

# Lenguajes Formales, Autómatas y Computabilidad

Clase Teórica: Funciones parciales computables y la función Halt

Segundo Cuatrimestre 2025

## **Bibliografía**

*Introduction to Automata Theory, Languages and Computation*, J. Hopcroft, R. Motwani, J. Ullman, Second Edition, Addison Wesley, 2001. Capítulo 8 y 9.

# Codificación de pares

Definimos la función

$$\langle x, y \rangle = 2^x(2 \cdot y + 1) - 1$$

Notar que  $2^x(2 \cdot y + 1) \neq 0$ .

## Proposición

*Hay una única solución  $(x, y)$  a la ecuación  $\langle x, y \rangle = z$ .*

## Demostración.

- ▶  $x$  es el máximo número tal que  $2^x | (z + 1)$
- ▶  $y = ((z + 1)/2^x - 1)/2$



# Observadores de pares

Los **observadores** del par  $z = \langle x, y \rangle$  son

- ▶  $\ell(z) = x$
- ▶  $r(z) = y$

## Proposición

*Los observadores de pares son totales computables.*

## Demostración.

Como  $x, y < z + 1$  tenemos que

- ▶  $\ell(z) = \min_{x \leq z} ((\exists y)_{\leq z} z = \langle x, y \rangle)$
- ▶  $r(z) = \min_{y \leq z} ((\exists x)_{\leq z} z = \langle x, y \rangle)$



Por ejemplo,

- ▶  $\langle 2, 5 \rangle = 2^2(2 \cdot 5 + 1) - 1 = 43$
- ▶  $\ell(43) = 2$
- ▶  $r(43) = 5$

# Codificación de secuencias

El **número de Gödel** de la secuencia de números naturales

$$a_1, \dots, a_n$$

es el número

$$[a_1, \dots, a_n] = \prod_{i=1}^n p_i^{a_i},$$

donde  $p_i$  es el  $i$ -ésimo primo ( $i \geq 1$ ).

Por ejemplo el número de Gödel de la secuencia

$$1, 3, 3, 2, 2$$

es

$$[1, 3, 3, 2, 2] = 2^1 \cdot 3^3 \cdot 5^3 \cdot 7^2 \cdot 11^2 = 40020750.$$

# Propiedades de la codificación de secuencias

## Teorema

Si  $[a_1, \dots, a_n] = [b_1, \dots, b_n]$  entonces  $a_i = b_i$  para todo  $i \in \{1, \dots, n\}$ .

## Demostración.

Por la factorización única en primos.



Observar que

$$[a_1, \dots, a_n] = [a_1, \dots, a_n, 0] = [a_1, \dots, a_n, 0, 0] = \dots$$

pero

$$[a_1, \dots, a_n] \neq [0, a_1, \dots, a_n]$$

# Observadores de secuencias

Los **observadores** de la secuencia  $x = [a_1, \dots, a_n]$  son

- ▶  $x[i] = a_i$
- ▶  $|x| = \text{longitud de } x$

## Proposición

*Los observadores de secuencias son totales computables.*

## Demostración.

- ▶  $x[i] = \min_{t \leq x} (\neg p_i^{t+1} | x)$
- ▶  $|x| = \min_{i \leq x} (x[i] \neq 0 \wedge (\forall j)_{\leq x} (j \leq i \vee x[j] = 0))$

Por ejemplo,

- ▶  $[1, 3, 3, 2, 2][2] = 3 = 40020750[2]$
- ▶  $[1, 3, 3, 2, 2][6] = 0 = 40020750[6]$
- ▶  $|[1, 3, 3, 2, 2]| = 5 = |40020750|$
- ▶  $|[1, 3, 3, 2, 2, 0]| = |[1, 3, 3, 2, 2, 0, 0]| = 5 = |40020750|$
- ▶  $x[0] = 0$  para todo  $x$
- ▶  $0[i] = 0$  para todo  $i$



