Resumen Computabilidad

Maquinas de Turing

Componentes

- **Cinta**
  - Celdas, infinita en ambas direcciones, cada celda contiene un símbolo de un alfabeto dado Σ.
  - $* \in \Sigma$
  - $L, R \notin \Sigma$
  - $*$ representa el blanco en una celda
- **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
  - **Σ** es el alfabeto, **Q** es el conjunto finito de estados, **A = Σ ∪ {L, R}** es el conjunto de acciones
  - Una **tabla de instrucciones T** es un subconjunto (finito) de **Q × Σ × A × Q**
  - La tupla **(q, s, a, q') ∈ 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'***
  - $q_0 \in$ Q es el estado inicial, $q_f \in$ Q es el estado final
  - cuando no hay restricciones sobre T decimos que M es una **máquina de Turing no determinística**
  - cuando no hay dos instrucciones en T que empiezan con las mismas primeras dos coordenadas, decimos que M es una **máquina de Turing determinística**

Funciones parciales

Una función parcial f es una función que puede estar indefinida para algunos (tal vez ninguno; tal vez todos) sus argumentos.

- notamos $f(x_1, \ldots, x_n)\downarrow$ cuando f está definida para $x_1, \ldots, x_n$.
- notamos $f(x_1, \ldots, x_n)\uparrow$ cuando f está indefinida para $x_1, \ldots, x_n$

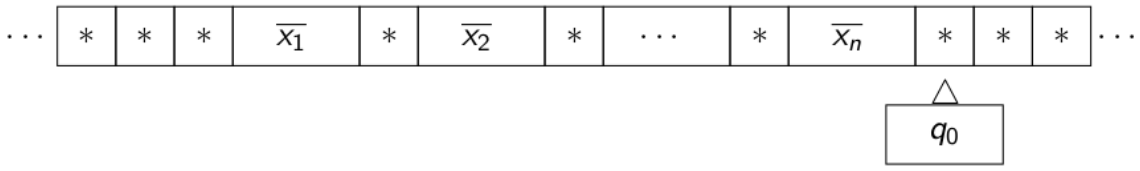
El conjunto de argumentos para los que f está definida se llama dominio de f , notado dom(f).

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

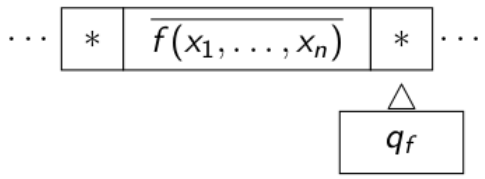
f es **total** si $dom(f) = \mathbb{N}^n$

Cómputo de funciones parciales en máquinas de Turing

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



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



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

Poder de computo

Teorema: Sea $f:\mathbb{N}^n \to \mathbb{N}$ una función parcial. Son equivalentes

f es computable en Java, C, Haskell, Turing

Funciones computables

Otra manera de formalizar la idea de función calculable de manera efectiva:

- empezar por funciones muy simples, efectivas intuitivamente
- si mezclamos de alguna manera efectiva dos o más funciones que ya eran efectivas, entonces obtenemos una función calculable de manera efectiva

Las siguientes funciones se llaman iniciales:

- $s(x)=x+1$
- $n(x)=0$
- proyecciones: $u_i^n(x_1,\ldots,x_n)=x_i$ para i ∈ {1, . . . , n}

Composición y recursión primitiva

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

- $h(x_1,\ldots,x_n,0)=f(x_1,\ldots,x_n)$
- $h(x_1,\ldots,x_n,t+1)=g(h(x_1,\ldots,x_n,t),x_1,\ldots,x_n,t)$

Clases PRC

Una clase **C** de funciones totales es **PRC** (primitive recursive closed) si

1. las funciones iniciales están en **C**

2. si una función f se obtiene a partir de otras pertenecientes a **C** por medio de composición o recursión primitiva, entonces f también está en **C**

Teorema: La clase de funciones totales Turing computables es una clase PRC.

Funciones primitivas recursivas

Una función es **primitiva recursiva (p.r.)** si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de composición y recursión primitiva.

Teorema: Una función es p.r. sii pertenece a toda clase PRC.

Corolario: Toda función p.r. es total y Turing computable.

