

# Teoría de la Computabilidad

Santiago Figueira

Departamento de Computación, FCEyN, UBA

Primer cuatrimestre 2021

# Contenido

- clase 0 - Introducción, modelos de cómputo, repaso de la parte de Computabilidad de LyC: p. 3
- clase 1 - Funciones Primitivas recursivas, iteración, esqueleto de funciones primitivas recursivas, recursión anidada en varias variables, jerarquía rápidamente creciente: p. 32
- clase 2 - Halting problem, Reducibilidades one-one y many one, Teorema de Rice, Teorema de Isomorfismo de Myhill, cilindros: p. 59
- clase 3 - Conjuntos 1-completos, conjuntos productivos y creativos: p. 78
- clase 4 - Reducibilidad tt, cómputos con oráculo, reducibilidad de Turing, principio del uso: p. 92
- clase 5 - Teorema del Salto, Lema del Límite, grados Turing: p. 107
- clase 6 - Jerarquía Aritmética, Teorema de Post, Conjuntos  $\Sigma_n$ -completos: p. 126
- clase 7 - Problema de Post, Conjuntos simples: p. 147
- clase 8 - Conjuntos hipersimples, funciones computablemente mayoradas: p. 159

# Teoría de la Computabilidad

## Clase 0

Introducción, modelos de cómputo, repaso de la parte de  
Computabilidad de LyC

## Organización de la materia

- 3 puntos para Licenciatura en Computación,
- 3 puntos para Doctorado en Computación (a confirmar)
- correlativa: LyC
- Martes de 14 a 18
- Evaluación: seguramente un parcial y un paper o capítulo de libro para exponer (tipo seminario).
- Campus virtual

# Bibliografía

En orden de importancia

- Robert I. Soare, Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets (Perspectives in Mathematical Logic), Springer, 1987.
- Hartley Rogers, Theory of Recursive Functions and Effective Computability, The MIT Press, 1987.
- Piergiorgio Odifreddi, Classical Recursion Theory, Vol. 1 (Studies in Logic and the Foundations of Mathematics, Vol. 125), North Holland, 1999.
- Piergiorgio Odifreddi, Classical Recursion Theory, Vol. 2 (Studies in Logic and the Foundations of Mathematics, Vol. 143), North Holland, 1999.
- Martin Davis, Ron Sigal, Elaine Weyuker, Computability, Complexity and Languages, fundamentals of theoretical computer science, Elsevier, 1994.

# Composición y recursión primitiva

## Definición

Sea  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  y  $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$ .  $h : \mathbb{N}^n \rightarrow \mathbb{N}$  se obtiene a partir de  $f$  y  $g_1, \dots, g_k$  por **composición** si

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

## Definición

$h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  se obtiene de  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$  y  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  por **recursión primitiva** si

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

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

# Funciones primitivas recursivas $\mathcal{P}_1$

## Funciones iniciales

- $\mathcal{O}(x) = 0$
- $\mathcal{S}(x) = x + 1$
- $\mathcal{I}_i^n(x_1, \dots, x_n) = x_i$  para todo  $n \geq 1$  y  $i \in \{1, \dots, n\}$

## Definición

La clase  $\mathcal{P}_1$  de funciones **primitivas recursivas** es la clase de funciones más chica:

- conteniendo las funciones iniciales
- cerrada por composición
- cerrada por recursión primitiva

## Ejemplos de funciones en $\mathcal{P}_1$

- **funciones aritméticas básicas**: suma, resta, producto, potencia
- **conectivos booleanos**:  $\wedge$ ,  $\vee$ ,  $\neg$  (verdadero se interpreta como 1 y falso como 0)
- **predicados aritméticos básicos**:  $=$ ,  $\leq$
- **definición por casos**: si  $g_1, \dots, g_m, h \in \mathcal{P}_1$  y  $p_1, \dots, p_m \in \mathcal{P}_1$  son predicados mutuamente excluyentes entonces también está en  $\mathcal{P}_1$  la función

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{si } p_1(\vec{x}) \\ & \vdots \\ g_m(\vec{x}) & \text{si } p_m(\vec{x}) \\ h(\vec{x}) & \text{si no} \end{cases}$$

- **sumatorias y productorias**: si  $f \in \mathcal{P}_1$  entonces también están en  $\mathcal{P}_1$  las funciones

$$g(y, \vec{x}) = \sum_{t=0}^y f(t, \vec{x}) \quad \text{y} \quad h(y, \vec{x}) = \prod_{t=0}^y f(t, \vec{x})$$

## Ejemplos de funciones en $\mathcal{P}_1$

- **cuantificadores acotados:** si  $p \in \mathcal{P}_1$  es un predicado entonces también están en  $\mathcal{P}_1$ :

$$(\forall t)_{\leq y} p(t, \vec{x}) \quad \text{y} \quad (\exists t)_{\leq y} p(t, \vec{x})$$

- **minimización acotada:** si  $p \in \mathcal{P}_1$  es un predicado entonces también está en  $\mathcal{P}_1$  la función

$$\min_{t \leq y} p(t, \vec{x}) = \begin{cases} \text{mínimo } t \leq y \text{ tal que} \\ \quad p(t, \vec{x}) \text{ es verdadero} & \text{si existe tal } t \\ y + 1 & \text{si no} \end{cases}$$

- **codificación y decodificación de  $k$ -uplas:**

$$\langle x_1, \dots, x_k \rangle \quad \pi_i(z)$$

- **codificación y decodificación de secuencias:**

$$[x_1, \dots, x_n] \quad z[i]$$

# Recursión *course-of-value*

## Teorema

$\mathcal{P}_1$  está cerrada por **recursión course-of-value** en donde la definición de  $f(\vec{x}, y + 1)$  involucra no solo el último valor  $f(\vec{x}, y)$  sino también cualquier número (quizá todos) los valores  $(f(\vec{x}, z))_{z \leq y}$ . Más formalmente, si definimos la función de historia

$$\hat{f}(\vec{x}, y) = [f(\vec{x}, 0), \dots, f(\vec{x}, y)],$$

tenemos que si  $g, h \in \mathcal{P}_1$  entonces también está en  $\mathcal{P}_1$  la función  $f$  definida como:

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, y + 1) &= h(\vec{x}, y, \hat{f}(\vec{x}, y)) \end{aligned}$$

# Recursión simultánea

## Teorema

$\mathcal{P}_1$  está cerrada por *recursión simultánea* en donde un número finito de funciones  $f_1, \dots, f_n$  se definen simultáneamente y la definición de  $f_m(\vec{x}, y + 1)$  puede involucrar no solo  $f_m(\vec{x}, y)$  sino también cualquier número (quizá todos) los valores  $(f_i(\vec{x}, y))_{1 \leq i \leq n}$ .

## Funciones parciales

Una **función parcial** es simplemente una función que puede indefinirse en algunos (quizá todos) sus argumentos.

Una función parcial es **total** si se define para todos sus argumentos.  
Las funciones primitivas recursivas son totales.

Solo trabajamos con funciones  $\mathbb{N}^k \rightarrow \mathbb{N}$ .

Notamos

- $f(\vec{x}) \downarrow$  cuando  $f$  está definida en  $\vec{x}$
- $f(\vec{x}) \uparrow$  cuando  $f$  está indefinida en  $\vec{x}$

El **dominio** de una función es conjunto de argumentos en el que la función se define:

$$\text{dom } f = \{\vec{x}: f(\vec{x}) \downarrow\}.$$

Si  $f$  y  $g$  son funciones parciales, por  $f(\vec{x}) = g(\vec{x})$  nos referimos a

$$[f(\vec{x}) \uparrow \wedge g(\vec{x}) \uparrow] \vee [f(\vec{x}) \downarrow \wedge g(\vec{x}) \downarrow \wedge \underbrace{f(\vec{x})}_{\in \mathbb{N}} = \underbrace{g(\vec{x})}_{\in \mathbb{N}}].$$

# Funciones parciales $\mu$ -recursivas

(Kleene, 1938)

## Funciones iniciales

FI

- $s(x) = x + 1$
- $n(x) = 0$
- $u_i^n(x_1, \dots, x_n) = x_i, i \in \{1, \dots, n\}$

## Recursión primitiva

RP

Dadas  $f$  y  $g$ , producir:

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, t + 1) &= g(h(\vec{x}, t), \vec{x}, t) \end{aligned}$$

## Composición

C

Dadas  $f$  y  $g_1, \dots, g_k$ , producir:

$$h(\vec{x}) = f(g_1(\vec{x}), \dots, g_k(\vec{x}))$$

## $\mu$ -recursión

$\mu$

Dada  $g$ , producir:

$$\begin{aligned} h(\vec{x}) &= \mu y [(\forall z \leq y) g(\vec{x}, z) \downarrow \wedge \\ &\quad g(\vec{x}, y) = 0] \end{aligned}$$

si tal  $y$  existe;  $h(\vec{x}) \uparrow$  si no.

## Definición

La clase de funciones *parciales  $\mu$ -recursivas* es la más chica de funciones que contiene a las FI y está cerrada por C, RP y  $\mu$ .

$$\text{parciales } \mu\text{-recursivas} = \mathcal{P}_1 + \mu$$

## 3 tipos de variables (almacenan $\mathbb{N}$ )

entrada

$X_1, X_2, X_3 \dots$

salida

$Y$

temporales

$Z_1, Z_2, Z_3 \dots$

## 3 instrucciones (cada una puede o no estar etiquetada)

$V \leftarrow V + 1$

$V$  se incrementa en 1

$V \leftarrow V - 1$

$V$  se decrementa en 1  
si era  $> 0$ ; si no queda en 0

IF  $V \neq 0$  GOTO  $A$

- $A$  es una etiqueta que denota una instrucción
- si el valor de  $V$  es distinto de 0, la ejecución sigue con la primera instrucción que tenga etiqueta  $A$
- si el valor de  $V$  es 0, sigue con la próxima instrucción

programa = sucesión finita de instrucciones

# $\lambda$ -cálculo

## Términos

átomos

$x, y, \dots$

las variables

aplicación

$(MN)$  es un término  
si  $M$  y  $N$  lo son

$\lambda$ -abstracción

$(\lambda x.M)$  es un término  
si  $M$  lo es.

## Reglas de reducción

regla  $\alpha$

$\lambda x.M \xrightarrow{\alpha} \lambda y.M[x/y]$   
si  $y$  no ocurre en  $M$ ;

regla  $\beta$

$(\lambda x.M)N \xrightarrow{\beta} M[x/N]$   
( $M$  debe mantener sus variables  
libres después de la sustitución)

$(M[x/N]$  indica  
sustitución de  $x$   
por  $N$  en  $M$ )

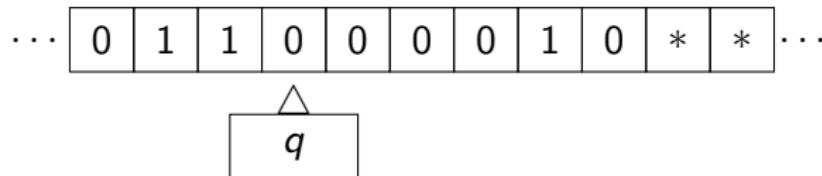
- Notamos  $t_1 \stackrel{\beta}{=} t_2$  si  $t_1$  y  $t_2$  reducen al mismo término  
aplicando un número finito de veces las reglas  $\alpha$  y  $\beta$ .
- Representamos al número  $n$  como  $\bar{n} = \lambda f. \lambda x. f^{(n)}(x)$ :

## Definición

Una función parcial  $f$  es  **$\lambda$ -definible** si existe un término  $F$  tal que

- si  $f(a_1, \dots, a_n) = b$ , entonces  $F\bar{a_1} \dots \bar{a_n} \stackrel{\beta}{=} \bar{b}$
- si  $f(a_1, \dots, a_n) \uparrow$ , entonces  $F\bar{a_1} \dots \bar{a_n}$  no tiene forma normal

# Máquinas de Turing



Se compone de :

- una **cinta**
  - dividida en celdas
  - infinita en ambas direcciones
  - cada celda contiene un símbolo de un alfabeto dado  $\Sigma$ .
    - $* \in \Sigma$
    - $L, R \notin \Sigma$ 
      - $*$  representa el blanco en una celda
      - $L$  y  $R$  son símbolos reservados (representarán acciones que puede realizar la cabeza)
- una **cabeza**
  - lee y escribe un símbolo a la vez
  - se mueve una posición a la izquierda o una posición a la derecha
- una **tabla finita de instrucciones**
  - dice qué hacer en cada paso

## Tabla de instrucciones

- $\Sigma$  es el alfabeto.  $L, R \notin \Sigma, * \in \Sigma$ .
- $Q$  es el conjunto finito de estados
- $A = \Sigma \cup \{L, R\}$  es el conjunto de acciones
  - un símbolo  $s \in \Sigma$  se interpreta como “escribir  $s$  en la posición actual”
  - $L$  se interpreta como “mover la cabeza una posición hacia la izquierda”
  - $R$  se interpreta como “mover la cabeza una posición hacia la derecha”

Una tabla de instrucciones  $T$  es un subconjunto (finito) de

$$Q \times \Sigma \times A \times Q$$

La tupla

$$(q, s, a, q') \in T$$

se interpreta como

*Si la máquina está en el estado  $q$  leyendo en la cinta el símbolo  $s$ , entonces realiza la acción  $a$  y pasa al estado  $q'$*

# Máquinas de Turing

Una **máquina de Turing** es una tupla

$$(\Sigma, Q, T, q_0, q_f)$$

donde

- $\Sigma$  (finito) es el conjunto **símbolos** ( $L, R \notin \Sigma, * \in \Sigma$ )
- $Q$  (finito) es el conjunto de **estados**
  - tiene dos estados distinguidos:
    - $q_0 \in Q$  es el **estado inicial**
    - $q_f \in Q$  es el **estado final**
- $T \subseteq Q \times \Sigma \times \Sigma \cup \{L, R\} \times Q$  es la **tabla de instrucciones**
  - va a ser finita porque  $\Sigma$  y  $Q$  lo son
- cuando no hay restricciones sobre  $T$  decimos que  $\mathcal{M}$  es una máquina de Turing **no determinísitca**
- cuando no hay dos instrucciones en  $T$  que empiezan con las mismas primeras dos coordenadas, decimos que  $\mathcal{M}$  es una máquina de Turing **determinísitca**
- Si no decimos nada, es una máquina de Turing determinística.

# Representación de números y tuplas en la cinta

Fijamos  $\Sigma = \{*, 1\}$ .

- representaremos a los **números** naturales en unario (con palotes).
  - el número  $x \in \mathbb{N}$  se representa como

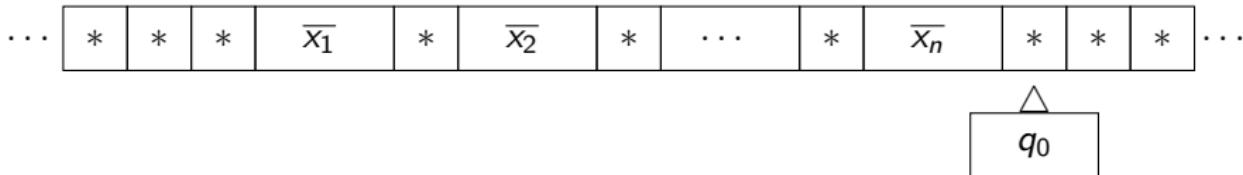
$$\overline{x} = \underbrace{1 \dots 1}_{x+1}$$

- 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

$$*\overline{x_1} * \overline{x_2} * \dots * \overline{x_n} *$$

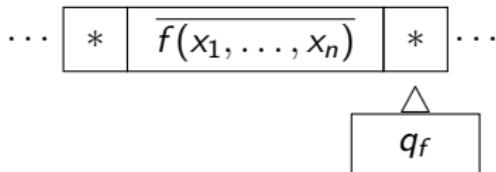
## Funciones parciales computables

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



(con los enteros  $x_i$  representados en unario y nada más en la cinta de entrada salvo la representación de la entrada):

- si  $f(x_1, \dots, x_n) \downarrow$  entonces siguiendo sus instrucciones en  $T$  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  $q_f$ .

Una función es **computable** si es parcial computable y total.

# Tesis de Church

Hay muchos modelos de cómputo:

- Lenguaje  $\mathcal{S}$
- Funciones parciales  $\mu$ -recursivas
- Máquinas de Turing (1936)
- Lambda cálculo (Church, 1933)
- Lenguajes de programación (Java, C++, Haskell, etc.)
- ...

¡La clase de funciones definida por cada modelo siempre coincide con la clase de funciones parciales **computables**!

## Tesis de Church

Cualquier función ‘efectiva’ es **computable**.

- Impone una cota superior precisa a la noción vaga de ‘algoritmo’

## 'Computabilidad' es una noción absoluta

*With this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e. one not depending on the formalism chosen. In all other cases treated previously, such as demonstrability or definability, one has been able to define them only relative to a given language, and for each individual language it is clear that the one thus obtained is not the one looked for. For the concept of computability however, although it is merely a special kind of demonstrability or definability, the situation is different.*

Gödel, 1946

# Enumeración de las máquinas de Turing

Habiendo fijado  $\Sigma$ ,  $q_0$ ,  $q_f$ , cada máquina de Turing puede ser identificada por su tabla de instrucciones.

Una tabla de instrucciones es un conjunto finito de 4-uplas. Cada tabla de instrucciones (i.e. cada máquina) puede ser codificado de manera efectiva por un número.

## Definición

$M_e$  es la máquina de Turing con tabla de instrucciones con código  $e$ .  $\varphi_e^{(n)}$  es la función parcial de  $n$  variables computada por la máquina  $M_e$ . Abreviamos  $\varphi^{(1)} = \varphi$ .

## Notación:

- “ $\varphi_e$  es la  $e$ -ésima función parcial computable”
- $e$  es el **índice** o **código** o **número de Gödel** de  $\varphi_e$ .

# Teoremas de Forma Normal y de Enumeración

## Teorema (Forma Normal)

Existe un predicado  $T(e, x, t)$  y una función  $U(y)$  primitivos recursivos tal que

$$\varphi_e(x) = U(\min_t T(e, x, t)).$$

## Teorema (Enumeración, existencia de máquina universal)

Para cada  $n \geq 1$  hay una función parcial computable de  $n + 1$  variables,  $\varphi_{z_n}^{(n+1)}$ , tal que

$$\varphi_{z_n}^{(n+1)}(e, x_1, \dots, x_n) = \varphi_e^{(n)}(x_1, \dots, x_n).$$

# Teorema del Parámetro (TP)

## Teorema

Para cada  $m, n \geq 1$  existe una función primitiva recursiva 1-1  $s_n^m$  de  $m + 1$  variables tal que para todo  $y_1, \dots, y_m, z_1, \dots, z_n \in \mathbb{N}$ ,

$$\varphi_{s_n^m(x, y_1, \dots, y_m)}(z_1, \dots, z_n) = \varphi_x(y_1, \dots, y_m, z_1, \dots, z_n).$$

# Teorema de la Recursión (TR)

## Teorema

Para cualquier función computable  $f$  existe un  $n \in \mathbb{N}$  (llamado punto fijo de  $f$ ) tal que  $\varphi_n = \varphi_{f(n)}$ .