# En resumen: codificación y decodificación de pares y secuencias

## Teorema (Codificación de pares)

- ▶  $\ell(\langle x, y \rangle) = x, r(\langle x, y \rangle) = y$
- ▶  $z = \langle \ell(z), r(z) \rangle$
- ▶  $\ell(z), r(z) \leq z$
- ▶ *la codificación y observadores de pares son p.c.*

## Teorema (Codificación de secuencias)

- ▶  $[a_1, \dots, a_n][i] = \begin{cases} a_i & \text{si } 1 \leq i \leq n \\ 0 & \text{si no} \end{cases}$
- ▶ *si  $n \geq |x|$  entonces  $[x[1], \dots, x[n]] = x$*
- ▶ *la codificación y observadores de secuencias son p.c.*

Usaremos variables en  $\mathbb{N}$ ,  $Y, X_1, Z_1, X_2, Z_2, \dots, X_k, Z_k$  donde  $Y$  es variable de salida, las  $X_i$  son de entrada y las  $Z_i$  son auxiliares.

Damos un lenguaje de programación  $\mathcal{S}^{++}$  con 4 de instrucciones:

- ▶  $V++$
- ▶  $V--$
- ▶ **while** ( $V \neq 0$ ) **do** {  
     $P$   
}
- ▶ **pass**

Observar que **pass** no tiene ninguna variable. Mientras que  $V++$ ,  $V--$  y la condición de **while** ( $V \neq 0$ ) { tienen exactamente una variable  $V$ . Tenemos que  $P$  es una lista de instrucciones, seguida de }

# Estados

Un **estado** de un programa  $P$  es una lista de igualdades de la forma  $V = m$  (donde  $V$  es una variable y  $m$  es un número) tal que

- ▶ hay una igualdad para cada variable que se usa en  $P$
- ▶ no hay dos igualdades para la misma variable

El **estado inicial** de  $P$  dados  $r_1, \dots, r_m$  es el estado que tiene

$$X_1 = r_1 \quad , \quad X_2 = r_2 \quad , \quad \dots \quad , \quad X_m = r_m \quad , \quad Y = 0$$

junto con

$$V = 0$$

para cada variable  $V$  que aparezca en  $P$  y no sea  $X_1, \dots, X_m, Y$ .

# Descripción instantánea

Supongamos que el programa  $P$  tiene  $n$  instrucciones.

Llamamos **descripción instantánea** a cada par  $(i, \sigma)$ , donde  $i$  es un número de instrucción y  $\sigma$  es un estado.

La descripción inicial es  $(1, \sigma)$ , para  $\sigma$  el estado inicial.

Una **descripción terminal** es  $(n + 1, \sigma)$ , para algun estado  $\sigma$ .

Si  $(i, \sigma)$  no es terminal, tiene un **sucesor**  $(j, \tau)$  donde:

Si la  $i$ -ésima instrucción de  $P$  es  $V++$ .

- ▶  $j = i + 1$
- ▶  $\tau$  tiene las mismas igualdades que  $\sigma$ , salvo que  $V = m$  se reemplaza por  $V = m + 1$

Si la  $i$ -ésima instrucción de  $P$  es  $V--$ .

- ▶  $j = i + 1$
- ▶  $\tau$  tiene las mismas igualdades que  $\sigma$ , salvo que  $V = m$  se reemplaza por  $V = \max\{m - 1, 0\}$

# Descripción instantánea

Si la  $i$ -ésima instrucción de  $P$  es **while** ( $V \neq 0$ ) **do**  $\{R\}$ , donde  $R$  tiene  $r$  instrucciones,

- ▶ si  $\sigma$  tiene  $V = 0$  entonces  $j = i + r + 1$  y  $\tau$  tiene las mismas igualdades que  $\sigma$  ;
- ▶ si  $\sigma$  tiene  $V = m$  para  $m \neq 0$  entonces  $j = i$  y  $\tau$  es el estado de la configuración terminal de  $R$  que se obtiene a partir de la configuración inicial para  $R$  determinada por la primera instrucción de  $R$  y el estado  $\sigma$ .

