém o número b (i.e. a execução de P a partir da configuração inicial $a_1, \ldots, a_n$ **converge para** b);
- $f(x) = -$ significa que f não está definida em x, i.e. $x \notin Dom(f)$.

Execução de um programa URM

Seja P um programa URM.

1. A máquina URM executa $P[1]$;
2. A execução da instrução $P[x]$, com $1 \leq x \leq \#P$ processa-se da seguinte maneira:
   a. $P[x] = Z(i)$: a máquina URM coloca o valor 0 no registo R_i e executa a instrução $P[x+1]$;
   b. $P[x] = S(i)$: a máquina URM incrementa o valor contido em R_i em uma unidade e executa a instrução $P[x+1]$;
   c. $P[x] = T(i,j)$: copia o valor contigo em R_i para R_j e executa a instrução $P[x+1]$;
   d. $P[x] = J(i,j,k)$:
      i. Se $r_i =_r j$, a máquina URm executa a instrução $P[k]$;
      ii. Caso contrário, executa a instrução $P[x+1]$.
   e. Quando a próxima instrução não existe (i.e. $P[x]$ com $x > \#P$), a máquina URM termina a execução de P.

Função Computável

Seja $f : \mathbb{N}_0^n \nrightarrow \mathbb{N}_0, n \geq 1$.

a. Um programa URM P calcula a *função* f, se e só se, para quaisquer $a_1, \ldots, a_n, b \in \mathbb{N}_0$ se tem que:
   i. $P(a_1, \ldots, a_n) \downarrow b$, se e só se, $(a_1, \ldots, a_n) \in Dom(f)$ e $f(a_1, \ldots, a_n) = b$.
   ii. $P(a_1, \ldots, a_n) \uparrow$, se e só se, $f(a_1, \ldots, a_n) = -$(i.e. $(a_1, \ldots, a_n) \notin Dom(f)$).

b. A *função* f é URM-computável, se e só se, existe um programa URm que calcula f.

- Denotamos por $\mathcal{C}$ a classe das funções parciais URM-computáveis e por $\mathcal{C}_n$ a classe das funções parciais *n*-árias URM-computáveis;
- *"Computável"* é uma abrevitura de *"URM-computável"*.

Funções Calculadas por Programa URM

Definição

Qualquer programa URM calcula apenas uma função (parcial) de cada aridade $n \geq 1$.

Seja P um programa URM e $n \in \mathbb{N}_1$. A única função *n*-ária calculada por P é denotada por $f_p^{(n)}$, definida da seguinte maneira:

$$\text{Para quaisquer } x_1, \ldots, x_n \in \mathbb{N}_0$$
$$f_P^{(n)}(x_1, \ldots, x_n) = \begin{cases} -, & P(x_1, \ldots, x_n) \uparrow \\ y, & P(x_1, \ldots, x_n) \downarrow y \end{cases}$$

Exemplo

Seja P o programa URM $(J(2,3,5), S(1), S(3), J(1,1,1))$. Vamos determinar as funções, de cada aridade, calculadas por P.

- $f_p^{(1)}$:
  - Para qualquer $x \in \mathbb{N}$, a execução de P a partir da configuração inicial $(x, 0, 0, \ldots)$ termina sem fazer qualquer alteração:

Instrução	r_1	r_2	r_3	r_4	r_5
1	3	0	0	0	0
STOP	3	0	0	0	0

  - Logo, $f_p^{(1)}$ é a função definida da seguinte maneira:

$$\begin{aligned} f_P^{(1)}: \quad & \mathbb{N}_0 \not\to \mathbb{N}_0 \\ & x \mapsto f_P^{(1)}(x) = x \end{aligned}$$

- $f_p^{(2)}$:
  - Para quaisquer $x, y \in \mathbb{N}$, a execução de P a partir da configuração inicial $(x, y, 0, \ldots)$ termina com a configuração final $(x+y, y, y, 0, \ldots)$:

Instruction	r_1	r_2	r_3	r_4	r_5
1	1	2	0	0	0
2	1	2	0	0	0
3	2	2	0	0	0
4	2	2	1	0	0
1	2	2	1	0	0
2	2	2	1	0	0
3	3	2	1	0	0
4	3	2	2	0	0
1	3	2	2	0	0
STOP	3	2	2	0	0