Predicados primitivos recursivos

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

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

Operadores lógicos

Teorema: Sea C una clase PRC. Si p y q son predicados en C entonces ¬p, p ∧ q y p ∨ q están en **C.**

Corolario: Si p y q son predicados p.r., entonces también lo son los predicados ¬p, p ∨ q y p ∧ q

Corolario: Si p y q son predicados totales Turing computables entonces también lo son los predicados ¬p, p ∨ q y p ∧ q

Definición por casos (2)

Teorema: Sea **C** una clase PRC. Sean $h, g: \mathbb{N}^n \to \mathbb{N}$ funciones en **C** y sea $p: \mathbb{N}^n \to \{0,1\}$ un predicado en **C**.

La siguiente función esta en **C**

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

Definición por casos (m+1)

Teorema: Sea **C** una clase PRC. Sean $g_1, \ldots, g_m, h: \mathbb{N}^n \to \mathbb{N}$ funciones en **C** y sean $p_1, \ldots, p_m: \mathbb{N}^n \to \{0,1\}$ predicados mutuamente excluyentes en **C**.

La siguiente función esta en **C**

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

Recursión primitiva

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

- la recursión siempre se hace en el último parámetro
- la función variante de $h(x_1,\ldots,x_n,x_{n+1})$ es x_{n+1}

Sumatorias y productorias (desde 0 y 1)

Teorema: Sea C una clase PRC. Si $f:\mathbb{N}^{n+1}\to\mathbb{N}$ esta en C entonces tambien estan las funciones

$$g(y,x_1,\ldots,x_n)=\sum_{t=0}^{y} f(t,x_1,\ldots,x_n) \qquad g(y,x_1,\ldots,x_n)=\sum_{t=1}^{y} f(t,x_1,\ldots,x_n)$$

$$h(y,x_1,\ldots,x_n)=\prod_{t=0}^{y} f(t,x_1,\ldots,x_n) \qquad h(y,x_1,\ldots,x_n)=\prod_{t=1}^{y} f(t,x_1,\ldots,x_n)$$

Cuantificadores acotados

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

$$(\forall t)_{\le y}\ p(t,x_1,\ldots,x_n)$$

$$(\exists t)_{\le y}\ p(t,x_1,\ldots,x_n)$$

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

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

Demostración.

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

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

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

Demostración.

$(\forall t)_{<y}\ p(t,x_1,\ldots,x_n)$ sii $(\forall t)_{\le y}\ (t=y\vee p(t,x_1,\ldots,x_n))$

$(\exists t)_{<y}\ p(t,x_1,\ldots,x_n)$ sii $(\exists t)_{\le y}\ (t\neq y\wedge p(t,x_1,\ldots,x_n))$

Minimizacion acotada

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

$$g(y,x_1,\ldots,x_n)=\sum_{u=0}^{y}\prod_{t=0}^{u}\alpha(p(t,x_1,\ldots,x_n)) \qquad \min_{t\le y} p(t,x_1,\ldots,x_n)=\begin{cases} \text{mínimo } t\le y \text{ tal que} \\ p(t,x_1,\ldots,x_n) \text{ es verdadero} & \text{si existe tal } t \\ 0 & \text{si no} \end{cases}$$

Teorema: Sea $p:\mathbb{N}^{n+1}\to\{0,1\}$ un predicado de una clase PRC C. La funcion $\min_{t\le y} p(t,x_1,\ldots,x_n)$ tmb esta en C

Codificacion de pares

Definimos la función primitiva recursiva $\langle x,y\rangle = 2^x(2\cdot y+1)\dot{-}1$ (solo tiene una solucion (x,y) que es = z)

Observadores de pares

Los observadores del par $z=\langle x,y\rangle$ son $l(z)=x$, $r(z)=y$ y son p.r.

Codificacion de secuencias

El número de Gödel de la secuencia $a_1,\ldots,a_n$ es el numero $[a_1,\ldots,a_n]=\prod_{i=1}^{n} p_i^{a_i}$ donde p_i es el i-esimo primo.

Teorema: Si $[a_1,\ldots,a_n]=[b_1,\ldots,b_n]$ entonces $a_i=b_i \forall i$

Observadores: (Son p.r.)