# Cómputos

Un **cómputo** de un programa  $P$  a partir de una descripción instantánea  $d_1$  es una lista

$$d_1, d_2, \dots, d_k$$

de descripciones instantáneas de  $P$  tal que

- ▶  $d_{i+1}$  es sucesor de  $d_i$  para  $i \in \{1, 2, \dots, k-1\}$
- ▶  $d_k$  es terminal

# Las funciones $\Psi_P^{(m)}(r_1, \dots, r_m)$

Sea  $P$  un programa y sean

- ▶  $r_1, \dots, r_m$  números dados
- ▶  $\sigma_1$  el estado inicial

Definimos la función  $\Psi_P^{(m)}$

Hay dos posibilidades:

Si  $P$  tiene un cómputo  $d_1, \dots, d_k$  tal que  $d_1 = (1, \sigma_1)$  entonces

$\Psi_P^{(m)}(r_1, \dots, r_m) = Y$ , donde  $Y$  es la variable de salida en  $d_k$ .

Decimos que  $\Psi_P^{(m)}(r_1, \dots, r_m)$  está definido, escribimos

$\Psi_P^{(m)}(r_1, \dots, r_m) \downarrow$ .

Si no hay tal cómputo: existe una secuencia infinita  $d_1, d_2, d_3, \dots$  donde

- ▶  $d_1 = (1, \sigma_1)$ .
- ▶  $d_{i+1}$  es sucesor de  $d_i$ .

Decimos que  $\Psi_P^{(m)}(r_1, \dots, r_m)$  está indefinido, escribimos

$\Psi_P^{(m)}(r_1, \dots, r_m) \uparrow$

## Teorema

Una función  $f : \mathbb{N}^m \rightarrow \mathbb{N}$  es **parcial computable** exactamente cuando existe un programa  $P$  en  $\mathcal{S}^{++}$  tal que para todo  $(r_1, \dots, r_m) \in \mathbb{N}^m$ ,

$$f(r_1, \dots, r_m) \downarrow, \Psi_P^{(m)}(r_1, \dots, r_m) \downarrow \text{ y } f(r_1, \dots, r_m) = \Psi_P^{(m)}(r_1, \dots, r_m)$$

o

$$f(r_1, \dots, r_m) \uparrow \text{ y } \Psi_P^{(m)}(r_1, \dots, r_m) \uparrow$$

Para demostrarlo debemos ver:

- 1) Que todas las funciones parciales computables se pueden definir en  $\mathcal{S}^{++}$ . Es decir, las funciones iniciales, composición, recursión primitiva, minimización acotada y no acotada, se pueden definir en  $\mathcal{S}^{++}$ .
- 2) Que todos los programas en  $\mathcal{S}^{++}$  son funciones parciales computables.

A continuación vemos el punto 1).

El punto 2) resulta evidente de la definición de  $\mathcal{S}^{++}$ .

## Demostración (las funciones parciales computables se definen en $\mathcal{S}^{++}$ )

### Funciones iniciales

$n(x) = 0$  se computa con macro

$Y \leftarrow 0$

que es el programa en  $\mathcal{S}^{++}$

**while** ( $Y \neq 0$ ) **do** {

$Y - -$

}

$s(x) = x + 1$  se computa con el programa en  $\mathcal{S}^{++}$

$X ++$

$Y \leftarrow X$

$u_i^n(x_1, \dots, x_n) = x_i$  se computa con la macro

$Y \leftarrow X_i$  que es el programa en  $\mathcal{S}^{++}$

$Y \leftarrow 0$

**while** ( $X \neq 0$ ) **do** {

$X - -$

$Y ++$

}

## Minimización no acotada

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  es un predicado computable, que suponemos tiene un programa en  $\mathcal{S}^{++}$ .

El siguiente programa en  $\mathcal{S}^{++}$  computa  $\min_t p(t, x_1, \dots, x_n)$ :

**while**  $(p(Y, X_1, \dots, X_n) \neq 1)$  **do**  $\{ Y++ \}$

## Clausura por composición

Asumamos que  $f$  y  $g_1, \dots, g_k$  ya las tenemos en  $\mathcal{S}^{++}$ .

Si  $h$  se obtiene a partir de las funciones (parcialmente) computables  $f, g_1, \dots, g_k$  por composición.

