Curso de Lógica Matemática
Teoría de la Computabilidad

Cristo Daniel Alvarado

6 de diciembre 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 finito 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.

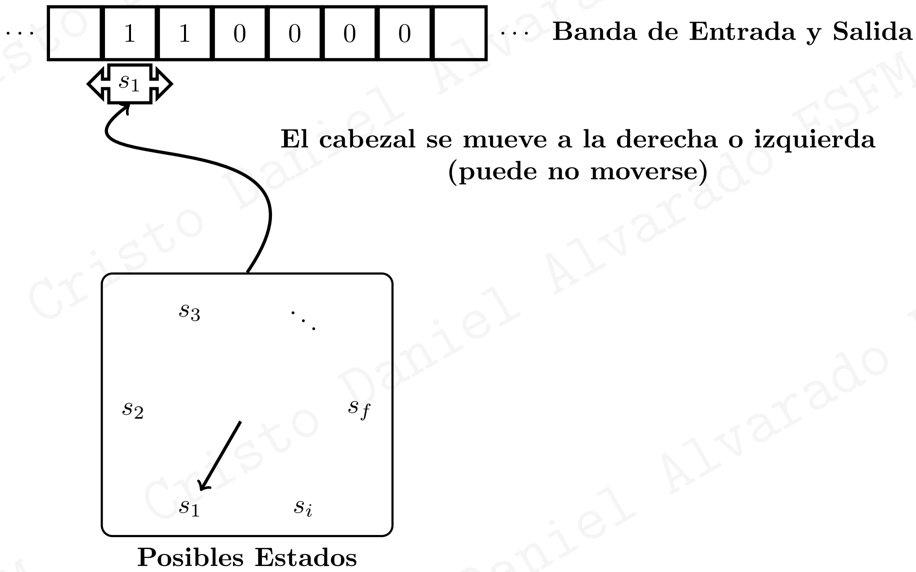

Figura 3.1: Ejemplo de Máquina de Turing

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,>
```

```
s1,_
sf,_,-

// < = left
// > = right
// - = hold
// use _ for blank cells

// States and symbols are case-sensitive

// Load your code and click COMPILE.
//  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 $(a_n, ..., a_1)$.

```
name: invertirCadena
init: s0
accept: s1,sf,l,c,u

//esto para que se empiece a mover
s0,_
s0,_,>

s0,0
x,0,<

s0,1
x,1,<

x,_
s1,2,>

s1,0
s1,0,>

s1,1
s1,1,>

//logica cuando encuentre cosas

s1,_
s2,_,<

s2,_
s2,_,<

s2,0
c00,_,>
```

```

        s2,1
        u00,_,>

        //mueve cosas al inicio

        c00,_
        m,0,<

        u00,_
        m,1,<

        //ya en ciclo

        //mueve derecha

        m,_
        l,_,<

        l,_
        l,_,<

        l,0
        c0,_,>

        l,1
        u0,_,>

        c0,_
        c0,_,>

        //mueve izquierda

        u0,_
        u0,_,>

        c0,0
        c1,0,>

        c0,1
        c1,1,>

        u0,0
        u1,0,>

        u0,1
        u1,1,>

        c1,0
        c1,0,>

        c1,1
```

```
            c1,1,>

            u1,0
            u1,0,>

            u1,1
            u1,1,>

            c1,_
            m,0,<

            u1,_
            m,1,<

            m,0
            m,0,<

            m,1
            m,1,<

            l,2
            sf,_,>

            sf,_
            sf,_,>

            sf,0
            sff,0,-

            sf,1
            sff,1,-
```

Definición 3.1.2
Una función f es **computable** si:

(1) $\operatorname{dom}(f) \subseteq \mathbb{N}$.

(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 $n \in \operatorname{dom}(f)$ 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);
}
```

$\blacksquare$

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);
}
```

$\blacksquare$

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 $\mathrm{dom}\,(f) \subseteq A$, 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 $\mathrm{dom}\,(f) = \mathbb{N}^k$. 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.