- $x[i]=a_i$
- $|x|=long(x)$

Lenguaje de programacion S

Variables: empiezan inicializadas en 0 y almacenan numeros naturales.

- *Variables de entrada:* $X_1 X_2 \ldots$
- *Variable de salida:* Y
- *Variables temporales:* $Z_1 Z_2 \ldots$

Instrucciones:

- $V \leftarrow V+1$
- $V \leftarrow V-1$ (si V es 0, queda en 0)
- $IF\ V \neq 0\ GOTO\ A$

Por conveniencia agregamos una mas que no hace nada:

- $V \leftarrow V$

Macros: Se pueden simular muchas operaciones a partir de las instrucciones.

Una vez que sepamos que se pueden escribir en el lenguaje S, las usamos como si fueran propias (son **pseudoinstrucciones**).

Estados: Un estado de un programa P es una lista de ecuaciones de la forma V = m (donde V es una variable y m es un número) tal que

- hay una ecuación para cada variable que se usa en P
- no hay dos ecuaciones para la misma variable

Descripción instantánea

Con un programa **P** de longitud n. Para un estado **σ** de **P** y un i ∈ {1, . . . , n + 1},

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

Para un (i, σ) no terminal, podemos definir su **sucesor** (j, τ) como:

1. si la i-ésima instrucción de P es V ← V + 1.
   - j = i +1
   - τ es σ, salvo que V = m se reemplaza por V = m + 1
2. si la i-ésima instrucción de P es V ← V − 1.

- j = i +1
- τ es σ, salvo que V = m se reemplaza por V = máx{m − 1, 0}

3. si la i-ésima instrucción de P es $IF\, V \neq 0\, GOTO\, L$
   - τ es idéntico a σ
   - si σ tiene V = 0 entonces j = i + 1
   - si σ tiene V = m para m ≠ 0 entonces
     1. si existe en P una instrucción con etiqueta L entonces j = mín{k : k-ésima instrucción de P tiene etiqueta L}
     2. si no j = n + 1

Cómputos

Un cómputo de un programa P a partir de una descripción instantánea d_1 es una lista $d_1, d_2, \ldots, d_k$ de descripciones instantáneas de P tal que

- d_{i+1} es sucesor de d_i para i ∈ {1, 2, . . . , k − 1}
- d_k es terminal

Estados y descripciones iniciales

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

- el estado inicial de P para $r_1, \ldots, r_m$ es el estado σ_1 , que tiene
  $X_1 = r_1, \ldots, X_m = r_m, Y = 0$ junto con V = 0
  para cada variable V que aparezca en P y no sea $X_1, \ldots, X_m, Y$
- la descripción inicial de P para $r_1, \ldots, r_m$ es $(1, \sigma_1)$

Cómputos a partir del estado inicial

Sea P un programa y sean $r_1, \ldots, r_m$ números dados, σ_1 el estado inicial

Dos casos

- *hay un cómputo de P* $d_1, \ldots, d_k$ *tal que* $d_1 = (1, \sigma_1)$

  Notamos $\psi_P^{(m)}(r_1, \ldots, r_m)$ al valor de Y en d_k (esta definido)
- *no hay tal cómputo, i.e. existe una secuencia infinita* $d_1, d_2, d_3, \ldots$

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

  Decimos que $\psi_P^{(m)}(r_1, \ldots, r_m)$ esta indefinido

Funciones computables

Una función (parcial) $f : \mathbb{N}^m \to \mathbb{N}$ es S-parcial computable (o simplemente parcial computable) si existe un programa P tal que $f(r_1, \ldots, r_m) = \psi_P^{(m)}(r_1, \ldots, r_m) \forall (r_1, \ldots, r_m)$

La **igualdad** (del meta-lenguaje) es verdadera si

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

La función f es **S-computable** (o simplemente computable) si es parcial computable y total.

- *Minimización no acotada*

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

  Teorema: Si $p: \mathbb{N}^{n+1} \to \{0,1\}$ es un predicado computable entonces $\min_t p(t, x_1, \ldots, x_n)$ es parcial computable

- *Clausura por composición*

  Teorema: Si h se obtiene a partir de las funciones (parciales) computables $f, g_1, \ldots, g_k$ por composición entonces h es (parcial) computable.