- Logo, $f_p^{(2)}$ é a função definida da seguinte maneira:

$$\begin{aligned} f_P^{(2)}: \quad & \mathbb{N}_0 \nrightarrow \mathbb{N}_0 \\ & (x,y) \mapsto f_P^{(2)}(x,y) = x + y \end{aligned}$$

Problemas, Predicados e Propriedades Decidíveis

Definição

Seja $M(x_1, \ldots, x_n)$ um predicado *n*-ário. A função caraterística do predicado M é a função:

$$\begin{aligned} C_M : \quad & \mathbb{N}_0{}^n \to \mathbb{N}_0 \\ & (x_1, \ldots, x_n) \mapsto C_M(x_1, \ldots, x_n) = \begin{cases} 1, & M(x_1, \ldots, x_n) \\ 0, & \neg M(x_1, \ldots, x_n) \end{cases} \end{aligned}$$

- O *predicado* $M(x_1, \ldots, x_n)$ é decidível se a sua função caraterística é computável:
  - **Uma função caraterística é uma função total.**

Exemplo

Seja M o predicado "$x = 0$".

$$\begin{aligned} C_M : \quad & \mathbb{N}_0{}^n \to \mathbb{N}_0 \\ & x \mapsto C_M(x) = \begin{cases} 1, & x = 0 \\ 0, & x \neq 0 \end{cases} \end{aligned}$$

Para mostrar que M é decidível, basta mostrar que C_M é computável. Para mostrar este último facto, basta apresentar um programa que calcule esta função, como por exemplo:

```
J(1,2,4)
Z(1)
J(1,1,5)
S(1)
```

- $x = 1$, $C_M(1, 0, 0, \ldots)$

Instrução	r_1	r_2	r_3	r_4	r_5
1	1	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
STOP	0	0	0	0	0

- $x = 0, C_M(0, 0, 0, \dots)$

Instrução	r_1	r_2	r_3	r_4	r_5
1	0	0	0	0	0
4	0	0	0	0	0
STOP	1	0	0	0	0

Construção de Programas a Partir de Outros Programas

Definição: Forma Padrão

Um programa P está na forma padrão se, para qualquer instrução de salto $J(m, n, q)$ em P, se tem que $q \leq \#P + 1$.

Definição: Programa P*

Seja P um programa URM. Denotamos por P^* o programa definido da seguinte forma:

a. $\#P^* = \#P$

b. Para cada $I \in \{1, \ldots, \#P\}$:

$$P^*[I] = \begin{cases} J(m, n, \#P + 1), & P[I] = J(m, n, k), \\ & \text{com k} > \#\text{P} + 1 \\ \\ P[I], & \text{caso contrário} \end{cases}$$

- **Exemplo:**

  Seja $P = (J(2, 1, 9), S(2), J(1, 1, 1))$. De acordo com a definição acima, P^* é definido da seguinte maneira:

  a. $\#P^* = \#P = 3$

  b. $P^* = (J(2, 1, 4), S(2), J(1, 1, 1))$

  P não está na forma padrão pois contém a instrução $J(2, 1, 9)$ e $9 > \#P + 1 \equiv 9 > 4$. Por outro lado, P^* está na forma padrão.

Definição: Programas Equivalentes

Dois programas P_1 e P_2 são equivalentes se, para qualquer configuração inicial se tem que:

i. $P_1(a_1, a_2, a_3, \ldots) \downarrow$ se $P_2(a_1, a_2, a_3, \ldots) \downarrow$ e;

ii. Se ambas as execuções terminam, as configurações finais da máquina URM resultante de cada uma destas execuções são idênticas.

Resultado: Programa na Forma Padrão

Seja P um programa URM. O programa P^* está na forma padrão e é equivalente a P.

Junção de Programas

Definição

Sejam P e Q dois programas URM. A junção de P e Q, denotado por $P;Q$ é o programa URm definido da seguinte forma:

a. $\#(P;Q) = \#P + \#Q$

b. Para cada $I \in \{1, \ldots, \#P\}, (P;Q)[I] = P^*[I]$

c. Para cada $k \in \{1, \ldots, \#Q\}$:

$$(P;Q)[\#P + k] = \begin{cases} Q[k], & Q[k] \text{ não é instrução } J \\ J(m, n, r + \#P), & Q[k] = J(m, n, r) \end{cases}$$

- **Exemplo:**

  Sejam P e Q dois programas URM definidos por $\begin{matrix} J(1,2,9) \\ S(2) \\ J(1,1,1) \end{matrix}$ e $\begin{matrix} J(3,2,5) \\ S(1) \\ S(3) \\ J(1,1,1) \end{matrix}$,

  respetivamente.

