

Curso de Lógica Matemática

Teoría de la Computabilidad

Cristo Daniel Alvarado

4 de diciembre de 2024

Índice general

3. Conjuntos y Funciones computables	2
3.1. Máquinas de Turing	2
3.2. Máquina Universal de Turing	21
3.3. Conjuntos y Relaciones Computables	23
3.4. Conjuntos computablemente numerables	25
4. Teoremas de Completud	31
4.1. Hilbert	31
4.2. Introducción	31

Capítulo 3

Conjuntos y Funciones computables

Todo de lo que se va a tratar esta parte es de: ¿Cómo formalizar la noción de *procedimiento mecánico, efectivo o sistemático*? Con esto nos referimos a:

- Tener un número finito de instrucciones.
- Terminar el procedimiento en un número finito de pasos.
- Usar únicamente *papel y lápiz*.
- No requiere razonamiento, solo se siguen reglas.

Básicamente se pretendía que dada una fórmula, encontrar un algoritmo que nos diga si esa fórmula es verdadera o falsa. Básicamente se pretendía formalizar las demostraciones para ver lo que nosotros podemos demostrar únicamente usando los axiomas.

Turing y Alonzo Church eventualmente se hicieron preguntas en la misma dirección. En la Tesis de Church-Turing se probó que estas tres preguntas en realidad se reducen a un mismo problema.

3.1. Máquinas de Turing

Definición 3.1.1

Una **máquina de Turing** consta de:

- Un *alfabeto*, un conjunto finito L .
- Un conjunto finito S de *estados*.
- Una función parcial $T : L^* \times S \rightarrow L^* \times S \times \{<, -, >\}$ llamada *función de transición*.

donde $L^* = L \cup \{*\}$.

Intuitivamente, uno debe imaginar que esto es una especie de *computadora rudimentaria*. Generalmente esto se conceptualiza como una cinta.

El cabezal c puede moverse a la derecha, izquierda o no moverse, dependiendo del estado en el que esté. En la Figura 3.1 se muestra que el hay al menos 5 diferentes estados, desde el estado inicial (s_i) hasta el final (s_f). Dependiendo de la entrada, la función T nos dirá lo que hará el cabezal, si cambia un elemento de la banda, si se mueve o si cambia de estado (o todas a la vez).

En este ejemplo, el alfabeto sería $L = \{0, 1\}$, el conjunto de estados es $S = \{s_i, s_1, \dots, s_f\}$ y la función sería representada por lo que sea que haga el cabezal.

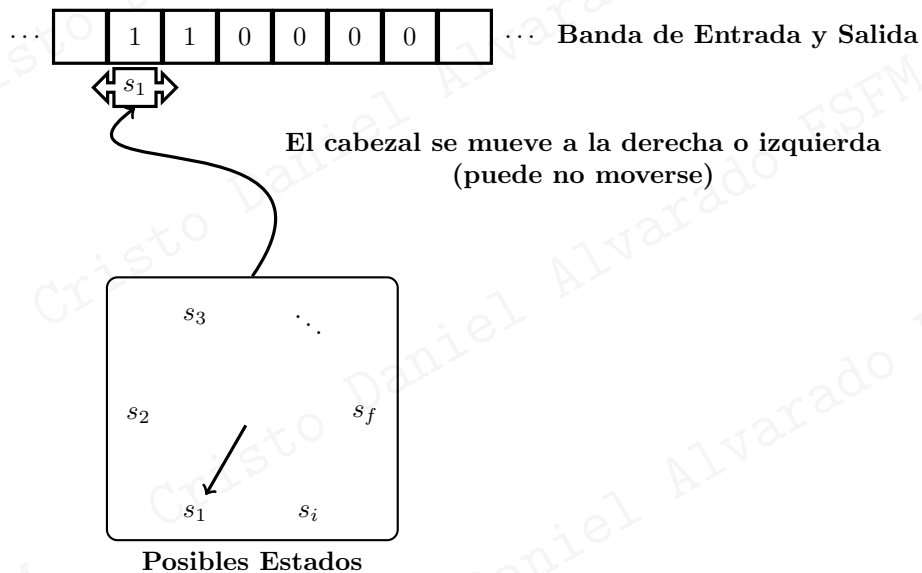


Figura 3.1: Ejemplo de Máquina de Turing

Ejemplo 3.1.1

Considere $L = \{1\}$, $S = \{s_i, s_1, s_2\}$ y,

$$T = \{(s_i, *, s_1, *, >), (s_i, 1, s_1, 1, >), (s_1, 1, s_1, 1, >), (s_1, 1, s_2, 1, -)\}$$

La cinta se ve más o menos así:

Para los siguientes ejercicios, ir a la página: [Simulador Máquina de Turing](#).

Ejercicio 3.1.1

Codifique una máquina de Turing que sume 1 a un número dado en binario.

```

1      name: Sumar uno en unario
2      init: s0
3      accept: sf
4
5
6      // Funciones de Transicion
7
8      s0, _
9      s0, _, >
10
11     s0, 1
12     s1, 1, -
13
14     s0, 0
15     s1, 0, -
16
17     s1, 1
18     s1, 0, >
19
20     s1, 0
21     s1, 1, >

```

```

22
23     s1,_
24     sf,_, -
25
26     // < = left
27     // > = right
28     // - = hold
29     // use _ for blank cells
30
31     // States and symbols are case-sensitive
32
33     // Load your code and click COMPILE.
34     // or load an example (top-right).

```

Ejercicio 3.1.2

Codifique una máquina de Turing que dada un número en binario, invierta su orientación, es decir, si la cadena es (a_1, \dots, a_n) , que la máquina de Turing la convierta en (a_n, \dots, a_1) .

```

1      name: invertirCadena
2      init: s0
3      accept: s1,sf,l,c,u
4
5      //esto para que se empiece a mover
6      s0,_
7      s0,_,>
8
9      s0,0
10     x,0,<
11
12     s0,1
13     x,1,<
14
15     x,_
16     s1,2,>
17
18     s1,0
19     s1,0,>
20
21     s1,1
22     s1,1,>
23
24     //logica cuando encuentre cosas
25
26     s1,_
27     s2,_,<
28
29     s2,_
30     s2,_,<
31
32     s2,0
33     c00,_,>

```

```

34
35     s2,1
36     u00,_,>
37
38     //mueve cosas al inicio
39
40     c00, _
41     m,0,<
42
43     u00, _
44     m,1,<
45
46     //ya en ciclo
47
48     //mueve derecha
49
50     m, _
51     l,_,<
52
53     l, _
54     l,_,<
55
56     l,0
57     c0,_,>
58
59     l,1
60     u0,_,>
61
62     c0, _
63     c0,_,>
64
65     //mueve izquierda
66
67     u0, _
68     u0,_,>
69
70     c0,0
71     c1,0,>
72
73     c0,1
74     c1,1,>
75
76     u0,0
77     u1,0,>
78
79     u0,1
80     u1,1,>
81
82     c1,0
83     c1,0,>
84
85     c1,1