## Teorema (Recursión con parámetros)

Para cualquier función computable  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$  existe una función computable  $n : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $\varphi_{n(y)} = \varphi_{f(n(y), y)}$ .

## Demostración.

Definir  $d$  computable tal que

$$\varphi_{d(x,y)}(z) = \begin{cases} \varphi_{\varphi_x(x,y)}(z) & \text{si } \varphi_x(x,y) \downarrow \\ \uparrow & \text{si no} \end{cases}$$

Sea  $v$  tal que  $\varphi_v(x, y) = f(d(x, y), y)$ . Definir  $n(y) = d(v, y)$ .

$$\varphi_{n(y)} = \varphi_{d(v,y)} = \varphi_{\varphi_v(v,y)} = \varphi_{f(d(v,y),y)} = \varphi_{f(n(y),y)}.$$

## Conjuntos, secuencias, cadenas

Cuando hablamos de un **conjunto** de naturales  $A$  pensamos siempre en la función característica de ese conjunto.

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

Así, un conjunto puede ser computable o primitivo recursivo.  $A$  puede ser visto también como una **secuencia** infinita de 0s y 1s:

$$A(0) \ A(1) \ A(2) \ A(3) \ A(4) \ A(5) \dots \in \{0, 1\}^\infty$$

Los primeros  $n$  bits de  $A$  (visto como tira de 0s y 1s) es

$$A \upharpoonright n = A(0) \ A(1) \ \dots \ A(n-1)$$

Las **cadenas** finitas de 0s y 1s pueden ser vistas como conjuntos finitos.

# Ejecución de a pasos

## Definición

Notamos  $\varphi_{e,s}(x) = y$  si  $x, y, e < s$  e  $y$  es la salida de la máquina con código  $e$  ejecutada por  $< s$  pasos.

- si tal  $y$  existe, decimos que  $\varphi_{e,s}(x)$  converge ( $\varphi_{e,s}(x) \downarrow$ ).
- en caso contrario decimos que  $\varphi_{e,s}(x)$  diverge ( $\varphi_{e,s}(x) \uparrow$ ).

## Teorema

*Los siguientes conjuntos son computables:*

- $\{\langle e, x, s \rangle : \varphi_{e,s}(x) \downarrow\}$
- $\{\langle e, x, y, s \rangle : \varphi_{e,s}(x) = y\}$

# Conjuntos computablemente enumerables

## Definición

- Un conjunto  $A \subseteq \mathbb{N}$  es **computablemente enumerable** (c.e.) si  $A$  es el dominio de alguna función parcial computable.
- El  $e$ -ésimo conjunto c.e. se denota

$$W_e = \text{dom } \varphi_e = \{x : \varphi_e(x) \downarrow\} = \{x : (\exists t) T(e, x, t)\}$$

- $W_{e,s} = \text{dom } \varphi_{e,s}$  (por convención si  $x \in W_{e,s}$  entonces  $x, e < s$ ).

## Teorema

$A$  es computable si  $A$  y  $\overline{A}$  son c.e.

## Teorema

$f$  es parcial computable y  $A$  es c.e. entonces  $f^{-1}(A)$  es c.e.

# Caracterizaciones de los conjuntos c.e.

## Teorema

*Si  $A \neq \emptyset$ , son equivalentes:*

1. *A es c.e.*
2. *A es el rango de una función primitiva recursiva*
3. *A es el rango de una función computable*
4. *A es el rango de una función parcial computable*

## Una función computable que no es primitiva recursiva

Se pueden enumerar las funciones primitivas recursivas de 1 variable:  $\psi_1, \psi_2, \dots$

### Teorema (Enumeración de funciones primitivas recursivas)

*Hay una función  $f$  computable de 2 variables tal que*

$$f(e, x) = \psi_e(x).$$

Analicemos  $g : \mathbb{N} \rightarrow \mathbb{N}$  definida como  $g(x) = f(x, x)$ .

- claramente  $g$  es computable porque  $f$  lo es.
- supongamos que  $g \in \mathcal{P}_1$ 
  - entonces  $h(x) = g(x) + 1 = f(x, x) + 1 \in \mathcal{P}_1$
  - existe un  $e$  tal que  $\psi_e = h$
  - tenemos  $f(e, x) = h(x) = f(x, x) + 1$
  - $e$  está fijo pero  $x$  es variable
  - instanciando  $x = e$  llegamos a un absurdo
- ¿por qué esto no funciona para funciones parciales computables en vez de primitivas recursivas?

# Teoría de la Computabilidad

## Clase 1

Funciones Primitivas recursivas, iteración, esqueleto de funciones primitivas recursivas, recursión anidada en varias variables, jerarquía rápidamente creciente

# Composición y recursión primitiva

## Definición

Sea  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  y  $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$ .  $h : \mathbb{N}^n \rightarrow \mathbb{N}$  se obtiene a partir de  $f$  y  $g_1, \dots, g_k$  por **composición** si

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

## Definición

$h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  se obtiene de  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$  y  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  por **recursión primitiva** si

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

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

# Funciones primitivas recursivas $\mathcal{P}_1$

## Funciones iniciales

- $\mathcal{O}(x) = 0$
- $\mathcal{S}(x) = x + 1$
- $\mathcal{I}_i^n(x_1, \dots, x_n) = x_i$  para todo  $n \geq 1$  y  $i \in \{1, \dots, n\}$

## Definición

La clase  $\mathcal{P}_1$  de funciones **primitivas recursivas** es la clase de funciones más chica:

- conteniendo las funciones iniciales
- cerrada por composición
- cerrada por recursión primitiva

# Recursión anidada en una variable

## Definición

$f$  se define a partir de  $g_1, \dots, g_n$  por **recursión anidada en una variable** si

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, y) &= h(\vec{x}, y) \end{aligned}$$

donde  $h(\vec{x}, y)$  es un término numérico construido por

- números naturales  $n$
- las variables  $\vec{x}$  e  $y$
- las funciones  $g_1, \dots, g_m$
- el símbolo  $f$
- los valores de  $f$  usados en la definición de  $f(\vec{x}, y)$  son de la forma  $f(\vec{t}, s)$  donde  $s$  tiene un valor numérico  $< y$

## Teorema

$\mathcal{P}_1$  está cerrada por recursión anidada en una variable.

# Caracterización alternativa de $\mathcal{P}_1$

## Definición

Una función  $f$  se define por **iteración** a partir de  $t$  si

$$f(x, n) = t^{(n)}(x)$$

donde  $t^{(n)}$  denota el resultado de  $n$  aplicaciones sucesivas de  $t$  (por convención  $t^{(0)}(x) = x$ ).

## Teorema

*La clase de funciones primitivas recursivas es la clase más chica:*

- *conteniendo las funciones iniciales y funciones de codificación y decodificación de pares*
- *cerrada por composición*
- *cerrada por iteración*

Sea  $\mathcal{C}$  la clase más chica de funciones que satisfacen las condiciones del teorema.

### Demostración ( $\mathcal{C} \subseteq \mathcal{P}_1$ )

Las funciones de codificación y decodificación son primitivas recursivas. La iteración  $f(x, n) = t^{(n)}(x)$  es un caso especial de recursión primitiva:

$$\begin{aligned}f(x, 0) &= x \\f(x, n + 1) &= t(f(x, n))\end{aligned}$$

### Demostración ( $\mathcal{P}_1 \subseteq \mathcal{C}$ )

Hay que ver que  $\mathcal{C}$  es cerrada por recursión primitiva. Observar que  $\mathcal{C}$  tiene cod./decod. de pares y composición, por lo tanto tiene cod./decod. de  $n$ -uplas para cualquier  $n$  fijo.

Sean  $g, h \in \mathcal{C}$  y sea  $f$  definida como

$$\begin{aligned}f(\vec{x}, 0) &= g(\vec{x}) \\f(\vec{x}, y + 1) &= h(\vec{x}, y, f(\vec{x}, y))\end{aligned}$$

Basta ver que esta función está en  $\mathcal{C}$ :

$$s(\vec{x}, n) = \langle \vec{x}, n, f(\vec{x}, n) \rangle$$

## Demostración ( $\mathcal{P}_1 \subseteq \mathcal{C}$ cont.)

La función

$$\begin{aligned}s(\vec{x}, 0) &= \langle \vec{x}, 0, f(\vec{x}, 0) \rangle \\ &= \langle \vec{x}, 0, g(\vec{x}) \rangle\end{aligned}$$

está en  $\mathcal{C}$  porque  $g$  y  $\langle \cdot \rangle$  están en  $\mathcal{C}$ . Podemos transformar

$$s(\vec{x}, n) = \langle \vec{x}, n, \underbrace{f(\vec{x}, n)}_z \rangle$$

en

$$\begin{aligned}s(\vec{x}, n + 1) &= \langle \vec{x}, n + 1, f(\vec{x}, n + 1) \rangle \\ &= \langle \vec{x}, n + 1, h(\vec{x}, n, \underbrace{f(\vec{x}, n)}_z) \rangle\end{aligned}$$

con esta función, que también está en  $\mathcal{C}$ :

$$t(\vec{x}, n, z) = \langle \vec{x}, n + 1, h(\vec{x}, n, z) \rangle.$$

Como  $\mathcal{C}$  está cerrada por iteración, está en  $\mathcal{C}$  esta función:

$$s(\vec{x}, n) = t^{(n)}(s(\vec{x}, 0)).$$

# La familia de funciones $(h_n)_{n \in \mathbb{N}}$

Sea

$$h_0(x) = x + 1 \quad \text{y} \quad h_{n+1}(x) = h_n^{(x)}(x)$$

## Proposición

$$x \leq h_n(x).$$

## Demostración.

Por inducción en  $n$ . Para  $n = 0$  es trivial. Para  $n + 1$ , suponer  $(\forall z) z \leq h_n(z)$ . Es fácil probar por inducción en  $m$  que  $x \leq h_n^{(m)}(x)$ . En particular

$$x \leq h_n^{(x)}(x) = h_{n+1}(x).$$



# Propiedades de $(h_n)_{n \in \mathbb{N}}$

## Proposición

Si  $n \leq m$  y  $x > 0$  entonces  $h_n(x) \leq h_m(x)$ .

## Demostración.

Basta probar  $(\forall x > 0) h_n(x) \leq h_{n+1}(x) = h_n^{(x)}(x)$ .

Basta probar  $h_n(z) \leq h_n^{(m)}(z)$  por inducción en  $m > 0$ .

Para  $m = 1$  es trivial. Para  $m + 1$ ,

$$\begin{aligned} h_n(z) &\leq h_n^{(m)}(z) && \text{por HI} \\ &\leq h_n(h_n^{(m)}(z)) && \text{por Proposición de pág. 39} \\ &= h_n^{(m+1)}(z) \end{aligned}$$



# Propiedades de $(h_n)_{n \in \mathbb{N}}$

## Proposición

*Si  $x \leq y$  entonces  $h_n(x) \leq h_n(y)$ .*

## Demostración.

Por inducción en  $n$ . Para  $n = 0$  es trivial. Para  $n + 1$ ,

$$\begin{aligned} h_{n+1}(x) &= h_n^{(x)}(x) \\ &\leq h_n^{(y)}(x) \quad \text{por Proposición de pág. 39 } (y - x \text{ veces}) \\ &\leq h_n^{(y)}(y) \quad \text{por HI } (y \text{ veces}) \\ &= h_{n+1}(y). \end{aligned}$$



# Propiedades de $(h_n)_{n \in \mathbb{N}}$

## Proposición

$$h_1(x) = 2x.$$

## Demostración.

Basta probar  $h_0^{(m)}(x) = x + m$  por inducción en  $m$ . Para  $m = 0$  es trivial. Para  $m + 1$ ,

$$\begin{aligned} h_0^{(m+1)}(x) &= h_0(h_0^{(m)}(x)) \\ &= h_0(x + m) \quad \text{por HI} \\ &= (x + m) + 1 \quad \text{porque } h_0(z) = z + 1 \\ &= x + (m + 1). \end{aligned}$$



# Propiedades de $(h_n)_{n \in \mathbb{N}}$

## Proposición

$$h_2(x) = x \cdot 2^x.$$

## Demostración.

Basta probar  $h_1^{(m)}(x) = x \cdot 2^m$  por inducción en  $m$ . Para  $m = 0$  es trivial. Para  $m + 1$ ,

$$\begin{aligned} h_1^{(m+1)}(x) &= h_1(h_1^{(m)}(x)) \\ &= h_1(x \cdot 2^m) \quad \text{por HI} \\ &= (x \cdot 2^m) \cdot 2 \quad \text{porque } h_1(z) = 2z \\ &= x \cdot 2^{m+1} \end{aligned}$$



# Todas las $h_n$ son primitivas recursivas

## Teorema

*Para cada  $n$ ,  $h_n$  es primitiva recursiva.*

## Demostración.

Obvio, porque  $\mathcal{P}_1$  está cerrada por iteración. □

Cada función primitiva recursiva es dominada por alguna  $h_n$

### Teorema

Para cada  $f$  primitiva recursiva, hay un  $n$  tal que  $f(\vec{x}) \leq h_n(\sum \vec{x})$  para casi todo  $\vec{x}$ .

### Demostración.

Por inducción en la caracterización alternativa de  $\mathcal{P}_1$  (pág. 36).

Veamos que:

- las funciones iniciales y las funciones de codificación y decodificación lo cumplen
- si  $h, g_1, \dots, g_m$  lo cumplen,

$$f(\vec{x}) = h(g_1(\vec{x}), \dots, g_m(\vec{x}))$$

también lo cumple

- si  $t$  lo cumple,

$$f(x, y) = t^{(y)}(x)$$

también lo cumple.



## Demostración - Funciones iniciales, codificación y decodificación.

- $\mathcal{O}(x) = 0$  está dominada por  $h_0$
- $\mathcal{S}(x) = x + 1$  está dominada por  $h_0$
- $\mathcal{I}_i^n(x_1, \dots, x_n) = x_i$  está dominada por  $h_0$  porque

$$\begin{aligned}\mathcal{I}_i^n(x_1, \dots, x_n) &\leq \sum_j x_j \\ &\leq h_0 \left( \sum_j x_j \right)\end{aligned}$$

- $\langle x, y \rangle = 2^x(y + 1) - 1$  está dominada por  $h_2$  porque

$$h_2(x + y) = (x + y) \cdot 2^{x+y} \geq 2^x(y + 1) - 1$$

para casi todo  $x, y$ .

- $\pi_1(x), \pi_2(x) \leq x$ , dominadas por  $h_0(x)$



## Demostración - Composición.

Supongamos que

$$f(\vec{x}) = h(g_1(\vec{x}), \dots, g_m(\vec{x}))$$

y por HI tenemos que para un  $n \geq 2$  suficientemente grande (usar Proposición de pág. 40),  $g_i(\vec{x}) \leq h_n(\sum \vec{x})$  para casi todo  $\vec{x}$  y  $h(\vec{y}) \leq h_n(\sum \vec{y})$  para casi todo  $\vec{y}$ .

$$\begin{aligned} f(\vec{x}) &= h(g_1(\vec{x}), \dots, g_m(\vec{x})) \\ &\leq h_n(\sum_{1 \leq i \leq m} g_i(\vec{x})) && \text{por HI sobre } h \\ &\leq h_n(\sum_{1 \leq i \leq m} h_n(\sum \vec{x})) && \text{por HI sobre } g_i \text{ y Prop. pág. 41} \\ &= h_n(m \cdot h_n(\sum \vec{x})) \\ &\leq h_n(h_2(h_n(\sum \vec{x}))) && \text{si } m \leq \sum \vec{x} \text{ y porque } h_2(z) = z \cdot 2^z \\ &\leq h_n(h_n(h_n(\sum \vec{x}))) && \text{por } n \geq 2 \text{ y Props. págs. 40 y 41} \\ &= h_n^{(3)}(\sum \vec{x}) \\ &\leq h_n^{(\sum \vec{x})}(\sum \vec{x}) && \text{si } 3 \leq \sum \vec{x} \text{ y Prop. pág. 39} \\ &= h_{n+1}(\sum \vec{x}) \end{aligned}$$



## Demostración - Iteración.

Supongamos que

$$f(x, y) = t^{(y)}(x)$$

y por HI para algún  $n$ ,  $t(x) \leq h_n(x)$  para casi todo  $x$ .

$$\begin{aligned} f(x, y) &= t^{(y)}(x) \\ &\leq h_n^{(y)}(x) \quad \text{por HI } y \text{ veces y Prop. pág. 41} \\ &\leq h_n^{(x+y)}(x + y) \quad \text{por Props. págs. 39 y 41} \\ &= h_{n+1}(x + y) \end{aligned}$$



# La función diagonal domina a cualquier primitiva recursiva

## Teorema

*La función diagonal  $d(x) = h_x(x)$  domina toda función primitiva recursiva  $\mathbb{N} \rightarrow \mathbb{N}$ .*

## Demostración.

Sea  $f$  una función primitiva recursiva. Existe  $n$  tal que para casi todo  $x$ ,

$$\begin{aligned} f(x) &\leq h_n(x) \\ &\leq h_x(x) \quad \text{si } x > n \text{ y por Prop. de pág. 40} \\ &= d(x) \end{aligned}$$

Entonces para casi todo  $x$ ,  $f(x) \leq d(x)$ . □

## Corolario

*$d$  no es primitiva recursiva.*

## Demostración.

Si lo fuera,  $d(x) + 1$  también lo sería. Por teorema anterior, para casi todo  $x$  tenemos  $d(x) + 1 \leq d(x)$ . Absurdo. □

# Orden lexicográfico

## Definición

El **orden lexicográfico** sobre  $\mathbb{N}^n$  se define como

$$(a_1, \dots, a_n) <_{\text{lex}} (b_1, \dots, b_n)$$

si y solo si existe un  $i \in \{1 \dots n\}$  tal que  $a_i < b_i$  y  $a_j = b_j$  para todo  $j \in \{1 \dots i - 1\}$ .

## Ejemplo

- $(1, 1000) <_{\text{lex}} (1, 1001) <_{\text{lex}} (2, 0)$
- $(1, 2, 3) <_{\text{lex}} (1, 2, 4) <_{\text{lex}} (1, 3, 1) <_{\text{lex}} (2, 0, 0)$

## Proposición

$(<_{\text{lex}}, \mathbb{N}^n)$  es un buen orden.