  A junção $P;Q$ de P e Q resulta no programa URM:

$$\begin{matrix} J(1,2,4) \\ S(2) \\ J(1,1,1) \\ J(3,2,8) \\ S(1) \\ S(3) \\ J(1,1,4) \end{matrix}$$

Utilização de Subprogramas na Máquina URM

Definição: Registo Máximo

Seja P um programa URM. $\rho(P)$ é o máximo do conjunto formado pelos índices dos registos que são mencionados no programa P.

Programa URM Generalizado

Definição: Programa Generalizado

Um programa URM generalizado Q é uma sequência finita de instruções URM generalizadas $(I_1, \ldots, I_k), k \geq 1$, onde cada instrução generalizada $I_r, r \in \{1, \ldots, k\}$ ou é uma instrução URM padrão ou uma instrução de um dos dois tipos seguintes:

i. $CallP$;

ii. $CallP[i_1, \ldots, i_n \to j]$;

onde $n, j \geq 1$ e $i_1, \ldots, i_n > n$ e P é um programa URM onde não ocorrem instruções que chamem o programa Q nem instruções que chamem outros programas que chamem Q.

Geração de Funções URM-computáveis: Funções Primitivas e Funções Parciais Recursivas

Funções Básicas

As seguintes funções são básicas:

a. zero: $zero = \lambda_x \cdot 0$;

b. sucessor: $suc = \lambda_x \cdot x + 1$;

c. Para cada $n \in \mathbb{N}$ e cada $i \in \{1, \ldots, n\}$, a função projeção: $U_i^n = \lambda_{x_1,\ldots,x_n} \cdot x_i$.

> 📖 **As funções básicas são computáveis.**

Composição de Funções

Técnica da Substituição

- Exemplo:

> Seja $f = \lambda_x \cdot 2x$.
>
> Verifica-se que f se obtém por substituição à custa de $soma = \lambda_{x,y} \cdot x + y$ pois, para qualquer $x \in \mathbb{N}_0$, $f(x) = soma(x, x)$.
>
> A função $soma$ é computável, logo a função f é computável.

> 📖 **O método de substituição para definição de novas funções preserva a computabilidade.**

Técnica da Recursão

- Resultado:

1. **Definição recursiva de uma função de vários argumentos:**

Seja $n \in \mathbb{N}_1$ e sejam $f : \mathbb{N}_0^n \not\to \mathbb{N}_0$ e $g : \mathbb{N}_0^{n+2} \not\to \mathbb{N}_0$ funções. Existe uma e só uma função $h : \mathbb{N}_0^{n+1} \not\to \mathbb{N}_0$ que satisfaz as seguitnes equações de recursão:

1. $h(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n)$
2. $h(x_1, \ldots, x_n, y+1) = g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y))$

A função h obtém-se por recursão a partir das funções f e g.

2. **Definição recursiva de uma função unária:**

   Seja $a \in \mathbb{N}_0$ e seja $g : \mathbb{N}_0^2 \not\to \mathbb{N}_0$ uma função. Existe uma e só uma função $h : \mathbb{N}_0 \not\to \mathbb{N}_0$ que satisfaz as equações de recursão:

   1. $h(0) = a$
   2. $h(y+1) = g(y, h(y))$

   A função h obtém-se por recursão a partir da constante a e da função g.

> 📖 **O método de recursão para definição de novas funções preserva a computabilidade.**

- Exemplo:

  Vamos mostrar como se pode definir recursivamente a função $fact = \lambda_x \cdot x!$.

  De acordo com a definição em (2), temos de mostrar que existem $a \in \mathbb{N}_0$ e $g : \mathbb{N}_0^2 \mapsto \mathbb{N}_0$ tais que:

  1. $fact(0) = a$
  2. $fact(y+1) = g(y, fact(y))$

  Temos que:

  a. $fact(0) = 1$ (por convenção)

  b. $fact(y+1) = (y+1) \cdot fact(y) = prod(suc(y), fact(y))$

- Exemplo:

  Vamos mostrar como se pode definir recursivamente a função $soma = \lambda_{x,y} \cdot x + y$.

