4. Numeração de funções computáveis

4.1 Numeração de programas

Definições

- Um conjunto X é **enumerável** se existe uma bijecção f : X $\rightarrow$ |N.
- Uma **enumeração** de um conjunto X é uma sobrejecção g: |N $\rightarrow$ X ;
  é representada por X = {x_0, x_1, x_2,…} , onde x_n = g(n).
  Se g é injectiva, a enumeração não tem repetições.
- Seja X um conjunto de objectos finitos (por ex: conjunto de inteiros, instruções, programas). X é **efectivamente enumerável** se existe uma bijecção f: X $\nrightarrow$ |N tal que f e f^{-1} são funções efectivamente computáveis.

Um conjunto é enumerável se e só se pode ser enumerado sem repetições

4.1 Numeração de programas

Teorema

Os seguintes conjuntos são efectivamente enumeráveis:

(a) $|N \times |N$;

(b) $|N^+ \times |N^+ \times |N^+$;

(c) $\bigcup_{k>0} |N^k$, o conjunto de todas as sequências finitas de números naturais.

Notação útil:

p_x designa o x-ésimo numero primo.

(Por convenção, $p_0 = 0$, logo, $p_1 = 2$, $p_2 = 3$, etc)

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

4.1 Numeração de programas

Prova:

(a) |N x |N é efectivamente numerável (?)

Uma bijecção $p : |\mathrm{N} \times |\mathrm{N} \longrightarrow |N$ é definida por:

$$p\ (\mathrm{m},\mathrm{n}) = 2^{\mathrm{m}}(2\mathrm{n}+1) - 1$$

p é uma função efectivamente computável.

p^{-1} é efectivamente computável, pois :

$$p^{-1}(\mathrm{x}) = (p_1(x),\ p_2(x))$$

$$p_1(x) = (\mathrm{x}+1)_1$$

$$p_2(x) = \frac{1}{2}\left(\frac{\mathrm{x}+1}{2^{p_1(x)}} - 1\right)$$

e p_1 e p_2 são funções computáveis.

4.1 Numeração de programas

(b) $|N^+ \times |N^+ \times |N^+$ é efectivamente enumerável (?)

Uma bijecção $z : |N^+ \times |N^+ \times |N^+ \longrightarrow |N$ é definida por:

$$z\ (m, n, q) = p\ (p\ (m-1, n-1),\ q-1)$$

e

$$z^{-1}(x) = (p_1(p_1(x)) + 1,\ p_2(p_2(x)) + 1,\ p_2(x) + 1)$$

Como as funções p , p_1 e p_2 são efectivamente computáveis, também z e z^{-1} são computáveis.

4.1 Numeração de programas

(c) $\bigcup_{k>0} |N^k$ é efectivamente enumerável (?)

Uma bijecção $t : \bigcup_{k>0} |N^k \longrightarrow |N$ é definida por:

$$t\ (a_1, a_2, \ldots, a_k) = 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots + 2^{a_1+a_2+\ldots+a_k+k-1} - 1$$

é uma função efectivamente computável.

Para calcular $t^{-1}(x)$, sabemos que dado x, podemos efectivamente encontrar valores únicos k > 0 e $0 \le b_1 < b_2 < \ldots < b_k$ tais que:

$$x + 1 = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_k}$$

Então $t^{-1}(x) = (a_1, a_2, \ldots, a_k)$, onde $a_1 = b_1$

$$a_{i+1} = b_{i+1} - b_i - 1 \ (1 \le i < k)$$

4.1 Numeração de programas

Designemos o conjunto de todas as instruções URM por I e o conjunto de todos os programas URM por P. Um programa consiste numa lista finita de instruções.

Teorema

I é efectivamente enumerável.

Prova: Defina-se uma bijecção $b : I \longrightarrow |N$ que, aos quatro tipos de instruções URM faz corresponder números naturais das formas **4u**, **4u+1**, **4u+2**, **4u+3**:

$$b(Z(n)) = 4(n-1)$$

$$b(S(n)) = 4(n-1)+1$$

$$b(T(m,n)) = 4\ p\ (m-1, n-1) + 2$$

$$b(J(m,n,q)) = 4\ z\ (m,n,q) + 3$$

4.1 Numeração de programas

Para calcular $b^{-1}(x)$:

(i) Determinar **u**, **r** tais que **x = 4 u + r**, com $0 \leq r < 4$

(ii) O valor de **r** diz-nos que tipo de instrução corresponde a

(iii) A partir de **u**, podemos calcular efectivamente a instrução particular:

Se **r = 0**, então $b^{-1}(x) = Z(u+1)$

Se **r = 1**, então $b^{-1}(x) = S(u+1)$

Se **r = 2**, então $b^{-1}(x) = T(p_1(u)+1,\ p_2(u)+1)$

Se **r = 3**, então $b^{-1}(x) = J(m,n,q)$, onde $(m,n,q) = z^{-1}(u)$

b e b^{-1} são efectivamente computáveis, logo I é efectivamente enumerável.