Ejercicio 3.1.3
Codifique una máquina de Turing que sume dos números en binario.

```
name: suma_numeros_binario
init: s0
accept: sf

s0,_
s0,_,>

s0,0
s1,0,>

s1,0
s1,0,>

s0,1
s1,1,>

s1,1
s1,1,>

//operaciones de suma

//el estado s_s0 nos dice que va a empezar a contar el otro numero
//el estado s_s1 nos dice que encontro un numero positivo para sumar

s1,+
s_s0,+,>

s_s0,0
s_s0,0,>

s_s0,1
s_s1,1,>

s_s1,1
s_s1,1,>

s_s1,0
s_s1,0,>

//llego al final de la cadena

s_s1,_
sr0,_,<

sr0,1
srf,0,>

sr0,0
sr0,0,<
```

```
srf,0
srf,1,>

srf,_
ss0,_,<

//ss es que ahora le va sumar uno a la cadena de la izquierda

ss0,0
ss0,0,<

ss0,1
ss0,1,<

ss0,+
ss1,+,<

ss1,0
s1,1,>

ss1,1
ss1,0,<

ss1,_
s1,1,>

//cuando no haya nada por sumar, simplemente se detiene

s_s0,_
sf0,_,<

sf0,0
sf0,_,<

sf0,+
sf,_,-

// < = left
// > = right
// - = hold
// use underscore for blank cells

//States and symbols are case-sensitive

//Load your code and click COMPILE.
//or load an example (top-right).
```

Ejercicio 3.1.4
Codifique una máquina de Turing que sume dos números en binario.

```
name: resta_numeros_1
init: s0
accept: sf

s0,_
s0,_,>

s0,0
s1,0,>

s1,0
s1,0,>

s0,1
s1,1,>

s1,1
s1,1,>

//operaciones de resta

//sr0 es estado resta inicial

//srf es estado resta final

s1,_
sr0,_,<

sr0,1
srf,0,>

sr0,0
sr0,0,<

srf,0
srf,1,>

srf,_
sf,_,-

// < = left
// > = right
// - = hold
// use underscore for blank cells

//States and symbols are case-sensitive

//Load your code and click COMPILE.
//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.

```
name: copiar_cadena
init: s0
accept: sf

//se empieza a mover y marca el inicio de la cadena

s0,_
s0,_,>

s0,0
s1,0,<

s2,0
s2,0,>

s0,1
s1,1,<

s2,1
s2,1,>

s1,_
s2,|,>

s2,0
s2,0,>

s2,1
s2,1,>

//coloca el inicio de la copia de la cadena

s2,_
sd,c,<

sd,0
sd,0,<

sd,1
sd,1,<

sd,2
sd,2,<

sd,3
sd,3,<

sd,c
sd,c,<

```

```
sd,|
sdp,|,>

//deteccion de si es 0 o 1

sdp,0
sdp0,2,>

sdp,1
sdp1,3,>

sdp,2
sdp,2,>

sdp,3
sdp,3,>

//movimiento a la derecha para colocar 0 o 1

sdp0,0
sdp0,0,>

sdp0,1
sdp0,1,>

sdp1,0
sdp1,0,>

sdp1,1
sdp1,1,>

//detecta la copia

sdp0,c
sdp0,c,>

sdp1,c
sdp1,c,>

sdp0,_
sd,0,<

sdp1,_
sd,1,<

//detecta que ya debe terminar

sdp,c
scam,c,<

scam,2
scam,0,<
```

```

scam,3
scam,1,<

scam,|
sf,_,-
```

Ejercicio 3.1.6
Programar una máquina de Turing que haga el producto de dos números.

