Lógica y Computabilidad

Clases de computabilidad

Santiago Figueira

Departamento de Computación, FCEyN, UBA

Verano 2007

Organización de la materia

- Docentes
  - Santiago Figueira (profesor)
  - Sergio Mera (JTP)
  - Daniel Gorín (Ayte. 1ra)
  - Mariano Moscato (Ayte. 1ra)
  - Lucía Cavatorta (Ayte. 2da)
  - Cynthia Disenfeld (Ayte. 2da)
  - Jorge Lucangeli (Ayte. 2da)
- Horario
  - Lunes a Jueves de 17 a 21.30
  - Primero la teórica, después la práctica
- Temas:
  - Computabilidad (8 clases)
  - Lógica proposicional (4 clases)
  - Lógica de primer orden (4 clases)
- Aprobación: dos parciales, cada uno con su recuperatorio
  - primer parcial: 20 febrero
  - segundo parcial: 8 marzo
- www.dc.uba.ar/lyc, lyc@dc.uba.ar

Lenguaje de programación $\mathscr{S}$

- imperativo, muy simple
  - variables de entrada: $X_1, X_2, \ldots$
  - variable de salida: Y
  - variables temporales: $Z_1, Z_2, \ldots$
    - no se declaran
    - todas empiezan en 0
- único tipo de dato: números naturales (con el 0)
  - no existen las constantes
- incrementar de a uno
  - $X_1 \leftarrow X_1 + 1$
- decrementar de a uno
  - $X_1 \leftarrow X_1 - 1$
  - si X_1 era 0 entonces no se modifica
- tiene un if muy primitivo
  - IF $X_1 \neq 0$ GOTO A
  - A es una etiqueta que denota una instrucción del programa
- no hay invocación de programas
  - no hay pasaje de parámetros

Ejemplo 1

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

- Escribimos X por X_1; Z por Z_1
- cuando $X = 0$ el programa termina porque no hay siguiente instrucción
- computa la función $f : \mathbb{N} \to \mathbb{N}$,

$$f(x) = \begin{cases} x & \text{si } x \neq 0 \\ 1 & \text{sino} \end{cases}$$

- siempre deja la variable X en 0

Ejemplo 2

$[A]$ IF $X \neq 0$ GOTO B
$Z \leftarrow Z + 1$
IF $Z \neq 0$ GOTO E
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
$Z \leftarrow Z + 1$
IF $Z \neq 0$ GOTO A

- computa la función $f : \mathbb{N} \to \mathbb{N}, f(x) = x$
- cuando intenta ir a E, termina
- En el ejemplo, Z solo sirve para un salto incondicional. En general GOTO L es equivalente a

$$V \leftarrow V + 1$$
IF $V \neq 0$ GOTO L

donde V es una variable nueva (en el ejemplo es Z)

Macros

- $\mathscr{S}$ no tiene salto incondicional
- pero podemos simularlo con GOTO L
- lo usamos, como si fuera del lenguaje, pero:
  - cada vez que aparece

    GOTO L

    en un programa P, lo reemplazamos con

    $V \leftarrow V + 1$
    IF $V \neq 0$ GOTO L

    dónde V tiene que ser una variable que no aparece en P.

Vamos a ver que se pueden simular muchas otras operaciones. Una vez que sepamos que se pueden escribir en el lenguaje $\mathscr{S}$, las usamos como si fueran propias (son pseudoinstrucciones).

- la forma abreviada se llama macro
- el segmento de programa que la macro abrevia se llama expansión del macro

Macro para la asignación de variables: $V \leftarrow V'$

$[A]$ IF $X \neq 0$ GOTO B
GOTO C
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
$Z \leftarrow Z + 1$
GOTO A
$[C]$ IF $Z \neq 0$ GOTO D
GOTO E
$[D]$ $Z \leftarrow Z - 1$
$X \leftarrow X + 1$
GOTO C

- el primer ciclo copia el valor de X en Y y en Z
- el segundo ciclo pone en X el valor que tenía originalmente y deja Z en cero
- se usa la macro GOTO A
  - no debe expandirse como

