

# Lenguajes Formales, Autómatas y Computabilidad

Clase Teórica: Funciones parcialmente computables y la función Halt

Segundo Cuatrimestre 2024

## **Bibliografía**

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

# Representación de números y tuplas

Fijamos  $\Sigma = \{\mathbf{B}, 1\}$ .

Representaremos a los **números** naturales en unario (con palotes).

- ▶ el número  $x \in \mathbb{N}$  se representa como

$$\overline{x} = \underbrace{1 \dots 1}_{x+1}$$

Representamos a las **tuplas**  $(x_1, \dots, x_n)$  como lista de (representaciones de) los  $x_i$  separados por blanco

- ▶ la tupla  $(x_1, \dots, x_n)$  se representa como

$$\mathbf{B}\overline{x_1}\mathbf{B}\overline{x_2}\mathbf{B}\dots\mathbf{B}\overline{x_n}\mathbf{B}$$

Por ejemplo,

- ▶ el número 0 se representa como 1
- ▶ el número 3 se representa como 1111
- ▶ la tupla (1, 2) se representa como **B11B111B**
- ▶ la tupla (0, 0, 1) se representa como **B1B1B11B**

# Funciones parciales

Una **función parcial**  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  es una función que puede estar indefinida para algunos (tal vez ninguno; tal vez todos) sus argumentos.

$f(x_1, \dots, x_n) \downarrow$  cuando  $f$  está definida para  $x_1, \dots, x_n$ .

En este caso  $f(x_1, \dots, x_n)$  es un número natural.

$f(x_1, \dots, x_n) \uparrow$  cuando  $f$  está indefinida para  $x_1, \dots, x_n$

El conjunto de argumentos para los que  $f$  está definida se llama **dominio** de  $f$ , notado  $\text{dom}(f)$ .

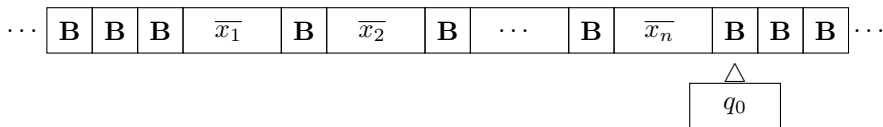
$$\text{dom}(f) = \{(x_1, \dots, x_n) : f(x_1, \dots, x_n) \downarrow\}$$

La función  $f$  es **total** si  $\text{dom}(f) = \mathbb{N}^n$ .

# Funciones parciales Turing computables

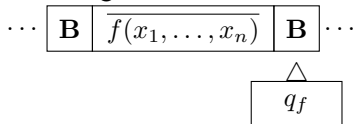
## Definición

Una función parcial  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  es **Turing computable** si existe una máquina de Turing determinística  $\mathcal{M} = (\Sigma, Q, \delta, q_0, q_f)$  con  $\Sigma = \{\mathbf{B}, 1\}$  tal que cuando empieza en la configuración inicial



(con los enteros  $x_i$  representados en unario ):

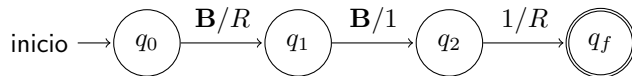
- ▶ si  $f(x_1, \dots, x_n) \downarrow$  entonces siguiendo sus instrucciones en  $\delta$  llega a una configuración final de la forma



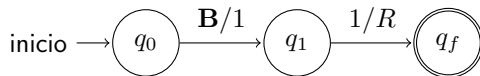
(quizá algo más en la cinta)

- ▶ si  $f(x_1, \dots, x_n) \uparrow$  entonces nunca termina en el estado  $q_f$ .

## Cómputo de la función $f(x) = 0$



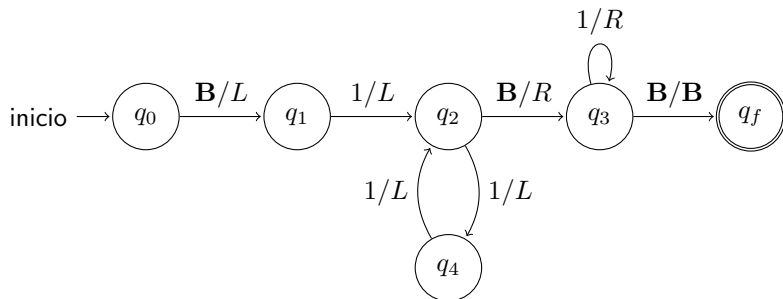
# Cómputo de la función $f(x) = x + 1$



# Cálculo de una función parcial

Supongamos

$$f(x) = \begin{cases} x & \text{si } x \text{ es par} \\ \uparrow & \text{si no} \end{cases}$$