# Recursión anidada en $n$ variables

## Definición

$f$  se define a partir de  $g, g_1, \dots, g_n$  por **recursión anidada en  $n$  variables** si

$$f(\vec{x}, \vec{0}) = g(\vec{x}) \quad (\vec{0} \in \mathbb{N}^n)$$

$$f(\vec{x}, \vec{y}) = h(\vec{x}, \vec{y}) \quad (\vec{y} \in \mathbb{N}^n, \vec{y} \neq \vec{0})$$

donde  $h(\vec{x}, \vec{y})$  es un término numérico construido por

- números naturales
- las variables  $\vec{x}$  e  $\vec{y}$
- las funciones  $g_1, \dots, g_n$
- el símbolo  $f$
- los valores de  $f$  usados en la definición de  $f(\vec{x}, \vec{y})$  son de la forma  $f(\vec{t}, \vec{s})$  donde  $\vec{s} \in \mathbb{N}^n$  tiene un valor numérico  $<_{\text{lex}} \vec{y}$ .

# Funciones $n$ -primitivas recursivas $\mathcal{P}_n$

## Definición

La clase  $\mathcal{P}_n$  de funciones  **$n$ -primitivas recursivas** es la clase de funciones más chica:

- conteniendo las funciones iniciales
- cerrada por composición
- cerrada por recursión anidada en a lo sumo  $n$  variables

$\mathcal{P}_1$  no está cerrado por recursión anidada en 2 variables

### Teorema

$$\mathcal{P}_1 \subsetneq \mathcal{P}_2.$$

### Demostración.

Las funciones  $(h_n)_{n \in \mathbb{N}}$  se pueden describir así:

$$\begin{array}{rcl} h_0(x) & = & x + 1 \\ h_{n+1}(x) & = & h_n^{(x)}(x) \end{array} \qquad \begin{array}{rcl} h_n^{(0)}(x) & = & x \\ h_n^{(z+1)}(x) & = & h_n(h_n^{(z)}(x)) \end{array}$$

Se pueden ver como una única función  $h(n, z, x) = h_n^{(z)}(x)$  definidas por recursión anidada en las variables  $n$  y  $z$  (pensar  $h_n$  como  $h_n^{(1)}$ ):

$$\begin{array}{rcl} h(n, 0, x) & = & x \\ h(0, 1, x) & = & x + 1 \end{array} \qquad \begin{array}{rcl} h(n+1, 1, x) & = & h(n, x, x) \\ h(n, z+1, x) & = & h(n, 1, h(n, z, x)) \end{array}$$

Si  $h$  fuese primitiva recursiva, también lo sería

$$d(x) = h_x(x) = h(x, 1, x)$$

# Jerarquía de funciones $n$ -recursivas

El resultado anterior se puede generalizar.

## Teorema

*Para todo  $n$ ,  $\mathcal{P}_n \subsetneq \mathcal{P}_{n+1}$ .*

# Al infinito y más allá

La familia  $(h_n)_{n \in \mathbb{N}}$  se puede extender con ordinales numerables:

## Definición (Jerarquía rápidamente creciente)

Sea la familia de funciones  $(h_\alpha)_{\alpha}$  definida por inducción en ordinales numerables como

$$h_0(x) = x + 1$$

$$h_{\alpha+1}(x) = h_\alpha^{(x)}(x)$$

$$h_\alpha(x) = h_{\alpha_x}(x) \quad \text{si } \alpha \text{ es un ordinal límite}$$

donde, para un ordinal límite  $\alpha$ ,  $\{\alpha_x\}_{x \in \omega}$  es una secuencia creciente fundamental de ordinales con límite  $\alpha$ .

## Ejemplo

- $\{x\}_{x \in \omega}$  tiene como límite  $\omega$ , entonces  $h_\omega(x) = h_x(x)$
- $\{\omega \cdot x\}_{x \in \omega}$  tiene como límite  $\omega^2$ , entonces  $h_{\omega^2}(x) = h_{\omega \cdot x}(x)$

# La jerarquía extendida de Grzegorczyk

## Definición

Si  $g$  es una función total, la clase  $\mathcal{E}(g)$  de **funciones elementales en  $g$**  es la clase más chica de funciones:

- que contiene a las funciones iniciales, la suma, la resta y  $g$ ;
- está cerrada por composición, suma acotada y producto acotado.

## Definición

Para  $3 \leq \alpha \leq \epsilon_0$  :  $(\epsilon_0 = \omega^{\omega^{\omega^{\omega^{\dots}}}})$

- si  $\alpha = \beta + 1$  entonces  $\mathcal{E}_\alpha = \mathcal{E}(h_\beta)$
- si  $\alpha$  es un límite, entonces  $\mathcal{E}_\alpha = \bigcup_{3 \leq \beta < \alpha} \mathcal{E}_\beta$

## Teorema

Para  $n \geq 1$ ,  $\mathcal{E}_{\omega^n} = \mathcal{P}_n$ .

# Secuencia de Goodstein

Dado  $b$  y  $n$ , escribir  $n$  en **pura base**  $b$  quiere decir escribir  $n$  en base  $b$ , después escribir los exponentes de la descomposición en base  $b$  y así sucesivamente.

## Ejemplo

- para  $n = 195$  y  $b = 2$ , tenemos

$$195 = 2^{2^2+2^1+2^0} + 2^{2^2+2^1} + 2^1 + 2^0$$

- para  $n = 100$  y  $b = 3$ , tenemos

$$100 = 3^{3^1+3^0} + \underbrace{3^2 + 3^2}_{2 \cdot 3^2} + 3^0$$

# Secuencia de Goodstein

La **secuencia de Goodstein** relativa a  $(n, b)$  es la secuencia de números que empieza en  $n$  y luego itera el siguiente proceso:

- escribir  $n$  en pura base  $b$
- cambiar la base de  $b$  a  $b + 1$
- sustraer 1

## Ejemplo

La secuencia de Goodstein relativa a  $(195, 2)$  empieza con

$$\textcircled{1} \quad 195 = 2^{2^2+2^1+2^0} + 2^{2^2+2^1} + 2^1 + 2^0$$

$$\textcircled{2} \quad 3^{3^3+3^1+3^0} + 3^{3^3+3^1} + 3^1 + 3^0 - 1 = 3^{3^3+3^1+3^0} + 3^{3^3+3^1} + 3^1$$

$$\textcircled{3} \quad 4^{4^4+4^1+4^0} + 4^{4^4+4^1} + 4^1 - 1 = 4^{4^4+4^1+4^0} + 4^{4^4+4^1} + \underbrace{4^0 + 4^0 + 4^0}_{4^1-1=3}$$

$$\textcircled{4} \quad \dots$$

## Teorema

Cualquier secuencia de Goodstein termina en 0.

(*Chan!*)

# Teoría de la Computabilidad

## Clase 2

Halting problem, Reducibilidades one-one y many one, Teorema de Rice, Teorema de Isomorfismo de Myhill, cilindros

# $K$ , el Halting Problem

## Definición

$$K = \{x : \varphi_x(x) \downarrow\} = \{x : x \in W_x\}.$$

## Teorema

$K$  es c.e.

## Demostración.

$g(x) = \varphi_x(x)$  es una función parcial computable.  $K = \text{dom } g$ . □

## Teorema

$K$  es no es computable.

## Demostración.

Si  $K$  fuera computable, entonces esta función también lo sería:

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{si } x \in K \\ 0 & \text{si no} \end{cases}$$

Pero  $f \neq \varphi_x$  para todo  $x$ . Absurdo. □

# $K_0$ , otro Halting Problem

## Definición

$$K_0 := \{\langle x, y \rangle : \varphi_x(y) \downarrow\}$$

## Proposición

$K_0$  es c.e.

## Demostración.

$g(\langle x, y \rangle) = \varphi_x(y)$  es una función parcial computable.

$K_0 = \text{dom } g$ .

□

## Corolario

$K_0$  es no es computable.

## Demostración.

$x \in K$  sii  $\langle x, x \rangle \in K_0$ . Si  $K_0$  fuera computable, también lo sería  $K$ .

□

## Reducibilidades many-one y one-one

La demostración del corolario anterior sugiere un método indirecto para demostrar resultados de indecidibilidad de nuevos conjuntos: reducir  $K$  a esos conjuntos.

### Definición

- $A$  es **many-one reducible** a  $B$  ( $A \leq_m B$ ) si hay una función computable  $f$  tal que

$$x \in A \quad \text{sii} \quad f(x) \in B.$$

- $A$  es **one-one reducible** a  $B$  ( $A \leq_1 B$ ) si  $A \leq_m B$  vía una función 1-1 (i.e. inyectiva)

Informalmente,  $A \leq_m B$  si  $A$  es a lo sumo tan difícil como  $B$ .

Conociendo  $B$  puedo computar  $A$ . Para todo  $x$ , haciendo **una sola pregunta** a  $B$ , decido si  $x \in A$  o  $x \notin A$ . La pregunta es:  $\{f(x) \in B\}$

## Propiedades de $\leq_1$ y $\leq_m$

### Proposición

$\leq_1$  y  $\leq_m$  son reflexivas y transitivas.

### Demostración.

Ejercicio.

### Proposición

1.  $A \leq_1 B \Rightarrow A \leq_m B$
2.  $A \leq_1 B \Rightarrow \bar{A} \leq_1 \bar{B}$
3.  $A \leq_m B \Rightarrow \bar{A} \leq_m \bar{B}$
4. Si  $A \leq_m B$  y  $B$  computable, entonces  $A$  es computable
5. Si  $A \leq_m B$  y  $B$  c.e., entonces  $A$  es c.e.

### Demostración de 1, 2 y 3.

Ejercicio.

Demostración de 4:

Si  $A \leq_m B$  y  $B$  computable, entonces  $A$  es computable.

Si  $A \leq_m B$  y  $B$  computable, entonces  $A$  es computable:

Supongamos que  $f$  es computable tal que

$$x \in A \quad \text{sii} \quad f(x) \in B.$$

Como  $B$  es computable, la función característica de  $A$ , definida como  $A(x) = B(f(x))$  también lo es. □

Demostración de 5: Si  $A \leq_m B$  y  $B$  c.e., entonces  $A$  es c.e.

Si  $A \leq_m B$  y  $B$  c.e., entonces  $A$  es c.e.:

Supongamos  $A \leq_m B$  vía  $f$ . Sea  $R$  computable tal que

$$x \in B \quad \text{sii} \quad (\exists y) R(x, y).$$

Entonces

$$x \in A \quad \text{sii} \quad f(x) \in B \quad \text{sii} \quad (\exists y) \overbrace{R(f(x), y)}^{\text{computable}}.$$



# Más propiedades de $\leq_1$ y $\leq_m$

## Proposición

- 1 Existen  $A, B$  incomparables con respecto a  $\leq_m$  ( $A \not\leq_m B$  y  $B \not\leq_m A$ )
- 2 Existen  $A, B$  no computables e incomparables con respecto a  $\leq_m$

## Demostración.

- 1  $\emptyset \not\leq_m \mathbb{N}$  y  $\mathbb{N} \not\leq_m \emptyset$ .
- 2 Considerar  $K$  y  $\overline{K}$ .  $K$  es c.e. pero  $\overline{K}$  no es c.e. Por teorema anterior (5)  $\overline{K} \not\leq_m K$ . Por teorema anterior (3)  $K \not\leq_m \overline{K}$ . □

## Corolario

$$A \leq_m B \not\Rightarrow A \leq_m \overline{B}.$$

## Demostración.

Tomar  $A = B = K$ . □

# Grados $m$ y grados 1

## Definición

- $A \equiv_m B$  si  $A \leq_m B$  y  $B \leq_m A$
- $A \equiv_1 B$  si  $A \leq_1 B$  y  $B \leq_1 A$
- $\deg_1(A) = \{B : B \equiv_1 A\}$  es el grado 1 de  $A$
- $\deg_m(A) = \{B : B \equiv_m A\}$  es el grado  $m$  de  $A$

## Semiretículo superior (upper semilattice)

### Definición

Un **semiretículo superior**  $\mathcal{L} = (L, \leq, \vee)$  es un conjunto parcialmente ordenado tal que cada par de elementos tiene un least upper bound (lub).

$c \in L$  es un **lub** de  $a, b \in L$  si:

- $a \leq c$
- $b \leq c$
- si  $d \in L$  verifica  $a \leq d$  y  $b \leq d$  entonces  $c \leq d$

Al lub de  $a$  y  $b$  también se lo llama **supremo** o **join**. Si  $a$  y  $b$  son elementos de  $\mathcal{L}$  entonces  $a \vee b$  denota el lub de  $a$  y  $b$ .

$\leq_m$  forma un semiretículo superior

### Definición

$$A \oplus B = \{2x : x \in A\} \cup \{2x + 1 : x \in B\}.$$

### Proposición

El orden de  $\leq_m$  forma un semiretículo superior sobre  $\deg_m$ . Es decir, cualesquiera dos grados  $m$  tienen un lub.

### Demostración.

Veamos que  $\deg_m(A \oplus B)$  es el lub de  $\deg_m(A)$  y  $\deg_m(B)$  en el orden  $\leq_m$ .

- $A \leq_m A \oplus B$  vía  $\lambda x.2x$
- $B \leq_m A \oplus B$  vía  $\lambda x.2x + 1$
- supongamos  $C$  tal que  $A \leq_m C$  vía  $f$  y  $B \leq_m C$  vía  $g$ . Entonces  $A \oplus B \leq_m C$  vía  $h$  definida como

$$h(2x) = f(x)$$

$$h(2x + 1) = g(x)$$



# Más conjuntos indecidibles

## Teorema

$$K \leq_1 \text{Tot} := \{x : \varphi_x \text{ es una función total}\}$$

## Demostración.

Definir la función parcial computable

$$g(x, y) = \begin{cases} 1 & \text{si } x \in K \\ \uparrow & \text{sino} \end{cases}$$

Por el Teorema del Parámetro, existe una función 1-1 computable  $f$  tal que  $\varphi_{f(x)}(y) = g(x, y)$ .

- $x \in K \Rightarrow \varphi_{f(x)} = \lambda y. 1 \Rightarrow f(x) \in \text{Tot}$
- $x \notin K \Rightarrow \varphi_{f(x)} = \lambda y. \uparrow \Rightarrow f(x) \notin \text{Tot}$

Entonces  $x \in K$  si  $f(x) \in \text{Tot}$ .



# Teorema de Rice

## Definición

$A \subseteq \mathbb{N}$  es un **conjunto de índices** si para todo  $x, y$ , si  $x \in A$  y  $\varphi_x = \varphi_y$  entonces  $y \in A$ .

## Teorema (Rice)

*Si  $A$  es un conjunto de índices no trivial (i.e.  $A \neq \emptyset, \mathbb{N}$ ) entonces  $A$  no es computable.*

## Demostración.

Sea  $b$  tal que  $(\forall x) \varphi_b(x) \uparrow$ . Supongamos que  $b \in \overline{A}$  y probemos que  $K \leq_1 A$  (del mismo modo, si  $b \in A$  entonces  $K \leq_1 \overline{A}$ ).

Como  $A \neq \emptyset$ , sea  $a \in A$ . Como  $A$  es un conjunto de índices,  $\varphi_a \neq \varphi_b$ .

Por el TP, sea  $f$  una función computable 1-1 tal que

$$\varphi_{f(x)}(y) = \begin{cases} \varphi_a(y) & \text{si } x \in K \\ \uparrow & \text{si no} \end{cases}$$

- $x \in K \Rightarrow \varphi_{f(x)} = \varphi_a \Rightarrow f(x) \in A$
- $x \in \overline{K} \Rightarrow \varphi_{f(x)} = \varphi_b \Rightarrow f(x) \in \overline{A}$



# Permutaciones computables

## Definición

- Una **permutación computable** es una función  $\mathbb{N} \rightarrow \mathbb{N}$  computable biyectiva.
- Una propiedad es **computablemente invariante** si es invariante bajo cualquier permutación computable.

### Ejemplos de propiedades computablemente invariantes

- $A$  es c.e.
- $\#A = n$
- $A$  es computable

### Ejemplos de propiedades no computablemente invariantes

- $2 \in A$
- $A$  contiene solo números pares
- $A$  es un conjunto de índices

## Definición

$A$  es **computablemente isomorfo** a  $B$  ( $A \equiv B$ ) si hay una permutación computable  $p$  tal que  $A = p(B)$ . Es decir,  $x \in A$  si  $p(x) \in B$ .

# Teorema de Isomorfismo de Myhill

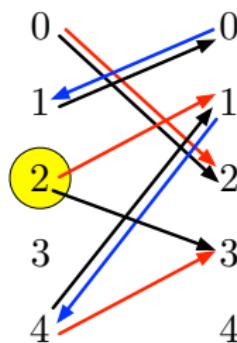
## Teorema

$A \equiv B$  si  $A \equiv_1 B$ .

## Demostración

$\Rightarrow$  Trivial.

$\Leftarrow$  Sea  $A \leq_1 B$  vía  $f$  y  $B \leq_1 A$  vía  $g$ . Definimos permutación computable  $h$  por etapas.  $h = \bigcup_s h_s$ .  $h_0 = \emptyset$ . Cada  $h_s$  es finita, 1-1, y cumple  $x \in A$  si  $h_s(x) \in B$  para  $x \in \text{dom } h_s$ .



- Paso 1:  $f(0) = 2 \quad h_1(0) = 2$
- Paso 2:  $g(0) = 1 \quad h_2^{-1}(0) = 1$  o sea  $h_2(1) = 0$
- Paso 3:  $h_3(1)$  ya está definido:  $h_3(1) = h_2(1) = 0$
- Paso 4:  $g(1) = 4 \quad h_4^{-1}(1) = 4$
- Paso 5:  $f(2) = 1$  está ocupado. Busco recorrido desde 2 con flecha roja - negra - roja - negra - ... - roja (las negras las tomo al revés) hasta encontrar uno libre.

Sup.  $f(4) = 3$ . Defino  $h_5(2) = f \circ h_1^{-1} \circ f(2) = 3$

## Demostración formal de ( $\Leftarrow$ ).

Supongamos definida  $h_s$  y definamos  $h_{s+1}$  de esta manera:

- si  $s + 1 = 2x + 1$ , definir  $h_{s+1}(x)$ . Si  $h_s(x)$  está definido, no hacer nada. Si no, enumerar

$$\{f(x), f(h_s^{-1}f(x)), \dots, f(h_s^{-1}f)^n(x), \dots\}$$

hasta encontrar el primer  $y \notin \text{rg } h_s$ . Definir  $h_{s+1}(x) = y$ .

Observar que  $h_{s+1}$  es finita, 1-1 y  $x \in A$  si y sólo si  $y \in B$ .

- si  $s + 1 = 2x + 2$ , definir  $h^{-1}(x)$  de forma similar, cambiando
  - $f \longleftrightarrow g$
  - $h_s \longleftrightarrow h_s^{-1}$
  - $\text{dom } h_s \longleftrightarrow \text{rg } h_s$



Vamos a clasificar conjuntos módulo permutaciones computable (lo mismo pasa en álgebra: clasifica estructuras módulo isomorfismo).