```

```

86      c1,1,>
87
88      u1,0
89      u1,0,>
90
91      u1,1
92      u1,1,>
93
94      c1,_
95      m,0,<
96
97      u1,_
98      m,1,<
99
100     m,0
101     m,0,<
102
103     m,1
104     m,1,<
105
106     l,2
107     sf,_,>
108
109     sf,_
110     sf,_,>
111
112     sf,0
113     sff,0,-
114
115     sf,1
116     sff,1,-

```

Definición 3.1.2

Una función f es **computable** si:

- (1) $\text{dom}(f) \subseteq \mathbb{N}$.
- (2) Existe un algoritmo tal que para cada $n \in \mathbb{N}$, el algoritmo al correrse con n como argumento, se detiene en tiempo finito si y sólo si $n \in \text{dom}(f)$ y en tal caso arroja $f(n)$ como salida.

¿Qué es un algoritmo? Resulta que hay muchas formas de definirlo, sin embargo, nosotros adoptaremos la siguiente definición:

Definición 3.1.3

Un **algoritmo** lo interpretaremos como una máquina de Turing.

Observación 3.1.1

Un algoritmo también puede verse como un código en C, C++, Python o \LaTeX (usando las librerías adecuadas).

En la Tesis de Church-Turing, cualquier noción es equivalente.

Observación 3.1.2

De ahora en adelante consideraremos a los naturales con el 0.

Ejemplo 3.1.2

La función $f : \mathbb{N} \setminus \{0, 1\} \rightarrow \mathbb{N}$ tal que $n \mapsto \min \{p \in \mathbb{N} \mid p \text{ es primo y } p \mid n\}$ es computable.

Demostración:

Se tiene el siguiente algoritmo:

```
1 int f(int n){
2     for(int k = 2; n % k != 0; k++) return k;
3 }
```

■

Ejemplo 3.1.3

Considere la función $g : \{n^2 \mid n \in \mathbb{N}\} \rightarrow \mathbb{N}$ dada por $n^2 \mapsto n$. Esta función es computable.

Demostración:

Se tiene el siguiente algoritmo:

```
1 int g(int m){
2     for(int k = 0; k*k != m; k++) return k;
3 }
```

■

Ejemplo 3.1.4

La función $+$: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ es computable.

Demostración:

Recordemos que existe una biyección entre $\mathbb{N} \times \mathbb{N}$ y \mathbb{N} dada por:

$$(k, l) \mapsto 2^k(2l + 1)$$

por lo cual, podemos ver a la función suma como una función de \mathbb{N} en \mathbb{N} .

■

Observación 3.1.3

Podemos ir más allá en el ejemplo anterior, podemos generalizar la idea anterior usando conjuntos que puedan ser representados mediante números naturales (recuerde la enumeración de Gödel).

Veremos más ejemplos que nos ayudarán más adelante a hacer cosas más complejas:

- La función sucesor (se vió en un ejercicio anterior).
- Cualquier función constante.
- La i -ésima proyección de una k -tupla.


```

1 int p_2(int a, int b, int c){
2     return b;
3 }

```

este ejemplo anterior es la 2-ésima proyección de una 3-tupla.

Ejemplo 3.1.5

La función exponencial: $(a, b) \mapsto a^b$ es computable.

Demostración:

En efecto, se tiene el siguiente algoritmo:

```

1 int exp(int a, int b){
2     if(b==0){
3         return 1;
4     }
5     else return a*exp(a,b-1);
6 }

```

Ejemplo 3.1.6

La función factorial $n \mapsto n!$ es computable.

Demostración:

En efecto, se tiene el siguiente algoritmo:

```

1 int fact(int n){
2     if(n==0) return 1;
3     else return n*fact(n-1);
4 }

```

Ejemplo 3.1.7

Las funciones máximo y mínimo son computables.

Ejemplo 3.1.8

El algoritmo de la división es computable.

Demostración:

En efecto, se tiene el siguiente algoritmo:

```

1 int div(int a, int b){
2     for(int i=1; i*b<=a;i++){ //se queda y acaba si es que se
        puede dividir
3         q = i-1;
4         r = a-b*q;
5         return exp(2,q)*(2*r-1); //codificamos de esta manera la
        salida del programa
6 }

```

Observación 3.1.4

Cuando coloquemos $f :: A \rightarrow B$, entenderemos que $\text{dom}(f) \subseteq A$, es decir que f es una función parcial.

Teorema 3.1.1

Sea $f :: \mathbb{N}^k \rightarrow \mathbb{N}$ una función computable, y sean $g_1, \dots, g_k :: \mathbb{N} \rightarrow \mathbb{N}$ funciones computables. Entonces:

- (1) La función $h_1 :: \mathbb{N} \rightarrow \mathbb{N}$ dada por: $h_1(x) = f(g_1(x), \dots, g_k(x))$ es computable.
- (2) La función $h_2 :: \mathbb{N}^{k-1} \rightarrow \mathbb{N}$ dada por:

$$h_2(x_2, \dots, x_k) = (\mu x)(f(x, x_2, \dots, x_k) = 0)$$

donde la función μx es el mínimo de x tal que lo de adentro se hace 0, siendo f tal que para todo $i \leq x$, $f(i, x_2, \dots, x_k)$ está bien definido, también es computable.

Demostración:

De (1): Considere el algoritmo:

```
1 int h_1(int x){
2   int y_1 = g_1(x);
3   int y_2 = g_2(x);
4   ...
5   int y_k = g_k(x);
6   return f(y_1, ..., y_k);
7 }
```

■ es una función computable, ya que si no puede calcular algún valor, se queda atorado.

De (2): Considere el algoritmo:

```
1 int h_2(int x_2, ..., int x_k){
2   for(int x=0; f(x, x_2, ..., x_k) != 0; x++) return x;
3 }
```

es de una función computable.

Definición 3.1.4

Una función computable $f :: \mathbb{N}^k \rightarrow \mathbb{N}$ es **total**, si $\text{dom}(f) = \mathbb{N}^k$. En computabilidad esto se denota por:

$$(\forall x_1, \dots, x_k)(f(x_1, \dots, x_k) \downarrow)$$

esto es, que para cualquier entrada f está bien definida.

Observación 3.1.5

En el teorema anterior, siempre se puede hacer (2) si la función f es total.