# Iniciales, composición y recursión primitiva

Las siguientes funciones se llaman **iniciales**:

$$s(x) = x + 1, \quad (\text{siguiente})$$

$$n(x) = 0, \quad (\text{cero}),$$

$$u_i^n(x_1, \dots, x_n) = x_i \text{ para } i \in \{1, \dots, n\}, \quad (\text{proyecciones})$$

Sean  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  y  $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$ .

La **composición** de  $f$  y  $g_1, \dots, g_k$  es  $h : \mathbb{N}^n \rightarrow \mathbb{N}$ ,

$$h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

Sean  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$  y  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ .

Definimos  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  por **recursión primitiva**

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

En este contexto, una función 0-aria es una constante  $k$ .

Si  $h$  es 1-aria y  $t = 0$ , entonces  $h(t) = k = s^{(k)}(n(t))$ .



## Definition (funciones recursivas primitivas)

Una función es **recursiva primitiva (r.p.)**,  $f : \mathbb{N}^k \rightarrow \mathbb{N}$ , si se define mediante las funciones iniciales y un número finito de aplicaciones de composición y recursión primitiva.

# Ejemplos de función recursivas primitivas

La función  $\text{suma}(x, y) = x + y$  es r.p. porque

$$\begin{aligned}\text{suma}(x, 0) &= u_1^1(x) \\ \text{suma}(x, y + 1) &= g(\text{suma}(x, y), x, y)\end{aligned}$$

donde  $g(x_1, x_2, x_3) = s(u_1^3(x_1, x_2, x_3))$

►  $x \cdot y$

►  $x!$

►  $x^y$

►  $x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y \\ 0 & \text{si no} \end{cases}$

►  $\alpha(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{si no} \end{cases}$

► y muchas más.

# Clases Cerradas por Recursión Primitiva (CRP)

Una clase  $\mathcal{C}$  de funciones totales es Cerrada por Recursión Primitiva (CRP ) si

1. las funciones iniciales están en  $\mathcal{C}$
2. si una función  $f$  se obtiene a partir de otras pertenecientes a  $\mathcal{C}$  por medio de composición o recursión primitiva, entonces  $f$  también está en  $\mathcal{C}$

## Teorema

*Una función es r.p. sii pertenece a toda clase CRP.*

## Demostración.

- ( $\Leftarrow$ ) La clase de funciones r.p. es una clase CRP. Luego, si  $f$  pertenece a toda clase CRP, en particular  $f$  es r.p.
- ( $\Rightarrow$ ) Sea  $f$  r.p. y sea  $\mathcal{C}$  una clase CRP. Como  $f$  es r.p., hay una lista

tal que  $f_1, f_2, \dots, f_n$

$$f = f_n$$

$f_i$  es inicial (está en  $\mathcal{C}$ ) o se obtiene por composición (está en  $\mathcal{C}$ ) o recursión primitiva a partir de funciones  $f_j$ ,  $j < i$  (están en  $\mathcal{C}$ ).

Entonces todas las funciones de la lista están en  $\mathcal{C}$  (por inducción).  
En particular,  $f_n \in \mathcal{C}$ . □

# Clase de funciones r.p. es la clase CRP más chica

## Teorema

*La clase de funciones totales Turing computables es una clase CRP.*

## Corolario

*Toda función r.p. es total y Turing computable.*

¿ Toda función total Turing computable es r.p.? No.

## Teorema

*La función de Ackermann es total Turing computable y no es recursiva primitiva.*

Damos a continuación la definición, pero no la demostración.