```
name: producto_numeros
init: p0
accept: pf

//input: [n]_2*[m]_2

//empieza el movimiento

p0,_
p0,_,>

p0,0
p1,0,<

p0,1
p1,1,<

p1,_
p2,|,>

p2,0
p2,0,>

p2,1
p2,1,>

p2,*
p2,*,>

//llego al final de la cadena

p2,_
sd,c,<

//PARTE PRIMERA COPIA

sd,0
sd,0,<

sd,1
sd,1,<
```

```

sd,2
sd,2,<

sd,3
sd,3,<

sd,*
sd,*,<

sd,c
sd,c,<

sd,|
sdp,|,>

//deteccion de si es 0 o 1

sdp,0
sdp0,2,>

sdp,1
sdp1,3,>

sdp,2
sdp,2,>

sdp,3
sdp,3,>

sdp,c
sdp,c,>

//movimiento a la derecha para colocar 0 o 1

sdp0,0
sdp0,0,>

sdp0,1
sdp0,1,>

sdp1,0
sdp1,0,>

sdp1,1
sdp1,1,>

//detecta la copia

sdp0,*
sdp0,*,>

```

```
sdp1,*
sdp1,*,>

sdp0,c
sdp0,c,>

sdp1,c
sdp1,c,>

sdp0,_
sd,0,<

sdp1,_
sd,1,<

//detecta que ya debe terminar y elimina los cambios que hizo

sdp,*
scam,*,<

scam,2
scam,0,<

scam,3
scam,1,<

scam,|
sf,_,>

//segunda copia

sf,0
sf,0,>

sf,1
sf,1,>

sf,*
sf,*,>

sf,c
sf,c,>

//ahora, hace la segunda copia

//coloca el inicio de la copia de la cadena

sf,_
rd,d,<

rd,0
rd,0,<
```

```
rd,1
rd,1,<

rd,2
rd,2,<

rd,3
rd,3,<

rd,d
rd,d,<

rd,c
rdp,c,>

//deteccion de si es 0 o 1

rdp,0
rdp0,2,>

rdp,1
rdp1,3,>

rdp,2
rdp,2,>

rdp,3
rdp,3,>

//movimiento a la derecha para colocar 0 o 1

rdp0,0
rdp0,0,>

rdp0,1
rdp0,1,>

rdp1,0
rdp1,0,>

rdp1,1
rdp1,1,>

//detecta la copia

rdp0,d
rdp0,d,>

rdp1,d
rdp1,d,>
```

```
rdp0,_
rd,0,<

rdp1,_
rd,1,<

//detecta que ya debe terminar

rdp,d
rcam,d,<

rcam,2
rcam,0,<

rcam,3
rcam,1,<

rcam,c
rf,c,<

rf,0
rf,0,<

rf,1
rf,1,<

rf,*
rf,*,<

//ahora ya puede empezar a sumar uno por uno

rf,_
rs,_,>

rs,0
rs,0,>

rs,1
rs,1,>

rs,*
rs,*,>

rs,c
rs,c,>

rs,d
rs,d,>

//hace la primera suma

rs,_
```

```
rsr,_,<

rsr,1
rss,0,>

rss,0
rss,1,>

rss,_
Rf,_,<

//en Rf

Rf,0
Rf,0,<

Rf,1
Rf,1,<

Rf,c
Rf,c,<

Rf,d
Rf,d,<

//ahora, si detecta el * es pq ahora tiene que sumar

Rf,*
estSum,*,<

estSum,0
estSumAcaba,1,>

estSum,_
estSumAcaba,1,>

estSum,1
estSum,0,<

//aqui acabo de sumar

rsr,0
RF,0,-

//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 $F:\mathbb{N}\to\mathbb{N}$ la función:

$$F(n)=\left\{\begin{array}{ccr} 1 & \text{si} & \text{conjetura Gölbach es verdadera.}\\ 0 & \text{en} & \text{caso contrario.}\end{array}\right.$$

esta función es computable.