  De acordo com a definição em (1), temos de mostrar que existem duas funções $f : \mathbb{N}_0 \not\to \mathbb{N}_0$ e $g : \mathbb{N}_0^3 \not\to \mathbb{N}_0$ tais que as seguintes equações de recursão são satisfeitas:

  1. $soma(x, 0) = f(x)$
  2. $soma(x, y+1) = g(x, y, soma(x, y))$

  Temos que:

a. $soma(x, 0) = x = U_1^1(x)$

b. $soma(x, y + 1) = x + (y + 1) = (x + y) + 1 = suc(soma(x, y))$

Logo, podemos concluir que as equações (1) e (2) são satisfeitas quando f é a função $U_1^1(x)$ e g é a função

$$\begin{array}{rccl} g: & \mathbb{N}_0^3 & \to & \mathbb{N}_0 \\ & (a, b, c) & \mapsto & g(a, b, c) = suc(c) \end{array}$$

Função Recursiva Primitiva

a. Uma **função recursiva primitiva**, função $\mathcal{RP}$ é uma função que pode ser obtida por substituição e recursão a partir das funções básicas;

b. Um predicado recursivo primitivo é um predicaod cuja função caraterística C_M é uma função $\mathcal{RP}$;

- As funções básicas são computáveis;
- O método da substituição preserva a computabilidade;
- O método da recursão preserva a computabilidade.

> 📖 **Toda a função $\mathcal{RP}$ é uma função computável.**
>
> Logo, para mostrar que uma função f é computável, basta mostrar que essa função é $\mathcal{RP}$, não havendo necessidade de formular um programa que a calcule.

> 📖 **Toda a função $\mathcal{RP}$ é uma função total.**

- Exemplo:

a. $soma = \lambda_{x,y} \cdot x + y$

Tem-se que:

i. $soma(0, x) = x = U_1^1(x)$

ii. $soma(x, y + 1) = x + (y + 1) = (x + y) + 1 = suc(soma(x, y))$

Portanto, $soma$ obtém-se por recursão e substituição à custa das funções básicas U_1^1 e suc. Logo, $soma$ é $\mathcal{RP}$.

b. $sg = \lambda_x \cdot \begin{cases} 0, & x = 0 \\ 1, & x > 0 \end{cases}$

Tem-se que:

i. $sg(0) = 0$

ii. $sg(y+1) = 1 = 1^{(2)}(y, sg(y))$

Portanto, sg obtém-se por recursão e substituição à custa da constante $0 \in \mathbb{N}_0$ e da função constante $1^{(2)}$. Como as funções constantes podem ser definidas através de funções básicas, $1^{(2)} \in \mathcal{RP}$. Portanto, sg é uma função $\mathcal{RP}$.

c. $\overline{sg} = \lambda_x \cdot \begin{cases} 1, & x = 0 \\ 0, & x > 0 \end{cases}$

Tem-se que:

i. $\overline{sg}(0) = 1^{(1)}(x)$

ii. $\overline{sg}(y+1) = 0^{(2)}(y, sg(y))$

Portanto, $\overline{sg}$ obtém-se por recursão e substituição à custa da constante $0 \in \mathbb{N}_0$ e da função constante $1^{(2)}$. Como as funções constantes podem ser definidas através de funções básicas, $1^{(2)} \in \mathcal{RP}$. Portanto, $\overline{sg}$ é uma função $\mathcal{RP}$.

d. $moddif = \lambda_{x,y} \cdot |x - y|$

Tem-se que:

$$\begin{aligned} & moddif(x, y) \\ & = (x \dot{-} y) + (y \dot{-} x) \\ & = soma(dif(x, y), dif(y, x)) \end{aligned}$$

Logo, $moddif$ obtém-se por substituição e recursão à custa das funções $\mathcal{RP}$ $soma$ e dif.

Álgebra da Decidibilidade (de Predicados) E Definição (de Funções) por Casos

Sejam $M(x_1, \ldots, x_n)$ e $Q(x_1, \ldots, x_n)$ predicados n-ários e sejam C_M, C_Q, $C_{\neg M}$, $C_{M \wedge Q}$, $C_{M \vee Q}$ as funções caraterísticas dos respetivos predicados em subscrito.

a. As funções $C_{\neg M}$, $C_{M \wedge Q}$ e $C_{M \vee Q}$ podem ser obtidas por substituição a partir das funções C_M, C_Q e de funções $\mathcal{RP}$;

b. Se os predicados M e Q são decidíveis, então os predicados $\neg M$, $M \wedge Q$ e $M \vee Q$ também são decidíveis;

c. Se M e Q são predicados $\mathcal{RP}$, então $\neg M$, $M \wedge Q$, $M \vee Q$ também são predicados $\mathcal{RP}$.