## La función de Ackermann (1928)

$$A(x, y, z) = \begin{cases} y + z & \text{si } x = 0 \\ 0 & \text{si } x = 1 \text{ y } z = 0 \\ 1 & \text{si } x = 2 \text{ y } z = 0 \\ y & \text{si } x > 2 \text{ y } z = 0 \\ A(x - 1, y, A(x, y, z - 1)) & \text{si } x, z > 0 \end{cases}$$

- ▶  $A_0(y, z) = A(0, y, z) = y + z = \underbrace{y + 1 + \cdots + 1}_{z \text{ veces}}$
- ▶  $A_1(y, z) = A(1, y, z) = y \cdot z = \underbrace{y + \cdots + y}_{z \text{ veces}}$
- ▶  $A_2(y, z) = A(2, y, z) = y \uparrow z = y^z = \underbrace{y \cdot \cdots \cdot y}_{z \text{ veces}}$
- ▶  $A_3(y, z) = A(3, y, z) = y \uparrow\uparrow z = \underbrace{y^{y^{\cdots^y}}}_{z \text{ veces}}$
- ▶ ...

Para cada  $i$ ,  $A_i : \mathbb{N}^2 \rightarrow \mathbb{N}$  es r.p. pero  $A : \mathbb{N}^3 \rightarrow \mathbb{N}$  no es r.p.

# Predicados en las clases CRP

Los **predicados** son simplemente funciones que toman valores en  $\{0, 1\}$ .

- ▶ 1 se interpreta como verdadero
- ▶ 0 se interpreta como falso

Los **predicados r.p.** son aquellos representados por funciones r.p. en  $\{0, 1\}$ .

Por ejemplo, el predicado  $x \leq y$  es r.p. ya que  $\alpha(x \dot{-} y)$ .



# Predicados en las clases CRP

## Teorema

*Sea  $\mathcal{C}$  una clase CRP. Si  $p$  y  $q$  son predicados en  $\mathcal{C}$  entonces  $\neg p$ ,  $p \wedge q$  y  $p \vee q$  están en  $\mathcal{C}$ .*

## Demostración.

- ▶  $\neg p$  se define como  $\alpha(p)$
- ▶  $p \wedge q$  se define como  $p \cdot q$
- ▶  $p \vee q$  se define como  $\neg(\neg p \wedge \neg q)$



## Corolario

*Si  $p$  y  $q$  son predicados r.p., entonces también lo son los predicados  $\neg p$ ,  $p \vee q$  y  $p \wedge q$*

## Corolario

*Si  $p$  y  $q$  son predicados totales Turing computables entonces también lo son los predicados  $\neg p$ ,  $p \vee q$  y  $p \wedge q$*

## Definición por casos (2) en las clases CRP

### Teorema

Sea  $\mathcal{C}$  una clase CRP. Sean  $h, g : \mathbb{N}^n \rightarrow \mathbb{N}$  funciones en  $\mathcal{C}$  y sea  $p : \mathbb{N}^n \rightarrow \{0, 1\}$  un predicado en  $\mathcal{C}$ . La función

$$f(x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n) & \text{si } p(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \text{si no} \end{cases}$$

está en  $\mathcal{C}$ .

### Demostración.

$$f(x_1, \dots, x_n) = g(x_1, \dots, x_n) \cdot p(x_1, \dots, x_n) + h(x_1, \dots, x_n) \cdot \alpha(p(x_1, \dots, x_n))$$



## Definición por casos ( $m + 1$ ) en las clases CRP

### Teorema

Sea  $\mathcal{C}$  una clase CRP. Sean  $g_1, \dots, g_m, h : \mathbb{N}^n \rightarrow \mathbb{N}$  funciones en  $\mathcal{C}$  y sean  $p_1, \dots, p_m : \mathbb{N}^n \rightarrow \{0, 1\}$  predicados mutuamente excluyentes en  $\mathcal{C}$ .

La función

$$f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n) & \text{si } p_1(x_1, \dots, x_n) \\ \vdots & \\ g_m(x_1, \dots, x_n) & \text{si } p_m(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \text{si no} \end{cases}$$

está en  $\mathcal{C}$ .

# Sumatorias y productorias (desde 0) en las clases CRP

## Teorema

Sea  $\mathcal{C}$  una clase CRP. Si  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  está en  $\mathcal{C}$  entonces también están las funciones

$$g(y, x_1, \dots, x_n) = \sum_{t=0}^y f(t, x_1, \dots, x_n)$$

$$h(y, x_1, \dots, x_n) = \prod_{t=0}^y f(t, x_1, \dots, x_n)$$

## Demostración.

$$g(0, x_1, \dots, x_n) = f(0, x_1, \dots, x_n)$$

$$g(t+1, x_1, \dots, x_n) = g(t, x_1, \dots, x_n) + f(t+1, x_1, \dots, x_n)$$

Idem para  $h$  con  $\cdot$  en lugar de  $+$ .