    $Z \leftarrow Z + 1$
    IF $Z \neq 0$ GOTO A

    sino como

    $Z_2 \leftarrow Z_2 + 1$
    IF $Z_2 \neq 0$ GOTO A

Macro para la asignación de variables: $V \leftarrow V'$

$Y \leftarrow 0$
$[A]$ IF $X \neq 0$ GOTO B
GOTO C
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
$Z \leftarrow Z + 1$
GOTO A
$[C]$ IF $Z \neq 0$ GOTO D
GOTO E
$[D]$ $Z \leftarrow Z - 1$
$X \leftarrow X + 1$
GOTO C

- se puede usar para asignar a la variable V el contenido de la variable V' y dejar V' sin cambios dentro de un programa P cualquiera: $V \leftarrow V'$.
  - cambiar Y por V
  - cambiar X por V'
  - cambiar Z por una variable temporal que no aparezca en P
- pero funciona bien solo cuando $V = 0$ y $Z = 0$
- lo arreglamos con $Y \leftarrow 0$ como primera pseudoinstrucción
  - no hace falta $Z \leftarrow 0$

Macro para la asignación de cero: $V \leftarrow 0$

En un programa P, la pseudoinstrucción $V \leftarrow 0$ se expande como

$$\begin{array}{ll} [L] & V \leftarrow V - 1 \\ & \text{IF } V \neq 0 \text{ GOTO } L \end{array}$$

donde L es una etiqueta que no aparece en P

Entonces el programa para $V \leftarrow V'$ queda:

$$\begin{array}{ll} & V \leftarrow 0 \\ [H] & V \leftarrow V - 1 \\ & \text{IF } V \neq 0 \text{ GOTO } H \\ [A] & \text{IF } V \neq 0 \text{ GOTO } B \\ & \text{GOTO } C \\ [B] & V \leftarrow V - 1 \\ & V' \leftarrow V' + 1 \\ & Z \leftarrow Z + 1 \\ & \text{GOTO } A \\ [C] & \text{IF } Z \neq 0 \text{ GOTO } D \\ & \text{GOTO } E \\ [D] & Z \leftarrow Z - 1 \\ & V \leftarrow V + 1 \\ & \text{GOTO } C \end{array}$$

Suma de dos variables

$$\begin{array}{ll} & Y \leftarrow X_1 \\ & Z \leftarrow X_2 \\ [B] & \text{IF } Z \neq 0 \text{ GOTO } A \\ & \text{GOTO } E \\ [A] & Z \leftarrow Z - 1 \\ & Y \leftarrow Y + 1 \\ & \text{GOTO } B \end{array}$$

computa la función $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$f(x_1, x_2) = x_1 + x_2$$

Resta de dos variables

$$\begin{array}{ll} & Y \leftarrow X_1 \\ & Z \leftarrow X_2 \\ [C] & \text{IF } Z \neq 0 \text{ GOTO } A \\ & \text{GOTO } E \\ [A] & \text{IF } Y \neq 0 \text{ GOTO } B \\ & \text{GOTO } A \\ [B] & Y \leftarrow Y - 1 \\ & Z \leftarrow Z - 1 \\ & \text{GOTO } C \end{array}$$

computa la función $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{si } x_1 \geq x_2 \\ \uparrow & \text{sino} \end{cases}$$

- g es una función parcial
- la indefinición se nota con $\uparrow$ (en el metalenguaje)
- el comportamiento del programa que se indefine es la no terminación
  - no hay otra causa de indefinición

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

Por ejemplo, para P:

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

- son estados de P:
  - $X = 3, Y = 1$
  - $X = 3, Y = 1, Z = 0$
  - $X = 3, Y = 1, Z = 8$
    - no hace falta que sea alcanzado
- no son estados de P:
  - $X = 3$
  - $X = 3, Z = 0$
  - $X = 3, Y = 1, X = 0$

Descripción instantánea

Supongamos que el programa P tiene longitud n.
Para un estado σ de P y un $i \in \{1, \ldots, n+1\}$,

- 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 $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ es σ, salvo que $V = m$ se reemplaza por $V = m + 1$
2. si la i-ésima instrucción de P es $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ es σ, salvo que $V = m$ se reemplaza por $V = \text{máx}\{m - 1, 0\}$
3. si la i-ésima instrucción de P es IF $V \neq 0$ GOTO L
   - τ es idéntico a σ

