Curso de Lógica Matemática
Teoría de la Computabilidad

Cristo Daniel Alvarado

4 de noviembre de 2024

Índice general

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 S de *estados*.
- Una función parcial $T : L^* \times S \to L^* \times S \times \{<, -, >\}$ 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 $S = \{s_i, s_1, ..., s_f\}$ y la función sería representada por lo que sea que haga el cabezal.

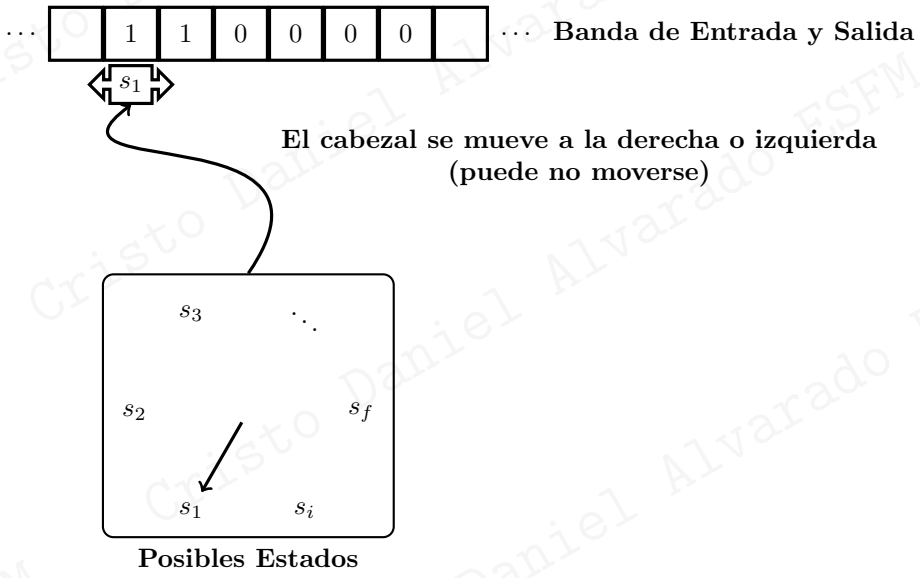

Figura 3.1: Ejemplo de Máquina de Turing

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.

```
name: Sumar uno en unario
init: s0
accept: sf


// Funciones de Transicion

s0,_
s0,_,>

s0,1
s1,1,-

s0,0
s1,0,-

s1,1
s1,0,>

s1,0
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 $(a_1, ..., a_n)$, que la máquina de Turing la convierta en $(a_n, ..., a_1)$.

```
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) $\mathrm{dom}(f) \subseteq \mathbb{N}$.

(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 $n \in \mathrm{dom}(f)$ 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 $f : \mathbb{N} \setminus \{0, 1\} \to \mathbb{N}$ tal que $n \mapsto \text{mín}\left\{p \in \mathbb{N} \middle| p \text{ es primo y } p \mid n\right\}$ es computable.

Demostración:
Se tiene el siguiente algoritmo:

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

■

Ejemplo 3.1.3
Considere la función $g : \left\{n^2 \middle| n \in \mathbb{N}\right\} \to \mathbb{N}$ dada por $n^2 \mapsto n$. Esta función es computable.

Demostración:
Se tiene el siguiente algoritmo:

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

■

Ejemplo 3.1.4
La función $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ 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.

```
int p_2(int a, int b, int c){
    return b;
}
```

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:

```
int exp(int a, int b){
    if(b==0){
        return 1;
    }
    else return a*exp(a,b-1);
}
```

■

Ejemplo 3.1.6
La función factorial $n \mapsto n!$ es computable.

Demostración:
En efecto, se tiene el siguiente algoritmo:

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

■

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:

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

■

Observación 3.1.4
Cuando coloquemos $f :; A \to B$, entenderemos que $\text{dom}\,(f) \subseteq A$, es decir que f es una función parcial.

Teorema 3.1.1
Sea $f :; \mathbb{N}^k \to \mathbb{N}$ una función computable, y sean $g_1, ..., g_k :; \mathbb{N} \to \mathbb{N}$ funciones computables. Entonces:

(1) La función $h_1 :; \mathbb{N} \to \mathbb{N}$ dada por: $h_1(x) = f(g_1(x), \cdots, g_k(x))$ es computable.

(2) La función $h_2 :; \mathbb{N}^{k-1} \to \mathbb{N}$ dada por:

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

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

Demostración:
De *(1)*: Considere el algoritmo:

```
int h_1(int x){
    int y_1 = g_1(x);
    int y_2 = g_2(x);
    ...
    int y_k = g_k(x);
    return f(y_1,...,y_k);
}
```

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

De *(2)*: Considere el algoritmo:

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

es de una función computable.

Definición 3.1.4
Una función computable $f :; \mathbb{N}^k \to \mathbb{N}$ es **total**, si $\text{dom}\,(f) = \mathbb{N}^k$. En computabilidad esto se denota por:

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

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.

Capítulo 4

Teoremas de Completud