- *Clausura por recursión primitiva*

  Teorema: Si h se obtiene a partir de g por recursión primitiva y g es computable entonces h es computable.

Teorema: Las funciones computables forman una clase PRC.

Codificación de programas en S

S solo tiene el tipo de datos naturales, pero podemos simular otros tipos a partir de estos.

Codificacion de variables y etiquetas

Ordenamos las variables: $Y, X_1, Z_1, X_2, Z_2, \ldots$

Y las etiquetas: $A, B, C, \ldots, Z, AA, AB, \ldots$

Escribimos #(W) para la posición que ocupa la variable/etiqueta W en la lista

Codificación de instrucciones

Codificamos la instrucción I con $\#(I) = \langle a, \langle b, c \rangle \rangle$ donde

1. si I tiene etiqueta L, entonces a = #(L); si no a = 0
2. si la variable mencionada en I es V entonces c = #(V) − 1
3. si la instrucción I es
   1. V ← V entonces b = 0
   2. V ← V + 1 entonces b = 1
   3. V ← V − 1 entonces b = 2
   4. IF V ≠ 0 GOTO L' entonces b = #(L') + 2

Codificación de programas

Un programa P es una lista (finita) de instrucciones $\#(P) = [\#(I_1), \ldots, \#(I_k)] - 1$

Ambigüedades

La instrucción final de un programa no puede ser Y ← Y. Con esto, cada número representa a un único programa.

Funciones no computables

Teorema: El conjunto de las funciones (totales) $\mathbb{N} \to \mathbb{N}$ no es numerable.

=> o sea, hay más funciones N → N que números naturales

- hay tantos programas como números naturales
- hay tantas funciones computables como números naturales
- tiene que haber funciones $\mathbb{N} \to \mathbb{N}$ no computables

Halting problem

Verdadero sii el programa con número y y entrada x no se indefine:

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

Donde P es el unico programa tal que $\#(P)=y$

Tesis de Church

Tesis de Church: Todos los algoritmos para computar en los naturales se pueden programar en S.

Entonces, el problema de la detención dice no hay algoritmo para decidir la verdad o falsedad de HALT(x, y)

Universalidad

Para cada n > 0 definimos

$\Phi^{(n)}(x_1,\ldots,x_n,e)$ = salida del programa e con entrada $x_1,\ldots,x_n$ = $\Psi^{(n)}(x_1,\ldots,x_n,e)$ donde $\#(P)=e$

Teorema: Para cada n > 0 la función $\Phi^{(n)}$ es parcial computable.

Observar que el programa para Φ (n) es un intérprete de programas. Se trata de un programa que interpreta programas (representados por números).

Notación: $\Phi^{(n)}(x_1,\ldots,x_n)=\Phi^{(n)}(x_1,\ldots,x_n,e)$ (también se puede omitir el superíndice si n = 1)

Step counter

$STP^{(n)}(x_1,\ldots,x_n,e,t)$ sii el programa e termina en t o menos pasos con entrada $x_1,\ldots,x_n$

sii hay un cómputo del programa e de longitud ≤ t + 1, comenzando con la entrada $x_1,\ldots,x_n$

Teorema: Para cada n > 0, el predicado $STP^{(n)}(x_1,\ldots,x_n,e,t)$ es p.r.

Snapshot

$SNAP^{(n)}(x_1,\ldots,x_n,e,t)$ = representación de la configuración instantánea del programa e con entrada $x_1,\ldots,x_n$ en el paso t

Se representa como $\langle \#instruccion, lista\ representando\ estado \rangle$

Teorema: Para cada n > 0, la funcion $SNAP^{(n)}(x_1,\ldots,x_n,e,t)$ es p.r.