   3.1 si σ tiene $V = 0$ entonces $j = i + 1$

   3.2 si σ tiene $V = m$ para $m \neq 0$ entonces
   - si existe en P una instrucción con etiqueta L entonces $j = \text{mín}\{k : k\text{-ésima instrucción de } P \text{ tiene etiqueta } L\}$
   - sino $j = n + 1$

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 $i \in \{1, 2, \ldots, k-1\}$
- 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

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

  junto con

  $$V = 0$$

  para cada variable V que aparezca en P y no sea $X_1, \ldots, X_m, Y$
- 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 la configuración instantánea d_k.
- 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_1

  Decimos que $\Psi_P^{(m)}(r_1, \ldots, r_m)$ está indefinido (notamos $\Psi_P^{(m)}(r_1, \ldots, r_m) \uparrow$)

Funciones computables

Una función (parcial) $f : \mathbb{N}^m \to \mathbb{N}$ es parcialmente computable si existe un programa P tal que

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

para todo $(r_1, \ldots, r_m) \in \mathbb{N}^m$.

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 computable si es parcialmente computable y total.

Notar que un mismo programa P puede servir para computar funciones de 1 variable, 2 variables, etc. Supongamos que en P aparece X_n y no aparece X_i para $i > n$

- si solo se especifican $m < n$ variables de entrada, $X_{m+1}, \ldots, X_n$ toman el valor 0
- si se especifican $m > n$ variables de entrada, P ignorará $X_{n+1}, \ldots, X_m$

Composición

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

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

Teorema

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

Demostración.

El siguiente programa computa h:

$$\begin{aligned} Z_1 &\leftarrow g_1(X_1, \ldots, X_n) \\ &\vdots \\ Z_k &\leftarrow g_k(X_1, \ldots, X_n) \\ Y &\leftarrow f(Z_1, \ldots, Z_k) \end{aligned}$$

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

Recursión para $h : \mathbb{N} \to \mathbb{N}$

$h : \mathbb{N} \to \mathbb{N}$ se obtiene de $g : \mathbb{N}^2 \to \mathbb{N}$ por recursión primitiva si

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

Teorema

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

Demostración.

El siguiente programa computa h:

$$\begin{aligned} & Y \leftarrow k \quad \text{(es una macro, se puede hacer fácil)} \\ [A] \quad & \text{IF } X = 0 \text{ GOTO } E \quad \text{(otra macro, condición del IF por } =) \\ & Y \leftarrow g(Z, Y) \\ & Z \leftarrow Z + 1 \\ & X \leftarrow X - 1 \\ & \text{GOTO } A \end{aligned}$$

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

Recursión para $h : \mathbb{N}^{n+1} \to \mathbb{N}$

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

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

Teorema

Si h se obtiene a partir de f y g por recursión primitiva y f y g son computables entonces h es computable.

¿Otra forma de caracterizar a las funciones computables?

Combinamos funciones computables por esquemas de

- composición
- recursión primitiva

y obtenemos nuevas funciones computables.

- Definimos a las funciones computables como las funciones totales programables en el lenguaje imperativo simple $\mathscr{S}$.
- Pensemos ahora en un lenguaje funcional simple (i.e. recursión muy restrictiva).
  - ¿podremos definir a las funciones computables a través de los esquemas de composición y recursión primitiva en lugar de usar $\mathscr{S}$?
  - es decir, ¿podremos caracterizar a las funciones computables por medio de un lenguaje funcional simple?

Nos faltan las funciones iniciales. Proponemos

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

Clases PRC

Una clase $\mathcal{C}$ de funciones totales es PRC (primitive recursive closed) si

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

Teorema

La clase de funciones computables es una clase PRC.

Demostración.

Ya vimos 2. Veamos 1:

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

$$Y \leftarrow X + 1$$

- $n(x) = 0$ se computa con el programa vacío
- $u_i^n(x_1, \ldots, x_n) = x_i$ se computa con el programa

$$Y \leftarrow X_i$$

□

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.

Demostración.

($\Leftarrow$) La clase de funciones p.r. es una clase PRC. Luego, si f pertenece a toda clase PRC, en particular f es p.r.