4.1 Numeração de programas

Teorema

P é efectivamente enumerável.

Prova: Defina-se uma bijecção $g : \mathrm{P} \longrightarrow |\mathrm{N}$. Seja **P = I_1, I_2, …, I_s**.

Então $g(P) = t\ (\ b\ (\mathrm{I}_1),\ldots,\ b\ (\mathrm{I}_s))$

Como t e b são bijecções, também g é uma bijecção;

Como t, b e as suas inversas são efectivamente computáveis, também g e g^{-1} são efectivamente computáveis.

4.1 Numeração de programas

Para um dado programa P,

o número g (P) chama-se o ***código*** de P ou ***número de Gödel*** de P.

Define-se P_n = o programa com o código n

= $g^{-1}(n)$

É fundamental que g e g^{-1} sejam efectivamente computáveis, ou seja:

(a) dado um programa P, podemos efectivamente calcular o seu código;

(b) dado um número n, podemos efectivamente determinar o programa $P_n = g^{-1}(n)$.

4.1 Numeração de programas

Exemplos:

(a) Seja P o programa T(1,3)
S(4)
Z(6)

Para calcular g (P):

$$b(\mathrm{T}(1,3)) = 4\,p\,(0,\,2) + 2 = 4\,(2^{0}(2\times 2+1)-1) + 2 = 18$$

$$b(\mathrm{S}(4)) = 4 \times 3 + 1 = 13$$

$$b(\mathrm{Z}(6)) = 4 \times 5 = 20$$

$$g(\mathrm{P}) = t(18,\,13,\,20) = 2^{18} + 2^{32} + 2^{53} - 1$$

$$= 9007203549970431$$

4.1 Numeração de programas

(b) Seja n = 4127. Qual é o programa P_{4127} ?

$g^{-1}(4127) = t^{-1}(4128) = (a_1,...,a_k)$

4127 = 2^5+2^{12} -1

(4128 = 2^5 (129) = 2^5 (1+128) = 2^5 (1+ 2^7) = **2^5+2^{12}**)

Então, **P_{4127}** é um programa com duas instruções (k=2): **I_1** e **I_2**

a_1 = 5 e **a_2** = 12 - 5 - 1 = 6

$a_1 = b\ (I_1) = 5$

$a_2 = b\ (I_2) = 6 = 4 \times 1 + 2 = 4\ p\ (1,0) + 2$

Pela definição, **I_1** = S(2) e **I_2** = T(2,1).

P_{4127} é o programa	**I_1 = S(2)** **I_2 = T(2,1).**

Mais conjuntos enumeráveis

Q = O conjunto dos números racionais positivos é enumerável

1/1	1/1	3/1	4/1	5/1	…
1/2	2/2	3/2	4/2	5/2	…
1/3	2/3	3/3	4/3	5/3	…
1/4	2/4	3/4	4/4	5/4	…
…	…	…	…	…	…

D = O conjunto de todos os conjuntos finitos de números inteiros positivos é enumerável

{1} : o conjunto cujo único elemento é o número 1

{2} : o conjunto cujo único elemento é o número 2

{1,2,5} : o conjunto cujos elementos são os números 1, 2 e 5

Todos os elementos de **D** podem ser descritos de forma finita (até por palavras), embora o conjunto seja infinito.

Conjuntos não enumeráveis

O conjunto dos números reais não é enumerável
Consideremos apenas os números reais entre 0 e 1. Suponhamos que conseguimos estabelecer uma correspondência entre cada número natural e cada número real entre 0 e 1.
Suponhamos ainda que cada número real pode ser expresso como uma dízima infinita, ou seja:
0.5 = 0.4999999……
Podemos então construir a seguinte tabela:

r(1)	**.1**	4	1	5	9	2	6	…
r(2)	.3	**3**	3	3	3	3	3	…
r(3)	.7	1	**8**	2	8	1	8	…
r(4)	.4	1	4	**2**	1	3	5	…
r(5)	.5	0	0	0	**0**	0	0	…
…	…	…	…	…	…	…	…	…

