Lógica y Computabilidad

2do cuatrimestre 2020 - **A DISTANCIA**

Departamento de Computación - FCEyN - UBA

Computabilidad - clase 5

Codificación de programas, Halting problem, diagonalización, tesis de Church, programa universal, step counter, snapshot

Tipos de datos en $\mathcal{S}$

- vimos que el único tipo de dato en $\mathcal{S}$ son los naturales
- sin embargo podemos simular otros tipos. Por ejemplo,
  - tipo bool: lo representamos con el 1 (verdadero) y el 0 (falso)
  - tipo par de números naturales: la codificación y decodificación de pares son funciones primitivas recursivas
  - tipo entero: podría ser codificada con un par
    $$\langle \text{bool}, \text{número natural} \rangle$$
  - tipo secuencias finitas de números naturales: la codificación y decodificación de secuencias son funciones primitivas recursivas
- ahora vamos a ver como simular el tipo programa en $\mathcal{S}$

Codificación de programas en $\mathcal{S}$

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

1. $V \leftarrow V + 1$
2. $V \leftarrow V - 1$
3. IF $V \neq 0$ GOTO L'

Por conveniencia vamos a agregar una cuarta instrucción

4. $V \leftarrow V$: no hace nada

Observar que toda instrucción

- puede o no estar etiquetada con L
- menciona exactamente una variable V
- el IF además menciona siempre una etiqueta L'

Codificación de variables y etiquetas de $\mathcal{S}$

Ordenamos las variables:

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

Ordenamos las etiquetas:

$$A, B, C, D, \ldots, Z, AA, AB, AC, \ldots, AZ, BA, BB, \ldots, BZ, \ldots$$

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

Por ejemplo,

- $\#(Y) = 1$
- $\#(X_2) = 4$
- $\#(A) = 1$
- $\#(C) = 3$

Codificación de instrucciones de $\mathcal{S}$

Codificamos a 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
   3.1 $V \leftarrow V$ entonces $b = 0$
   3.2 $V \leftarrow V + 1$ entonces $b = 1$
   3.3 $V \leftarrow V - 1$ entonces $b = 2$
   3.4 IF $V \neq 0$ GOTO L' entonces $b = \#(L') + 2$

Por ejemplo,

- $\#(X \leftarrow X + 1) = \langle 0, \langle 1, 1 \rangle \rangle = \langle 0, 5 \rangle = 10$
- $\#([A] \quad X \leftarrow X + 1) = \langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 21$
- $\#(\text{IF } X \neq 0 \text{ GOTO } A) = \langle 0, \langle 3, 1 \rangle \rangle = \langle 0, 23 \rangle = 46$
- $\#(Y \leftarrow Y) = \langle 0, \langle 0, 0 \rangle \rangle = \langle 0, 0 \rangle = 0$

Todo número x representa a una única instrucción I.

Codificación de programas en $\mathcal{S}$

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

Codificamos al programa P con

$$\#(P) = [\#(I_1), \ldots, \#(I_k)] - 1$$

Por ejemplo, para el programa P

$$\begin{array}{ll} [A] & X \leftarrow X + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \end{array}$$

tenemos

$$\#(P) = [\#(I_1), \#(I_2)] - 1 = [21, 46] - 1 = 2^{21} \cdot 3^{46} - 1$$

Ambigüedades

Dijimos que P

$$\begin{array}{ll} [A] & X \leftarrow X + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \end{array}$$

tiene número $[21, 46] - 1$. Pero

$$[21, 46] = [21, 46, 0]$$

¡Un mismo número podría representar a más de un programa!
Por suerte, el programa $[21, 46, 0]$ es

$$\begin{array}{ll} [A] & X \leftarrow X + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \\ & Y \leftarrow Y \end{array}$$

y es equivalente a P.

De todos modos, eliminamos esta ambigüedad estipulando que

la instrucción final de un programa no puede ser $Y \leftarrow Y$

Con esto, cada número representa a un único programa.

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

Teorema (Cantor)

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

Demostración.