Teorema de la forma normal

Teorema: Sea f : $f:\mathbb{N}^n \to \mathbb{N}$ una función parcial computable. Entonces existe un predicado p.r. R : N n+1 → N tal que

$$f(x_1,\ldots,x_n) = l\left(\min_z R(x_1,\ldots,x_n,z)\right)$$

Otra caracterización de funciones computables

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

- composición,
- recursión primitiva y
- minimización

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

- composición,
- recursión primitiva y
- minimización propia

Teorema del parámetro

Hay un programa P_{x2} para la función $f_{x2}(x_1)=f(x_1,x_2)$

La transformación $(x_2,\#(P))\to\#(P_{x2})$ es p.r., es decir, existe una función $S:\mathbb{N}^2\to\mathbb{N}$ p.r. tal que dado x_2 e $y=\#(P)$ calcula $\#(P_{x2})$:

$$S(x_2,y) = \left(2^{109}\cdot 3^{110}\cdot\prod_{j=1}^{x_2} p_{j+2}^{26}\cdot\prod_{j=1}^{|y+1|} p_{j+x_2+2}^{(y+1)[j]}\right) - 1$$

Teorema: Hay una función p.r. $S:\mathbb{N}^2\to\mathbb{N}$ tal que

$$\Phi_y^{(2)}(x_1,x_2) = \Phi_{S(x_2,y)}^{(1)}(x_1)$$

Teorema: Para cada n, m > 0 hay una función p.r. inyectiva $S_m^n:\mathbb{N}^{n+1}\to\mathbb{N}$ tal que

$$\Phi_y^{(n+m)}(x_1,\ldots,x_m,u_1,\ldots,u_n) = \Phi_{S_m^n(u_1,\ldots,u_n,y)}^{(m)}(x_1,\ldots,x_m)$$

Teorema de la Recursión

Teorema: Si $g:\mathbb{N}^{n+1}\to\mathbb{N}$ es parcial computable, existe un e tal que

$$\Phi_e^{(n)}(x_1,\ldots,x_n) = g(e,x_1,\ldots,x_n)$$

Corolario: Si $g:\mathbb{N}^{n+1}\to\mathbb{N}$ es parcial computable, existen infinitos e tal que

$$\Phi_e^{(n)}(x_1,\ldots,x_n) = g(e,x_1,\ldots,x_n)$$

Quines

Un quine es un programa que cuando se ejecuta, devuelve como salida el mismo programa.

Proposición: Hay infinitos e tal que $\Phi_e(x)=e$ y $\Phi_e(x)=h(e)$

Teorema del punto fijo

Teorema: Si $f:\mathbb{N}\to\mathbb{N}$ es computable, existe un e tal que $\Phi_{f(e)}(x)=\Phi(e)$.

Conjuntos en teoría de la computabilidad

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

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

Un conjunto puede ser computable y/o primitivo recursivo.

Teorema: Sean A, B conjuntos de una clase PRC C. Entonces A ∪ B, A ∩ B y A están en C.

Conjuntos computablemente enumerables

Definición: Un conjunto A es computablemente enumerable (c.e.) cuando existe una función parcial computable $g:\mathbb{N}\to\mathbb{N}$ tal que $A = \{x : g(x)\downarrow\} = \text{dom } g$

- podemos decidir algoritmicamente si un elemento **sí** pertenece a A, pero para elementos que **no** pertenecen a A, el algoritmo se indefine
- se llaman algoritmos de **semi-decisión**: resuelven una aproximación al problema de decidir la pertenencia de un elemento al conjunto A

Propiedades de los conjuntos c.e.

- **Teorema:** Si A y B son c.e. entonces A ∪ B y A ∩ B también son c.e.
- **Teorema:** A es computable sii A y $\bar{A}$ son c.e.
- **Teorema:** Si A es c.e., existe un predicado p.r. $R:\mathbb{N}^2\to\mathbb{N}$ tal que $A = \{x : (\exists t)\ R(x,t)\}$
- **Teorema:** Si $A\neq\emptyset$ es c.e., existe una función p.r. $f:\mathbb{N}\to\mathbb{N}$ tal que $A = \{f(0), f(1), f(2), \ldots\}$
- **Teorema:** Si $f:\mathbb{N}\to\mathbb{N}$ es parcial computable, $A = \{f(x) : f(x)\downarrow\}$ es c.e.

Teorema de la enumeracion

Definimos $W_n = \{x : \Phi_n(x)\downarrow\} = \text{dominio del } n\text{-ésimo programa}$

Teorema: Un conjunto A es c.e. sii existe un n tal que $A=W_n$.

Existe una enumeración de todos los conjuntos c.e. $W_0, W_1, W_2, \ldots$

Teorema de la detencion

Definimos $K = \{n : n \in W_n\}$

Observar que $n \in W_n$ sii $\Phi_n(n)\downarrow$ sii $\text{HALT}(n, n)$

Teorema: K es c.e. pero no computable.