# Cilindros

## Definición

$A$  es un **cilindro** si  $A \equiv B \times \mathbb{N}$  para algún  $B$ .

## Teorema

1.  $A \leq_1 A \times \mathbb{N}$
2.  $A \times \mathbb{N} \leq_m A$  (*por lo tanto*  $A \equiv_m A \times \mathbb{N}$ )
3.  $A$  es un cilindro sii  $(\forall B) [B \leq_m A \Rightarrow B \leq_1 A]$
4.  $A \leq_m B$  sii  $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$

## Demostración de 1 y 2.

Ejercicio.



## Demostración de 3

$A$  es un cilindro sii  $(\forall B) B \leq_m A \Rightarrow B \leq_1 A$ .

$A$  es un cilindro sii  $(\forall B) [B \leq_m A \Rightarrow B \leq_1 A]$ :

⇒ Sea  $A \equiv C \times \mathbb{N}$  y supongamos que  $B \leq_m A$ . Entonces  $B \leq_m C$  vía alguna función computable  $f$ . Sea  $g(x) = \langle f(x), x \rangle$ .

- $g$  es 1-1
- $x \in B$  sii  $g(x) \in C \times \mathbb{N}$

Entonces  $B \leq_1 C \times \mathbb{N} \equiv_1 A$ .

⇐ De 2 sabemos que  $A \times \mathbb{N} \equiv_m A$ . Por hipótesis,  $A \times \mathbb{N} \leq_1 A$ . De 1 sabemos  $A \leq_1 A \times \mathbb{N}$ . Luego  $A \equiv_1 A \times \mathbb{N}$ . Por Teorema de Isomorfismo,  $A \equiv A \times \mathbb{N}$ , de modo que  $A$  es un cilindro.



## Demostración de 4

$A \leq_m B$  si  $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$ .

$A \leq_m B$  si  $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$ :

$\Rightarrow$  Supongamos  $A \leq_m B$ . Entonces

$$A \times \mathbb{N} \leq_m A \leq_m B \leq_1 B \times \mathbb{N}.$$

Luego  $A \times \mathbb{N} \leq_m B \times \mathbb{N}$ . Por 3 (y porque  $B \times \mathbb{N}$  es trivialmente un cilindro),  $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$ .

$\Leftarrow$  Supongamos que  $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$ . Tenemos

$$A \leq_1 A \times \mathbb{N} \leq_1 B \times \mathbb{N} \leq_m B.$$



# Teoría de la Computabilidad

## Clase 3

Conjuntos 1-completos, conjuntos productivos y creativos

# Conjuntos 1-completos

## Definición

Un conjunto c.e.  $A$  es  **$r$ -completo** ( $r \in \{1, m\}$ ) si  $W_e \leq_r A$  para todo  $e$ .

Recordar

$$K_0 := \{\langle x, y \rangle : \varphi_x(y) \downarrow\}.$$

## Proposición

$K_0$  es 1-completo.

## Demostración.

$x \in W_e$  sii  $\langle e, x \rangle \in K_0$ .

$x \mapsto \langle e, x \rangle$  es 1-1.

□

Como  $K$  es c.e., tenemos  $K \leq_1 K_0$  (esto ya lo habíamos probado por otro lado antes).

$K \equiv_1 K_0$

Ya vimos  $K \leq_1 K_0$ .

### Proposición

$K_0 \leq_1 K$ .

### Demostración.

Misma idea que demostración de  $K \leq_1 \text{Tot}$  funciona.

Definir la función parcial computable

$$g(x, z) = \begin{cases} 1 & \text{si } x \in K_0 \\ \uparrow & \text{sino} \end{cases}$$

Por el TP, existe una función 1-1 computable  $f$  tal que  $\varphi_{f(x)}(z) = g(x, z)$ .

- $x \in K_0 \Rightarrow \varphi_{f(x)} = \lambda z. 1 \Rightarrow f(x) \in K$
- $x \notin K_0 \Rightarrow \varphi_{f(x)} = \lambda z. \uparrow \Rightarrow f(x) \notin K$

Entonces  $x \in K_0$  si  $f(x) \in K$ .

□

$K$ ,  $K_0$  y  $K_1$  son 1-completos

Definición

$$K_1 := \{x : W_x \neq \emptyset\}.$$

Proposición

$$K \leq_1 K_1.$$

Demostración.

Ejercicio.



Como  $K_1$  es c.e. y  $K_0$  es 1-completo,  $K_1 \leq_1 K_0$ .

Ya habíamos visto  $K \equiv_1 K_0$  y  $K \leq_1 K_1$ .

Entonces

$$K \equiv_1 K_0 \equiv_1 K_1$$

Corolario

$K$ ,  $K_0$  y  $K_1$  son 1-completos.

## Conjuntos productivos y creativos

Una forma de ver que  $\overline{K}$  no es c.e. es el hecho de que dado el índice de un conjunto  $W$  c.e. incluido en  $\overline{K}$  uno puede obtener un número que está en  $\overline{K}$  pero no en  $W$ :

$$W_x \subseteq \overline{K} \Rightarrow x \in \overline{K} \setminus W_x.$$

### Definición

- Un conjunto  $P$  es **productivo** si existe una función parcial computable  $\psi$ , llamada **función productiva** para  $P$ , tal que

$$(\forall x)[W_x \subseteq P \Rightarrow [\psi(x) \downarrow \wedge \psi(x) \in P \setminus W_x]].$$

- Un conjunto  $C$  es **creativo** si es c.e. y  $\overline{C}$  es productivo.

Por ejemplo,  $K$  es creativo vía la identidad  $\psi(x) = x$ .

### Idea

Un conjunto  $C$  es creativo si es “efectivamente” no computable: para cualquier candidato  $W_x$  para  $\overline{C}$ ,  $\psi(x)$  es un contraejemplo efectivo.

# Conjuntos productivos

## Proposición

Cualquier conjunto productivo es productivo vía una función total.

### Demostración.

Sea  $P$  productivo vía  $\psi$ . Por el TP, sea  $g$  computable tal que

$$\varphi_{g(x)}(y) = \begin{cases} 1 & \text{si } \psi(x) \downarrow \quad \wedge \quad \varphi_x(y) \downarrow \\ \uparrow & \text{si no} \end{cases}$$

Entonces

$$W_{g(x)} = \begin{cases} W_x & \text{si } \psi(x) \downarrow \\ \emptyset & \text{si no} \end{cases}$$

Definir  $q(x)$  como  $\psi(x) \circ \psi(g(x))$ , la que converja primero.

- $W_x \subseteq P \Rightarrow \psi(x) \downarrow \Rightarrow W_{g(x)} = W_x$   
 $\psi(g(x)), \psi(x) \in P \setminus W_x$ . Entonces  $P$  es productivo vía  $q$ .
- $q$  es total:

- $\psi(x) \downarrow \Rightarrow q(x) \downarrow$
- $\psi(x) \uparrow \Rightarrow W_{g(x)} = \emptyset \subseteq P \Rightarrow \psi(g(x)) \downarrow \Rightarrow q(x) \downarrow$

□

# Conjuntos productivos

## Proposición

*Cualquier conjunto productivo es productivo vía una función 1-1 total.*

## Demostración

Sea  $P$  productivo vía  $q$  total. Sea  $h$  computable tal que

$$\varphi_{h(x)}(y) = \begin{cases} 1 & \text{si } \varphi_x(y) \downarrow \quad \vee \quad y = q(x) \\ \uparrow & \text{si no} \end{cases}$$

Tenemos  $W_{h(x)} = W_x \cup \{q(x)\}$ . Observar que

- $W_x \subseteq P \Rightarrow W_{h(x)} \subseteq P \Rightarrow (\forall n) W_{h^{(n)}(x)} \subseteq P$
- $W_{h^{(n)}(x)} = W_x \cup \{q(x), qh(x), \dots, qh^{(n-1)}(x)\}$

(sigue →)

## Demostración (Cont.)

Transformamos  $q$  total en  $p$  total y 1-1. Definimos  $p(0), p(1), \dots$  en ese orden. Definir  $p(0) = q(0)$ . Para  $x > 0$ , computar  $p(x)$  así: enumerar  $\{q(x), qh(x), qh^{(2)}(x), \dots\}$  hasta que ocurre alguno de estos dos casos:

- 1 algún  $qh^{(n)}(x) \notin \{p(0), \dots, p(x-1)\}$ . Definir  $p(x) = qh^{(n)}(x)$ .

- supongamos  $W_x \subseteq P$ . Como  $W_{h^{(n)}(x)} \subseteq P$  entonces

$$p(x) = qh^{(n)}(x) \in P \setminus W_{h^{(n)}(x)} \subseteq P \setminus W_x$$

- 2 existe  $m < n$  tal que  $qh^{(m)}(x) = qh^{(n)}(x)$

- Veamos que  $W_x \not\subseteq P$ : Supongamos  $W_x \subseteq P$   
 $W_x \subseteq P \Rightarrow$  (mismo argumento de arriba)  $\Rightarrow$

$$(\forall k) qh^{(k)}(x) \in P \setminus W_{h^{(k)}(x)}$$

En particular,  $qh^{(n)}(x) \in P \setminus W_{h^{(n)}(x)}$ .

Pero  $qh^{(m)}(x) \in W_{h^{(n)}(x)}$  por definición. Absurdo.

- Entonces da lo mismo cómo definir  $p(x)$ . Basta con que sea un número adecuado para que  $p$  no deje de ser 1-1. Por ejemplo,  
 $p(x) = \min\{y : y \notin \{p(0), \dots, p(x-1)\}\}$ .

□

# Conjuntos Productivos

## Teorema

*Si  $P$  es productivo entonces*

1.  $P$  no es c.e.
2.  $P$  contiene un conjunto c.e. infinito  $W$
3. si  $P \leq_m A$  entonces  $A$  es productivo

## Demostración de 1: $P$ no es c.e.

Inmediato (ejercicio). □

## Demostración de 2: $P$ contiene un conjunto c.e.

Sea  $P$  productivo vía  $p$  total y 1-1. Sea  $W_n = \emptyset$ . Por TP sea  $h$  1-1 tal que  $W_{h(x)} = W_x \cup \{p(x)\}$ . Definir

$$W = \{p(n), ph(n), ph^{(2)}(n), \dots\}$$

- $W$  es c.e.
- $W$  es infinito porque  $p$  y  $h$  son 1-1, y porque  $n \neq h(n)$
- $W \subseteq P$  porque  $P$  es productivo vía  $p$  (ver demo p. 84)



Demostración de 3:  $P \leq_m A$  entonces  $A$  es productivo.

Sea  $P \leq_m A$  vía  $f$ .

Sea  $P$  productivo vía  $p$  total. Sea  $g$  computable tal que

$$\varphi_{g(x)}(y) = \begin{cases} 0 & \text{si } f(y) \in W_x \\ \uparrow & \text{si no} \end{cases}$$

Tenemos

$$W_{g(x)} = f^{-1}(W_x) = \{y : f(y) \in W_x\}$$

Veamos que  $A$  es productivo vía  $fpg$  (total):

- $W_x \subseteq A \Rightarrow f^{-1}(W_x) \subseteq f^{-1}(A)$
- como  $x \in P \Leftrightarrow f(x) \in A$ , entonces  $f^{-1}(A) \subseteq P$
- entonces  $W_{g(x)} \subseteq P$
- luego  $pg(x) \in P \setminus W_{g(x)}$
- veamos que  $fpg(x) \in A \setminus W_x$ :
  - $pg(x) \in P \Rightarrow fpg(x) \in A$
  - $pg(x) \notin W_{g(x)} \Rightarrow fpg(x) \notin W_x$

□

# Productivo implica $\geq_1 \overline{K}$

## Teorema

Si  $P$  es productivo entonces  $\overline{K} \leq_1 P$ .

## Demostración.

Sea  $P$  productivo vía  $p$  total y 1-1. Por TP definir  $f$  computable tal que

$$W_{f(x,y)} = \begin{cases} \{p(x)\} & \text{si } y \in K \\ \emptyset & \text{si no} \end{cases}$$

Por TR con parám., existe una función computable  $n$  1-1 tal que

$$W_{n(y)} = W_{f(n(y),y)} = \begin{cases} \{pn(y)\} & \text{si } y \in K \\ \emptyset & \text{si no} \end{cases}$$

- $y \in K \Rightarrow W_{n(y)} = \{pn(y)\} \Rightarrow W_{n(y)} \not\subseteq P \Rightarrow pn(y) \in \overline{P}$
- $y \in \overline{K} \Rightarrow W_{n(y)} = \emptyset \Rightarrow W_{n(y)} \subseteq P \Rightarrow pn(y) \in P$

Entonces  $\overline{K} \leq_1 P$  vía  $\lambda y. pn(y)$ .



# Creativo implica 1-completo

## Corolario

*Si  $C$  es creativo, entonces  $C$  es 1-completo.*

## Demostración.

$C$  es c.e. y  $\overline{C}$  es productivo.

Por teorema anterior  $\overline{K} \leq_1 \overline{C}$ .

Entonces  $K \leq_1 C$ .

Como  $K$  es 1-completo,  $C$  es 1-completo. □

# Equivalencias

## Corolario

*Los siguientes son equivalentes:*

1.  $P$  es productivo
2.  $\overline{K} \leq_1 P$
3.  $\overline{K} \leq_m P$

## Demostración.

- $1 \Rightarrow 2$  sale de teorema de pág. 89.
- $2 \Rightarrow 3$  es trivial.
- $3 \Rightarrow 1$  sale del hecho de que  $\overline{K}$  es productivo y teorema de pág. 86.

## Corolario

*Los siguientes son equivalentes:*

1.  $C$  es creativo
2.  $C$  es 1-completo
3.  $C$  es  $m$ -completo

# Teoría de la Computabilidad

## Clase 4

Reducibilidad tt, cómputos con oráculo, reducibilidad de Turing, principio del uso

# Tablas de verdad

## Definición

- Una **tabla de verdad**  $k$ -aria es una función  $\{0, 1\}^k \rightarrow \{0, 1\}$ .
- Un par  $\langle \vec{x}, \alpha \rangle$ ,  $\vec{x} \in \mathbb{N}^k$  y  $\alpha$  una tabla de verdad  $k$ -aria es una **tt-condición**
- Una tt-condición  $\langle (x_1, \dots, x_k), \alpha \rangle$  es **satisficha** por  $A$  si  $\alpha(A(x_1), \dots, A(x_k)) = 1$ .

## Ejemplo

$$\alpha = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

- $\langle (n, m), \alpha \rangle$  es satisficha por  $A$  si  $n \in A$  y  $m \in A$

Las tablas de verdad son objetos finitos. Las tt-condiciones también.

Suponemos una codificación de las tt-condiciones con  $\mathbb{N}$ .

## Reducibilidad tabla de verdad (*truth table*)

Sabemos que  $K$ ,  $\overline{K}$  y  $K \oplus \overline{K}$  están en clases  $m$  distintas. Pero intuitivamente son interreducibles.

### Definición

$A$  es *truth table* reducible a  $B$  ( $A \leq_{tt} B$ ) si hay una función computable  $f$  tal que para todo  $x$ ,  $x \in A$  si la tt-condición (codificada por)  $f(x)$  es satisfecha por  $B$

### Ejemplo

Para todo  $A$ :

- $A \leq_{tt} \overline{A}$  vía la función  $f(x)$  que codifica la tt-condición  $\langle x, \alpha \rangle$ , donde  $\alpha(0) = 1$ ,  $\alpha(1) = 0$ .
- $\overline{A} \leq_{tt} A \oplus \overline{A}$  vía la función  $f(x)$  que codifica la tt-condición  $\langle 2x + 1, \alpha \rangle$ , donde  $\alpha(0) = 0$ ,  $\alpha(1) = 1$ .
- $A \oplus \overline{A} \leq_{tt} A$  vía  $f(2x) = \langle x, \alpha \rangle$ , donde  $\alpha(0) = 0$ ,  $\alpha(1) = 1$ ;  
 $f(2x + 1) = \langle x, \alpha \rangle$ , donde  $\alpha(0) = 1$ ,  $\alpha(1) = 0$ .

## Propiedades de $\leq_{tt}$

### Definición

$A \equiv_{tt} B$  si  $A \leq_{tt} B$  y  $B \leq_{tt} A$ .

### Definición

Un conjunto c.e.  $A$  es **tt-completo** si  $W_e \leq_{tt} A$  para todo  $e$ .

### Teorema

- ①  $A \leq_m B \Rightarrow A \leq_{tt} B$
- ②  $A \leq_{tt} \overline{A}$  (*por lo tanto  $A \equiv_{tt} \overline{A}$* )
- ③  $\leq_{tt}$  forma un semireticulo superior
- ④ Si  $A \leq_{tt} B$  y  $B$  computable, entonces  $A$  es computable
- ⑤ Si  $A$  es computable entonces  $(\forall B) A \leq_{tt} B$

### Demostración.

Ejercicio.



$\leq_{tt}$  tampoco alcanza

## Consideremos

$$C = \{x : \varphi_x(x) \downarrow \text{ y la tt-condición } \varphi_x(x) \text{ es satisfecha por } K\}$$

Intuitivamente  $C$  es reducible a  $K$ , es decir

$C \leq_{intuitivamente} K$ ,

pero...

## Proposición

$$C \not\leq_{tt} K.$$

## Demostración.

Supongamos  $C \leq_{tt} K$ . Entonces  $\overline{C} \leq_{tt} K$  vía  $\varphi_e$  total.



# Máquinas de Turing

- una **cinta**
  - dividida en celdas
  - infinita en ambas direcciones
  - cada celda contiene un símbolo de un alfabeto dado  $\Sigma$ .
    - $* \in \Sigma$
    - $L, R \notin \Sigma$ 
      - \* representa el blanco en una celda
      - $L$  y  $R$  son símbolos reservados (representarán acciones que puede realizar la cabeza)
- una **cabeza**
  - lee y escribe un símbolo a la vez
  - se mueve una posición a la izquierda o una posición a la derecha
- una **tabla finita de instrucciones**
  - dice qué hacer en cada paso

# Máquinas de Turing

Una **máquina de Turing (determinística)** es una tupla

$$(\Sigma, Q, \delta, q_0, q_f)$$

donde

- $\Sigma$  (finito) es el conjunto **símbolos** (Fijamos  $\Sigma = \{*, 1\}$ )
  - $Q$  (finito) es el conjunto de **estados**
    - tiene dos estados distinguidos:
      - $q_0 \in Q$  es el **estado inicial**
      - $q_f \in Q$  es el **estado final**
  - $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{R, L\}$  es la **tabla de instrucciones** o **programa**
    - va a ser finita porque  $\Sigma$  y  $Q$  lo son
    - $\delta(q, s) = (q', s', d)$  se interpreta como
- Si la máquina está en el estado  $q$  leyendo en la cinta el símbolo  $s$ , entonces escribe  $s'$  sobre  $s$ , se mueve hacia la izquierda si  $d = L$  o derecha si  $d = R$ , y pasa al estado  $q'$*

## Máquinas de Turing con oráculo

Es una Máquina de Turing con una cinta adicional que es solo de lectura.