Ejemplo 3.1.10
La funcion $f:\mathbb{N}\to\mathbb{N}$ tal que:

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

en efecto, se tiene el siguiente algoritmo de f:

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

3.2. Máquina Universal de Turing

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

Observación 3.2.1
Una máquina de Turing es un conjunto finito $\{a_1,...,a_n\}$, donde $a_i=(s_{i_1},t_{i_2},s_{i_3},t_{i_4},l)$ con $l\in\{<,-,>\}$. Podemos hacer una codificación estas 6-tuplas:

0	-	0
1	-	—
2	-	*
3	-	¿
⋮	-	⋮

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

Teorema 3.2.1
El conjunto

$$\left\{x\in X\Big|x\text{ es máquina de Turing}\right\}$$

y,

$$\left\{n\in\mathbb{N}\Big|n\text{ codifica una máquina de Turing}\right\}$$

son computables.

Demostración:
Ver simulador de máquina de Turing. ■

Definición 3.2.1
Denotaremos la **máquina universal de Turing** por φ, donde

$$\varphi :; \mathbb{N}^2 \to \mathbb{N}$$

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 $\varphi(e,-) : \mathbb{N} \to \mathbb{N}$ 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:

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

pero,

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

Ejemplo 3.2.1
Se define la función **Busy Beaver**, $\sigma : \mathbb{N} \to \mathbb{N}$ dada por:

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

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 $f : \mathbb{N} \to \mathbb{N}$ 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 $g :; \mathbb{N} \to \mathbb{N}$ dada por:

$$g(n) = \text{mín}\left\{ f(2n), f(2n+1) \right\} + 1$$

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) $\left\{ n \in \mathbb{N} \middle| n \text{ es cuadrado perfecto} \right\}$.

(4) $\left\{ p \in \mathbb{N} \middle| p \text{ es primo} \right\}$.

(5) $\left\{ (n, m) \in \mathbb{N}^2 \middle| n \mid m \right\}$.

Solución:

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

```
int uno(int n) return 1;

int dos(int n) return 0;

int tres(int n){
    for(int i=0; i*i<n;i++){
        if(i*i=n) return 1;
    }
    return 0;
}
```

□

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:

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

para todo $n \in \mathbb{N}$. ■

Teorema 3.3.1
El conjunto
$$\left\{e \in \mathbb{N} \middle| \varphi(e, -) \text{ es una función total}\right\}$$
no es computable.

Demostración:
En caso contrario, se tendría que la funcioń $h : n \mapsto n$-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 $u :; \mathbb{N}^2 \to \mathbb{N}$:
$$(x, y) \mapsto \varphi(h(x), y)$$
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, Enschleidung problem**)
El conjunto
$$H = \left\{(e, n) \middle| \varphi(e, n) \text{ está definido}\right\}$$
(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
$$D = \left\{e \in \mathbb{N} \middle| (e, e) \downarrow\right\}$$
es computable. Definimos la siguiente función $f :; \mathbb{N} \to \mathbb{N}$ con algoritmo:

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

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

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

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 $F : \mathbb{N} \to \left\{ F \subseteq \mathbb{N} \middle| F \text{ es finito} \right\}$ 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:

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

$\blacksquare$

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:

$$H=\left\{(e,n)\middle|\varphi(e,n)\downarrow\right\}$$

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:

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

Lo cual lo hacemos de la siguiente manera:

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

$\blacksquare$

Observación 3.4.3
Note que H no es computable, pero el conjunto:

$$\left\{(e,n,t)\middle|\varphi(e,n)\text{ se detiene en }\leq t\text{ pasos}\right\}$$

sí es computable.

Teorema 3.4.1
Sea $X\subseteq\mathbb{N}$. Entonces las siguientes son equivalentes:

(1) X es computablemente numerable.

(2) La función semicaracterística σ_X dada por:

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

es computable.