Ejercicio 3.1.3

Codifique una máquina de Turing que sume dos números en binario.

```

1  name: suma_numeros_binario
2  init: s0
3  accept: sf
4
5  s0,_
6  s0,_,>
7
8  s0,0
9  s1,0,>
10
11 s1,0
12 s1,0,>
13
14 s0,1
15 s1,1,>
16
17 s1,1
18 s1,1,>
19
20 //operaciones de suma
21
22 //el estado s_s0 nos dice que va a empezar a contar el otro
    numero
23 //el estado s_s1 nos dice que encuentro un numero positivo para
    sumar
24
25 s1,+
26 s_s0,+,>
27
28 s_s0,0
29 s_s0,0,>
30
31 s_s0,1
32 s_s1,1,>
33
34 s_s1,1
35 s_s1,1,>
36
37 s_s1,0
38 s_s1,0,>
39
40 //llego al final de la cadena
41
42 s_s1,_
43 sr0,_,<
44
45 sr0,1
46 srf,0,>
47
48 sr0,0
49 sr0,0,<

```

```

50
51 srf,0
52 srf,1,>
53
54 srf,_
55 ss0,_,<
56
57 //ss es que ahora le va sumar uno a la cadena de la izquierda
58
59 ss0,0
60 ss0,0,<
61
62 ss0,1
63 ss0,1,<
64
65 ss0,+
66 ss1,+,<
67
68 ss1,0
69 s1,1,>
70
71 ss1,1
72 ss1,0,<
73
74 ss1,_
75 s1,1,>
76
77 //cuando no haya nada por sumar, simplemente se detiene
78
79 s_s0,_,_
80 sf0,_,<
81
82 sf0,0
83 sf0,_,<
84
85 sf0,+
86 sf,_,-
87
88 // < = left
89 // > = right
90 // - = hold
91 // use underscore for blank cells
92
93 //States and symbols are case-sensitive
94
95 //Load your code and click COMPILE.
96 //or load an example (top-right).

```

Ejercicio 3.1.4

Codifique una máquina de Turing que sume dos números en binario.

```

1 name: resta_numeros_1
2 init: s0
3 accept: sf
4
5 s0,_
6 s0,_,>
7
8 s0,0
9 s1,0,>
10
11 s1,0
12 s1,0,>
13
14 s0,1
15 s1,1,>
16
17 s1,1
18 s1,1,>
19
20 //operaciones de resta
21
22 //sr0 es estado resta inicial
23
24 //srf es estado resta final
25
26 s1,_
27 sr0,_,<
28
29 sr0,1
30 srf,0,>
31
32 sr0,0
33 sr0,0,<
34
35 srf,0
36 srf,1,>
37
38 srf,_
39 sf,_,-
40
41 // < = left
42 // > = right
43 // - = hold
44 // use underscore for blank cells
45
46 //States and symbols are case-sensitive
47
48 //Load your code and click COMPILE.
49 //or load an example (top-right).

```

Ejercicio 3.1.5

Dada una cadena en binario, escribir un programa que haga una copia de la misma.

```
1 name: copiar_cadena
2 init: s0
3 accept: sf
4
5 //se empieza a mover y marca el inicio de la cadena
6
7 s0,_
8 s0,_,>
9
10 s0,0
11 s1,0,<
12
13 s2,0
14 s2,0,>
15
16 s0,1
17 s1,1,<
18
19 s2,1
20 s2,1,>
21
22 s1,_
23 s2,|,>
24
25 s2,0
26 s2,0,>
27
28 s2,1
29 s2,1,>
30
31 //coloca el inicio de la copia de la cadena
32
33 s2,_
34 sd,c,<
35
36 sd,0
37 sd,0,<
38
39 sd,1
40 sd,1,<
41
42 sd,2
43 sd,2,<
44
45 sd,3
46 sd,3,<
47
48 sd,c
49 sd,c,<
50
```

```

51 sd,|
52 sdp,|,>
53
54 //deteccion de si es 0 o 1
55
56 sdp,0
57 sdp0,2,>
58
59 sdp,1
60 sdp1,3,>
61
62 sdp,2
63 sdp,2,>
64
65 sdp,3
66 sdp,3,>
67
68 //movimiento a la derecha para colocar 0 o 1
69
70 sdp0,0
71 sdp0,0,>
72
73 sdp0,1
74 sdp0,1,>
75
76 sdp1,0
77 sdp1,0,>
78
79 sdp1,1
80 sdp1,1,>
81
82 //detecta la copia
83
84 sdp0,c
85 sdp0,c,>
86
87 sdp1,c
88 sdp1,c,>
89
90 sdp0,_
91 sd,0,<
92
93 sdp1,_
94 sd,1,<
95
96 //detecta que ya debe terminar
97
98 sdp,c
99 scam,c,<
100
101 scam,2
102 scam,0,<

```

```

103
104 scam,3
105 scam,1,<
106
107 scam,|
108 sf,_,-

```

Ejercicio 3.1.6

Programar una máquina de Turing que haga el producto de dos números.

```

1 name: producto_numeros
2 init: p0
3 accept: pf
4
5 //input: [n]_2*[m]_2
6
7 //empieza el movimiento
8
9 p0,_
10 p0,_,>
11
12 p0,0
13 p1,0,<
14
15 p0,1
16 p1,1,<
17
18 p1,_
19 p2,|,>
20
21 p2,0
22 p2,0,>
23
24 p2,1
25 p2,1,>
26
27 p2,*
28 p2,*,>
29
30 //llego al final de la cadena
31
32 p2,_
33 sd,c,<
34
35 //PARTE PRIMERA COPIA
36
37 sd,0
38 sd,0,<
39
40 sd,1
41 sd,1,<

```



```

42
43 sd,2
44 sd,2,<
45
46 sd,3
47 sd,3,<
48
49 sd,*
50 sd,*,<
51
52 sd,c
53 sd,c,<
54
55 sd,|
56 sdp,|,>
57
58 //deteccion de si es 0 o 1
59
60 sdp,0
61 sdp0,2,>
62
63 sdp,1
64 sdp1,3,>
65
66 sdp,2
67 sdp,2,>
68
69 sdp,3
70 sdp,3,>
71
72 sdp,c
73 sdp,c,>
74
75 //movimiento a la derecha para colocar 0 o 1
76
77 sdp0,0
78 sdp0,0,>
79
80 sdp0,1
81 sdp0,1,>
82
83 sdp1,0
84 sdp1,0,>
85
86 sdp1,1
87 sdp1,1,>
88
89 //detecta la copia
90
91 sdp0,*
92 sdp0,*,>
93

```

```

94 sdp1,*
95 sdp1*,*,>
96
97 sdp0,c
98 sdp0,c,>
99
100 sdp1,c
101 sdp1,c,>
102
103 sdp0,_
104 sd,0,<
105
106 sdp1,_
107 sd,1,<
108
109 //detecta que ya debe terminar y elimina los cambios que hizo
110
111 sdp,*
112 scam,*,<
113
114 scam,2
115 scam,0,<
116
117 scam,3
118 scam,1,<
119
120 scam,|
121 sf,_,>
122
123 //segunda copia
124
125 sf,0
126 sf,0,>
127
128 sf,1
129 sf,1,>
130
131 sf,*
132 sf,*,>
133
134 sf,c
135 sf,c,>
136
137 //ahora, hace la segunda copia
138
139 //coloca el inicio de la copia de la cadena
140
141 sf,_
142 rd,d,<
143
144 rd,0
145 rd,0,<