- la nueva cinta se llama **cinta oracular** (fijamos  $\Sigma' = \{*, 0, 1\}$ )
- la vieja cinta se llama **cinta de trabajo** y funciona igual que antes (fijamos  $\Sigma = \{*, 1\}$ )

La nueva tabla de instrucciones o programa de una máquina de Turing oracular es una función

$$\delta : Q \times \Sigma \times \Sigma' \rightarrow Q \times \Sigma \times \{R, L\}$$

$\delta(q, s, x) = (q', s', d)$  se interpreta como

*Si la máquina está en el estado  $q$  leyendo en la cinta el símbolo  $s$  y en la cinta oracular está el símbolo  $x$ , entonces escribe  $s'$  sobre  $s$  en la cinta de trabajo, pasa al estado  $q'$  y se mueve hacia la izquierda si  $d = L$  o derecha si  $d = R$  en ambas cintas*

## Enumeración de máquinas con oráculo

Igual que antes, los programas se pueden enumerar:  $\Phi_e$  es el  $e$ -ésimo programa, o la  $e$ -ésima máquina de Turing oracular.

$\Phi_e$  necesita dos entradas

- una entrada finita  $x_1, \dots, x_n$  para la cinta de trabajo
- una entrada posiblemente infinita  $A \subseteq \mathbb{N}$  para la cinta oracular

Se suele escribir  $\Phi_e^A(x_1, \dots, x_n)$  para referirse a la  $e$ -ésima máquina de Turing oracular con oráculo  $A$  y entrada  $x_1, \dots, x_n$ .

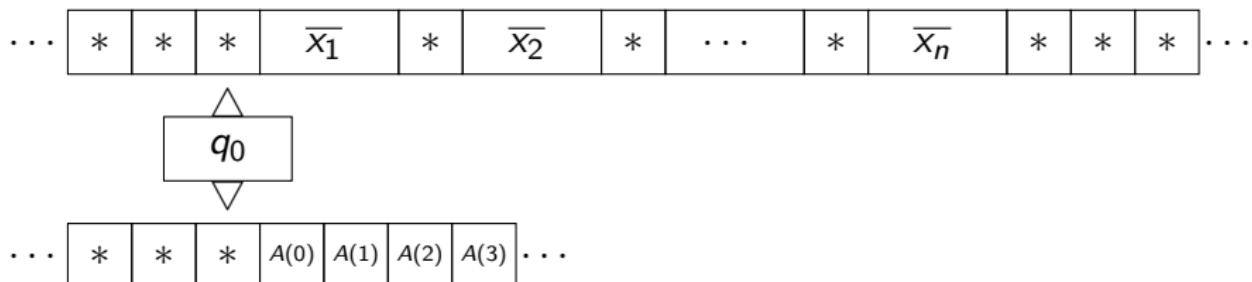
# Configuración inicial

Se codifica

- la entrada  $x_1, \dots, x_n$  como antes (sobre alfabeto  $\Sigma = \{*, 1\}$ )
- el oráculo  $A$  como una lista de ceros y unos  $A(0), A(1), A(2) \dots$  (sobre alfabeto  $\Sigma' = \{*, 0, 1\}$ ). Como siempre

$$A(n) = \begin{cases} 1 & \text{si } n \in A \\ 0 & \text{si no} \end{cases}$$

La configuración inicial es esta:



(con los números  $x_i$  representados en unario)

# Cómputos relativos a oráculos

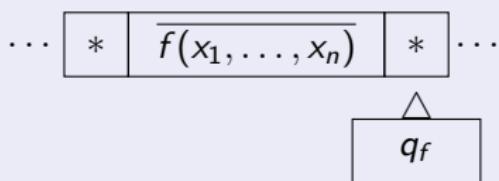
## Definición

Una función parcial  $f$  es **Turing computable en  $A$**  (o  **$A$ -computable**), notado  $f \leq_T A$ , si existe un programa  $e$  tal que

$$f(x_1, \dots, x_n) = \Phi_e^A(x_1, \dots, x_n),$$

es decir:

- si  $f(x_1, \dots, x_n) \downarrow$  entonces  $\Phi_e^A(x_1, \dots, x_n)$ , partiendo de la configuración inicial y siguiendo sus instrucciones, llega a una configuración final de la forma



(quizá algo más en la cinta de trabajo; recordar que la cinta oracular es solo de lectura y se mueve a la par de la cinta de trabajo)

- si  $f(x_1, \dots, x_n) \uparrow$  entonces  $\Phi_e^A(x_1, \dots, x_n)$  nunca llega al estado final.

## El uso

En cualquier cómputo  $\Phi_e^A(x_1, \dots, x_n) \downarrow$  la máquina  $e$  solo mira una cantidad finta de elementos del oráculo. Sea  $y$  el máximo número de celdas no  $*$  consultadas durante ese cómputo. Decimos que los elementos  $z \leq y$  son **usados** en el cómputo.

### Definición

- Notamos  $\Phi_{e,s}^A(x) = y$  para  $x, y, e < s, s > 0$ , si  $\Phi_e^A(x) = y$  en  $< s$  pasos y usando solo números  $z < s$  del oráculo.
- Notamos  $\Phi_{e,s}^A(x) \downarrow$  para  $x, y, e < s, s > 0$ , si  $\Phi_{e,s}^A(x) = y$  para algún  $y$ .
- El **uso**  $u(A, e, x, s)$  se define como

$$u(A, e, x, s) = \begin{cases} 1 + \text{el máximo número usado} & \text{si } \Phi_{e,s}^A(x) \downarrow \\ \text{en el cómputo } \Phi_{e,s}^A(x) \\ 0 & \text{si no} \end{cases}$$

- Para  $\sigma \in \{0, 1\}^*$ , notamos  $\Phi_{e,s}^\sigma(x) = y$ , si  $\Phi_{e,s}^A(x) = y$  para algún  $A \supset \sigma$ , y solo elementos menores a  $|\sigma|$  son usados en este último cómputo.

## Propiedades del uso

Si  $\Phi_{e,s}^A(x) = y$  entonces

- $x, y, e < s$  y  $u(A, e, x, s) \leq s$
- $\Phi_{e,s}^\sigma(x) = y$  donde  $\sigma = A \upharpoonright u(A, e, x, s)$
- $(\forall t \geq s) [\Phi_{e,t}^A(x) = y \wedge u(A, e, x, s) = u(A, e, x, t)]$

Habíamos definido  $u(A, e, x, s)$ .

### Definición

$u(A, e, x)$  es

$$u(A, e, x) = \begin{cases} u(A, e, x, s) & \text{si } \Phi_{e,s}^A(x) \downarrow \text{ para algún } s \\ \uparrow & \text{si no} \end{cases}$$

# El principio del uso (*use principle*)

## Teorema

- ①  $\Phi_e^A(x) = y \Rightarrow (\exists s)(\exists \sigma \subset A) \Phi_{e,s}^\sigma(x) = y$
- ②  $\Phi_{e,s}^\sigma(x) = y \Rightarrow (\forall t \geq s)(\forall \tau \supset \sigma) \Phi_{e,t}^\tau(x) = y$
- ③  $\Phi_e^\sigma(x) = y \Rightarrow (\forall A \supset \sigma) \Phi_e^A(x) = y$

## Corolario

Si  $\Phi_{e,s}^A(x) = y$  y  $B \upharpoonright u = A \upharpoonright u$  entonces  $\Phi_{e,s}^B(x) = y$ , donde  $u = u(A, e, x, s)$ .

# Propiedades de $\Phi_e^A$ y $\Phi_e^\sigma$

## Teorema

- 1  $\{\langle e, \sigma, x, s \rangle : \Phi_{e,s}^\sigma(x) \downarrow\}$  es computable
- 2  $\{\langle e, \sigma, x \rangle : \Phi_e^\sigma(x) \downarrow\}$  es c.e.

## Teorema

Son funciones parciales A-computables:

- 1  $\lambda e, s, x. \Phi_{e,s}^A(x)$  *(esta no necesariamente es total)*
- 2  $\lambda e, s, x. \begin{cases} 1 & \text{si } \Phi_{e,s}^A(x) \downarrow \\ 0 & \text{si no} \end{cases}$  *(esta es total)*

# Teoría de la Computabilidad

## Clase 5

Teorema del Salto, Lema del Límite, grados Turing

## Relativizaciones de resultados conocidos

### Teorema (Teorema del Parámetro Relativizado - TPR)

Para cada  $m, n \geq 1$  existe una función primitiva recursiva 1-1  $s_n^m$  de  $m + 1$  variables tal que para todo  $A \subseteq \mathbb{N}$  y para todo  $y_1, \dots, y_m, z_1, \dots, z_n \in \mathbb{N}$ ,

$$\Phi_{s_n^m(x, y_1, \dots, y_m)}^A(z_1, \dots, z_n) = \Phi_x^A(y_1, \dots, y_m, z_1, \dots, z_n)$$

### Teorema (Teorema de la Recursión Relativizado - TRR)

Para cualquier función  $A$ -computable  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$  existe una función computable  $n$  tal que

$$\Phi_{n(y)}^A = \Phi_{f(n(y), y)}^A.$$

Es más, la definición de  $n$  no depende de  $A$ . Es decir, si  $f(x, y) = \Phi_e^A(x, y)$  entonces  $n$  se puede definir uniformemente en  $e$ .

# Conjuntos $A$ -computables, $A$ -c.e. y $\Sigma_1^A$

## Definición

- $W_e^A = \text{dom } \Phi_e^A$
- $W_{e,s}^A = \text{dom } \Phi_{e,s}^A$
- $W_e^\sigma = \text{dom } \Phi_e^\sigma$
- $W_{e,s}^\sigma = \text{dom } \Phi_{e,s}^\sigma$

## Definición

- $B$  es **computable en  $A$**  (o  **$A$ -computable**), notado  $B \leq_T A$ , si  $B = \Phi_e^A$  para algún  $e$ .
  - notamos  $A \equiv_T B$  cuando  $A \leq_T B$  y  $B \leq_T A$
  - notamos  $A <_T B$  cuando  $A \leq_T B$  y  $B \not\leq_T A$
  - notamos  $\deg_T(A) = \{B : B \equiv_T A\}$
- $B$  es **c.e. en  $A$**  (o  **$A$ -c.e.**) si  $B = W_e^A$  para algún  $e$
- $B$  es de la forma  $\Sigma_1^A$  (abreviado  $B \in \Sigma_1^A$ ) cuando  $B = \{x : (\exists \vec{y}) R^A(x, \vec{y})\}$  para algún predicado  $A$ -computable  $R^A$ .

## Convención

Definimos  $\Phi_{e,s}^A$  de modo que

$$\Phi_{e,s}^A(x) \downarrow \Rightarrow x, e < s$$

Si  $B$  es  $A$ -c.e., entonces  $B = \text{dom } \Phi_e^A$  para algún  $e$ . Tenemos una  $A$ -enumeración natural de  $B$ . En general se la nota  $(B_s)_{s \in \mathbb{N}}$  y está definida como

$$B_s = \text{dom } \Phi_{e,s}^A \subseteq \{0, \dots, s-1\}.$$

En realidad,  $B_s$  abrevia una función  $A$ -computable  $b : \mathbb{N} \rightarrow \mathbb{N}$  tal que

$$b(s) = \text{codificación de } B_s \text{ usando } \mathbb{N}$$

$B_s$  satisface

- para todo  $x$ ,  $B(x) = \lim_s B_s(x)$
- $(\forall n)(\exists s) B \upharpoonright n = B_s \upharpoonright n$

# Equivalencias entre $A$ -c.e. y $\Sigma_1^A$

## Teorema

Son equivalentes:

1.  $B$  es  $A$ -c.e.
2.  $B$  es  $\Sigma_1^A$
3.  $B = \emptyset$  o  $B$  es el rango de alguna función  $A$ -computable

Demostración de  $1 \Rightarrow 2$ :

$B$  es  $A$ -c.e.  $\Rightarrow B$  es  $\Sigma_1^A$ .

Sea  $B = \text{dom } \Phi_e^A$  entonces  $B = \{x : (\exists s) \Phi_{e,s}^A(x) \downarrow\}$ .

Recordar que

$$\lambda e, s, x. \begin{cases} 1 & \text{si } \Phi_{e,s}^A(x) \downarrow \\ 0 & \text{si no} \end{cases}$$

es  $A$ -computable.



Demostración de 2  $\Rightarrow$  3:

$B$  es  $\Sigma_1^A$   $\Rightarrow$   $B = \emptyset$  o  $B$  es el rango de alguna función  $A$ -computable.

Sea  $b \in B$  y sea  $B = \{x : (\exists \vec{y}) R^A(x, \vec{y})\}$ . Supongamos  $R^A$  es  $(n+1)$ -ario. Definimos la función  $A$ -computable

$$f(\langle x, y_1, \dots, y_n \rangle) = \begin{cases} x & \text{si } R^A(x, y_1, \dots, y_n) \\ b & \text{si no} \end{cases}$$

- veamos que  $x \in B \Rightarrow (\exists z) f(z) = x$ :

$$x \in B \Rightarrow (\exists \vec{y}) R^A(x, \vec{y}) \Rightarrow f(\langle x, \vec{y} \rangle) = x$$

- veamos que  $(\forall z) f(z) \in B$ :

Sea  $z = \langle x, \vec{y} \rangle$ .

- $\neg R^A(x, \vec{y}) \Rightarrow f(z) = b \in B$

- $R^A(x, \vec{y}) \Rightarrow x \in B$ . Luego  $f(z) = f(\langle x, \vec{y} \rangle) = x \in B$



Demostración de 3  $\Rightarrow$  1:

$B = \emptyset$  o  $B$  es el rango de alguna función  $A$ -computable  $\Rightarrow$   
 $B$  es  $A$ -c.e.

Sea  $e$  tal que  $\Phi_e^A(\mathbb{N}) = B$ . Definir la función parcial

$$f(y) = \begin{cases} 0 & \text{si } (\exists x, s) \Phi_{e,s}^A(x) = y \\ \uparrow & \text{si no} \end{cases}$$

$f$  es parcial  $A$ -computable y  $\text{dom } f = B$ .



# Relativizaciones de resultados anteriores

## Teorema

*Si  $B$  es  $A$ -computable entonces  $\overline{B}$  también.*

## Teorema

*Si  $B$  es  $A$ -computable entonces  $B$  es  $A$ -c.e.*

## Teorema (Complementación)

*Si  $B$  y  $\overline{B}$  son  $A$ -c.e. entonces  $B$  es  $A$ -computable.*

# El salto

Recordar que

$$\begin{aligned}K &= \{x : \varphi_x(x) \downarrow\} = \{x : x \in W_x\} \\K_0 &= \{\langle x, y \rangle : \varphi_x(y) \downarrow\}\end{aligned}$$

Se pueden relativizar

$$\begin{aligned}K^A &= \{x : \Phi_x^A(x) \downarrow\} = \{x : x \in W_x^A\} \\K_0^A &= \{\langle x, y \rangle : \Phi_x^A(y) \downarrow\}\end{aligned}$$

## Definición

El **salto** (*jump*) de  $A$ , notado  $A'$ , se define como

$$A' = K^A = \{x : \Phi_x^A(x) \downarrow\} = \{x : x \in W_x^A\}$$

$A^{(n)}$  es el  **$n$ -esimo salto** de  $A$ . Se define como  $A^{(0)} = A$  y  $A^{(n+1)} = (A^{(n)})'$ .

Observar que  $\emptyset' = K$ .

## Teorema

$$K^A \equiv_1 K_0^A$$

# El Teorema del Salto (TS)

## Teorema

1.  $A'$  es  $A$ -c.e.
2.  $A' \not\leq_T A$
3.  $B$  es  $A$ -c.e. si  $B \leq_1 A'$
4. si  $A$  es  $B$ -c.e. y  $B \leq_T C$  entonces  $A$  es  $C$ -c.e.
5.  $B \leq_T A$  si  $B' \leq_1 A'$
6. si  $B \equiv_T A$  entonces  $B' \equiv_1 A'$
7.  $A$  es  $B$ -c.e. si  $A$  es  $\overline{B}$ -c.e.

### Demostración de 1: $A'$ es $A$ -c.e.

La función parcial  $f(x) = \Phi_x^A(x)$  es  $A$ -computable y  $\text{dom } f = A'$ .  
Entonces  $A'$  es  $A$ -c.e. □

### Demostración de 2: $A' \not\leq_T A$ .

Supongamos que  $A'$  es  $A$ -computable. Entonces  $\overline{A'}$  es  $A$ -c.e. Luego existe  $e$  tal que

$$W_e^A = \overline{A'} = \{x : x \notin W_x^A\}.$$

Entonces  $e \in W_e^A$  si  $e \notin W_e^A$ . Absurdo. □

### Demostración de 3: $B$ es $A$ -c.e. sii $B \leq_1 A'$ .

- ⇒ Existe  $e$  tal que  $B = W_e^A = \text{dom } \Phi_e^A$ . Entonces  $x \in B$  sii  $\Phi_e^A(x) \downarrow$  sii  $\langle x, e \rangle \in K_0^A$ . Como  $\lambda x. \langle x, e \rangle$  es 1-1, entonces  $B \leq_1 K_0^A$ . Además,  $K_0^A \equiv_1 K^A = A'$ .
- ⇐ Supongamos  $f$  computable tal que  $x \in B$  sii  $f(x) \in A'$ . Entonces  $x \in B$  sii  $\Phi_{f(x)}^A(f(x)) \downarrow$ . La función parcial  $\lambda x. \Phi_{f(x)}^A(f(x))$  es  $A$ -computable, entonces su dominio (que es  $B$ ) es  $A$ -c.e.



Demostración de 4:

si  $A$  es  $B$ -c.e. y  $B \leq_T C$  entonces  $A$  es  $C$ -c.e.

Por Teorema pág. 107, si  $A \neq \emptyset$ ,  $A$  es el rango de una función  $f$  tal que  $f \leq_T B$ . Como  $B \leq_T C$  entonces  $f \leq_T C$ . Luego  $A$  es el rango de una función  $C$ -computable.  $\square$

Demostración de 5:  $B \leq_T A$  sii  $B' \leq_1 A'$ .

$\Rightarrow$  Por TS1  $B'$  es  $B$ -c.e. Como  $B \leq_T A$ , por TS4,  $B'$  es  $A$ -c.e.

Por TS3,  $B' \leq_1 A'$ .