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

Teorema de Rice

$A \subseteq \mathbb{N}$ es un conjunto **de índices** si existe una clase de funciones $\mathbb{N} \to \mathbb{N}$ parciales computables C tal que $A = \{x : \Phi_x \in C\}$

Teorema: Si A es un conjunto de índices tal que $\emptyset \neq A \neq \mathbb{N}$, A no es computable.

Aplicaciones del teorema de Rice:

El teorema da una fuente de conjuntos no computables:

- $\{x : \Phi_x \text{ es total}\}$
- $\{x : \Phi_x \text{ es creciente}\}$
- $\{x : \Phi_x \text{ tiene dominio infinito}\}$
- $\{x : \Phi_x \text{ es primitiva recursiva}\}$

¡Todos son no computables porque todos son conjuntos de índices no triviales!

Oraculos

El lenguaje S se extiende:

- tiene entradas $X_1, \ldots, X_n \in \mathbb{N}$ (como antes) y una entrada especial que se llama **oráculo**
- en el oráculo no se pone un número natural sino un **conjunto** $A \subseteq \mathbb{N}$
- un nuevo término para leer el oráculo seteado en A

$$ORACULO[i] = \begin{cases} 1 & \text{si } i \in A \\ 0 & \text{sino} \end{cases}$$

Todo se puede aritmetizar como antes. Existe un programa universal:

$$\Phi_e^A(x_1, \ldots, x_n) = \begin{array}{l} \text{salida del } e\text{-ésimo programa} \\ \text{con entrada } x_1, \ldots, x_n \text{ y oráculo} = A \end{array}$$

Los oraculos tienen mas poder de computo:

Por ejemplo, hay un programa que calcula el halting problem con oraculo K: $K = \{x : \Phi_x(x)\downarrow\}$

Reducibilidad de Turing

Para $A, B \subseteq \mathbb{N}$, decimos que $A \leq_T B$ cuando se puede calcular el conjunto A con oráculo B, i.e. existe p tal que $\Phi_p^B = A$.

o sea, para todo $x \in \mathbb{N}$

$$\Phi_p^B(x) = A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{sino} \end{cases}$$

También se dice que A es **B-computable**

- $A \leq_T A$ para todo A
- $\overline{A} \leq_T A$ para todo A
- $\{x : \text{dom } \Phi_x = \emptyset\} \leq_T K$
- $\{x : \Phi_x \text{ es total}\} \not\leq_T K$
- $\{x : \Phi_x^K(x) \downarrow\} \not\leq_T K$

El salto

Para $A \subseteq \mathbb{N}$ se define el **salto de A** como $A' = \{x : \Phi_x^A(x) \downarrow\}$

Decimos $A <_T B$ cuando $A \leq_T B$ y $B \not\leq_T A$.

En general, para cualquier $A \subseteq \mathbb{N}$

$$A <_T A' <_T A'' <_T A''' \ldots$$

Conjuntos $\leq_T \emptyset$

- A es **c.e.** sii
  - existe e tal que $A = \text{dom}\Phi_e^{\emptyset}$
  - existe P p.r. tal que $A(x)=1$ sii $(\exists y)P(x,y)$
  - se los llama conjuntos Σ_1
- A es **co-c.e.** sii $\overline{A}$ es c.e. sii
  - existe e tal que $\overline{A} = \text{dom}\Phi_e^{\emptyset}$
  - existe R p.r. tal que $A(x)=1$ sii $(\forall y)R(x,y)$
  - se los llama conjuntos Π_1
- A es **computable** sii $A \leq_T \emptyset$ sii
  - A es c.e. y co-c.e.
  - existen P y R p.r. tal que $A(x)=1$ sii $(\exists y)P(x,y)$ sii $(\forall y)R(x,y)$
  - se los llama conjuntos Δ_1