La función  $h$  se define en  $\mathcal{S}^{++}$  :

El siguiente programa computa  $h$ :

$$Z_1 \leftarrow g_1(X_1, \dots, X_n)$$

$$\vdots$$

$$Z_k \leftarrow g_k(X_1, \dots, X_n)$$

$$Y \leftarrow f(Z_1, \dots, Z_k)$$

Si  $f, g_1, \dots, g_k$  son totales entonces  $h$  es total.

## Clausura por recursión (corregida respecto a la dada en clase)

Supongamos  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  se obtiene a partir de  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  y  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$  por recursión primitiva y  $f$  y  $g$  son parcial computables:

$$\begin{aligned}h(0, x_1, \dots, x_n) &= f(x_1, \dots, x_n), && \text{caso base} \\h(t+1, x_1, \dots, x_n) &= g(t, h(x_1, \dots, x_n), x_1, \dots, x_n)\end{aligned}$$

Supongamos tenemos un programa en  $\mathcal{S}^{++}$  para  $f$  y para  $g$ .

El siguiente programa en  $\mathcal{S}^{++}$  computa  $h$ .

Aquí  $Y$  hace el rol de  $h(\cdot)$ ,  $X_1$  el de  $t+1$ , y  $Z_1$  varía desde 0 hasta  $t$ .

```
Y ← f(X2, ... Xn+1)
Z1 ← 0
while (X1 ≠ 0) {
  Y ← g(Z1, Y, X2, ... Xn+1)
  Z1 ++
  X1 --
}
```

El programa primero computa  $h(0)$ , luego  $h(1)$ , luego ... hasta  $h(t+1)$

No es importante

- ▶ qué base usamos para representar a los números
  - ▶ usamos representación unaria ( $\Sigma = \{\mathbf{B}, 1\}$ )
  - ▶ pero podríamos haber elegido la binaria ( $\Sigma = \{\mathbf{B}, 0, 1\}$ )
  - ▶ o base 10 ( $\Sigma = \{\mathbf{B}, 0, 1, 2, \dots, 9\}$ )
- ▶ si permitimos que al terminar la cinta tenga otras cosas escritas además de la salida o solo contenga la salida
- ▶ si usamos esta variante de arquitectura:
  - ▶ una cinta de entrada (solo de lectura)
  - ▶ una cinta de salida (solo de escritura)
  - ▶ una o varias cintas de trabajo, de lectura/escritura

¡Siempre computamos la misma clase de funciones!

## Teorema

Sea  $f : \mathbb{N}^m \rightarrow \mathbb{N}$  una función parcial. Son equivalentes:

1.  $f$  es parcial computable en Python
2.  $f$  es parcial computable en C
3.  $f$  es parcial computable en Haskell
4.  $f$  es parcial Turing computable

# Tipos de datos en $\mathcal{S}^{++}$

- ▶ vimos que el único tipo de dato en  $\mathcal{S}^{++}$  son los naturales
- ▶ sin embargo podemos simular otros tipos. Por ejemplo,
  - ▶ **tipo bool**: lo representamos con el 1 (verdadero) y el 0 (falso)
  - ▶ **tipo par de números naturales**: la codificación y decodificación de pares son funciones totales computables
  - ▶ **tipo entero**: podría ser codificada con un par

$\langle \text{bool}, \text{número natural} \rangle$

- ▶ **tipo secuencias finitas de números naturales**: la codificación y decodificación de secuencias son totales computables
- ▶ ahora vamos a ver como simular el **tipo programa en  $\mathcal{S}^{++}$**

# Codificación de programas en $\mathcal{S}^{++}$

Recordemos que las instrucciones de  $\mathcal{S}^{++}$  eran:

1.  $V++$
2.  $V--$
3. **while** ( $V \neq 0$ ) **do**  $\{P\}$
4. **pass**

Observar que **pass** no tiene ninguna variable. Mientras que  $V++$ ,  $V--$  y la condición de **while** ( $V \neq 0$ )  $\{$  tienen exactamente una variable  $V$ . Tenemos que  $P$  es una lista de instrucciones, seguida de  $\}$

# Codificación de variables de $\mathcal{S}^{++}$

Ordenamos las variables:

$$Y, X_1, Z_1, X_2, Z_2, X_3, Z_3, \dots$$

Escribimos  $\#(V)$  para la posición que ocupa la variable  $V$  en la lista.