$\Leftarrow$   $B, \overline{B} \leq_1 A'$  (ejercicio). Por TS3  $B$  y  $\overline{B}$  son  $A$ -c.e. Por

Teorema de Complementación (pág. 110),  $B \leq_T A$ .



Demostración de 6: si  $B \equiv_T A$  entonces  $B' \equiv_1 A'$ .

Directo de TS5.



Demostración de 7:  $A$  es  $B$ -c.e. si  $A$  es  $\overline{B}$ -c.e.

Directo de TS4.



## Lema del límite

### Teorema

Para toda función total  $f$ ,  $f \leq_T A'$  si y sólo si hay una función  $A$ -computable  $g$  de dos variables tal que  $f(x) = \lim_s g(x, s)$ .

## Demostración de $(\Rightarrow)$ .

Supongamos que  $f = \Phi_e^{A'}$ . Por TS1  $A'$  es  $A$ -c.e. Sea  $(A'_s)_{s \in \mathbb{N}}$  una enumeración  $A$ -computable de  $A'$ . Definir:

$$g(x, s) = \begin{cases} \Phi_{e,s}^{A'_s}(x) & \text{si } \Phi_{e,s}^{A'_s}(x) \downarrow \\ 0 & \text{si no} \end{cases}$$

Claramente  $g \leq_T A$ . Sea  $t$  suficientemente grande tal que

$$(\exists z \leq t) [\Phi_{e,t}^{A'_t \upharpoonright z}(x) \downarrow \wedge A'_t \upharpoonright z = A' \upharpoonright z]$$

Entonces para todo  $s \geq t$  tenemos

$$\Phi_{e,s}^{A'_s \upharpoonright z}(x) = \Phi_{e,t}^{A'_s \upharpoonright z}(x) = \Phi_e^{A' \upharpoonright z}(x) = \Phi_e^{A'}(x) = f(x)$$

Luego  $\lim_s g(x, s) = f(x)$ .

□

## Demostración de ( $\Leftarrow$ ).

Sup.  $f(x) = \lim_s g(x, s)$  para  $g$   $A$ -computable. Definir

$$A_x = \{s : (\exists t) [s \leq t \wedge g(x, t) \neq g(x, t + 1)]\}$$

- $A_x$  es finito porque existe el límite  $\lim_s g(x, s)$
- $s \notin A_x \Rightarrow (\forall t \geq s) g(x, t) = g(x, t + 1) \Rightarrow g(x, s) = f(x)$

$$\begin{aligned} B &= \{\langle s, x \rangle : s \in A_x\} \\ &= \{z : (\exists t) [\pi_1(z) \leq t \wedge g(\pi_2(z), t) \neq g(\pi_2(z), t + 1)]\} \end{aligned}$$

es claramente  $\Sigma_1^A$ , luego  $A$ -c.e., luego  $B \leq_T A'$  (TS).

La siguiente función es  $B$ -computable:

$$m(x) = \min\{s : s \notin A_x\} = \min\{s : \langle s, x \rangle \notin B\}.$$

Entonces  $g(x, m(x)) = f(x)$ . Como  $g \leq_T A \leq_T A'$  y  $m \leq_T B \leq_T A'$  entonces  $f \leq A'$ .



# Grados Turing

Recordar que

$$\deg_T(A) = \{B : B \equiv_T A\}$$

Si  $\mathbf{a}$  es un grado Turing,  $\mathbf{a}' = \deg_T(A')$  para  $A \in \mathbf{a}$ .

Sabemos que

- $\mathbf{a}' >_T \mathbf{a}$
- $\mathbf{a}'$  es c.e. en  $\mathbf{a}$

Sea  $\mathbf{0}^{(n)} = \deg_T(\emptyset^{(n)})$ . Entonces tenemos una jerarquía de grados

$$\mathbf{0} <_T \mathbf{0}' <_T \mathbf{0}'' <_T \mathbf{0}''' \dots$$

## Ejemplos

### Algunos conjuntos en los primeros saltos de $\mathbf{0}$

$$\mathbf{0} = \deg_T(\emptyset) = \{B : B \text{ es computable}\}$$

$$\mathbf{0}' = \deg_T(\emptyset') = \deg_T(K) = \deg_T(K_0) = \deg_T(K_1)$$

$$\mathbf{0}'' = \deg_T(\emptyset'') = \deg_T(\text{Tot}) = \deg_T(\text{Fin})$$

$$\mathbf{0}''' = \deg_T(\emptyset''') = \deg_T(\text{Cof}) = \deg_T(\text{Rec}) = \deg_T(\text{Ext})$$

donde

$$\text{Tot} := \{x : W_x = \mathbb{N}\} = \{x : \Phi_x \text{ es total}\}$$

$$\text{Fin} := \{x : W_x \text{ es finito}\}$$

$$\text{Cof} := \{x : W_x \text{ es cofinito}\}$$

$$\text{Rec} := \{x : W_x \text{ es computable}\}$$

$$\text{Ext} := \{x : \Phi_x \text{ es extendible a una función total computable}\}$$

# Teoría de la Computabilidad

## Clase 6

Jerarquía Aritmética, Teorema de Post, Conjuntos  
 $\Sigma_n$ -completos

# Conjuntos $\Sigma_n$ y $\Pi_n$

## Definición

- Un conjunto  $B$  es  $\Sigma_0$  o  $\Pi_0$  si  $B$  es computable
- Para  $n \geq 1$ , un conjunto  $B$  es  $\Sigma_n$  (notado  $B \in \Sigma_n$ ) si existe una relación computable  $R(x, y_1, \dots, y_n)$  tal que

$$x \in B \quad \text{sii} \quad \underbrace{(\exists y_1)(\forall y_2)(\exists y_3) \dots (Qy_n)}_{n-1 \text{ alternancias}} R(x, y_1, \dots, y_n)$$

donde  $Q = \exists$  si  $n$  es par y  $Q = \forall$  si  $n$  es impar.

- Para  $n \geq 1$ , un conjunto  $B$  es  $\Pi_n$  (notado  $B \in \Pi_n$ ) si existe una relación computable  $R(x, y_1, \dots, y_n)$  tal que

$$x \in B \quad \text{sii} \quad \underbrace{(\forall y_1)(\exists y_2)(\forall y_3) \dots (Qy_n)}_{n-1 \text{ alternancias}} R(x, y_1, \dots, y_n)$$

donde  $Q = \forall$  si  $n$  es par y  $Q = \exists$  si  $n$  es impar.

- $B$  es  $\Delta_n$  (notado  $B \in \Delta_n$ ) si  $B \in \Sigma_n \cap \Pi_n$ .

# Relaciones y fórmulas $\Sigma_n$ y $\Pi_n$

## Definición

- Una **relación**  $R \subseteq \mathbb{N}^k$  es  $\Sigma_n$  ( $\Pi_n$ ) si es  $\Sigma_n$  ( $\Pi_n$ ) el conjunto

$$\{\langle r_1, \dots, r_k \rangle : (r_1, \dots, r_k) \in R\}$$

- Una **fórmula**  $\varphi(x_1, \dots, x_n)$  con variables libres  $x_1, \dots, x_n$  es  $\Sigma_n$  ( $\Pi_n$ ) si es  $\Sigma_n$  ( $\Pi_n$ ) la relación

$$\{(a_1, \dots, a_n) : \mathbb{N} \models \varphi(a_1, \dots, a_n)\}$$

## Ejemplo

- $R \subseteq \mathbb{N}^3$  es  $\Sigma_2$  si existe una relación  $R'$  computable tal que

$$(x, y, z) \in R \quad \text{sii} \quad (\exists u)(\forall v) R'(x, y, z, u, v).$$

- si  $R$  es computable, la siguiente fórmula es  $\Pi_3$ :

$$(\forall x)(\exists y)(\forall z) R(x, y, z, w)$$

# Propiedades de $\Sigma_n$ y $\Pi_n$

## Teorema

1.  $A \in \Sigma_n \Rightarrow \overline{A} \in \Pi_n$
2.  $A \in \Sigma_n(\Pi_n) \Rightarrow (\forall m > n) A \in \Sigma_m \cap \Pi_m$
3.  $A, B \in \Sigma_n(\Pi_n) \Rightarrow A \cup B, A \cap B \in \Sigma_n(\Pi_n)$
4.  $R \in \Sigma_n \wedge n > 0 \wedge A = \{x : (\exists y) R(x, y)\} \Rightarrow A \in \Sigma_n$
- 4'.  $R \in \Pi_n \wedge n > 0 \wedge A = \{x : (\forall y) R(x, y)\} \Rightarrow A \in \Pi_n$
5.  $B \leq_m A \wedge A \in \Sigma_n(\Pi_n) \Rightarrow B \in \Sigma_n(\Pi_n)$
6. Si  $R \in \Sigma_n(\Pi_n)$  entonces están en  $\Sigma_n(\Pi_n)$ :  
$$A = \{\langle x, y \rangle : (\forall z < y) R(x, y, z)\}$$
$$B = \{\langle x, y \rangle : (\exists z < y) R(x, y, z)\}$$

Demostración de 1:  $A \in \Sigma_n \Rightarrow \bar{A} \in \Pi_n$ .

Si  $A = \{x : (\exists y_1)(\forall y_2)(\exists y_3) \dots R(x, y_1, \dots, y_n)\}$  entonces  
 $\bar{A} = \{x : (\forall y_1)(\exists y_2)(\forall y_3) \dots \neg R(x, y_1, \dots, y_n)\}$

□

Demostración de 2:  $A \in \Sigma_n(\Pi_n) \Rightarrow (\forall m > n) A \in \Sigma_m \cap \Pi_m$ .

Observar que

$$(\exists x_1)(\forall x_2)(\exists x_3) \dots (Qx_n) R(x, x_1, \dots, x_n) \quad \text{sii}$$

$$(\forall x_0)(\exists x_1)(\forall x_2)(\exists x_3) \dots (Qx_n) \underbrace{R(x, x_1, \dots, x_n)}_{R'(x, x_0, x_1, \dots, x_n)} \quad \text{sii}$$

$$(\exists x_1)(\forall x_2)(\exists x_3) \dots (Qx_n)(\tilde{Q}x_{n+1}) \underbrace{R(x, x_1, \dots, x_n)}_{R'(x, x_1, \dots, x_n, x_{n+1})}$$

donde  $\tilde{Q} = \exists$  si  $Q = \forall$  y viceversa.

Lo mismo para fórmulas  $\Pi_n$ .

□

Demostración de 3:  $A, B \in \Sigma_n(\Pi_n) \Rightarrow A \cup B, A \cap B \in \Sigma_n(\Pi_n)$ .

Supongamos

$$\begin{array}{ll} x \in A & \text{sii } (\exists y_1)(\forall y_2) \dots R(x, \vec{y}) \\ x \in B & \text{sii } (\exists z_1)(\forall z_2) \dots S(x, \vec{z}) \end{array}$$

Entonces

$$\begin{aligned} x \in A \cup B & \text{sii } (\exists y_1)(\forall y_2) \dots R(x, \vec{y}) \vee (\exists z_1)(\forall z_2) \dots S(x, \vec{z}) \\ & \text{sii } (\exists y_1)(\exists z_1)(\forall y_2)(\forall z_2) \dots R(x, \vec{y}) \vee S(x, \vec{z}) \\ & \text{sii } (\exists u_1)(\forall u_2) \dots R(x, \pi_1(u_1), \pi_1(u_2), \dots) \vee \\ & \quad S(x, \pi_2(u_1), \pi_2(u_2), \dots) \end{aligned}$$

El mismo truco para  $A \cap B$  y para  $\Pi_n$ . □

Demostración de 4:

$$R \in \Sigma_n \wedge n > 0 \wedge A = \{x : (\exists y) R(x, y)\} \Rightarrow A \in \Sigma_n.$$

Supongamos

$$(x, y) \in R \quad \text{sii} \quad (\exists z_1)(\forall z_2) \dots R'(x, y, z_1, z_2, \dots).$$

Entonces

$$\begin{aligned} x \in A &\quad \text{sii} \quad (\exists y) R(x, y) \\ &\quad \text{sii} \quad (\exists y)(\exists z_1)(\forall z_2) \dots R'(x, y, z_1, z_2, \dots) \\ &\quad \text{sii} \quad (\exists u)(\forall z_2) \dots R'(x, \pi_1(u), \pi_1(u), z_2, \dots) \end{aligned}$$



Demostración de 4':

$$R \in \Pi_n \wedge n > 0 \wedge A = \{x : (\forall y) R(x, y)\} \Rightarrow A \in \Pi_n.$$

Análogo a 4.



Demostración de 5:  $B \leq_m A \wedge A \in \Sigma_n(\Pi_n) \Rightarrow B \in \Sigma_n(\Pi_n)$ .

Sea  $A = \{x : (\exists y_1)(\forall y_2) \dots R(x, \vec{y})\}$  y sea  $B \leq_m A$  via  $f$ . Entonces

$$B = \{x : (\exists y_1)(\forall y_2) \dots \underbrace{R(f(x), \vec{y})}_{\text{computable}}\}$$

Lo mismo para el caso  $\Pi_n$ .



Demostración de 6: Si  $R \in \Sigma_n(\Pi_n)$  entonces están en  $\Sigma_n(\Pi_n)$ :  
 $A = \{\langle x, y \rangle : (\forall z < y) R(x, y, z)\}$  y  
 $B = \{\langle x, y \rangle : (\exists z < y) R(x, y, z)\}.$

Por inducción en  $n$ . Si  $n = 0$ ,  $A$  y  $B$  son computables. Sea  $n > 0$  y supongamos cierta la propiedad para todo  $< n$ . Supongamos  $R \in \Sigma_n$ . Por 4,  $B \in \Sigma_n$ . Existe  $S \in \Pi_{n-1}$  tal que

$$R(x, y, z) \quad \text{sii} \quad (\exists u) S(x, y, z, u).$$

Entonces

$$\begin{aligned} \langle x, y \rangle \in A &\quad \text{sii} \quad (\forall z < y) R(x, y, z) \\ &\quad \text{sii} \quad (\forall z < y) (\exists u) S(x, y, z, u) \\ &\quad \text{sii} \quad (\exists \sigma) (\forall z < y) S(x, y, z, \pi_z(\sigma)) \end{aligned}$$

donde  $\sigma$  se interpreta como una  $y$ -upla.

Por HI

$$\{\langle x, y, \sigma \rangle : (\forall z < y) S(x, y, z, \pi_z(\sigma))\} \in \Pi_{n-1}.$$

Entonces  $A \in \Sigma_n$ .

El caso  $R \in \Pi_n$  es análogo.



## Ejemplos

$\text{Fin} := \{x : W_x \text{ es finito}\}$

### Proposición

$\text{Fin} \in \Sigma_2$ .

### Demostración.

$x \in \text{Fin}$  si  $W_x$  es finito si

$(\exists s)(\forall t) [t > s \Rightarrow \overbrace{W_{x,s} = W_{x,t}}^{\text{computable}}]$



## Ejemplos

$\text{Cof} := \{x: W_x \text{ es cofinito}\}$

### Proposición

$\text{Cof} \in \Sigma_3$ .

### Demostración.

$x \in \text{Cof}$  sii  $\overline{W_x}$  es finito sii

$$(\exists y)(\forall z) [z > y \Rightarrow z \in W_x]$$

sii

$$(\exists y)(\forall z)(\exists s) [z > y \Rightarrow \underbrace{z \in W_{x,s}}_{\text{computable}}]$$



# Conjuntos $\Sigma_n$ y $\Pi_n$ -completos y Teorema de Post (TP)

## Definición

Un conjunto  $A$  es  **$\Sigma_n$ -completo** ( **$\Pi_n$ -completo**) si  $A \in \Sigma_n$  ( $\Pi_n$ ) y  $B \leq_1 A$  para todo  $B \in \Sigma_n$  ( $\Pi_n$ ).

## Teorema (Post)

Para todo  $n \geq 0$ :

1.  $B \in \Sigma_{n+1}$  si  $B$  es c.e. en algún conjunto  $\Pi_n$  si  $B$  es c.e. en algún conjunto  $\Sigma_n$
2.  $\emptyset^{(n)}$  es  $\Sigma_n$ -completo, para  $n > 0$
3.  $B \in \Sigma_{n+1}$  si  $B$  es c.e. en  $\emptyset^{(n)}$
4.  $B \in \Delta_{n+1}$  si  $B \leq_T \emptyset^{(n)}$

Demostración de 1:  $B \in \Sigma_{n+1}$  sii  $B$  es c.e. en algún conjunto  $\Pi_n$  sii  $B$  es c.e. en algún conjunto  $\Sigma_n$ .

Habíamos visto que  $B$  es  $C$ -c.e. sii  $B$  es  $\overline{C}$ -c.e. (TS7). El último 'sii' sale de ahí. Veamos el primer 'sii':

- ⇒ Sea  $B \in \Sigma_{n+1}$ . Entonces existe  $R \in \Pi_n$  tal que  $x \in B$  sii  $(\exists y) R(x, y)$ . Entonces  $B$  es  $\Sigma_1^R$  y por lo tanto  $B$  es  $R$ -c.e.
- ⇐ Supongamos que  $B$  es  $C$ -c.e. para  $C \in \Pi_n$ . Entonces  $\exists e$  tal que

$$\begin{array}{ll} x \in B & \text{sii} \\ & \quad x \in W_e^C \\ & \text{sii} \quad (\exists s)(\exists \sigma) [\sigma \subset C \wedge \underbrace{x \in W_{e,s}^\sigma}_{\text{computable}}] \end{array}$$

Basta ver que ' $\sigma \subset C$ ' es  $\Sigma_{n+1}$

$$\begin{array}{ll} \sigma \subset C & \text{sii} \quad (\forall i < |\sigma|) [\sigma(i) = C(i)] \\ & \text{sii} \quad (\forall i < |\sigma|) [\underbrace{[\sigma(i) = 1 \wedge i \in C]}_{\Pi_n} \vee \underbrace{[\sigma(i) = 0 \wedge i \notin C]}_{\Sigma_n}] \end{array}$$

Sabemos  $\Pi_n, \Sigma_n \subseteq \Sigma_{n+1}$  y que  $\Sigma_{n+1} \vee \Sigma_{n+1} \subseteq \Sigma_{n+1}$ . Como el  $\forall$  externo está acotado,  $C \in \Sigma_{n+1}$ .



Demostración de 2:  $\emptyset^{(n)}$  es  $\Sigma_n$ -completo, para  $n > 0$ .