```

```

146
147 rd,1
148 rd,1,<
149
150 rd,2
151 rd,2,<
152
153 rd,3
154 rd,3,<
155
156 rd,d
157 rd,d,<
158
159 rd,c
160 rdp,c,>
161
162 //deteccion de si es 0 o 1
163
164 rdp,0
165 rdp0,2,>
166
167 rdp,1
168 rdp1,3,>
169
170 rdp,2
171 rdp,2,>
172
173 rdp,3
174 rdp,3,>
175
176 //movimiento a la derecha para colocar 0 o 1
177
178 rdp0,0
179 rdp0,0,>
180
181 rdp0,1
182 rdp0,1,>
183
184 rdp1,0
185 rdp1,0,>
186
187 rdp1,1
188 rdp1,1,>
189
190 //detecta la copia
191
192 rdp0,d
193 rdp0,d,>
194
195 rdp1,d
196 rdp1,d,>
197

```

```

198 rdp0,_
199 rd,0,<
200
201 rdp1,_
202 rd,1,<
203
204 //detecta que ya debe terminar
205
206 rdp,d
207 rcam,d,<
208
209 rcam,2
210 rcam,0,<
211
212 rcam,3
213 rcam,1,<
214
215 rcam,c
216 rf,c,<
217
218 rf,0
219 rf,0,<
220
221 rf,1
222 rf,1,<
223
224 rf,*
225 rf,*,<
226
227 //ahora ya puede empezar a sumar uno por uno
228
229 rf,_
230 rs,_,>
231
232 rs,0
233 rs,0,>
234
235 rs,1
236 rs,1,>
237
238 rs,*
239 rs,*,>
240
241 rs,c
242 rs,c,>
243
244 rs,d
245 rs,d,>
246
247 //hace la primera suma
248
249 rs,_

```

```

250 rsr,_,<
251
252 rsr,1
253 rss,0,>
254
255 rss,0
256 rss,1,>
257
258 rss,_
259 Rf,_,<
260
261 //en Rf
262
263 Rf,0
264 Rf,0,<
265
266 Rf,1
267 Rf,1,<
268
269 Rf,c
270 Rf,c,<
271
272 Rf,d
273 Rf,d,<
274
275 //ahora, si detecta el * es pq ahora tiene que sumar
276
277 Rf,*
278 estSum,*,<
279
280 estSum,0
281 estSumAcaba,1,>
282
283 estSum,_
284 estSumAcaba,1,>
285
286 estSum,1
287 estSum,0,<
288
289 //aqui acabo de sumar
290
291 rsr,0
292 RF,0,-
293
294 //se mueve para ahora sumar a la otra cadena

```

Definición 3.1.5

Un conjunto X es **computable**, si puede ser visto como input de elementos de \mathbb{N} , y su función característica es computable.

Ejemplo 3.1.9

Sea $F : \mathbb{N} \rightarrow \mathbb{N}$ la función:

$$F(n) = \begin{cases} 1 & \text{si conjetura G\"{o}lbach es verdadera.} \\ 0 & \text{en caso contrario.} \end{cases}$$

esta funci3n es computable.

Ejemplo 3.1.10

La funci3n $f : \mathbb{N} \rightarrow \mathbb{N}$ tal que:

$$f(n) = \min \left\{ p \mid p > n \text{ y tanto } p \text{ como } p + 2 \text{ son primos} \right\}$$

en efecto, se tiene el siguiente algoritmo de f :

```

1 int f(int n){
2     for(int i=n+1; i no es primo || i+2 tampoco lo es; i++){
3         return i;
4     }
5 }

```

3.2. Máquina Universal de Turing

En general, los input de mis algoritmos son \mathbb{N} , pero podemos tambi3n codificar parejas por medio de n3meros naturales, por lo que realmente tambi3n podemos introducir tuplas de \mathbb{N}^k .

Observaci3n 3.2.1

Una m3quina de Turing es un conjunto finito $\{a_1, \dots, a_n\}$, donde $a_i = (s_{i1}, t_{i2}, s_{i3}, t_{i4}, l)$ con $l \in \{<, -, >\}$. Podemos hacer una codificaci3n estas 6-tuplas:

0	-	0
1	-	—
2	-	*
3	-	.
\vdots	-	\vdots

en este caso, codificamos elementos de $\left[\{0, 1, *, <, >, -, s_1, \dots, s_n\}^k \right]^{<\aleph_0}$ (cadenas finitas de tuplas finitas). Lo que podemos hacer entonces es codificar m3quinas de Turing.

Teorema 3.2.1

El conjunto

$$\left\{ x \in X \mid x \text{ es m3quina de Turing} \right\}$$

y,

$$\left\{ n \in \mathbb{N} \mid n \text{ codifica una m3quina de Turing} \right\}$$

son computables.

Demostración:

Ver simulador de máquina de Turing. ■

Definición 3.2.1

Denotaremos la **máquina universal de Turing** por φ , donde

$$\varphi :: \mathbb{N}^2 \rightarrow \mathbb{N}$$

dada por $\varphi(e, n)$ es el resultado de correr la e -ésima máquina de Turing con input n .

Observación 3.2.2

φ es una función computable.

Una forma de interpretar a φ es el simulador de máquina de Turing, pues en este simulador introducimos una máquina de Turing y un input, así que nuestro código sería e y el input sería n .

Definición 3.2.2

La función $\varphi(e, -) : \mathbb{N} \rightarrow \mathbb{N}$ tal que $n \mapsto \varphi(e, n)$ es llamada la **e -ésima función computable**.

Teorema 3.2.2 (Lema del Relleno)

Para cada función computable f , existen una infinidad de $e \in \mathbb{N}$ tales que $f = \varphi(e, -)$.

Existen funciones que no son computables.

Observación 3.2.3

Como se pueden enumerar todas las funciones computables, el conjunto:

$$\left| \left\{ f : \mathbb{N} \rightarrow \mathbb{N} \mid f \text{ es computable} \right\} \right| = \aleph_0$$

pero,

$$\left| \left\{ f : \mathbb{N} \rightarrow \mathbb{N} \mid f \text{ es función} \right\} \right| = \aleph_1$$

Ejemplo 3.2.1

Se define la función **Busy Beaver**, $\sigma : \mathbb{N} \rightarrow \mathbb{N}$ dada por:

$$\sigma(n) = \max \{ \text{output en unario de una máquina de Turing con } \leq n + 1 \text{ estados e input } 0 \}$$

de forma inmediata se tiene que $\sigma(0) = 1$, ya que solo hay una máquina de Turing con un estado (el estado final), y el mayor output que puede hacer es 1.