($\Rightarrow$) Sea f p.r. y sea $\mathcal{C}$ una clase PRC. Como f es p.r., hay una lista

$$f_1, f_2, \ldots, f_n$$

tal que

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

Entonces todas las funciones de la lista están en $\mathcal{C}$.

□

¿Funciones computables = funciones primitivas recursivas?

Entonces la clase de funciones p.r. es la clase PRC más chica.

Corolario

Toda función p.r. es computable.

Demostración.

Ya probamos que la clase de funciones computables es PRC. Por el teorema anterior, si f es p.r., entonces f pertenece a la clase de funciones computables. □

Ciertamente no toda función parcialmente computable es p.r porque toda función p.r. es total. Pero...

¿toda función computable es p.r.?

Ejemplo de función p.r.

La $suma(x, y) = x + y$ es p.r.
Pensemos cómo se escribe en funcional

```
suma :: nat -> nat
suma x y = x + y
```

pensemos sin usar el $+$ general; solo el $+1$

```
suma :: nat -> nat
suma x 0 = x
suma x (y+1) = (suma x y) + 1
```

Podemos reescribirlo como

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

donde

$$g(x_1, x_2, x_3) = s(u_2^3(x_1, x_2, x_3))$$

Otras funciones primitivas recursivas

- $x \cdot y$
- $x!$
- x^y
- $x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y \\ 0 & \text{sino} \end{cases}$
- $\alpha(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{sino} \end{cases}$
- y muchas más ¿Todas las computables..?

Predicados primitivos recursivos

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

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

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

Por ejemplo, el predicado $x \leq y$ es p.r. porque se puede definir como

$$\alpha(x \dot{-} y)$$

Operadores lógicos

Teorema

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

Demostración.

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

□

Corolario

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

Corolario

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

Definición por casos (2)

Teorema

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

$$f(x_1,\ldots,x_n) = \begin{cases} g(x_1,\ldots,x_n) & \text{si } p(x_1,\ldots,x_n) \\ h(x_1,\ldots,x_n) & \text{sino} \end{cases}$$

está en $\mathcal{C}$.

Demostración.

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

Definición por casos ($m+1$)

Teorema

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

$$f(x_1,\ldots,x_n) = \begin{cases} g_1(x_1,\ldots,x_n) & \text{si } p_1(x_1,\ldots,x_n) \\ \quad\vdots & \\ g_m(x_1,\ldots,x_n) & \text{si } p_m(x_1,\ldots,x_n) \\ h(x_1,\ldots,x_n) & \text{sino} \end{cases}$$

está en $\mathcal{C}$.

Recursión primitiva

- todavía no respondimos si p.r. = computable
- no lo vamos a responder todavía

Observar que el esquema de recursión

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

es muy simple:

- 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}

Programas-for

Si volvemos a un lenguaje imperativo conocido como Pascal, las p.r. son las funciones que se computan con programas-for:

- el único tipo de ciclos es de la forma

```
for i=1 to x {
    S(i)
}
```

- no es el `for` de C (el de C funciona como `while`)
- no es el tipo de ciclos de $\mathscr{S}$ (hay ciclos tipo `while`)

Lo que sabemos:

- $\mathscr{S}$ puede simular un `while`
- un `for` se puede simular con un `while`
- todo programa-for se puede reescribir en $\mathscr{S}$

Preguntas:

- ¿todo programa de $\mathscr{S}$ se puede reescribir como un programa-for?
- ¿es igual de poderosa la recursión primitiva que la recursión general de funcional?
- ¿computable = primitivo recursivo?

Sumatorias y productorias (desde 0)

Teorema

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

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

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

Demostración.

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

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

Observar que no importa la variable en la que se hace la recursión (podemos definir $g'(x, t)$ como la clase pasada y luego $g(t, x) = g'(x, t)$)

Sumatorias y productorias (desde 1)

Teorema

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

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

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

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

Demostración.

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

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

Cuantificadores acotados

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

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

- $p(0, x_1, \ldots, x_n)$ es verdadero y

  ⋮

- $p(y, x_1, \ldots, x_n)$ es verdadero

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

- $p(0, x_1, \ldots, x_n)$ es verdadero o

  ⋮

- $p(y, x_1, \ldots, x_n)$ es verdadero

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

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

Cuantificadores acotados (con $\leq$)

Teorema

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

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

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

Demostración.