Soma Limitada

Sejam $n \in \mathbb{N}_0$ e $f : \mathbb{N}_0^{n+1} \not\to \mathbb{N}_0$.

A soma limitada de f, sl_f, é a seguinte função:

$$\begin{array}{rccc} sl_f : & \mathbb{N}_0^{n+1} & \not\to & \mathbb{N}_0 \\ & (x_1, \ldots, x_n, k) & \mapsto & sl_f(x_1, \ldots, x_n, k) = \sum_{y<k} f(x_1, \ldots, x_n, y) \end{array}$$

onde $\sum_{y<k} f(x_1, \ldots, x_n, y)$ é dada por:

$$\begin{cases} 0, & k = 0 \\ f(x_1, \ldots, x_n, 0) + \cdots + f(x_1, \ldots, x_n, k-1), & k > 0 \end{cases}$$

📖 Se f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável}\\ \text{total}\end{array}\right.$, então sl_f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável}\\ \text{total}\end{array}\right.$.

Produto Limitado

Sejam $n \in \mathbb{N}_0$ e $f : \mathbb{N}_0^{n+1} \not\to \mathbb{N}_0$.

O produto limitado de f, pl_f, é a seguinte função:

$$\begin{array}{rccc} pl_f : & \mathbb{N}_0^{n+1} & \not\to & \mathbb{N}_0 \\ & (x_1, \ldots, x_n, k) & \mapsto & pl_f(x_1, \ldots, x_n, k) = \prod_{y<k} f(x_1, \ldots, x_n, y) \end{array}$$

onde $\prod_{y<k} f(x_1, \ldots, x_n, y)$ é dado por:

$$\begin{cases} 1, & k = 0 \\ f(x_1, \ldots, x_n, 0) \times \cdots \times f(x_1, \ldots, x_n, k-1), & k > 0 \end{cases}$$

📖 Se f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável}\\ \text{total}\end{array}\right.$, então pl_f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável}\\ \text{total}\end{array}\right.$.

Minimização Limitada

Sejam $n \in \mathbb{N}_0$ e $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$. A minimização limitada de f, ml_f, é a seguinte função:

$$\begin{array}{rccc} ml_f: & \mathbb{N}_0^{n+1} & \to & \mathbb{N}_0 \\ & (x_1,\ldots,x_n,k) & \mapsto & ml_f(x_1,\ldots,x_n,k) = \mu_{y<k}(f(x_1,\ldots,x_n,y)=0) \end{array}$$

onde $\mu_{y<k}(f(x_1,\ldots,x_n,y)=0)$ é dado por:

$$\begin{cases} \text{menor } y<k \text{ tal que } f(x_1,\ldots,x_n,y)=0, & \text{se tal y existe} \\ k, & \text{c.c.} \end{cases}$$

📖 Se f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável}\end{array}\right.$, então pl_f é $\left|\begin{array}{l}\text{recursiva primitiva}\\ \text{computável (e total)}\end{array}\right.$.

- Exemplo:

  Seja $f:\mathbb{N}_0^1\to\mathbb{N}_0$ tal que:

$$\begin{array}{lll} f(1,0)=1 & f(1,1)=5 & f(1,2)=4 \\ f(1,3)=0 & f(1,4)=0 & f(1,5)=1 \end{array}$$

De acordo com a definição acima:

- $ml_f(1,2)=\mu_{y<2}(f(1,y)=0)=2$
- $ml_f(1,5)=\mu_{y<5}(f(1,y)=0)=3$

Minimização Limitada de um Predicado

Sejam $n\in\mathbb{N}_0$ e $R(x_1,\ldots,x_n,y)$ um predicado de $n+1$ argumentos. A minimização limitada de R, ml_R, é a seguinte função:

$$\begin{array}{rccc} ml_R: & \mathbb{N}_0^{n+1} & \to & \mathbb{N}_0 \\ & (x_1,\ldots,x_n,k) & \mapsto & ml_R(x_1,\ldots,x_n,k) = \mu_{y<k}(R(x_1,\ldots,x_n,y)=0) \end{array}$$

onde $\mu_{y<k}(f(x_1,\ldots,x_n,y)=0)$ é dado por:

$$\begin{cases} \text{menor } y<k \text{ tal que } R(x_1,\ldots,x_n,y), & \text{se tal y existe} \\ k, & \text{c.c.} \end{cases}$$