Por ejemplo,

- ▶  $\#(Y) = 0$
- ▶  $\#(X_1) = 1$
- ▶  $\#(Z_1) = 2$
- ▶  $\#(X_2) = 3$

# Codificación de instrucciones de $\mathcal{S}^{++}$

Definimos la función  $\#$  : expresiones de  $\mathcal{S}^{++} \rightarrow \mathbb{N}$

$$\#(\mathbf{pass}) = 0$$

$$\#(I) = \langle a, b \rangle + 1, \text{ para } I \neq \mathbf{pass}$$

donde

$a = \#(V)$  donde  $V$  es la variable mencionada en  $I$ ,

si  $V$  es variable de salida  $Y$ ,  $\#(Y) = 0$

si  $V$  es variable de entrada  $X_i$ ,  $\#(X_i) = 2i - 1$

si  $V$  es variable local  $Z_i$ ,  $\#(Z_i) = 2i$

$b = 0$  si  $I$  es  $V++$

$b = 1$  si  $I$  es  $V--$

$b = \#(P) + 2$  si  $I$  es **while**  $V \neq 0$  **do**  $\{P\}$

Todo número  $x$  representa a una única instrucción  $I$ .

# Codificación de programas en $\mathcal{S}^{++}$

Un programa  $P$  es una lista (finita) de instrucciones  $I_1, \dots, I_k$

Codificamos al programa  $P$  con

$$\#(P) = [\#(I_1), \dots, \#(I_k)]$$

Prohibimos los programas que terminan con la instrucción **pass** y con esto cada número representa a un **único** programa.

# Hay más funciones $\mathbb{N} \rightarrow \mathbb{N}$ que números naturales

## Teorema (Cantor)

*El conjunto de las funciones (totales)  $\mathbb{N} \rightarrow \mathbb{N}$  no es numerable.*

## Demostración.

Supongamos que lo fuera. Las enumero:  $f_0, f_1, f_2 \dots$

valores de $f_0$	=	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...
valores de $f_1$	=	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	...
valores de $f_2$	=	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	...
$\vdots$						
valores de $f_k$	=	$f_k(0)$	$f_k(1)$	$f_k(2)$	$f_k(3)$	...
$\vdots$						$\ddots$

Defino la siguiente función  $g : \mathbb{N} \rightarrow \mathbb{N}$

$$g(x) = f_x(x) + 1.$$

Para todo  $k$ ,  $f_k \neq g$  (en particular difieren en el punto  $k$ ). Entonces  $g$  no está listada. Absurdo:  $f_0, f_1, f_2 \dots$  era una enumeración de **todas** las funciones  $\mathbb{N} \rightarrow \mathbb{N}$ .

# Hay funciones no computables

- ▶ hay una cantidad no numerable de funciones  $\mathbb{N} \rightarrow \mathbb{N}$
  - ▶ o sea, hay más funciones  $\mathbb{N} \rightarrow \mathbb{N}$  que números naturales
  - ▶ hay tantos programas como números naturales
  - ▶ hay tantas funciones parcial computables como números naturales
  - ▶ tiene que haber funciones  $\mathbb{N} \rightarrow \mathbb{N}$  no parciales computables
- Pero ¿qué ejemplo concreto tenemos?

## El problema de la detención (Halting problem)

$\text{HALT}(x, e) : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}$  es verdadero exactamente cuando el programa con número  $y$  y entrada  $x$  no se indefine,

$$\text{HALT}(x, e) = \begin{cases} 1 & \text{si } \Psi_P^{(1)}(x) \downarrow \\ 0 & \text{si no} \end{cases}$$

donde  $P$  es el único programa tal que  $\#(P) = e$ .