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

$$
\begin{array}{lcccccc}
\text{valores de } f_0 & = & f_0(0) & f_0(1) & f_0(2) & f_0(3) & \ldots \\
\text{valores de } f_1 & = & f_1(0) & f_1(1) & f_1(2) & f_1(3) & \ldots \\
\text{valores de } f_2 & = & f_2(0) & f_2(1) & f_2(2) & f_2(3) & \ldots \\
\vdots & & & & & & \\
\text{valores de } f_k & = & f_k(0) & f_k(1) & f_k(2) & f_k(3) & \ldots \\
\vdots & & & & & & \ddots
\end{array}
$$

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

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

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

Hay funciones no computables

- hay una cantidad no numerable de funciones $\mathbb{N} \to \mathbb{N}$
- o sea, hay más funciones $\mathbb{N} \to \mathbb{N}$ 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
  Pero ¿qué ejemplo concreto tenemos?

El problema de la detención (halting problem)

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

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

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

Ejemplo:

(pseudo)programa P

	$Y \leftarrow 2$
$[A]$	$Y \leftarrow Y + 2$
	IF $(\exists a)_{\leq Y} (\exists b)_{\leq Y}$ $[\text{Primo}(a) \wedge \text{Primo}(b) \wedge a + b = Y]$ GOTO A

Supongamos que $\#(P) = e$. ¿Cuánto vale $\text{HALT}(x, e)$?

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

HALT no es computable

Teorema (Turing, 1936)

HALT *no es computable.*

Demostración.

Supongamos que HALT es computable.
Construimos el siguiente programa:

Programa Q

[A]	IF $\mathrm{HALT}(X, X) = 1$ GOTO A

Supongamos que $\#(Q) = e$. Entonces

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

Entonces

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

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

Diagonalización

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

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

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

$$f(x)\downarrow \text{ sii } \Psi_{P_x}(x)\uparrow \qquad \begin{array}{cccc} \boxed{\Psi_{P_0}(0)} & \Psi_{P_0}(1) & \Psi_{P_0}(2) & \cdots \\ \Psi_{P_1}(0) & \boxed{\Psi_{P_1}(1)} & \Psi_{P_1}(2) & \cdots \\ \Psi_{P_2}(0) & \Psi_{P_2}(1) & \boxed{\Psi_{P_2}(2)} & \cdots \end{array}$$

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

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

Tesis de Church

Hay muchos modelos de cómputo.
Está probado que tienen el mismo poder que $\mathcal{S}$

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

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

Entonces, el problema de la detención dice

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

Universalidad

Para cada $n > 0$ definimos

$$\begin{aligned}\Phi^{(n)}(x_1, \ldots, x_n, e) &= \text{salida del programa } e \text{ con entrada } x_1, \ldots, x_n \\ &= \Psi_P^{(n)}(x_1, \ldots, x_n) \qquad \text{donde } \#(P) = e\end{aligned}$$

Teorema

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

Observar que el programa para $\Phi^{(n)}$ es un intérprete de programas. Se trata de un programa que interpreta programas (representados por números).

Para demostrar el teorema, construimos el programa U_n que computa $\Phi^{(n)}$.

Idea de U_n

U_n es un programa que computa

$$\begin{aligned}\Phi^{(n)}(x_1,\dots,x_n,e) &= \text{salida del programa } e \text{ con entrada } x_1,\dots,x_n\\ &= \Psi_P^{(n)}(x_1,\dots,x_n) \qquad \text{donde } \#(P)=e\end{aligned}$$

U_n necesita

- averiguar quién es P (decodifica e)
- llevar cuenta de los estados de P en cada paso
  - parte del estado inicial de P cuando la entrada es $x_1,\dots,x_n$
  - codifica los estados con listas
  - por ejemplo $Y=0, X_1=2, X_2=1$ lo codifica como $[0,2,0,1]=63$

En el código de U_n

- K indica el número de instrucción que se está por ejecutar (en la simulación de P)
- S describe el estado de P en cada momento

Inicialización

$$
\begin{array}{ll}
// & \text{entrada} = x_1, \ldots, x_n, e \\
// & \#(P) = e = [i_1, \ldots, i_m] - 1 \\
 & Z \leftarrow X_{n+1} + 1 \\
// & Z = [i_1, \ldots, i_m] \\
 & S \leftarrow \prod_{j=1}^{n} (p_{2j})^{X_j} \\
// & S = [0, X_1, 0, X_2, \ldots, 0, X_n] \text{ es el estado inicial} \\
 & K \leftarrow 1 \\
// & \text{la primera instrucción de } P \text{ a analizar es la } 1
\end{array}
$$