Minimização Ilimitada

Sejam $n\in\mathbb{N}_0$ e $f:\mathbb{N}_0^{n+1}\not\to\mathbb{N}_0$. A minimização ilimitada de f, m_f, é a seguinte função:

$$\begin{array}{rccc} m_f: & \mathbb{N}_0^{n} & \to & \mathbb{N}_0 \\ & (x_1,\ldots,x_n) & \mapsto & m_f(x_1,\ldots,x_n) = \mu_y(f(x_1,\ldots,x_n,y)=0) \end{array}$$

onde $\mu_y(f(x_1,\ldots,x_n,y)=0)$ é dado por:

$$\begin{cases} \text{menor } y \in \mathbb{N}_0 \text{ tal que} & \\ \text{i. } f(x_1,\ldots,x_n,z) \text{ está definido para qualquer } z \leq y & \\ \text{ii. } f(x_1,\ldots,x_n,y)=0, & \text{se um tal } y \text{ existe} \\ & \\ -, & \text{c.c.} \end{cases}$$

- Exemplo:

  Seja $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ tal que:

  $$\begin{array}{lll} f(1,0)=1 & f(1,1)=5 & f(1,2)=4 \\ f(1,3)=0 & f(1,4)=0 & f(1,5)=1 \end{array}$$

  Então, pela definiçãoacima:

  $m_f(1)=\mu_y(f(1,y)=0)=3$

  Seja $g : \mathbb{N}_0^2 \to \mathbb{N}_0$ tal que:

  $$\begin{array}{lll} g(1,0)=1 & g(1,1)=- & g(1,2)=4 \\ g(1,3)=0 & g(1,4)=0 & g(1,5)=1 \end{array}$$

  Então, pela definição:

  $m_g(1)=\mu_y(f(1,y)=0)=-$

  pois $g(1,3)=0$, mas $g(1,1)$ não está definido como necessário em (i) na definição.

> 📖 **O método da minimização para definição de novas funções preserva a computabilidade.**

Minimização Ilimitada de um Predicado

Sejam $n \in \mathbb{N}_0$ e $R(x_1,\ldots,x_n,y)$ um predicado de $n+1$ argumentos. A minimização ilimitada de R, m_R, é a seguinte função:

$$\begin{array}{rccc} m_R: & \mathbb{N}_0^n & \to & \mathbb{N}_0 \\ & (x_1,\ldots,x_n) & \mapsto & m_f(x_1,\ldots,x_n)=\mu_y(R(x_1,\ldots,x_n,y)) \end{array}$$

onde $\mu_y(R(x_1,\ldots,x_n,y))$ é dado por:

$$\begin{cases} \text{menor } y \text{ tal que } R(x_1, \ldots, x_n, y), & \text{se um tal } y \text{ existe} \\ -, & \text{c.c.} \end{cases}$$

- Exemplo:

  Consideremos a função

  $$\begin{array}{rccc} RaizInt: & \mathbb{N}_0 & \not\to & \mathbb{N}_0 \\ & x & \mapsto & RaizInt(x) = [\sqrt{x}] \end{array}$$

  onde $[x]$ denota o maior natural cujo quadrado é menor ou igual a $x \in \mathbb{N}_0$.

  Pode-se obter $RaizInt$ através da minimização de um predicado. Para qualquer $x \in \mathbb{N}_0$,

  $$RaizInt(x) = u_y((y+1)^2 > x) = m_R(x)$$

  onde R é o predicado binário

  $$R(x, y) = (y+1)^2 > x$$

  De acordo com o resultado, para mostrar que $RaizInt$ é computável, basta mostrar que R é decidível.

  Seja, então, $C_R : \mathbb{N}_0^2 \to 0$. Tem-se que $\forall_{(x,y)} \in \mathbb{N}_0^2$,

  $$C_R(x, y) = C_>((y+1)^2, x) = C > (prod(suc(y), suc(y)), x)$$

  C_R obtém-se por substituição a partir das funções $C_>$, $prod$ e suc. Todas estas funções são computáveis, e visto que o método da substituição preserva a computabilidade, C_R é computável. Logo, R é decidível, e como $RaizInt = m_R$, $RaizInt$ é computável.