Ejemplo:

Supongamos programa  $P$  busca incrementalmente el primer número par que no es la suma de dos primos. Sea  $\#(P) = e$ .

¿Cuánto vale  $\text{HALT}(x, e)$ ?

$\text{HALT}(x, e) = 1$     sii     $\Psi_P(x) \downarrow$     sii    la conjetura de Goldbach es falsa

# HALT no es computable

Definimos  $\text{HALT}:\mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}$ ,

$$\text{HALT}(x, e) = \begin{cases} 1 & \text{si } \Psi_P(x) \downarrow, \\ 0 & \text{si } \Psi_P(x) \uparrow \end{cases}$$

donde  $e = \#(P)$ .

Notar que HALT es una función: todo par  $x, e$  tiene un valor booleano.

## Teorema (Turing, 1936)

*HALT no es total computable.*

## Demostración (versión ampliada de la clase teórica).

Supongamos que HALT es total computable. Entonces hay un programa en  $\mathcal{S}^{++}$  que computa HALT con dos argumentos de entrada.

Podemos definir este otro programa  $Q$  que escribimos en color azul.

Programa  $Q$

```
 $Z \leftarrow \text{HALT}(X, X)$   
while ( $Z \neq 0$ ) do {pass }
```

Luego,

$$\Psi_Q(x) = \begin{cases} \uparrow & \text{si } \text{HALT}(x, x) \neq 0 \\ 0 & \text{si } \text{HALT}(x, x) = 0 \end{cases}$$

Entonces, por definicion de  $\text{HALT}(x, e)$  (diapositiva anterior), con  $e = \#(Q)$ ,

$$\text{HALT}(x, \#(Q)) = 1 \quad \text{sii} \quad \Psi_Q(x) \downarrow$$

Y, por el programa  $Q$  tenemos

$$\Psi_Q(x) \downarrow \quad \text{sii} \quad \text{HALT}(x, x) = 0$$

Ponemos las dos lineas juntas y nos queda

$$\text{HALT}(x, \#(Q)) = 1 \quad \text{sii} \quad \Psi_Q(x) \downarrow \quad \text{sii} \quad \text{HALT}(x, x) = 0$$

$\#(Q)$  está fijo pero  $x$  es variable. Llegamos a un absurdo con  $x = \#Q$ .



# Diagonalización

En general, sirve para definir una función distinta a muchas otras.

En el caso de  $\text{HALT}(x, e)$ ,

- ▶ sea  $P_i$  el programa con número  $i$
- ▶ supongo que  $\text{HALT}(x, e)$  es computable
- ▶ defino una función  $f$  computable
- ▶ núcleo de la demostración: ver que  $f \notin \{\Psi_{P_0}, \Psi_{P_1}, \Psi_{P_2}, \dots\}$ 
  - ▶ para esto, me aseguro que  $f(x) \neq \Psi_{P_x}(x)$ , en particular:

$f(x) \downarrow$ sii $\Psi_{P_x}(x) \uparrow$	$\Psi_{P_0}(0)$	$\Psi_{P_0}(1)$	$\Psi_{P_0}(2)$	$\dots$
	$\Psi_{P_1}(0)$	$\Psi_{P_1}(1)$	$\Psi_{P_1}(2)$	$\dots$
	$\Psi_{P_2}(0)$	$\Psi_{P_2}(1)$	$\Psi_{P_2}(2)$	$\dots$

- ▶ ¡pero  $f$  era computable! Absurdo: tenía que estar en

$$\{\Psi_{P_0}, \Psi_{P_1}, \Psi_{P_2}, \dots\}.$$

# Tesis de Church

Hay muchos modelos de cómputo.

Está probado que tienen el mismo poder que  $\mathcal{S}^{++}$

- ▶ C
- ▶ Java
- ▶ Haskell
- ▶ máquinas de Turing
- ▶ ...

**Tesis de Church.** Todos los algoritmos para computar en los naturales se pueden programar en  $\mathcal{S}^{++}$ .

Entonces, el problema de la detención dice

*no hay algoritmo para decidir la verdad o falsedad de  $\text{HALT}(x, e)$*