# Sumatorias y productorias (desde 1) en las clases CRP

## Teorema

Sea  $\mathcal{C}$  una clase CRP. Si  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  está en  $\mathcal{C}$  entonces también están las funciones

$$g(y, x_1, \dots, x_n) = \sum_{t=1}^y f(t, x_1, \dots, x_n)$$

$$h(y, x_1, \dots, x_n) = \prod_{t=1}^y f(t, x_1, \dots, x_n)$$

(como siempre, sumatoria vacía = 0, productoria vacía = 1)

## Demostración.

$$g(0, x_1, \dots, x_n) = 0$$

$$g(t+1, x_1, \dots, x_n) = g(t, x_1, \dots, x_n) + f(t+1, x_1, \dots, x_n)$$

Idem para  $h$  con  $\cdot$  en lugar de  $+$  y 1 en lugar de 0 en el caso base. □

# Cuantificadores acotados en las clases CRP

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  un predicado.

$(\forall t)_{\leq y} p(t, x_1, \dots, x_n)$  es verdadero sii

►  $p(0, x_1, \dots, x_n)$  es verdadero y

⋮

►  $p(y, x_1, \dots, x_n)$  es verdadero

$(\exists t)_{\leq y} p(t, x_1, \dots, x_n)$  es verdadero sii

►  $p(0, x_1, \dots, x_n)$  es verdadero o

⋮

►  $p(y, x_1, \dots, x_n)$  es verdadero

Lo mismo se puede definir con  $< y$  en lugar de  $\leq y$ .

$$(\exists t)_{< y} p(t, x_1, \dots, x_n) \quad \text{y} \quad (\forall t)_{< y} p(t, x_1, \dots, x_n)$$

# Cuantificadores acotados (con $\leq$ ) en las clases CRP

## Teorema

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  un predicado perteneciente a una clase CRP  $\mathcal{C}$ .  
Los siguientes predicados también están en  $\mathcal{C}$ :

$$(\forall t)_{\leq y} p(t, x_1, \dots, x_n)$$

$$(\exists t)_{\leq y} p(t, x_1, \dots, x_n)$$

## Demostración.

$$(\forall t)_{\leq y} p(t, x_1, \dots, x_n) \text{ sii } \prod_{t=0}^y p(t, x_1, \dots, x_n) = 1$$

$$(\exists t)_{\leq y} p(t, x_1, \dots, x_n) \text{ sii } \sum_{t=0}^y p(t, x_1, \dots, x_n) \neq 0$$

- ▶ la sumatoria y productoria están en  $\mathcal{C}$
- ▶ la comparación por  $=$  está en  $\mathcal{C}$



# Cuantificadores acotados (con $<$ ) en las clases CRP

## Teorema

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  un predicado perteneciente a una clase CRP  $\mathcal{C}$ .  
Los siguientes predicados también están en  $\mathcal{C}$ :

$$(\forall t)_{<y} p(t, x_1, \dots, x_n)$$

$$(\exists t)_{<y} p(t, x_1, \dots, x_n)$$

## Demostración.

$$(\forall t)_{<y} p(t, x_1, \dots, x_n) \text{ sii } (\forall t)_{\leq y} (t = y \vee p(t, x_1, \dots, x_n))$$

$$(\exists t)_{<y} p(t, x_1, \dots, x_n) \text{ sii } (\exists t)_{\leq y} (t \neq y \wedge p(t, x_1, \dots, x_n))$$





# Más ejemplos de funciones recursivas primitivas

- ▶  $y|x$  sii  $y$  divide a  $x$ . Se define como

$$(\exists t)_{\leq x} y \cdot t = x$$

Notar que con esta definición  $0|0$ .

- ▶  $\text{primo}(x)$  sii  $x$  es primo.

# Minimización acotada en las clases CRP

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  un predicado de una clase CRP  $\mathcal{C}$ .

$$g(y, x_1, \dots, x_n) = \sum_{u=0}^y \prod_{t=0}^u \alpha(p(t, x_1, \dots, x_n))$$

¿ Qué hace  $g$ ?