Ejercicio 3.2.1

Verifique que $\sigma(2) \geq 4$.

Teorema 3.2.3

Si $f : \mathbb{N} \rightarrow \mathbb{N}$ es cualquier función computable, entonces existe $N \in \mathbb{N}$ tal que para todo $n \geq N$ se cumple que $f(n) < \sigma(n)$. En particular, σ **no es computable**.

Demostración:

Sea f computable, entonces la función $g : \mathbb{N} \rightarrow \mathbb{N}$ dada por:

$$g(n) = \min \{f(2n), f(2n + 1)\} + 1$$

también es computable. Por lo tanto, existe alguna máquina de Turing T que calcula a g en unario. Sea k el número de estados (además del estado inicial) de dicha máquina.

Consideremos la función



3.3. Conjuntos y Relaciones Computables

Definición 3.3.1

Un conjunto $X \subseteq \mathbb{N}$ es **computable** si su función característica χ_X es total computable.

Ejemplo 3.3.1

Los siguientes son conjuntos computables:

- (1) \mathbb{N} .
- (2) \emptyset .
- (3) $\{n \in \mathbb{N} \mid n \text{ es cuadrado perfecto}\}$.
- (4) $\{p \in \mathbb{N} \mid p \text{ es primo}\}$.
- (5) $\{(n, m) \in \mathbb{N}^2 \mid n \mid m\}$.

Solución:

En efecto, se tienen los siguientes algoritmos de cada una de las funciones características de los conjuntos anteriores:

```
1  int uno(int n) return 1;
2
3  int dos(int n) return 0;
4
5  int tres(int n){
6      for(int i=0; i*i<n; i++){
7          if(i*i==n) return 1;
8      }
9      return 0;
10 }
```



Proposición 3.3.1

Sean $X, Y \subseteq \mathbb{N}$ conjuntos computables, entonces $X \cap Y$, $X \cup Y$ y $X \setminus Y$ son también computables (en otras palabras, el conjunto de conjuntos computables es un álgebra booleana).

Demostración:

Basta con ver que:

- $\chi_X(n)\chi_Y(n) = \chi_{X \cap Y}(n)$.
- $\chi_{X \cup Y}(n) = \max\{\chi_X(n), \chi_Y(n)\}$.
- $\chi_{X \setminus Y}(n) = \max\{1 - \chi_X(n), 0\}$.

para todo $n \in \mathbb{N}$. ■

Teorema 3.3.1

El conjunto

$$\left\{e \in \mathbb{N} \mid \varphi(e, -) \text{ es una función total} \right\}$$

no es computable.

Demostración:

En caso contrario, se tendría que la función $h : n \mapsto n$ -ésima máquina de Turing que induce una función total, también sería computable (ya que la característica detecta si una función computable es total o no, luego en particular existe una función que codifica a todas las funciones totales).

Se sigue así que la función $u : \mathbb{N}^2 \rightarrow \mathbb{N}$:

$$(x, y) \mapsto \varphi(h(x), y)$$

sería total computable. Así que, la función $g(n) = u(n, n) + 1$ también es total computable.

Por ende, existe $m \in \mathbb{N}$ tal que $g = \varphi(h(m), -)$, entonces:

$$\begin{aligned}
 u(m, m) &= g(m) \\
 &= u(m, m) + 1
 \end{aligned}$$

lo cual es una contradicción. ■

Teorema 3.3.2 (Halting Problem, Entscheidungs problem)

El conjunto

$$H = \left\{(e, n) \mid \varphi(e, n) \text{ está definido} \right\}$$

(esto es que $(e, n) \in \text{dom}(g)$ también denotado por $(e, n) \downarrow$) no es computable.

Demostración:

Supongamos que sí lo es, entonces el conjunto

$$D = \left\{e \in \mathbb{N} \mid (e, e) \downarrow \right\}$$

es computable. Definimos la siguiente función $f : \mathbb{N} \rightarrow \mathbb{N}$ con algoritmo:

```

1  int f(int n){
2      int i;
3      if(chi_X(n)) while(1) i++;
4      return 1;
5  }
```

claramente esta función es computable, y está definida si $n \notin D$ (en caso contrario, se sigue y ejecuta el programa infinitamente). Por ser computable, existe $e \in \mathbb{N}$ tal que $f = \varphi(e, -)$. Se tiene:

$$\begin{aligned} f(e) \text{ está definido si y sólo si } e \notin D \\ \text{si y sólo si } \varphi(e, e) \text{ no está definido} \\ \text{si y sólo si } f(e) \text{ no está definido} \end{aligned}$$

lo cual es una contradicción. Por tanto, el conjunto anterior no puede ser computable. ■

Observación 3.3.1

El teorema anterior se conoce como el **brinco de Turing** y es llamado el **problema de la detección**.

Observación 3.3.2

Del teorema anterior se deduce de forma inmediata que el conjunto

$$D = \left\{ e \in \mathbb{N} \mid \varphi(e, e) \text{ está definido} \right\}$$

no es computable.

Ejemplo 3.3.2

El problema 10 de Hilbert.

3.4. Conjuntos computablemente numerables

Definición 3.4.1

Un conjunto $X \subseteq \mathbb{N}$ es **computablemente enumerable** o **listable** si existe una función total computable $F : \mathbb{N} \rightarrow \left\{ F \subseteq \mathbb{N} \mid F \text{ es finito} \right\}$ tal que:

- (0) $F(0) = \emptyset$.
- (1) $\forall n \in \mathbb{N}, F(n) \subseteq F(n+1)$.
- (2) $\forall n \in \mathbb{N}, |F(n+1) \setminus F(n)| \leq 1$.
- (3) $X = \bigcup_{n=0}^{\infty} F(n)$.

Observación 3.4.1

Intuitivamente, diremos que $X \subseteq \mathbb{N}$ es computablemente enumerable si existe un algoritmo con instrucción *print* que *sucesivamente* (no necesariamente los imprime en orden) imprime a los elementos de X .

Ejemplo 3.4.1

Todo conjunto computable es computablemente enumerable.

Demostración:

Suponga que X es un conjunto computable, entonces existe un algoritmo para su función característica χ_X . Considere la función:

```
1 void main(void){
2     for(i=0; 1;i++){
3         if(chi_X(i)) print(i);
4     }
5 }
```

■

Observación 3.4.2

En el caso en que un conjunto sea computable, su función característica es total computable.

Ejemplo 3.4.2

El problema de la detección:

$$H = \{(e, n) \mid \varphi(e, n) \downarrow\}$$

es computablemente enumerable.