Ciclo principal

$//$ S codifica el estado, K es el número de instrucción

$//$ $Z = [i_1, \ldots, i_m]$

$[C]$ IF $K = |Z| + 1 \vee K = 0$ GOTO F

$//$ si llegó al final, terminar (ya veremos $K = 0$)

$//$ si no, sea $Z[K] = i_K = \langle a, \langle b, c\rangle\rangle$

$U \leftarrow r(Z[K])$

$//$ $U = \langle b, c\rangle$

$P \leftarrow p_{r(U)+1}$

$//$ la variable que aparece en i_K es la $c + 1$-ésima

$//$ P es el primo para la variable que aparece en i_K

Ciclo principal (cont.)

```
//    S codifica el estado, K es el número de instrucción
//    Z = [i_1, ..., i_m], i_K = ⟨a, ⟨b, c⟩⟩, U = ⟨b, c⟩
//    P es el primo para la variable V que aparece en i_K
      IF l(U) = 0 GOTO N
//    si se trata de una instrucción V ← V salta a N
      IF l(U) = 1 GOTO S
//    si se trata de una instrucción V ← V + 1 salta a S
//    si no, es de la forma V ← V − 1 o IF V ≠ 0 GOTO L
      IF ¬(P|S) GOTO N
//    si P no divide a S (i.e. V = 0), salta a N
      IF l(U) = 2 GOTO R
//    V ≠ 0. Si se trata de una instrucción V ← V − 1, salta a R
```

Caso IF $V \neq 0$ GOTO L y $V \neq 0$

$//$ S codifica el estado, K es el número de instrucción

$//$ $Z = [i_1, \dots, i_m], i_K = \langle a, \langle b, c \rangle \rangle, U = \langle b, c \rangle$

$//$ P es el primo para la variable V que aparece en i_K

$//$ $V \neq 0$ y se trata de la instrucción IF $V \neq 0$ GOTO L

$//$ $b \geq 2$, por lo tanto L es la $(b-2)$-ésima etiqueta

$$K \leftarrow \min_{j \leq |Z|} (l(Z[j]) + 2 = l(U))$$

$//$ K pasa a ser la primera instrucción con etiqueta L

$//$ si no hay tal instrucción, $K = 0$ (saldrá del ciclo)

GOTO C

$//$ vuelve a la primera instrucción del ciclo principal

Caso R (Resta)

// S codifica el estado, K es el número de instrucción
// $Z = [i_1, \ldots, i_m], i_K = \langle a, \langle b, c\rangle\rangle, U = \langle b, c\rangle$
// P es el primo para la variable V que aparece en i_K
// se trata de $V \leftarrow V - 1$ con $V \neq 0$
[R] $S \leftarrow S$ div P
GOTO N
// S=nuevo estado de P (resta 1 a V) y salta a N

Caso S (Suma)

$//$ S codifica el estado, K es el número de instrucción

$//$ $Z = [i_1, \ldots, i_m], i_K = \langle a, \langle b, c \rangle \rangle, U = \langle b, c \rangle$

$//$ P es el primo para la variable V que aparece en i_K

$//$ se trata de $V \leftarrow V + 1$

[S] $S \leftarrow S \cdot P$

GOTO N

$//$ S = nuevo estado de P (suma 1 a V) y salta a N

Caso N (Nada)

//	S codifica el estado, K es el número de instrucción
//	$Z = [i_1, \ldots, i_m], i_K = \langle a, \langle b, c \rangle \rangle, U = \langle b, c \rangle$
//	P es el primo para la variable V que aparece en i_K
//	la instrucción no cambia el estado
$[N]$	$K \leftarrow K + 1$
	GOTO C
//	S no cambia
//	K pasa a la siguiente instrucción
//	vuelve al ciclo principal

Devolución del resultado

//	S codifica el estado final de P
//	sale del ciclo principal
$[F]$	$Y \leftarrow S[1]$
//	$Y =$ el valor que toma la variable Y de P al terminar