Os números que se encontram na diagonal serão usados para construir um número que não se encontra na tabela.
Subtraímos 1 a cada dígito do número da diagonal.
Obtemos **d = 0.02719**.
Dada a forma como a tabela foi construída podemos afirmar que :
0 1º dígito de **d** não é o 1º dígito de r(1);
0 2º dígito de **d** não é o 2º dígito de r(2);
…
0 20º dígito de **d** não é o 20º dígito de r(20); **d não se encontra na tabela, logo |R não é enumerável**

4.2 Numeração de funções computáveis

$f_P^{(n)}$ designa a função n-ária computada por P. Para cada **a** de |N e para cada **n > 0**, define-se:

$$f_a^{(n)} = f_{P_a}^{(n)}$$

$$W_a^{(n)} = \text{Dom}\,(f_a^{(n)}) = \{(x_1,...,x_n):\ \text{P}_\text{a}(x_1,...,x_n)\ \downarrow\ \}$$

$$E_a^{(n)} = \text{Ran}\,(f_a^{(n)})$$

Quando a função é unária, omite-se o índice n =1, ficando apenas f_a , W_a , E_a .

Ex: P_{4127} é o programa $I_1 = S(2)$
$I_2 = T(2,1)$

Então

$$f_{4127}(x) = 1 \qquad f_{4127}^{n}(x_1,...,x_n) = x_2 + 1$$

$$W_{4127} = |N \qquad W_{4127}^{n} = |N^{n}$$

$$E_{4127} = \{1\} \qquad E_{4127}^{n} = |N^{+}$$

4.2 Numeração de funções computáveis

Seja ***f*** uma função unária computável. Então existe um programa **P** que computa ***f***.

Sendo $a = g(P)$ o código desse programa, então $f = f_a$.

Diz-se que **a é um índice** para f.
(Cada função computável tem um número infinito de índices).

Podemos concluir que cada função unária aparece na enumeração (com repetições):

$$f_0,\ f_1,\ f_2,\ f_3,\ldots$$

As mesmas conclusões podem ser tiradas para funções n-árias.

4.2 Numeração de funções computáveis

Teorema

O conjunto C_n é enumerável.

Prova: Vamos usar uma enumeração com repetições para construir uma outra sem repetições.

Seja $f_0^{(n)},\ f_1^{(n)},\ f_2^{(n)},\ f_3^{(n)},\ldots$ a enumeração com repetições.

Seja $\begin{cases} f(0) = 0 \\ f(m+1) = m\ z(f_z^{(n)} \neq f_{f(0)}^{(n)},\ldots,f_{f(m)}^{(n)}) \end{cases}$

Então $f_{f(0)}^{(n)},\ f_{f(1)}^{(n)},\ f_{f(2)}^{(n)},\ f_{f(3)}^{(n)},\ldots$ é uma enumeração de C_n sem repetições.

Nota: Não se afirma que f é computável.

4.2 Numeração de funções computáveis

Teorema

O conjunto C é enumerável.

Prova: $C = \bigcup_{n \geq 1} C_n$

C é a reunião enumerável de conjuntos enumeráveis, logo também é enumerável.

O teorema seguinte mostra que existem funções que não são computáveis !!

4.3 O método da diagonal

Teorema

Existe uma função unária total que não é computável.

Prova:
Vamos construir uma função total f que é simultaneamente diferente de todas as funções na enumeração $f_0, f_1, f_2, f_3, \ldots$

Defina-se $$f(n) = \begin{cases} f_n(n) + 1, & \text{se } f_n(n) \text{ esta definida} \\ 0, & \text{se } f_n(n) \text{ nao esta definida} \end{cases}$$

Note-se que *f* foi construída por forma que *f* é diferente de f_n para cada n:

- Se f_n está definida, $f(n) = f_n(n) + 1 \neq f_n(n)$

- Se f_n não está definida: $f(n) = 0 \neq f_n(n)$

Como *f* é diferente de todas as funções computáveis, ***f* não é computável**.

4.3 O método da diagonal

Vejamos como a função anterior foi construída: $f(n) = \begin{cases} f_n(n)+1, & \text{se } f_n(n) \text{ esta definida} \\ 0, & \text{se } f_n(n) \text{ nao esta definida} \end{cases}$

Construímos a seguinte tabela:

	0	1	2	3	…
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	…
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	…
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	…
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	…
…	…	…	…	…	…

Tomamos os elementos da diagonal e alteramo-los sucessivamente, construindo f(n).