Supongamos que existe un  $t \leq y$  tal que  $p(t, x_1, \dots, x_n)$  es verdadero

► sea  $t_0$  el mínimo tal  $t$

►  $p(t, x_1, \dots, x_n) = 0$  para todo  $t < t_0$

►  $p(t_0, x_1, \dots, x_n) = 1$

►  $\prod_{t=0}^u \alpha(p(t, x_1, \dots, x_n)) = \begin{cases} 1 & \text{si } u < t_0 \\ 0 & \text{si no} \end{cases}$

►  $g(y, x_1, \dots, x_n) = \underbrace{1 + 1 + \dots + 1}_{t_0 \text{ veces}} \overbrace{+ 0 + 0 + \dots + 0}^{y+1 \text{ veces}} = t_0$

► entonces  $g(y, x_1, \dots, x_n)$  es el mínimo  $t \leq y$  tal que  $p(t, x_1, \dots, x_n)$  es verdadero

Si no existe tal  $t$ ,  $g(y, x_1, \dots, x_n) = y + 1$

# Minimización acotada en las clases CRP

Notamos

$$\min_{t \leq y} p(t, x_1, \dots, x_n) = \begin{cases} \text{mínimo } t \leq y \text{ tal que} \\ p(t, x_1, \dots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ 0 & \text{si no} \end{cases}$$

## Teorema

Sea  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  un predicado de una clase CRP  $\mathcal{C}$ . La función

$$\min_{t \leq y} p(t, x_1, \dots, x_n)$$

también está en  $\mathcal{C}$ .

# Más ejemplos de funciones recursivas primitivas

- ▶  $x \text{ div } y$  es la división entera de  $x$  por  $y$

$$\min_{t \leq x} ((t+1) \cdot y > x)$$

Notar que con esta definición  $0 \text{ div } 0$  es  $0$ .

- ▶  $x \text{ mód } y$  es el resto de dividir a  $x$  por  $y$
- ▶  $p_n$  es el  $n$ -ésimo primo ( $n > 0$ ). Se define  $p_0 = 0, p_1 = 2, p_2 = 3, p_3 = 5, \dots$

$$\begin{aligned} p_0 &= 0 \\ p_{n+1} &= \min_{t \leq K(n)} (\text{primo}(t) \wedge t > p_n) \end{aligned}$$

Necesitamos una cota  $K(n)$  que sea buena, es decir

- ▶ suficientemente grande y
- ▶ recursiva primitiva

$K(n) = p_n! + 1$  funciona (ver que  $p_{n+1} \leq p_n! + 1$ ).

# Codificación de pares

Definimos la función recursiva primitiva

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

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 recursivas primitivas.*

## 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 recursivas primitivas.*

## 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 r.p.*

## 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 r.p.*

# Minimización no acotada

Definimos la **minimización no acotada**

$$\min_t p(t, x_1, \dots, x_n) = \begin{cases} \text{mínimo } t \text{ tal que} \\ p(t, x_1, \dots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ \uparrow & \text{si no} \end{cases}$$

# Funciones computables

## Definición

Una función parcial es **parcial computable** si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de

- ▶ composición,
- ▶ recursión primitiva y
- ▶ **minimización** no acotada

## Definición

Una función total es **computable** si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de

- ▶ composición,
- ▶ recursión primitiva y
- ▶ **minimización propia**

(del tipo  $\min_t q(x_1, \dots, x_n, t)$  donde siempre existe al menos un  $t$  tal que  $q(x_1, \dots, x_n, t)$  es verdadero)

# S++

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 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

Por ejemplo, para  $P$ :

```
while ( $X \neq 0$ ) do {  
     $X - -$   
     $Y + +$   
}
```

Son estados de  $P$ :

- ▶  $X = 3, Y = 1$
- ▶  $X = 2, Y = 1$

# Descripción instantánea

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

Para un estado  $\sigma$  de  $P$  y un  $i \in \{1, \dots, n+1\}$ ,

- ▶ el par  $(i, \sigma)$  es una **descripción instantánea** de  $P$
- ▶  $(i, \sigma)$  se llama **terminal** si  $i = n+1$

# Descripción instantánea

Para un  $(i, \sigma)$  no terminal, podemos definir su **sucesor**  $(j, \tau)$  como:

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  $V = \max\{m - 1, 0\}$

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

# Estados y descripciones iniciales

Sea  $P$  un programa y sean  $r_1, \dots, r_m$  números dados.