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

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

$$X = \left\{ x \in \mathbb{N} \middle| (\exists y \in \mathbb{N})((x,y) \in Y) \right\}$$

Demostración:

(1)⇒*(2)*: Veamos que existe el siguiente algoritmo para la función semicaracterística:

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

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

(3)⇒*(4)*: Considere el algoritmo de la función característica de Y, χ_Y:

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

note que Y es precisamente:

$$Y = \left\{ (a,b) \middle| f(a) \text{ se detiene en } \leq b \text{ pasos} \right\}$$

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

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

(4)⇒*(1)*: Considere el algoritmo de la función característica de X:

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

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:

```
int chi_X(int n){
    for(int i = 0; 1; i++){
        correr X en i pasos;
            if(se imprimio n) return 1;
        correr N\X en i pasos;
        if(se imprimio n) return 0;
    }
}
```

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

Corolario 3.4.1
Los conjuntos:

$$\mathbb{N}^2 \setminus H = \left\{(e,d) \middle| \varphi(e,d)\uparrow\right\}$$

y,

$$\mathbb{N} \setminus D = \left\{n \in \mathbb{N} \middle| (n,n)\uparrow\right\}$$

no son computablemente enumerables.

Demostración:

Es inmediata del teorema de Kleene. $\blacksquare$

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. $\blacksquare$

Teorema 3.4.3
Sea $f :; \mathbb{N} \to \mathbb{N}$ una función parcial.

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

$$\Gamma(f) = \left\{(x,y) \in \mathbb{N}^2 \middle| y = f(x)\right\}$$

(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:

```
void main(void){
    for(int a = 0; 1; a++){
        Decodificar a=(n,t);
        Correr f(n) t pasos{
            si el algoritmo termino{
                print(n,f(n));
            }
        }
    }
}
```

$\Leftarrow$): Considere el algoritmo:

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

De *(2)*:

$\Rightarrow$): Considere el algoritmo:

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

$\Leftarrow$): Considere el algoritmo:

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

$\blacksquare$

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 $f : \mathbb{N} \to \mathbb{N}$ tal que $X = \operatorname{ran}(f) = \operatorname{im}(f)$.

Demostración:

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

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

```
int f(int k){
    switch(k){
        case 0: return x_0;
        ... : ... ;
        case n: return x_n;
    }
    return x_n;
}
```

- X **es infinito**. Considere el algoritmo:

```
int f(int k){
    correr algoritmo que lista los elementos de  X;
    tomar el k-esimo elemento de la lista de X y almacenarlo
       en la variable y;
    return y;
}
```

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

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

$\blacksquare$

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) Demuestrar 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 codificamoss ímbolo sdel 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	–	$2 \cdot 3^n$
$\vdots$	$\vdots$	$\vdots$

$$\begin{array}{ccc}
\hline
R_1 & - & 10 \\
R_2 & - & 50 \\
\vdots & \vdots & \vdots \\
R_n & - & 2 \cdot 5^n \\
\vdots & \vdots & \vdots \\
\hline
\end{array}$$

$$\begin{array}{ccc}
\hline
c_1 & - & 14 \\
c_2 & - & 98 \\
\vdots & \vdots & \vdots \\
c_n & - & 2(7^n) \\
\hline
\end{array}$$

En este sentido, lo hacemos para que el conjunto:

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

es computable.

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

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

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

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

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

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

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

Análogamente, podemos codificar sucesoines finitas de cuesiones 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:

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

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

y,

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

son conjuntos computables.

Demostración:

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

Teorema 4.2.2

Los conjuntos:

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

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:

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

es computable.

Demostración:

Inmediato del teorema anterior. ■

Corolario 4.2.2

El conjunto:

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

es computablemente enumerable.

Demostración:

Considere:

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

donde $P(a, b)$ es la propiedad: b codifica el número de Gödel de φ, $a \in$ Dem y el código $(a, \varphi) \in$ 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:

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

es computablemente enumerable. ■

Teorema 4.2.3
Sea Σ un conjunto computable de fórmulas.