Todo junto

$$
\begin{array}{ll}
 & Z \leftarrow X_{n+1} + 1 \\
 & S \leftarrow \displaystyle\prod_{i=1}^{n} (p_{2i})^{X_i} \\
 & K \leftarrow 1 \\
[C] & \text{IF } K = |Z| + 1 \lor K = 0 \text{ GOTO } F \\
 & U \leftarrow r(Z[K]) \\
 & P \leftarrow p_{r(U)+1} \\
 & \text{IF } l(U) = 0 \text{ GOTO } N \\
 & \text{IF } l(U) = 1 \text{ GOTO } S \\
 & \text{IF } \neg(P|S) \text{ GOTO } N \\
 & \text{IF } l(U) = 2 \text{ GOTO } R \\
 & K \leftarrow \min\limits_{i \leq |Z|} (l(Z[i]) + 2 = l(U)) \\
 & \text{GOTO } C
\end{array}
$$

$$
\begin{array}{ll}
[R] & S \leftarrow S \text{ div } P \\
 & \text{GOTO } N \\
[S] & S \leftarrow S \cdot P \\
 & \text{GOTO } N \\
[N] & K \leftarrow K + 1 \\
 & \text{GOTO } C \\
[F] & Y \leftarrow S[1]
\end{array}
$$

Notación

A veces escribimos

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

A veces omitimos el superíndice cuando $n = 1$

$$\Phi_e(x) = \Phi(x, e) = \Phi^{(1)}(x, e)$$

Step Counter

Definimos

$$
\begin{array}{lll}
\mathrm{STP}^{(n)}(x_1, \ldots, x_n, e, t) & \text{sii} & \text{el programa } e \text{ termina en} \\
 & & t \text{ o menos pasos con entrada } x_1, \ldots, x_n \\
 & \text{sii} & \text{hay un cómputo del programa } e \\
 & & \text{de longitud } \leq t + 1 \text{, comenzando} \\
 & & \text{con la entrada } x_1, \ldots, x_n
\end{array}
$$

Teorema

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

Snapshot

Definimos

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

La configuración instantánea se representa como

$$\langle \text{número de instrucción}, \text{lista representando el estado} \rangle$$

Teorema

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

Una función computable que no es primitiva recursiva

- se pueden codificar los programas de $\mathcal{S}$ con constructores y observadores p.r.
- se pueden codificar las definiciones de funciones p.r. con constructores y observadores p.r.
- existe $\Phi_e^{(n)}(x_1, \ldots, x_n)$ parcial computable que simula al e-ésimo programa con entrada $x_1, \ldots, x_n$
- existe $\tilde{\Phi}_e^{(n)}(x_1, \ldots, x_n)$ computable que simula a la e-ésima función p.r. con entrada $x_1, \ldots, x_n$.

Analicemos $g : \mathbb{N} \to \mathbb{N}$ definida como $g(x) = \tilde{\Phi}_x^{(1)}(x)$

- claramente g es computable
- supongamos que g es p.r.
  - entonces también es p.r. la función $f(x) = g(x) + 1 = \tilde{\Phi}_x^{(1)}(x) + 1$
  - existe un e tal que $\tilde{\Phi}_e = f$
  - tendríamos $\tilde{\Phi}_e(x) = f(x) = \tilde{\Phi}_x(x) + 1$
  - e está fijo pero x es variable
  - instanciando $x = e$, $\tilde{\Phi}_e(e) = f(e) = \tilde{\Phi}_e(e) + 1$. Absurdo.
- ¿por qué esto no funciona para parcial comp. en lugar de p.r.?

La función de Ackermann (1928)

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

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

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

Versión de Robinson & Peter (1948)

$$B(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ B(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ B(m-1, B(m, n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

- $B_0(n) = B(0,n) = n+1$
- $B_1(n) = B(1,n) = 2 + (n+3) - 3$
- $B_2(n) = B(2,n) = 2 \cdot (n+3) - 3$
- $B_3(n) = B(3,n) = 2 \uparrow (n+3) - 3$
- $B_4(n) = B(4,n) = 2 \uparrow\uparrow (n+3) - 3$
- $\ldots$

Para cada i, $B_i : \mathbb{N} \to \mathbb{N}$ es p.r. pero $B : \mathbb{N}^2 \to \mathbb{N}$ no es p.r.

- $B(4,2) \simeq 2 \times 10^{19728}$

$B'(x) = B(x,x)$ crece más rápido que cualquier función p.r.

$$(\forall f \text{ p.r.})(\exists n)(\forall x > n)\ B'(x) > f(x)$$