Demostración:

La idea es ir corriendo el programa con cada uno de los inputs posibles recorriendo (ya sea en diagonal o no), a todas y cada una de las parejas:

$$\begin{array}{cccc} \vdots & \vdots & \vdots & \ddots \\ (1,0) & (1,1) & (1,2) & \dots \\ (0,0) & (0,1) & (0,2) & \dots \end{array}$$

Lo cual lo hacemos de la siguiente manera:

```
1 void main(void){
2     for(int m = 1; 1;m++){
3         encuentra e,n tales que m=2^(2n+1);
4         correr varphi(e,n) m pasos, si se detiene imprime;
5         print(e,n);
6     }
7 }
```

■

Observación 3.4.3

Note que H no es computable, pero el conjunto:

$$\{(e, n, t) \mid \varphi(e, n) \text{ se detiene en } \leq t \text{ pasos}\}$$

sí es computable.

Teorema 3.4.1

Sea $X \subseteq \mathbb{N}$. Entonces las siguientes son equivalentes:

- (1) X es computablemente enumerable.

(2) La función semicaracterística σ_X dada por:

$$\sigma_X(n) = \begin{cases} 1 & \text{si } n \in X \\ \uparrow & \text{en otro caso} \end{cases}$$

es computable.

(3) Hay una función computable $f : \mathbb{N} \rightarrow \mathbb{N}$ tal que $X = \text{dom}(f)$.

(4) Hay un conjunto computable $Y \subseteq \mathbb{N}^2$ tal que:

$$X = \{x \in \mathbb{N} \mid (\exists y \in \mathbb{N})((x, y) \in Y)\}$$

Demostración:

(1) \Rightarrow (2): Veamos que existe el siguiente algoritmo para la función semicaracterística:

```
1 int sigma(int n){
2     correr algoritmo que lista a X{
3         if(se imprimio n) return 1;
4     }
5 }
```

(2) \Rightarrow (3): Es inmediato (tomar $f = \sigma_X$).

(3) \Rightarrow (4): Considere el algoritmo de la función característica de Y , χ_Y :

```
1 int chi_Y(int a, int b){
2     correr f(a) b pasos;
3     if(f(a) se calculo) return 1;
4     else return 0;
5 }
```

note que Y es precisamente:

$$Y = \{(a, b) \mid f(a) \text{ se detiene en } \leq b \text{ pasos}\}$$

y es tal que su proyección coincide con X , pues note que:

$a \in X$ si y sólo si $a \in \text{dom}(f)$
si y sólo si $\exists b$ tal que $f(a)$ se detiene en b pasos
si y sólo si $(a, b) \in Y$ para algún b

(4) \Rightarrow (1): Considere el algoritmo de la función característica de X :

```
1 void main(void){
2     for(m=1;1;m++){
3         [se decodifica la pareja dada por m=2^n(2d+1)];
4         if(chi_Y(n,d)) print(n);
5     }
6 }
```

pues la función característica de Y es total.



Observación 3.4.4

La flecha hacia arriba significa que el algoritmo se queda estancado y nunca termina.

Teorema 3.4.2 (Kleene)

Si $X \subseteq \mathbb{N}$, entonces X es computable si y sólo si tanto X como $\mathbb{N} \setminus X$ son computablemente enumerables.

Demostración:

\Rightarrow): Suponga que X es computable, entonces $\mathbb{N} \setminus X$ es computable, luego los dos (en particular) son enumerablemente computables.

\Leftarrow): Considere el algoritmo de la función característica de X :

```
1 int chi_X(int n){
2     for(int i = 0; 1; i++){
3         correr X en i pasos;
4         if(se imprimio n) return 1;
5         correr N\X en i pasos;
6         if(se imprimio n) return 0;
7     }
8 }
```

este algoritmo siempre termina por ser ambos conjuntos enumerablemente computables, luego la función característica de X es total. ■

Corolario 3.4.1

Los conjuntos:

$$\mathbb{N}^2 \setminus H = \{(e, d) \mid \varphi(e, d) \uparrow\}$$

y,

$$\mathbb{N} \setminus D = \{n \in \mathbb{N} \mid (n, n) \uparrow\}$$

no son computablemente enumerables.

Demostración:

Es inmediata del teorema de Kleene. ■

Observación 3.4.5

Si X y Y son conjuntos enumerablemente computables, entonces $X \cap Y$ y $X \cup Y$ también lo son.

Demostración:

Ejercicio. ■

Teorema 3.4.3

Sea $f :: \mathbb{N} \rightarrow \mathbb{N}$ una función parcial.

- (1) f es una función computable si y sólo si $\Gamma(f)$ es un conjunto enumerablemente computable, donde:

$$\Gamma(f) = \{(x, y) \in \mathbb{N}^2 \mid y = f(x)\}$$

(2) Más aún, si f es total, entonces f es computable si y sólo si $\Gamma(f)$ es un conjunto computable.

Demostración:

De (1): Probaremos la doble implicación:

\Rightarrow): Considere el algoritmo:

```
1 void main(void){
2     for(int a = 0; 1; a++){
3         Decodificar a=(n,t);
4         Correr f(n) t pasos{
5             si el algoritmo termino{
6                 print(n,f(n);
7             }
8         }
9     }
10 }
```

\Leftarrow): Considere el algoritmo:

```
1 int f(int n){
2     Correr el algoritmo que imprime la lista de elementos de
        Gamma(f);
3     Cada vez que se imprime (x,y){
4         if(x == n) return y;
5     }
6 }
```

De (2):

\Rightarrow): Considere el algoritmo:

```
1 int chi_Gamma_f(int x; int y){
2     if(y == f(x)) return 1;
3     else return 0;
4 }
```

\Leftarrow): Considere el algoritmo:

```
1 int f(int n){
2     for(int y = 0; 1-chi_Gamma_f(n,y),y++);
3     //la instruccion return va afuera del ciclo FOR
4     return y;
5 }
```

■

Teorema 3.4.4

Sea $X \subseteq \mathbb{N}$. Entonces, X es computablemente enumerable si y sólo si o bien $X = \emptyset$ o existe una función total computable $f : \mathbb{N} \rightarrow \mathbb{N}$ tal que $X = \text{ran}(f) = \text{im}(f)$.