(1) Sea:

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

es un conjunto computable.

(2) El conjunto:

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

es un conjunto computablemente enumerable.

Demostración:
Análoga al teorema anterior. ■

Teorema 4.2.4
Sea Σ un conjunto computablemente enumerable de fórmulas.

(1) Sea:

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

es un conjunto computablemente enumerable.

(2) El conjunto:

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

es un conjunto computablemente enumerable.

Demostración:
Análoga al teorema anteanterior. ■

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

Teorema 4.2.5
Si una función f es parcial computable, entonces hay una fórmula φ tal que $\Gamma(f) = \left\{ (l, m) \middle| \mathbb{N} \vDash \varphi[n, m] \right\}$.

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:

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

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

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

El código de $(m_0, ..., m_{k-1})$ 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 $p \mid 1\#_c$. 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 $i = 0, ..., k-1$ 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 $i = 0, ..., k-1$ 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 $i = 0, ..., k-1$, 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:

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

De lo contrario, por el lema anterior $q = q_j$ para algun $j = 0, ..., k-1$, entonces q codifica tanto a (m, j) como a (m_i, i). Por la inyectividad de esta codificación se sigue que $i = j$ y $m = m_i = m_j\#_c$. 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 $i, \alpha, m \in \mathbb{N}$:

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

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 llego 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:

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

es válida siempre que $y = f(x)$. ■

Observación 4.2.2
El macro $a \mid b$ abrevia la fórmula $(\exists c)(a \cdot c = b)$.

Recordemos el ideal de Hilbert: Encontrar un conjunto Γ tal que:

(1) Computable.

(2) Si $\varphi \in \Gamma$ entonces $\mathbb{N} \vDash \varphi$.

(3) Completa.

(4) *(Opcional)*. $\Gamma = \text{Teor}(\Sigma)$ para un conjunto Σ computable.

Teorema 4.2.6 (**Indecibilidad de la Aritmética**)
El conjunto:

$$\text{Th}(\mathbb{N}) = \left\{ n \in \mathbb{N} \middle| n \text{ es el número de Gödel de una } \varphi \text{ tal que } \mathbb{N} \vDash \varphi \right\}$$

no es computable, es decir habrá fórmulas para las cuáles no se podrá verificar con un algoritmo si éstas son verdaderas o no.

Demostración:

Supongamos que Th($\mathbb{N}$) sí es computable. Sea $\varphi_1, ..., \varphi_n, ...$ una enumeración computable de todas las fórmulas de la aritmética $\mathcal{L}_A$ con una variable libre.

Sea:

$$X = \left\{m \in \mathbb{N} \middle| \mathbb{N} \vDash \varphi_m[m]\right\}$$

Este conjunto es computable por ser Th($\mathbb{N}$) computable, por tanto del teorema anterior existe una fórmula $\psi(x, y)$ de $\mathcal{L}_A$ tal que:

$$\left\{(n,1) \in \mathbb{N}^2 \middle| n \in X\right\} \cup \left\{(n,0) \in \mathbb{N}^2 \middle| n \notin X\right\} = \Gamma(\chi_X) = \left\{(n,m) \in \mathbb{N}^2 \middle| \mathbb{N} \vDash \psi[n,m]\right\}$$

Considere la fórmula $\neg\psi[1/y]$. Note que:

(1) Esta fórmula tiene una variable libre (por tanto, es igual a φ_e para algún $e \in \mathbb{N}$).

(2) $X = \left\{n \in \mathbb{N} \middle| \mathbb{N} \vDash \neg\varphi_e[n]\right\}$.