Funções Parciais Recursivas

- $\mathcal{C}$: classe das funções parciais URM-computáveis;
- $\mathcal{RP}$: classe das funções recursivas primitivas (funções geradas a partir das funções básicas usando as técnicas da substituição e recursão):
  - Estas funções são totais porque as funções básicas e as (novas) funções que são definidas por substituição e/ou recursão a partir de funções totais são também totais.
- $\mathcal{R}$: classe das funções **funções parciais recursivas:**

    - Uma função $f \in \mathcal{R}$ é uma função que pode ser obtida a partir das funções básicas usando as técnicas da substituição, recursão e minimização ilimitada.
- $\mathcal{R}_{\mathcal{T}}$: classe das **funções totais recursivas:**
    - Uma função $f \in \mathcal{R}_{\mathcal{T}}$ é uma função de $\mathcal{R}$ que e total.

Resultado: Inclusão de Conjuntos

$$\mathcal{RP} \subseteq \mathcal{R}_{\mathcal{T}} \subseteq \mathcal{R} \subseteq \mathcal{C}$$

$$\mathcal{RT} \subset \mathcal{R}$$

Resultado: Igualdade $\mathcal{R} = \mathcal{C}$

$$\mathcal{R} = \mathcal{C}$$

> 📖 Uma qualquer função é computável se e só se é uma função parcial recursiva.

Máquina de Turing e Funções Turing-computáveis

Máquinas de Turing

- Uma máquina de Turing Mrealiza operações sobre uma fita de papel infinita em ambas as direções e está dividida em células ao longo de todo o seu comprimento.
- Em qualquer instante, uma célula da fita está em branco (i.e. preenchida com o **símbolo** β) ou contém um único símbolo de um conjunto finito de símbolos $\{s_0, s_1, \ldots, s_n\}$ (**alfabeto de** M).

Instruções

a. Escrever um símbolo do seu alfabeto na célula que está a ser lida;

b. Mover a cabeça de leitura uma célula para a direita daquela que está a ser lida.

c. Mover a cabeça de leitura uma célula para a esquerda daquela que está a ser lida.

Em cada momento, M está num certo estado de conjunto finito de estados, representados por $\{q_1, q_2, \ldots, q_n\}$.

A ação a ser executada por M depende do estado atual de Me do símbolo que está a ser lido no momento.

Um quádruplo $q_i\ s_j\ \alpha\ q_l$ (onde $\alpha \in \{s_k, R, L\}$) em Q (conjunto finito de quadruplos) **especifica** a ação a ser **executada** por M quando está **no estado** q_i e a ler o símbolo s_i**:**

1. Executar a seguinte operação na fita:
    a. Se $\alpha = s_k$, apagar s_j e escrever s_k na célula que está a ser lida no momento;
    b. Se $\alpha = R$, mover a cabeça de leitura uma célula para a direita;
    c. Se $\alpha = L$, mover a cabeça de leitura uma célula para a esquerda;
2. Mudar o estado para q_l.

Convenção

- Dada uma máquina de Turing M e uma fita de papel infinita:
  - M inicia a sua execução no estado q_1;
  - A cabeça de leitura está na célula não em branco (i.e. para a i-ésima célula mais à esquerda, $s_i \neq \beta$);

Exemplo de Execução

Seja Q a seguinte especificação:

$$
\begin{array}{l}
q_1\ a\ b\ q_2 \\
q_1\ b\ a\ q_2 \\
q_2\ a\ R\ q_1 \\
q_2\ b\ R\ q_1
\end{array}
$$

A configuração inicial é a seguinte:

$$\ldots \mid \beta \mid \beta \mid {}'a' \mid b \mid b \mid \beta \mid \beta \mid \ldots$$

A execução é a seguinte ($'s_i'$ representa a posição da cabeçade leitura da máquina URM):

$$\ldots \mid \beta \mid \beta \mid {}'b'[q_2] \mid b \mid b \mid \beta \mid \beta \mid \ldots$$

$$\ldots \mid \beta \mid \beta \mid b \mid {}'b'[q_1] \mid b \mid \beta \mid \beta \mid \ldots$$

$$\ldots \mid \beta \mid \beta \mid b \mid {}'a'[q_2] \mid b \mid \beta \mid \beta \mid \ldots$$

$$\ldots \mid \beta \mid \beta \mid b \mid a \mid {}'b'[q_1] \mid \beta \mid \beta \mid \ldots$$

$$\ldots \mid \beta \mid \beta \mid b \mid a \mid {}'a'[q_2] \mid \beta \mid \beta \mid \ldots$$

$$\ldots \mid \beta \mid \beta \mid b \mid a \mid a \mid {}'\beta'[q_1] \mid \beta \mid \ldots \ \text{STOP}$$