Demostración:

\Leftarrow): Suponga que $X \neq \emptyset$. Se tienen dos casos:

- X es finito, digamos $X = \{x_0, \dots, x_n\}$. Considere el algoritmo.

```

1  int f(int k){
2      switch(k){
3          case 0: return x_0;
4          ... : ... ;
5          case n: return x_n;
6      }
7      return x_n;
8  }

```

- X es infinito. Considere el algoritmo:

```

1  int f(int k){
2      correr algoritmo que lista los elementos de  $X$ ;
3      tomar el  $k$ -esimo elemento de la lista de  $X$  y almacenarlo
4      en la variable  $y$ ;
5      return  $y$ ;
6  }

```

\Leftarrow): Suponga que $X \neq \emptyset$. Considere el algoritmo:

```

1  void main(void){
2      for(int i = 0; 1; i++) print(f(n));
3  }

```

■

Capítulo 4

Teoremas de Completud

4.1. Hilbert

Hilbert propuso algunos problemas (en su lista de 100 problemas), los cuales son los siguientes:

- (1) Lenguaje formal de las matemáticas (lógica de primer orden).
- (2) Codificar las reglas de la lógica (cálculo deductivo).
- (3) Encontrar un conjunto razonable de axiomas (algún conjunto de fórmulas).
- (4) Demostrar que lo elegido en (3) sea consistente.

4.2. Introducción

Sea \mathcal{L} un lenguaje de primer orden y considere una enumeración de Gödel, esto es que codificamos el símbolo del alfabeto por medio de los naturales \mathbb{N} .

En este caso, retomamos todo lo del capítulo antepasado. Hacemos:

\exists	—	1
\Rightarrow	—	2
\neg	—	3
$=$	—	4
v_1	—	5
\vdots	\vdots	\vdots
v_n	—	$2n + 3$
\vdots	\vdots	\vdots

y hacemos lo análogo para los símbolos de relación, función y constantes.

F_1	—	6
F_2	—	18
\vdots	\vdots	\vdots
F_n	—	$2 \cdot 3^n$
\vdots	\vdots	\vdots

R_1	—	10
R_2	—	50
\vdots	\vdots	\vdots
R_n	—	$2 \cdot 5^n$
\vdots	\vdots	\vdots

c_1	—	14
c_2	—	98
\vdots	\vdots	\vdots
c_n	—	$2(7^n)$

En este sentido, lo hacemos para que el conjunto:

$$\left\{ n \in \mathbb{N} \mid n \text{ representa algún símbolo del alfabeto} \right\}$$

es computable.

Ahora, lo que queremos es codificar sucesiones finitas de símbolos del alfabeto. Recordemos que una forma de hacerlo era mediante el teorema fundamental de la aritmética, hacinedo:

$$(n_1, \dots, n_k) \rightsquigarrow p_1^{n_1} \cdots p_{k-1}^{n_{k-1}} p_k^{n_k+1}$$

recordando que estamos haciendo una enumeración creciente de ls números primos $\{p_n\}_{n=1}^{\infty}$. Nueva-mente, hacemos esto para que el conjunto:

$$\left\{ n \mid n \text{ codifica una cadena de símbolos} \right\}$$

es un conjunto computable. Esto se hace para que las funciones:

$$\begin{aligned} n &\mapsto 1^\circ \text{ término de la tupla codificada por } n \\ n &\mapsto 2^\circ \text{ término de la tupla codificada por } n \\ &\vdots \\ n &\mapsto k^\circ \text{ término de la tupla codificada por } n \\ &\vdots \\ n &\mapsto \text{longitud de la tupla codificada por } n \end{aligned}$$

sean funciones *total* computables (va a suceder algo con la condición de totalidad).

Análogamente, podemos codificar sucesiones finitas de cusiones finitas del símbolos del alfabeto.

Observación 4.2.1

Recuerde que un enunciado es una fórmula en la que no aparecen variables libres.

Teorema 4.2.1

El conjunto de términos de \mathcal{L} , el conjunto de fórmulas y el conjunto de enunciados:

$$\text{Term}(\mathcal{L}) = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de un término} \right\}$$

$$\text{Form}(\mathcal{L}) = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de una fórmula} \right\}$$

y,

$$\text{Enun}(\mathcal{L}) = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de un enunciado} \right\}$$

son conjuntos computables.

Demostración:

Ejercicio. Es posible probarlo de forma inductiva y usando la tesis de Church-Turing. ■

Teorema 4.2.2

Los conjuntos:

$$\begin{aligned} & \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de un axioma lógico} \right\} \\ & \left\{ n \in \mathbb{N} \mid n \text{ codifica una terna } (a, b, c) \right\} \end{aligned}$$

donde en el segundo conjunto, a es el número de Gödel de la fórmula $\Rightarrow \varphi\psi$, donde además b es el número de φ y c es el número de ψ , son conjuntos computables.

Demostración:

Ejercicio. Es posible probarlo de forma inductiva y usando la tesis de Church-Turing. ■

Corolario 4.2.1

El conjunto:

$$\text{Dem} = \left\{ n \in \mathbb{N} \mid n \text{ codifica una demostración válida} \right\}$$

es computable.

Demostración:

Inmediato del teorema anterior. ■

Corolario 4.2.2

El conjunto:

$$\text{Teor} = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de alguna } \varphi \text{ tal que } \vdash \varphi \right\}$$

es computablemente enumerable.

Demostración:

Considere:

$$Y = \left\{ (a, b) \mid P(a, b) \right\}$$

donde $P(a, b)$ es la propiedad: b codifica el número de Gödel de φ , $a \in \text{Dem}$ y el código $(a, \varphi) \in \text{Dem}$ (donde en esta parte, a está decodificando las líneas de la demostración). Este conjunto Y es computable (en la misma descripción de la propiedad P viene el algoritmo) y se tiene que el conjunto:

$$\text{Teor} = \left\{ b \in \mathbb{N} \mid \exists a \in \mathbb{N} \text{ tal que } (a, b) \in Y \right\}$$

es computablemente enumerable. ■

Teorema 4.2.3

Sea Σ un conjunto computable de fórmulas.

(1) Sea:

$$\text{Dem}(\Sigma) = \left\{ n \in \mathbb{N} \mid n \text{ codifica una demostración a partir de } \Sigma \right\}$$

es un conjunto computable.

(2) El conjunto:

$$\text{Teor}(\Sigma) = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de alguna fórmula } \varphi \text{ tal que } \Sigma \vdash \varphi \right\}$$

es un conjunto computablemente enumerable.

Demostración:

Análoga al teorema anterior. ■

Teorema 4.2.4

Sea Σ un conjunto computablemente enumerable de fórmulas.

(1) Sea:

$$\text{Dem}(\Sigma) = \left\{ n \in \mathbb{N} \mid n \text{ codifica una demostración a partir de } \Sigma \right\}$$

es un conjunto computablemente enumerable.

(2) El conjunto:

$$\text{Teor}(\Sigma) = \left\{ n \in \mathbb{N} \mid n \text{ es el número de Gödel de alguna fórmula } \varphi \text{ tal que } \Sigma \vdash \varphi \right\}$$

es un conjunto computablemente enumerable.

Demostración:

Análoga al teorema anterior. ■

El ideal de Hilbert: Quería encontrar un conjunto computable Σ tal que $\text{Teor}(\Sigma)$ sea computable.