Conjuntos $\leq_T \emptyset'$

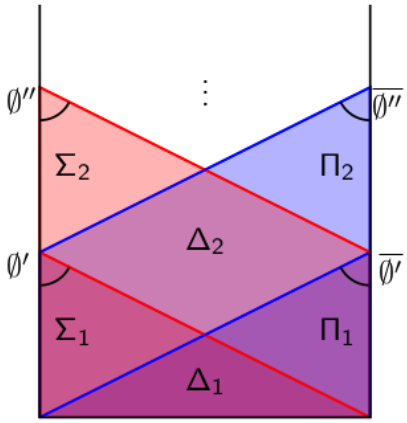
- A es $\emptyset'$-c.e. sii
  - existe e tal que $A = \mathrm{dom}\Phi_e^{\emptyset'}$
  - existe P p.r. tal que $A(x)=1$ sii $(\exists y)(\forall z)P(x,y,z)$
  - se los llama conjuntos Σ_2
- A es $\emptyset'$-co-c.e. sii $\overline{A}$ es $\emptyset'$-c.e. sii
  - existe e tal que $\overline{A} = \mathrm{dom}\Phi_e^{\emptyset'}$
  - existe R p.r. tal que $A(x)=1$ sii $(\forall y)(\exists z)R(x,y,z)$
  - se los llama conjuntos Π_2
- A es $\emptyset'$-computable sii $A \leq_T \emptyset'$ sii
  - A es $\emptyset'$-c.e. y $\emptyset'$-co-c.e.
  - existen P y R p.r. tal que $A(x)=1$ sii $(\exists y)(\forall z)P(x,y,z)$ sii $(\forall y)(\exists z)R(x,y,z)$
  - se los llama conjuntos Δ_2

Conjuntos $\leq_T \emptyset''$

- A es $\emptyset''$-c.e. sii
  - existe e tal que $A = \mathrm{dom}\Phi_e^{\emptyset''}$
  - existe P p.r. tal que $A(x)=1$ sii $(\exists y)(\forall z)(\exists w)P(x,y,z,w)$
  - se los llama conjuntos Σ_3
- A es $\emptyset''$-co-c.e. sii $\overline{A}$ es $\emptyset'$-c.e. sii
  - existe e tal que $\overline{A} = \mathrm{dom}\Phi_e^{\emptyset''}$
  - existe R p.r. tal que $A(x)=1$ sii $(\forall y)(\exists z)(\forall w)R(x,y,z,w)$
  - se los llama conjuntos Π_3
- A es $\emptyset''$-computable sii $A \leq_T \emptyset''$ sii
  - A es $\emptyset''$-c.e. y $\emptyset''$-co-c.e.
  - existen P y R p.r. tal que $A(x)=1$ sii $(\exists y)(\forall z)(\exists w)P(x,y,z,w)$ sii $(\forall y)(\exists z)(\forall w)R(x,y,z,w)$
  - se los llama conjuntos Δ_3

La jerarquia aritmetica

COMPLETAR



Problema de Post

Ejemplos de conjuntos c.e.:

- $K = \{x : \Phi_x(x)\downarrow\} \equiv_T \emptyset'$
- $\emptyset \equiv_T \emptyset$
- $\{\langle x, y\rangle : \Phi_x(y)\downarrow\} \equiv_T \emptyset'$
- $\{x : x \text{ es primo}\} \equiv_T \emptyset$
- $\{\langle x, \langle y, z\rangle\rangle : \Phi_x(y) = z\} \equiv_T \emptyset'$
- $\mathbb{N} \equiv_T \emptyset$
- $\{x : \text{dom } \Phi_x \neq \emptyset\} \equiv_T \emptyset'$
- $\{x : \Phi_x(x) \text{ tarda más de 10 pasos en terminar}\} \equiv_T \emptyset$
- $\{x : 0 \in \text{dom } \Phi_x\} \equiv_T \emptyset'$

Decimos $A \equiv_T B$ cuando $A \leq_T B$ y $B \leq_T A$.

Problema de Post: Existe un A c.e. tal que $\emptyset <_T A <_T \emptyset'$?

Solucion: Construir un conjunto A c.e. tal que

1. A es **bajo** (low): $A' \equiv_T \emptyset'$. Esto garantiza $A <_T A' \equiv_T \emptyset'$
2. A es **simple:** $\overline{A}$ es infinito y $\overline{A}$ no contiene ningun conjunto infinito c.e.
   No puede ser $A' \equiv_T \emptyset'$. Si lo fuera $\overline{A}$ seria c.e. e infinito