- el **estado inicial** de  $P$  para  $r_1, \dots, r_m$  es el estado  $\sigma_1$ , 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  $X_1, \dots, X_m, Y$

- la **descripción inicial** de  $P$  para  $r_1, \dots, r_m$  es

$$(1, \sigma_1)$$

# 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

Hay dos posibilidades:

## Definition

Si hay un cómputo de  $P$   $d_1, \dots, d_k$  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 parcial  $f : \mathbb{N}^m \rightarrow \mathbb{N}$  es *parcialmente computable* sii existe un programa  $P$  en  $\mathcal{S}++$  tal que para todo  $(r_1, \dots, r_m) \in \mathbb{N}^m$ ,

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

La igualdad (del meta-lenguaje) es verdadera si

- ▶ los dos lados están definidos y tienen el mismo valor o
- ▶ los dos lados están indefinidos

Para demostrarlo debemos ver que  $\mathcal{S}++$  permite definir todas las funciones parcialmente computables. Debemos ver que podemos definir en  $\mathcal{S}++$  las funciones iniciales, composición, recursión primitiva, minimización acotada y no acotada.

# Minimización no acotada

## Teorema

Si  $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$  es un predicado computable entonces

$$\min_t p(t, x_1, \dots, x_n)$$

se define en S++ .

## Demostración.

El siguiente programa 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  $X \leftarrow Y . X \leftarrow f(Y, Z)$  ya las tenemos definidas

## Teorema

*Si  $h$  se obtiene a partir de las funciones (parciales) computables  $f, g_1, \dots, g_k$  por composición entonces  $h$  se define en  $S^{++}$ .*

## Demostración.

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 primitiva

## Teorema

*Si  $h$  se obtiene a partir de  $g$  por recursión primitiva y  $g$  es computable entonces  $h$  se define en  $S++$ .*

## Demostración.

El siguiente programa computa  $h$ :

```
Y ← k
while X ≠ 0 do {
  Y ← g(Y, Z)
  Z ++
  X -- }
```

Si  $g$  es total entonces  $h$  es total.



## Teorema

*La clase de funciones parcialmente computables es una clase CRP.*

## Demostración.

Ya vimos que la clase de funciones computables está cerrada por composición y recursión primitiva. Veamos que las funciones iniciales son computables:

- ▶  $s(x) = x + 1$  se computa con el programa

$$Y \leftarrow X + 1$$

- ▶  $n(x) = 0$  se computa con el programa vacío

- ▶  $u_i^n(x_1, \dots, x_n) = x_i$  se computa con el programa

$$Y \leftarrow X_i$$





## Teorema

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

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

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!

# 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 primitivas recursivas
  - ▶ **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 funciones primitivas recursivas
- ▶ 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 primitiva recursiva  $\#$  : 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 computables como números naturales
  - ▶ tiene que haber funciones  $\mathbb{N} \rightarrow \mathbb{N}$  no computables
- Pero ¿qué ejemplo concreto tenemos?



## El problema de la detención (Halting problem)

$\text{HALT}(x, y) : \mathbb{N}^2 \rightarrow \{0, 1\}$  es verdadero sii el programa con número  $y$  y entrada  $x$  no se indefine,

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

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

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

Supongamos  $\#(P) = e$ .

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

## Teorema (Turing, 1936)

HALT *no es computable*.

## Demostración.

Supongamos que HALT es computable. Construimos el siguiente programa:

Programa  $P$  `while HALT( $X, X$ )  $\neq 0$  do {pass }`

Supongamos que  $\#(P) = e$ . Entonces, por el programa  $P$  escrito en azul,

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

Entonces

$$\text{HALT}(x, e) = 1 \quad \text{sii} \quad \Psi_P(x) \downarrow \quad \text{sii} \quad \text{HALT}(x, x) \neq 1$$

$e$  está fijo pero  $x$  es variable. Llegamos a un absurdo con  $x = e$ .



# Diagonalización

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

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

- ▶ sea  $P_i$  el programa con número  $i$
- ▶ supongo que  $\text{HALT}(x, y)$  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

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