Por inducción en  $n$ . Es claro que vale para  $n = 1$ . Supongamos  $\emptyset^{(n)}$  es  $\Sigma_n$ -completo. Entonces  $\emptyset^{(n)}$  es  $\Pi_n$ -completo.

$$\begin{aligned} B \in \Sigma_{n+1} &\quad \text{sii} \quad B \text{ es c.e. en algún } C \in \Sigma_n \quad \text{por TP1} \\ &\quad \text{sii} \quad B \text{ es c.e. en } \emptyset^{(n)} \quad \text{por HI, } C \leq_1 \emptyset^{(n)} \\ &\quad \text{sii} \quad B \leq_1 \emptyset^{(n+1)} \quad \text{por TS3} \end{aligned}$$

Esto prueba

- $\emptyset^{(n+1)}$  es  $\Sigma_{n+1}$  (tomar  $B = \emptyset^{(n+1)}$ )
- si  $B \in \Sigma_{n+1}$  entonces  $B \leq_1 \emptyset^{(n+1)}$



Demostración de 3:  $B \in \Sigma_{n+1}$  sii  $B$  es c.e. en  $\emptyset^{(n)}$ .

$$\begin{aligned} B \in \Sigma_{n+1} &\quad \text{sii} \quad B \text{ es c.e. en algún conjunto } \Sigma_n \quad \text{por TP1} \\ &\quad \text{sii} \quad B \text{ es c.e. en } \emptyset^{(n)} \quad \text{por TP2} \end{aligned}$$



Demostración de 4:  $B \in \Delta_{n+1}$  sii  $B \leq_T \emptyset^{(n)}$ .

$B \in \Delta_{n+1}$  sii  $B, \overline{B} \in \Sigma_{n+1}$

sii  $B, \overline{B}$  son c.e. en  $\emptyset^{(n)}$  por TP3

sii  $B \leq_T \emptyset^{(n)}$



# Jerarquía estricta

Sabemos

$$\begin{aligned}\Pi_n, \Sigma_n &\supseteq \bigcup_{m < n} \Sigma_m \cup \Pi_m \\ \Pi_n, \Sigma_n &\supseteq \Delta_n\end{aligned}$$

## Corolario

Si  $n > 0$  entonces  $\Delta_n \subsetneq \Pi_n, \Sigma_n$ .

## Demostración.

$\emptyset^{(n)} \in \Sigma_n$ . Supongamos que  $\emptyset^{(n)} \in \Delta_n$ . Por TP4  $\emptyset^{(n)} \leq_T \emptyset^{(n-1)}$  y esto contradice TS2. Entonces  $\emptyset^{(n)} \in \Sigma_n \setminus \Delta_n$ .

Análogamente  $\overline{\emptyset^{(n)}} \in \Pi_n \setminus \Delta_n$ .



# Ejemplo

## Proposición

$\text{Fin}$  es  $\Sigma_2$ -completo.

## Demostración.

Ya vimos que  $\text{Fin} \in \Sigma_2$ . Sea  $A \in \Sigma_2$  y veamos  $A \leq_1 \text{Fin}$ . Por definición

$$x \in \overline{A} \quad \text{sii} \quad (\forall y)(\exists z) \ R(x, y, z),$$

con  $R$  computable. Existe  $f$  computable 1-1 tal que

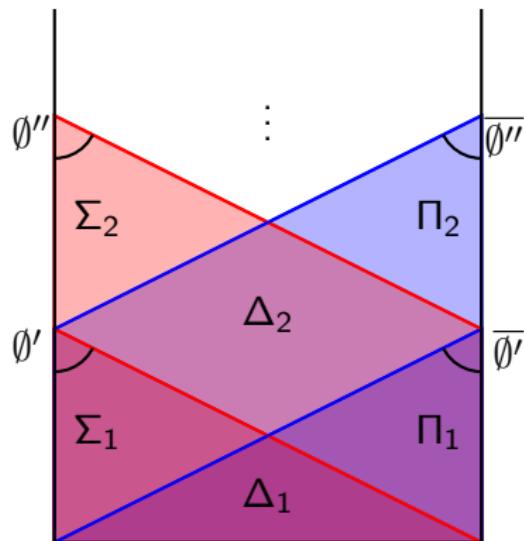
$$\varphi_{f(x)}(u) = \begin{cases} 0 & \text{si } (\forall y \leq u) (\exists z) \ R(x, y, z) \\ \uparrow & \text{si no} \end{cases}$$

Tenemos

- si  $x \in A$  entonces existe  $y'$  tal que no es cierto  $(\exists z) \ R(x, y', z)$ . Luego, no es cierto  $(\forall y \leq u) (\exists z) \ R(x, y, z)$  para  $u \geq y'$ . Entonces  $W_{f(x)}$  es finito, o sea  $f(x) \in \text{Fin}$ .
- $x \in \overline{A} \Rightarrow W_{f(x)} = \mathbb{N} \Rightarrow f(x) \notin \text{Fin}$

□

# La jerarquía aritmética



# Conjuntos $\Sigma_n^A$ y $\Pi_n^A$

## Definición

- Un conjunto  $B$  es  $\Sigma_0^A$  ( $\Pi_0^A$ ) si  $B$  es  $A$ -computable
- Para  $n \geq 1$ , un conjunto  $B$  es  $\Sigma_n^A$  (notado  $B \in \Sigma_n^A$ ) si existe una relación  $A$ -computable  $R(x, y_1, \dots, y_n)$  tal que

$$x \in B \quad \text{sii} \quad \underbrace{(\exists y_1)(\forall y_2)(\exists y_3) \dots (Qy_n)}_{n-1 \text{ alternancias}} R(x, y_1, \dots, y_n)$$

donde  $Q = \exists$  si  $n$  es par y  $Q = \forall$  si  $n$  es impar.

- Para  $n \geq 1$ , un conjunto  $B$  es  $\Pi_n^A$  (notado  $B \in \Pi_n^A$ ) si existe una relación  $A$ -computable  $R(x, y_1, \dots, y_n)$  tal que

$$x \in B \quad \text{sii} \quad \underbrace{(\forall y_1)(\exists y_2)(\forall y_3) \dots (Qy_n)}_{n-1 \text{ alternancias}} R(x, y_1, \dots, y_n)$$

donde  $Q = \forall$  si  $n$  es par y  $Q = \exists$  si  $n$  es impar.

- $B$  es  $\Delta_n^A$  (notado  $B \in \Delta_n^A$ ) si  $B \in \Sigma_n^A \cap \Pi_n^A$ .

# Jerarquía Aritmética relativizada

## Definición

Un conjunto  $A$  es  $\Sigma_n^C$ -completo ( $\Pi_n^C$ -completo) si  $A \in \Sigma_n^C$  ( $\Pi_n^C$ ) y  $B \leq_1 A$  para todo  $B \in \Sigma_n^C$  ( $\Pi_n^C$ ).

## Teorema (Post relativizado, TPR)

Para todo  $n \geq 0$ :

1.  $B \in \Sigma_{n+1}^A$  si  $B$  es c.e. en algún conjunto  $\Pi_n^A$  si  $B$  es c.e. en algún conjunto  $\Sigma_n^A$
2.  $A^{(n)}$  es  $\Sigma_n^A$ -completo, para  $n > 0$
3.  $B \in \Sigma_{n+1}^A$  si  $B$  es c.e. en  $A^{(n)}$
4.  $B \in \Delta_{n+1}^A$  si  $B \leq_T A^{(n)}$

# Observaciones sobre $\Sigma_n^A$ y $\Pi_n^A$

## Proposición

1. si  $A \leq_T B$  entonces  $\Sigma_n^A \subseteq \Sigma_n^B$
2. si  $A \leq_T B$  entonces  $\Pi_n^A \subseteq \Pi_n^B$
3.  $\Sigma_{n+1} = \Sigma_n^{\emptyset'}$
4.  $\Pi_{n+1} = \Pi_n^{\emptyset'}$

## Demostración.

1 y 2 son triviales.

Para 3, de TPR3  $A \in \Sigma_{n+1}$  si y sólo si  $A$  es c.e. en  $\emptyset^{(n)}$ . Por TPR3,  $A \in \Sigma_n^{\emptyset'}$  si y sólo si  $A$  es c.e. en  $(\emptyset')^{(n-1)} = \emptyset^{(n)}$ .

Análogo para 4. □

# Teoría de la Computabilidad

## Clase 7

Conjuntos simples

# Problema de Post

## Problema de Post

¿Existe un conjunto  $A$  c.e. tal que  $\emptyset <_T A <_T \emptyset'$ ?

### Idea

Creemos que sí, entonces construyamos un conjunto  $A$  c.e. tal que

- $A$  sea incompleto ( $K \not\leq_T A$ ).
  - para esto, definir una propiedad sencilla sobre  $\bar{A}$  que diga que  $\bar{A}$  es muy “fino” (en el sentido de que no contiene a ningún c.e. infinito)
- $A$  no sea computable ( $A \not\leq_T \emptyset$ )
  - la propiedad que definamos tiene que ser compatible con ser no computable.

# Inmunidad y simpleza

## Definición

- Un conjunto es **inmune** si es infinito pero no contiene a ningún conjunto infinito c.e.
- Un conjunto  $A$  es **simple** si  $A$  es c.e. y  $\overline{A}$  es inmune

## Proposición

*Si  $A$  es simple entonces:*

1.  $A$  no es computable
2.  $A$  no es creativo
3.  $A$  no es  $m$ -completo

## Demostración.

Ejercicio.



## Existe un conjunto simple

Vamos a ver dos pruebas:

- Una larga pero didáctica
  - nos va a servir para lo que viene
- Una corta y elegante

Primero la larga.

# Existencia de conjuntos simples (I)

## Requerimientos

Tratamos de construir un conjunto  $A$  c.e. y satisfacer los siguientes requerimientos, para todo  $e \in \mathbb{N}$ :

### Requerimientos

- $P_e : W_e$  es infinito  $\Rightarrow W_e \cap A \neq \emptyset$
- $N_e : \|\overline{A}\| \geq e$

Intuitivamente,

- $P_e$  es un requerimiento “positivo”: trata de poner elementos en  $A$ , así  $A$  crece
- $N_e$  es un requerimiento “negativo”: trata de evitar poner elementos en  $A$ , así  $\overline{A}$  crece

Si el conjunto c.e.  $A$  que construimos satisface  $P_e$  y  $N_e$  para todo  $e \in \mathbb{N}$  entonces  $A$  es simple:

- supongamos que  $W_e$  es infinito y  $W_e \subseteq \overline{A}$ . Entonces  $W_e \cap A = \emptyset$ , o sea, no se cumple  $P_e$ .
- supongamos que  $\|\overline{A}\| = e$ , entonces no se cumple  $N_{e+1}$ .

# Existencia de conjuntos simples (I)

## Construcción

- $P_e$  :  $W_e$  es infinito  $\Rightarrow W_e \cap A \neq \emptyset$
- $N_e$  :  $\|\overline{A}\| \geq e$

Definimos una enumeración de  $A$ . Llamamos  $A_s$  a la aproximación de  $A$  en el paso  $s$  (como siempre,  $A = \bigcup_s A_s$ ). Cada  $A_s$  es finito, y definimos

$$\overline{A_s} = \{a_{0,s} < a_{1,s} < \dots\}.$$

## Construcción

**Paso 0** Definir  $A_0 := \emptyset$ .

**Paso  $s+1$**  Ya está definido  $A_s$ . Buscar el primer  $e \leq s$  tal que

1.  $W_{e,s} \cap A_s = \emptyset$  y
2. para algún  $m \geq e$ ,  $a_{m,s} \in W_{e,s}$

Si no hay ningún tal  $e$ , pasar al paso  $s+2$ . Si no,

1. definimos  $A_{s+1} := A_s \cup \{a_{n,s}\}$   
donde  $n$  es el mínimo  $m$  que satisface 2
2. decimos que "atendemos a  $P_e$  en el paso  $s+1$ "

## Existencia de conjuntos simples (I)

## Verificación

Hecho

Una vez que  $P_e$  es atendido en el paso  $s$  no vuelve a ser atendido.

Hecho

Los  $a_{n,s}$  van creciendo a medida que aumenta  $s$  ( $a_{n,s} \leq a_{n,s+1}$ ).

Cuando atendemos  $P_e$  en el paso  $s + 1$  y metemos  $a_{n,s}$  en  $A$ , crecen los  $a_{m,s}$  para  $m \geq n$ , pero quedan iguales  $a_{0,s}, \dots, a_{n-1,s}$ .

# Existencia de conjuntos simples (I)

## Verificación

### Hecho

Para todo  $n$ ,  $\lim_t a_{n,t} < \infty$ .

### Demostración.

Supongamos lo contrario. Existe  $n$  y sucesión

$$s_1 < s_2 < \dots$$

tal que para todo  $i$ ,  $a_{n,s_i} < a_{n,s_i+1}$ .

Entonces para todo  $i \in \mathbb{N}$ , en el paso  $s_i + 1$  atendemos a cierto  $P_{e_i}$  y metemos  $a_{n,s_i}$  en  $A$ . Cada  $P_{e_i}$  es atendido una sola vez. Entonces si  $i \neq j$ ,  $e_i \neq e_j$ . Luego  $\{e_i : i \in \mathbb{N}\}$  es infinito. Pero por la condición 2 de la construcción,  $e_i \leq n$  para todo  $i$ . Absurdo. □

# Existencia de conjuntos simples (I)

## Verificación

### Hecho

Para todo  $e$ , se satisface  $N_e$ . Es decir,  $\overline{A}$  es infinito.

### Demostración.

Fijemos  $e$ . Sea  $s$  suficientemente grande tal que

$$a_{0,s}, \dots, a_{e-1,s}$$

son estables (i.e. para todo  $t \geq s$  e  $i = 0, \dots, e-1$ , tenemos  $a_{i,s} = a_{i,t}$ ). Entonces

$$\{a_{0,s}, \dots, a_{e-1,s}\} \subseteq \overline{A}.$$

De esta manera,  $\|\overline{A}\| \geq e$ .



# Existencia de conjuntos simples (I)

## Verificación

### Hecho

Para todo  $e$ , se satisface  $P_e$ .

### Demostración.

Supongamos que  $\|W_e\| = \infty$  y  $W_e \cap A = \emptyset$ . Sea

$$\bar{A} = \{a_0 < a_1 < a_2 < \dots\}$$

$$y = \min\{x : x \in W_e \wedge x > a_e\}$$

Sea  $s$  suficientemente grande tal que

- $y \in W_{e,s}$
- $a_{n,s} = a_n$  para  $n \in \{0, \dots, e\}$
- si  $P_i$ ,  $i < e$ , es atendido, ya fue atendido para el paso  $s + 1$

En el paso  $s + 1$ , el item 1 de la construcción vale trivialmente.

- si  $y \in A_s$  entonces  $y \in A \cap W_e$ . Absurdo.
- si  $y \in \bar{A}_s$ , tenemos  $y = a_{m,s}$  para algún  $m > e$ . Entonces vale el item 2 y  $a_{n,s} \in A_{s+1} \cap W_{e,s} \subseteq A \cap W_e$ . Absurdo.



## Existencia de conjuntos simples (II)

Definir  $f$  parcial computable como

$f(e) = \text{primer elemento } > 2e \text{ que aparece en } W_e$

Sea  $S = \text{rg } f$ .

- $S$  es c.e. porque es el rango de una función parcial computable
- $\overline{S}$  es infinito: como  $f(e) > 2e$ ,  $S$  contiene a lo sumo  $e$  elementos de entre  $\{0, 1, \dots, 2e\}$ , a saber:  
 $f(0), f(1), \dots, f(e-1)$ .

Entonces

$$\begin{aligned}\|\overline{S} \upharpoonright 2e + 1\| &\geq 2e + 1 - \|\{f(0), f(1), \dots, f(e-1)\}\| \\ &\geq 2e + 1 - e = e + 1,\end{aligned}$$

de modo que  $\overline{S}$  es infinito.

- si  $W_e$  es infinito,  $f(e) \in S \cap W_e$ , de modo que  $S \cap W_e \neq \emptyset$ . Entonces  $S$  es simple.

## Los conjuntos simples no nos alcanzan

Sabemos que si  $A$  es simple entonces

- $A$  es c.e. (y por lo tanto  $A \leq_m \emptyset'$ )
- no computable ( $A \not\leq_T \emptyset$ )
- $m$ -incompleto ( $\emptyset' \not\leq_m A$ )

Sin embargo  $A$  puede ser T-completo ( $A \geq_T \emptyset'$ ) (lo veremos más adelante).

Nueva idea

Definir  $\overline{A}$  más y más “fino”.

# Teoría de la Computabilidad

## Clase 8

Conjuntos hipersimples

# Codificación de conjuntos finitos y arreglos

## Definición

Dado un conjunto finito  $A = \{a_1 < a_2 < \dots < a_k\}$ ,

- el número  $y = \sum_{1 \leq i \leq k} 2^{a_i}$  es el **índice canónico** de  $A$
- notamos  $D_y$  al conjunto finito cuyo índice canónico es  $y$ , y  $D_0$  denota  $\emptyset$ .

Si  $y$  se escribe en binario,  $D_y$  son las posiciones donde el dígito 1 ocurre.

## Ejemplo

Si  $y = 5 = 2^0 + 2^2$ , que en binario es 101, entonces  $D_y = D_5 = \{0, 2\}$ .

## Definición

- Una secuencia  $\{F_n\}_{n \in \mathbb{N}}$  de conjuntos finitos es un **arreglo** si hay una función computable  $f$  tal que  $F_n = D_{f(n)}$ .
- Un arreglo es **disjunto** si todos sus miembros son disjuntos dos a dos.

# Hipersimpleza

## Definición

- Un conjunto infinito  $B$  es **hiperinmune** (h-immune) si no existe un arreglo disjunto  $\{F_n\}_{n \in \mathbb{N}}$  tal que  $(\forall n) F_n \cap B \neq \emptyset$ .
- Un conjunto  $A$  c.e. es **hipersimple** (h-simple) si  $\overline{A}$  es h-inmune.

## Nueva idea

“más fino” = hipersimple

- Si  $A$  es **simple** entonces no hay un conjunto infinito c.e.

$$\{a_n\}_{n \in \mathbb{N}} \subseteq \overline{A}.$$

- Si  $A$  es **hipersimple** entonces no hay un arreglo disjunto

$$\{F_n\}_{n \in \mathbb{N}} \text{ tal que } (\forall n) F_n \cap \overline{A} \neq \emptyset$$

- $\overline{A}$  contiene a algún elemento de  $F_n$ , pero no sabemos a cuál.

# Hipersimpleza

## Teorema

*Si  $A$  es hipersimple entonces  $A$  es simple.*

## Demostración.

## Ejercicio.



# Mayorar funciones y función principal

## Definición

- Una función  $f$  **mayora** a una función  $g$  si  $(\forall x) f(x) \geq g(x)$ .
- Si  $A = \{a_0 < a_1 < \dots\}$  es un conjunto infinito, la **función principal** de  $A$  es  $p_A(n) = a_n$ .
- Una función  $f$  **mayora** un conjunto infinito  $A$  si  $f$  mayora a  $p_A$ .