Nota: existe uma grande liberdade na escolha dos f(n);
apenas temos de assegurar que $f(n) \neq f_n(n)$ para cada n .

Por exemplo: $g(n) = \begin{cases} f_n(n) + 27^n, & \text{se } f_n(n) \text{ esta definida} \\ n^2, & \text{se } f_n(n) \text{ nao esta definida} \end{cases}$ é outra função não computável

4.3 O método da diagonal

Podemos generalizar o método da diagonal da seguinte forma:

Seja $c_0, c_1, c_2, \ldots$ uma enumeração de objectos de um certo tipo
(funções ou conjuntos de números naturais)

Podemos construir um objecto c do mesmo tipo, mas diferente de todos os c_n

"Fazer c e c_n diferentes em n"

4.3 O método da diagonal

Exemplos:

(a) Seja ***f(x,y)*** uma função total computável.
Para cada **m**, seja g_m a função computável dada por $g_m(y) = f(m,y)$
Construa uma função total computável **h** tal que, para cada **m**: $h \neq g_m$

$g_m(m) = f(m,m)$

Seja **h(m) = f(m,m)+1**

h é computável (pois f é computável) e total

(b) Seja $f_0, f_1, f_2, \ldots$ uma enumeração de funções parciais de |N em |N.
Construa uma função **g: |N -> |N** tal que, para cada ***i***: $Dom(g) \neq Dom(f_i)$

$$g(i) = \begin{cases} 1 & \text{, se } f_i(i) \text{ nao esta definida} \\ \text{indefinida} & \text{, se } f_i(i) \text{ esta definida} \end{cases}$$

ou seja, $i \in Dom(g)$ sse $i \notin Dom(f_i)$

4.4 O Teorema s-m-n

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

Então, ***f*** tem um índice, ***a***, tal que $f_a^{(2)} = f$

Para cada ***b***, defina-se $g_b\ (y) = f\ (b,y)$

Por substituição, cada g_b é computável, logo tem um índice.

Este índice depende de ***a*** e de ***b***, logo, defina-se a função binária ***k***, onde ***k(a,b)*** é um índice para a função g_b . Logo, $f_{k(a,b)}\ (y) = g_b\ (y)$

Mostraremos que ***k*** é uma função computável.

Sabemos que a função ***f*** é computada por P_a. Considere-se:

T(1,2)
Z(1)
S(1) ↕
… **b vezes**
S(1) ↕
P_a

Este programa computa g_b

4.4 O Teorema s-m-n

A função ***k(a,b)*** dá-nos o índice deste programa.

Para mostrar que ***k*** é computável usaremos a Tese de Church, argumentando que o algoritmo:

1. Determinar qual o programa que corresponde ao código ***a***, obtendo-se P_a

2. Construir o programa para computar g_b (depende apenas de ***b***)

3. Codificar esse programa, obtendo-se ***k(a,b)***.

é efectivamente computável, logo ***k*** é computável.

Notas:
- ***k*** é primitiva recursiva.
- Sem o recurso à Tese de Church, esta demonstração seria longa e tediosa, mas é óbvio que funcionaria.
- Podemos agora construir funções que computam índices de funções.

4.4 O Teorema s-m-n

Exemplo: Seja $f(x,y) = y^x$

Esta função é computável, logo seja a um índice para f : $f_a^{(2)} = f$

Para cada *n*, defina-se $g_n(y) = f(n,y) = y^n$

Seja k(a,n) o índice de g_n

Então $f_{k(a,n)}(y) = g_n(y) = y^n$

Seja ***p(n) = k(a,n)***.

p é a função que, para cada ***n*** calcula um índice da função **"n-ésima potência"**.

4.4 O Teorema s-m-n

Teorema s-m-n

Para cada m, n > 0, existe uma função (m+1)-ária total e computável $s_n^m(\mathrm{e},\mathbf{x})$

tal que $$f_e^{(m+n)}(\mathbf{x},\mathbf{y}) = f_{s_n^m(e,\mathbf{x})}^{(n)}(\mathbf{y})$$

Prova: (semelhante ao caso m=n=1)

Designe-se por **Q(i,x)** o programa URM:

Z(i)
S(i) ↕ x vezes
…
S(i)

Defina-se $s_n^m(\mathrm{e},\mathbf{x})$ como o código do programa:

T(n,m+n)
…
T(1,m+1)
Q(1,x_1)
…
Q(m,x_m)
P_e

↓

Esta função é computável, pela Tese de Church