- A execução da máquina M termina porque não existe, na especificação Q, nenhum quadruplo $q_1\ \beta\ \{s_k, R, L\}\ q_l$

Funções Turing-computáveis

Representação de Números

Número	Representação
0	$\ldots \mid \beta \mid 1 \mid \beta \mid \ldots$
1	$\ldots \mid \beta \mid 1 \mid 1 \mid \beta \mid \ldots$
2	$\ldots \mid \beta \mid 1 \mid 1 \mid 1 \mid \beta \mid \ldots$
x	$\ldots \mid \beta \mid \underbrace{x \mid \ldots \mid x}_{x+1 \text{ vezes}} \mid \beta \mid \ldots \mid$

Cálculo de Função Unária

Seja $f : \mathbb{N}_0 \nrightarrow \mathbb{N}_0$.

Consideremos a execução de uma máquina M sobre uma fita com a seguinte configuração inicial:

$$\ldots \mid \beta \mid '1' \mid \ldots \mid 1 \mid \beta \mid \ldots$$

Esta configuração tem exatamente $x + 1$ células consecutivas com o símbolo 1 e todos as restantes células estão em branco. M inicia-se no estado q_1 na célula contendo o 1 mais à esquerda. Então,

$$f(x) = \begin{cases} \text{número total de ocorrências} \\ \text{do símbolo 1 na configuração final}, & \text{se a execução termina} \\ -, & \text{caso contrário} \end{cases}$$

Exemplo de Execução

Seja M uma máquina de Turing e seja $f : \mathbb{N}_0 \nrightarrow \mathbb{N}_0$ a função unária calculada por M. Se a execução de M inicia com configuração inicial

$$\ldots \mid \beta \mid '1' \mid 1 \mid 1 \mid \beta \mid \ldots$$

e termina com a configuração final

$$\ldots \mid \beta \mid 1 \mid \beta \mid 1 \mid 1 \mid \beta \mid 1 \mid \beta \ldots$$

então tem-se que $f(2) = 4$.

Cálculo de Função *n*-ária

- Um *n*-uplo $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$ é representado numa fita pelas sequencias de 1s consecutivos que representam cada uma das componentes de $(x_1, \ldots, x_n)$, estando duas dessas sequencias separadas, **exatamente**, por uma célula em branco (e.g. preenchida com β):

$$\ldots \mid \beta \mid 1 \mid \ldots \mid 1 \mid \beta \mid 1 \mid \ldots \mid 1 \mid \beta \mid 1 \mid \ldots \mid 1 \mid \beta \mid \text{...}$$

- Exemplo:
  - O triplo $(1, 0, 2)$ representa-se:

$$\ldots \mid \beta \mid 1 \mid 1 \mid \beta \mid 1 \mid \beta \mid 1 \mid 1 \mid 1 \mid \beta \mid \ldots$$

Exemplo de Execução

Seja M uma máquina de Turing e seja $f : \mathbb{N}_0^2 \nrightarrow \mathbb{N}_0$ a função calculada por M. Se a execução de M a partir da configuração inicial

$$\ldots \mid \beta \mid {}'1' \mid 1 \mid 1 \mid \beta \mid 1 \mid 1 \mid \beta \mid \ldots$$

termina com a configuração final

$$\ldots \mid \beta \mid 1 \mid \beta \mid 1 \mid 1 \mid \beta \mid 1 \mid \beta \mid 1 \quad \beta \quad \ldots$$

então $f(2, 1) = 5$.

Função Turing-computável

Uma função $f : \mathbb{N}_0^n \nrightarrow \mathbb{N}_0$, com $n \geq 1$, é Turing-computável ($f \in \mathcal{T}$) se e só se existe uma máquina de Turing que calcula f.

Resultado Fundamental

Três formas de formalizar a noção de função computável:

- Funções URM-computáveis, $\mathcal{C}$;
- Funções parcias recursivas, $\mathcal{R}$;
- Funções Turing-computáveis, $\mathcal{T}$.

Todas as formalizações para caraterizar a noção de uma função computável, até hoje propostas,. originam na mesma classe de funções. Logo,

$$\mathcal{C} = \mathcal{R} = \mathcal{T} = \ldots$$