## *h-inmune sii no computablemente mayorado*

### **Teorema**

*Un conjunto infinito  $A$  es  $h$ -inmune sii ninguna función computable mayorá a  $A$ .*

### **Corolario**

*Un conjunto coinfinito c.e.  $A$  es  $h$ -simple sii ninguna función computable mayorá a  $\overline{A}$ .*

## Demostración del Teorema ( $\Leftarrow$ ).

Por el absurdo. Supongamos que  $A$  no es  $h$ -inmune. Sea  $\{D_{g(x)}\}_{x \in \mathbb{N}}$  un arreglo disjunto tal que

$$(\forall x) D_{g(x)} \cap A \neq \emptyset.$$

Sea

$$f(x) = \max \bigcup_{y \leq x} D_{g(y)}.$$

- $f$  es computable
- como  $\{D_{g(x)}\}_{x \in \mathbb{N}}$  es disjunto,  $f(x) \geq p_A(x)$



## Demostración del Teorema ( $\Rightarrow$ ).

Sea  $f$  computable tal que  $(\forall x) f(x) \geq p_A(x)$ . Definir  $g$  de esta manera:

- $g(0)$  es tal que  $D_{g(0)} = \{0, \dots, f(0)\}$
- supongamos que tenemos definido  $g(0), \dots, g(n)$ . Sea

$$k_n = 1 + \max \bigcup_{i \leq n} D_{g(i)}$$

y definir  $g(n+1)$  tal que

$$D_{g(n+1)} = \{k_n, \dots, f(k_n)\}$$

Claramente  $\{D_{g(n)}\}_{n \in \mathbb{N}}$  es un arreglo disjunto.

Veamos que  $(\forall n) D_{g(n)} \cap A \neq \emptyset$ :

- es claro para  $n = 0$  pues  $p_A(0) \leq f(0)$  y  $D_{g(0)} = \{0, \dots, f(0)\}$ .
- basta ver que  $p_A(k_n) \in D_{g(n+1)} = \{k_n, \dots, f(k_n)\}$ 
  - como  $p_A$  es función principal,  $p_A(k_n) \geq k_n$
  - por hipótesis  $p_A(k_n) \leq f(k_n)$
  - luego  $k_n \leq p_A(k_n) \leq f(k_n)$

# Existe un conjunto h-simple

## Teorema

Para cada conjunto c.e. no computable  $A$  hay un conjunto h-simple  $B \equiv_T A$ .

## Demostración

Sea  $f$  computable 1-1 tal que  $\text{rg } f = A$ . Sea  $a_s = f(s)$ . Entonces

$$A = \{a_0, a_1, a_2, \dots\}.$$

Definimos el **conjunto de deficiencia de  $A$  para la enumeración  $f$** :

$$B = \{s : (\exists t > s) a_t < a_s\}$$

- $B \in \Sigma_1$ , de modo que  $B$  es c.e.
- $\overline{B}$  es infinito (ejercicio)
- veamos que  $a_{p_{\overline{B}}(x)} < a_{p_{\overline{B}}(x+1)}$  (y luego  $a_{p_{\overline{B}}(x)} \geq x$ ):
  - $p_{\overline{B}}(x) \in \overline{B}$ , de modo que  $(\forall t > p_{\overline{B}}(x)) a_t \geq a_{p_{\overline{B}}(x)}$
  - como  $f$  es 1-1,  $(\forall t > p_{\overline{B}}(x)) a_t > a_{p_{\overline{B}}(x)}$
  - instanciar  $t = p_{\overline{B}}(x+1) > p_{\overline{B}}(x)$

(sigue →)

## Demostración (continuación)

- $A \leq_T B$ :
  - basta ver que  $x \in A$  si y sólo si  $x \in \{a_0, a_1, \dots, a_{p_{\bar{B}}(x)}\}$
  - sabemos que  $(\forall t > p_{\bar{B}}(x)) a_t > a_{p_{\bar{B}}(x)} \geq x$
  - si  $x \in A$  entonces  $x = a_t$  para algún  $t$ . Además,  $t \leq p_{\bar{B}}(x)$
  - dado  $B$  podemos computar  $\bar{B}$  y de ahí  $p_{\bar{B}}$
- $B \leq_T A$ :
  - definir  $A_s = \{a_0, a_1, \dots, a_s\}$  (observar  $\#A_s = s + 1$ )
  - dado  $s$  enumerar  $A_0, A_1, A_2, \dots$  hasta encontrar el primer  $t$  tal que  $A_t \upharpoonright 1 + a_s = A \upharpoonright 1 + a_s$ . Esto se puede hacer de manera  $A$ -computable.
  - es claro que  $t \geq s$ , pues al menos tengo que 'encontrar'  $a_s$
  - por construcción,  $a_t \leq a_s$  y  $(\forall t' > t) a_{t'} > a_s$
  - si  $t = s$  entonces  $s \notin B$
  - si  $t > s$  entonces  $a_t < a_s$ . Luego  $s \in B$ .
- $\bar{B}$  es h-inmune:
  - supongamos  $p_{\bar{B}}$  es mayorada por  $g$  computable
  - entonces  $x \in A$  si y sólo si  $x \in \{a_0, a_1, \dots, a_{g(x)}\}$
  - entonces  $A$  sería computable. Absurdo.



## Simpleza e hipersimpleza no alcanzan

Queríamos resolver el problema de Post:

*¿Existe un conjunto A c.e. tal que  $\emptyset <_T A <_T \emptyset'$ ?*

Buscábamos una respuesta afirmativa tratando de encontrar una propiedad sobre A que garantizara

1. no computabilidad ( $A \not\leq_T \emptyset$ )
2.  $T$ -incompletitud ( $\emptyset' \not\leq_T A$ )

Propuestas:

- A es simple
- A es hipersimple
- (hay más: A es hiperhipersimple)

Garantizan 1 pero no pueden garantizar 2: por el teorema anterior, hay un conjunto hipersimple  $\equiv_T \emptyset'$

# Teoría de la Computabilidad

## Clase 9

Conjuntos bajos, solución al problema de Post

# Conjuntos altos y bajos

Sabemos que si  $\emptyset \leq_T A \leq_T \emptyset'$  entonces  $\emptyset' \leq_T A' \leq_T \emptyset''$ .

## Definición

- $A$  es **bajo (low)** si  $A' \equiv_T \emptyset'$ .
- $A$  es **alto (high)** si  $A' \equiv_T \emptyset''$ .

Intuitivamente,

- los conjuntos bajos están cerca de ser computables
- los conjuntos altos están cerca de ser completos

# Equivalencias de conjuntos bajos

## Teorema

Si  $A \leq_T \emptyset'$ , los siguientes son equivalentes:

1.  $A$  es bajo
2.  $\Sigma_1^A \subseteq \Pi_2$
3.  $A' \leq_1 \overline{\emptyset''}$

## Demostración.

- $A$  es bajo    sii     $A' \leq_T \emptyset'$   
                          sii     $A' \in \Delta_2$     por TPR4  
                          sii     $\Sigma_1^A \subseteq \Delta_2$     porque  $A'$  es  $\Sigma_1^A$ -completo  
                          sii     $\Sigma_1^A \subseteq \Pi_2$     porque  $\Sigma_1^A \subseteq \Sigma_1^{\emptyset'} = \Sigma_2$   
                          sii     $A' \leq_1 \overline{\emptyset''}$

último sii:

$\Rightarrow$  Como  $A' \in \Sigma_1^A$ , entonces  $A' \in \Pi_2$ . Como  $\overline{\emptyset''}$  es  $\Pi_2$ -completo,  
 $A' \leq_1 \overline{\emptyset''}$ .

$\Leftarrow$   $B \in \Sigma_1^A \xrightarrow{\text{TPR}} B$  es  $A$ -c.e.  $\xrightarrow{\text{TS5}} B \leq_1 A' \leq_1 \overline{\emptyset''} \Rightarrow B \in \Pi_2$

□

# Equivalencias de conjuntos altos

## Teorema

*Si  $A \leq_T \emptyset'$ , los siguientes son equivalentes:*

1.  $A$  es alto
2.  $\Sigma_2 \subseteq \Pi_2^A$
3.  $\emptyset'' \leq_1 \overline{A''}$

## Demostración.

- $A$  es alto    sii     $\emptyset'' \leq_T A'$   
                          sii     $\emptyset'' \in \Delta_2^A$     por TPR4  
                          sii     $\Sigma_2 \subseteq \Delta_2^A$     porque  $\emptyset''$  es  $\Sigma_2$ -completo  
                          sii     $\Sigma_2 \subseteq \Pi_2^A$     porque  $\Sigma_2 \subseteq \Sigma_2^A$   
                          sii     $\emptyset'' \leq_1 \overline{A''}$

último sii:

$\Rightarrow$  Como  $\emptyset'' \in \Sigma_2$ , entonces  $\emptyset'' \in \Pi_2^A$ . Como  $\overline{A''}$  es  $\Pi_2^A$ -completo,  
 $\emptyset'' \leq_1 \overline{A''}$ .

$\Leftarrow B \in \Sigma_2 \Rightarrow B$  es  $\emptyset'$ -c.e.  $\xrightarrow{\text{TS5}} B \leq_1 \emptyset'' \leq_1 \overline{A''} \Rightarrow B \in \Pi_2^A$

□

# Solución al problema de Post

Friedberg y Muchnik dieron una solución al problema de Post:

## Idea

Construir un conjunto  $A$  que sea simple y bajo.

- si  $A$  es **simple** entonces no es computable ( $\emptyset <_{\mathcal{T}} A$ )
- si  $A$  es **bajo** entonces es incompleto ( $A <_{\mathcal{T}} \emptyset'$ )
  - en efecto, supongamos que  $A \equiv_{\mathcal{T}} \emptyset'$ . Entonces  $A' \equiv_{\mathcal{T}} \emptyset''$ . Como  $A$  es bajo,  $A' \equiv_{\mathcal{T}} \emptyset'$ . Absurdo.

El problema está en construir un conjunto que cumpla con las dos condiciones.

## Lesiones finitas (*finite injury*)

Como en la demostración (I) de existencia de conjuntos simples, vamos a tener dos tipos de requerimientos.

Informalmente:

- $P_e$  trata de poner elementos en  $A$  para que  $A$  sea simple
- $N_e$  trata de no poner elementos en  $A$  para que  $A$  sea bajo

Los  $P_e$  y  $N_e$  tienen objetivos en conflicto.

Cada  $P_e$  y cada  $N_e$  tiene asignada una **prioridad**.

Cuando un  $P_i$  actúa, puede destruir (lesionar) un  $N_j$  solo si la prioridad de  $P_i$  es mayor que la prioridad de  $N_j$ .

La construcción va a garantizar que:

- solo los  $N_j$  pueden ser lesionados por  $P_i$  de mayor prioridad
- para cada  $N_j$  hay una cantidad finita de  $i$  tal que  $P_i$  tiene más prioridad que  $N_j$ . Entonces cada  $N_j$  puede ser lesionado solo una cantidad finita de veces.

# Existe un conjunto simple y bajo

Basta construir un conjunto  $A$  c.e. y coinfinito que cumpla estos

## Requerimientos

- $P_e : W_e$  es infinito  $\Rightarrow W_e \cap A \neq \emptyset$
- $N_e : (\exists^\infty s) \Phi_{e,s}^{A_s}(e) \downarrow \Rightarrow \Phi_e^A(e) \downarrow$

- **Los  $P_e$  quieren agregar elementos a  $A$**  para que  $A$  se interseque con todos los  $W_e$  infinitos.
- **Los  $N_e$  no quieren que  $A$  cambie.** Si  $N_e$  vale y cambiamos  $A$ , corremos el riesgo de que  $N_e$  deje de valer. En efecto si  $A_{t+1} = A_t \cup \{x\}$  podría pasar que  $\Phi_e^{A_t}(e) \downarrow$  pero  $\Phi_e^{A_{t+1}}(e) \uparrow$ . Entonces hay que tener cuidado con qué  $x$  podemos agregar a  $A$ .

## Los $P_e$ garantizan simpleza; los $N_e$ garantizan bajura

Los requerimientos  $P_e$  son los mismos que vimos para probar la existencia de conjuntos simples (demostración (I)).

### Hecho

*Si  $\overline{A}$  es infinito y para todo  $e$  vale  $P_e$  entonces  $A$  es simple.*

### Hecho

*Si para todo  $e$  vale  $N_e$  entonces  $A$  es bajo ( $A' \leq_T \emptyset'$ ).*

### Demostración.

Definir la función computable  $g$  de esta manera:

$$g(e, s) = \begin{cases} 1 & \text{si } \Phi_{e,s}^{A_s}(e) \downarrow \\ 0 & \text{si no} \end{cases}$$

Si para todo  $e$  vale  $N_e$  entonces  $\hat{g}(e) = \lim_s g(e, s)$  existe para todo  $e$ . Por el Lema del Límite,  $\hat{g} \leq_T \emptyset'$ . Pero  $\hat{g}$  es la función característica de  $A'$ , de modo que  $A' \leq_T \emptyset'$ . □

## Cómo ayudamos a los $N_e$

Recordar:

- $N_e : (\exists^\infty s) \Phi_{e,s}^{A_s}(e) \downarrow \Rightarrow \Phi_e^A(e) \downarrow$

Vimos que si  $N_e$  vale y agregamos un elemento  $x$  a  $A$ , el cálculo  $\Phi_e^A(e)$  se puede **destruir** ( $N_e$  se puede lesionar). Es decir, si  $A_{t+1} = A_t \cup \{x\}$  podría pasar que  $\Phi_e^{A_t}(e) \downarrow$  pero  $\Phi_e^{A_{t+1}}(e) \uparrow$ .

Recordar que la función de uso es computable y está definida como

$$u(A_s; e, x, s) = 1 + \text{máximo elemento usado en el cálculo de } \Phi_{e,s}^{A_s}(x)$$

y definimos  $r(e, s) = u(A_s; e, e, s)$ .

## Cómo ayudamos a los $N_e$

Ayudamos a  $N_e$  si evitamos agregar en  $A$  elementos  $x \leq r(e, s)$ .

1. si ponemos en  $A$  un elemento  $x \leq r(e, s)$  estamos amenazando  $N_e$  y posiblemente lo lesionamos
2. si ponemos en  $A$  un elemento  $x > r(e, s)$ , a  $N_e$  no le importa y seguro que no lo lesionamos

Si siempre pudiéramos garantizar 2, si  $\Phi_e^{A_t}(e) \downarrow$  y  $r = r(e, t)$  entonces  $A_t \upharpoonright r = A \upharpoonright r$  y luego  $\Phi_e^A(e) \downarrow$ .

No podemos garantizar 2 para todo  $e$ , pero fijamos esta regla de prioridad:

*En el paso  $t$ ,  $P_i$  puede meter elementos  $\leq r(e, t)$  en  $A$  (lesionando a  $N_e$ ) solo si  $i < e$ .*

Para  $i < e$ ,  $P_i$  tiene **mayor prioridad** que  $N_e$ . Así,  $N_e$  puede ser lesionado por  $P_0, \dots, P_{e-1}$  pero nunca por  $P_e, P_{e+1}, \dots$

# Construcción de $A$

## Construcción

Paso 0  $A_0 = \emptyset$ .

Paso  $s+1$  Dado  $A_s$  podemos computar  $r(e, s)$  para el  $e$  que queramos.

Elegir el menor  $i \leq s$  tal que cumpla estas dos condiciones

1.  $W_{i,s} \cap A_s = \emptyset$  y
2.  $(\exists x) [x \in W_{i,s} \wedge x > 2i \wedge (\forall e \leq i) r(e, s) < x]$

Si tal  $i$  existe, definir  $A_{s+1} := A_s \cup \{x\}$ . En este caso decimos que **atendemos a  $P_i$** . Si tal  $i$  no existe, definir  $A_{s+1} := A_s$ .

Finalmente  $A = \bigcup_s A_s$ .

Decimos que  $x$  **lesiona** a  $N_e$  en el paso  $s + 1$  si  $x \in A_{s+1} \setminus A_s$  y  $x \leq r(e, s)$ .

# Verificación

## Lema

$P_e$  es atendido a lo sumo una sola vez.

## Demostración.

Una vez que  $P_e$  es atendido en el paso  $s + 1$ , la condición 1 de la construcción es falsa para pasos  $> s + 1$ . □

## Lema

$N_e$  es lesionado a lo sumo una cantidad finita de veces.

## Demostración.

Por la condición 2 de la construcción,  $N_e$  puede ser lesionado solo por un  $x$  agregado a  $A$  cuando un  $P_i$  es atendido, con  $i < e$ . Como cada  $P_i$  es atendido a lo sumo una vez,  $N_e$  es lesionado a lo sumo  $e$  veces. □

# Verificación

## Lema

Para cada  $e$ ,  $N_e$  es verdadero y  $r(e) = \lim_s r(e, s)$  existe.

## Demostración.

Fijemos  $e$ . Sea  $s_e$  suficientemente grande tal que  $N_e$  no es lesionado en etapas  $> s_e$ .

Recordar:

- $N_e : (\exists^\infty s) \Phi_{e,s}^{A_s}(e) \downarrow \Rightarrow \Phi_e^A(e) \downarrow$

Supongamos cierto el antecedente de  $N_e$ . Tenemos  $\Phi_{e,s}^{A_s}(e) \downarrow$  para algún  $s > s_e$ . Entonces (por inducción en  $t$ ) para todo  $t \geq s$  tenemos

$$r(e, t) = r(e, s) \quad \text{y} \quad \Phi_e^{A_t}(e) = \Phi_e^{A_s}(e).$$

De modo que para  $r = r(e, s)$  tenemos  $A_s \upharpoonright r = A \upharpoonright r$  y  $\Phi_e^A(e) = \Phi_{e,s}^{A_s}(e) \downarrow$ .



# Verificación

## Lema

Para cada  $i$ ,  $P_i$  es verdadero.

## Demostración.

Fijemos  $i$ . Recordar:

- $P_i : W_i$  es infinito  $\Rightarrow W_i \cap A \neq \emptyset$

Supongamos que  $W_i$  es infinito. Sea  $s$  tal que

$$(\forall t \geq s)(\forall e \leq i) r(e, t) = r(e).$$

Sea  $s' \geq s$  tal que ningún  $P_j$ ,  $j < i$  recibe atención después de la etapa  $s'$ . Sea  $t > s'$  tal que

$$(\exists x) [x \in W_{i,t} \wedge x > 2i \wedge (\forall e \leq i) r(e) < x]$$

O bien  $W_{i,t} \cap A_t \neq \emptyset$  o bien  $P_i$  es atendido en la etapa  $t + 1$ . En cualquier caso,  $P_i$  es verdadero al finalizar la etapa  $t + 1$ . □

Observar que  $\bar{A}$  es infinito por la cláusula  $x > 2i$  de la condición 2.