Entonces:

$$\begin{aligned} \mathbb{N} \vDash \neg\varphi_e[e] &\text{ si y sólo si } \mathbb{N} \vDash \neg\neg\psi[1/y][e] \\ &\text{ si y sólo si } \mathbb{N} \vDash \psi[1/y][e] \\ &\text{ si y sólo si } \mathbb{N} \vDash \psi[e,1] \\ &\text{ si y sólo si } e \in X \\ &\text{ si y sólo si } \mathbb{N} \vDash \varphi_e[e] \end{aligned}$$

una contradicción. Por tanto, Th($\mathbb{N}$) no es computable. ■

Teorema 4.2.7 (**Indefinibilidad de la verdad de Tarski**)
El conjunto:

$$\mathrm{Th}(\mathbb{N}) = \left\{n \in \mathbb{N} \middle| n \text{ es el número de Gödel de una } \varphi \text{ tal que } \mathbb{N} \vDash \varphi\right\}$$

no es definible.

Demostración:

Suponga que si es definible, entonces existe una fórmula $\psi(x, y)$ tal que ... (ver la demostración anterior). ■

Corolario 4.2.3
Si $\Sigma \subseteq$ Th($\mathbb{N}$) es computable, entonces Σ no es una teoría completa (es decir, o demuestras φ o $\neg\varphi$ para toda fórmula φ en $\mathcal{L}_A$).

Demostración:

Suponga que existe $\Sigma \subseteq$ Th($\mathbb{N}$) computable y completa. Entonces:

(1) Teor(Σ) sería computable.

(2) Teor(Σ) $\subseteq$ Th($\mathbb{N}$).

por *(2)* al ser Teor(Σ) completa, se sigue que Teor(Σ) = Th($\mathbb{N}$), pues si $\varphi \notin$ Teor(Σ) entonces $\neg \in$ Th(Σ), así que $\mathbb{N} \vDash \neg\varphi$ lo cual implica que $\varphi \notin$ Th($\mathbb{N}$).

Por ende, Teor(Σ) = Th($\mathbb{N}$), donde el conjunto de la derecha es computable y el de la izquierda no$\#_c$. Por ende, Σ no es completo. ■

4.3. Dos teorías en $\mathcal{L}_A$

Adotaremos los siguientes axiomas:

Definición 4.3.1 (**Axiomas de Peano**)
Se definen los siguientes axiomas:

(1) $\forall x \neg(\mathbb{1} = Sx)$.

(2) $\forall x \forall y(Sx = Sy \Rightarrow x = y)$.

(3) $\forall x \neg(x < \mathbb{1})$.

(4) $\forall x \forall y(x < Sy \Rightarrow (x < y \vee x = y))$.

(5) $\forall x \forall y(x < y \vee x = y \vee y < x)$.

(6) $\forall x(x + \mathbb{1} = Sx)$.

(7) $\forall x \forall y(x + Sy = S(x + y))$.

(8) $\forall x(x \cdot \mathbb{1} = x)$.

(9) $\forall x \forall y(x \cdot Sy = x \cdot y + x)$.

los axiomas *(1)* a *(2)* son llamados **de sucesor**, los *(3)* a *(5)* son llamados **de orden**, los *(6)* a *(7)* son llamados **de suma** y los *(8)* a *(9)* son llamados **de producto**. Se define además el siguiente axioma:

(10) *(Esquema)*. Para cada fórmula $\varphi(x)$ de $\mathcal{L}_A$ con una variable libre: $\varphi[1/x] \wedge (\forall x)(\varphi \Rightarrow \varphi[Sx/x]) \Rightarrow \forall x \varphi$.

La **Aritmética de Peano** (abreviado por PA), es el conjunto:

$$\mathrm{PA} = \{\textit{(1)}, \textit{(2)}, ..., \textit{(9)}, \textit{(10a)}, \textit{(10b)}, ...\}$$

Además, se define la **Aritmética de Robinson**:

$$\mathrm{Q} = \mathrm{RA} = \{\textit{(1)}, \textit{(2)}, ..., \textit{(9)}\}$$

En la aritmética de Robinson es posible probar que dados dos números, su suma conmuta, pero como no hay inducción no es posible probar la fórmula:

$$\forall x \forall y(x + y = y + x)$$