Teorema 4.2.5

Si una función f es **computable**, entonces hay una fórmula φ tal que $\Gamma(f) = \left\{ (l, m) \mid \mathbb{N} \models \varphi[n, m] \right\}$.

Demostración:

A la pareja (m, n) la codificamos mediante $(m + n)^2 + m$ (este mapeo de $\mathbb{N}^2 \mapsto \mathbb{N}$ es inyectivo y es sencillo de explicar en el lenguaje de la aritmética). Sea $P.O.(x, y, z) \equiv z = (x + y) \cdot (x + y) + x$, ésta es una fórmula en el lenguaje de la aritmética que codifica. Se tiene que:

$$\mathbb{N} \models P.O.(a, m, n) \text{ si y sólo si } a \text{ codifica a } m, n$$

Ahora, vamos a codificar una k -tupla. Sea (m_0, \dots, m_{k-1}) , esta la codificamos de la siguiente manera: para cada $i = 0, \dots, k - 1$ sea q_i el código de la pareja ordenada (m_i, i) . Sea $n = \max \{q_0, \dots, q_{k-1}\}$ y

$$u = \prod_{i < k} (1 + (q_i + 1)n!) \tag{4.1}$$

El código de (m_0, \dots, m_{k-1}) es el código de $(u, n!)$.

Antes de seguir, debemos probar tres lemas adicionales:

Lema 4.2.1

Si $q < r \leq n$, entonces:

$$1 + (q + 1)n! \text{ y } 1 + (r + 1)n!$$

son primos relativos.

Demostración:

Supongamos que $p \in \mathbb{N}$ es primo tal que p divide a ambos números, entonces p divide a:

$$p \mid 1 + (r + 1)n! - 1 - (1 + 1)n! = n!(r - q)$$

Si $p \mid n!$ entonces $p \mid 1 \nmid_c$. Por ende, $p \mid r - q$ en particular $p \mid n!$ ya que $0 < r - q \leq n$. Así que no existe tal primo que divida a ambos números, luego éstos deben ser primos relativos. ■

Lema 4.2.2

Si $q < n$ y u es como en la ecuación (4.1), entonces $(1 + (q + 1)n!) \mid u$ si y sólo si existe $i = 0, \dots, k - 1$ tal que $q = q_i$.

Demostración:

\Rightarrow): Supongamos que:

$$(1 + (q + 1)n!) \mid \prod_{i < k} (1 + (q_i + 1)n!)$$

luego existe $p \in \mathbb{N}$ primo tal que $p \mid (1 + (q + 1)n!)$ y $p \mid \prod_{i < k} (1 + (q_i + 1)n!)$, luego al ser todos los elementos en el producto primos relativos, debe existir $i = 0, \dots, k - 1$ tal que $p \mid (1 + (q_i + 1)n!)$, así que $(1 + (q + 1)n!)$ y $(1 + (q_i + 1)n!)$ no son primos relativos, del lema anterior se sigue que $q = q_i$.

\Leftarrow): Es inmediata. ■

Lema 4.2.3

Para cada $i = 0, \dots, k - 1$, m_i es el mínimo número m tal que:

$$(1 + (c(m, i) + 1)n!) \mid u$$

donde $c(m, i)$ es el número que codifica a (m, i) .

Demostración:

Sabemos que m_i satisface la condición de arriba (ya que q_i en particular codifica a la tupla (m_i, i)). Basta demostrar que si $m < m_i$ y q es el código de la pareja (m, i) , entonces:

$$(1 + (q + 1)n!) \mid u$$

De lo contrario, por el lema anterior $q = q_j$ para algun $j = 0, \dots, k - 1$, entonces q codifica tanto a (m, j) como a (m_i, i) . Por la inyectividad de esta codificación se sigue que $i = j$ y $m = m_i = m_j \nmid_c$. Por ende, se sigue el resultado. ■

De los tres lemas anteriores, entonces:

$$\begin{aligned} \text{Tupla}(x, y, z) \equiv & (\exists u \exists N P.O.(x, u, N) \wedge \exists q P.O.(q, z, y) \wedge) \wedge \\ & (1 + (q + 1)N) \mid u \wedge (\forall w < z) (\forall p P.O.(p, w, y) \Rightarrow (1 + (p + 1)N \nmid u)) \end{aligned}$$

donde la función de la izquierda es la y -ésima coordenada de z es x . Gracias al lema anterior, se tiene que dados $i, \alpha, m \in \mathbb{N}$:

$$\mathbb{N} \models \text{Tupla}[\alpha, i, m]$$

si y sólo si m es la i -ésima entrada de la tupla codificada por α .

Ahora sí vamos con la **pseudodemostración**: dada una máquina de Turing, una **foto instantánea** es una terna $(s, i, t) \in \mathbb{N}^3$ donde s es el estado, i es la posición del cabezal y t es el contenido de la cinta.

Cada máquina de Turing σ inducirá una fórmula $\psi_\sigma(a, b, c, x, y, z)$ si y sólo si al correr la máquina de turing σ con foto instantánea (a, b, c) un paso luego a la foto instantánea (x, y, z) . Por ejemplo:

Ejemplo 4.2.1

Considere la máquina de Turing:

$$\sigma = \{(s_1, 1, s_1, 1, >), (s_1, 0, s_2, 1, -)\}$$

Entonces, ψ_s es la fórmula:

$$\begin{aligned} \psi_\sigma \equiv & (a = 1 \wedge \text{Tupla}(b, c, 1) \wedge x = 1 \wedge y = b + 1 \wedge \text{Tupla}(b, z, 1)) \vee \\ & (a = 1 \wedge \text{Tupla}(b, c, 0) \wedge x = 2 \wedge y = b \wedge \text{Tupla}(b, z, 1)) \end{aligned}$$

Entonces, si f es computable por medio de la máquina de Turing σ , la fórmula:

$$\begin{aligned} \chi_\sigma(x, y) \equiv & (\exists t)(\exists l)(\exists w \text{Tupla}(t, 1, w) \wedge w \text{ codifica a } (1, 1, x) \wedge w \text{ codifica a } (1, 1, x) \\ & \forall i < l \exists u \exists v \text{Tupla}(t, i, u) \wedge \text{Tupla}(t, i + 1, v) \exists a \exists b \exists c \exists x \exists y \exists z \\ & u \text{ codifica } (a, b, c) \wedge v \text{ codifica } (x, y, z) \wedge \psi_\sigma(a, b, c, x, y, z) \wedge \exists d \text{Tupla}(t, l, d) \\ & \wedge d \text{ codifica } (x, y, z)) \end{aligned}$$

■

Observación 4.2.2

El macro $a \mid b$ abrevia la fórmula $(\exists c)(a \cdot c = b)$.