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

□

Cuantificadores acotados (con $<$)

Teorema

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

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

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

Demostración.

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

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

Más ejemplos de funciones primitivas recursivas

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

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

  Notar que con esta definición $0|0$.
- $primo(x)$ sii x es primo.

Minimización

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

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

¿Qué hace g?

- supongamos que existe un $t \leq y$ tal que $p(t, x_1, \ldots, x_n)$ es verdadero
  - sea t_0 el mínimo tal t
  - $p(t, x_1, \ldots, x_n) = 0$ para todo $t < t_0$
  - $p(t_0, x_1, \ldots, x_n) = 1$
  - $\prod_{t=0}^{u} \alpha(p(t, x_1, \ldots, x_n)) = \begin{cases} 1 & \text{si } u < t_0 \\ 0 & \text{sino} \end{cases}$
  - $g(y, x_1, \ldots, x_n) = \underbrace{1 + 1 + \cdots + 1}_{t_0 \text{ veces}} = t_0$
  - entonces $g(x_1, \ldots, x_n)$ es el mínimo t tal que $p(t, x_1, \ldots, x_n)$ es verdadero
- si no existe tal t, $g(y, x_1, \ldots, x_n) = y + 1$

Minimización

Notamos

$$\min_{t \leq y} p(t, x_1, \ldots, x_n) = \begin{cases} \text{mínimo } t \leq y \text{ tal que} \\ p(t, x_1, \ldots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ 0 & \text{sino} \end{cases}$$

Teorema

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

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

también está en $\mathcal{C}$.

Más ejemplos de funciones primitivas recursivas

- x div y es la división entera de x por y

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

  Notar que con esta definición 0 div 0 es falso.
- x mod y es el resto de dividir a x por y
- p_n es el n-ésimo primo ($n > 0$). Se define $p_0 = 0, p_1 = 2, p_2 = 3, p_3 = 5, \ldots$

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

  Necesitamos una cota $K(n)$ que sea buena, i.e.
  - suficientemente grande y
  - primitiva recursiva

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

Minimización no acotada

Recordar la definición de minimización acotada:

$$\min_{t \leq y} p(t, x_1, \ldots, x_n) = \begin{cases} \text{mínimo } t \leq y \text{ tal que} \\ p(t, x_1, \ldots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ 0 & \text{sino} \end{cases}$$

¿Qué pasa cuando no hay cota? Definimos la minimización no acotada

$$\min_{t} p(t, x_1, \ldots, x_n) = \begin{cases} \text{mínimo } t \text{ tal que} \\ p(t, x_1, \ldots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ \uparrow & \text{sino} \end{cases}$$

Minimización no acotada

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 parcialmente computable.

Demostración.
El siguiente programa computa $\min_t p(t, x_1, \ldots, x_n)$:

$$\begin{aligned} [A] \quad & \text{IF } p(X_1, \ldots, X_n, Y) = 1 \text{ GOTO } E \\ & Y \leftarrow Y + 1 \\ & \text{GOTO } A \end{aligned}$$

□

Minimización acotada y no acotada

Lo que sabemos:

- un `while` puede simular una minimización no acotada
- un `for` puede simular una minimización acotada

Preguntas:

- ¿todo programa de $\mathscr{S}$ se puede reescribir como un programa-for?
- ¿en cualquier minimización no acotada se puede encontrar una cota válida y convertirla en minimización acotada?
- ¿computable = primitivo recursivo?

Tipos de datos en $\mathscr{S}$

Vimos que el único tipo de dato en $\mathscr{S}$ son los naturales

Sin embargo podemos simular otros tipos. Por ejemplo, el tipo Bool lo representamos con el 1 (verdadero) y el 0 (falso).

En esta clase, veremos como codificar

- pares de naturales
- secuencias finitas de naturales

Codificación de pares

Definimos la función primitiva recursiva

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

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

Proposición

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

Demostración.

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

□

Observadores de pares

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

- $l(z) = x$
- $r(z) = y$

Proposición

Los observadores de pares son primitivas recursivas.

Demostración.

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

- $l(z) = \text{mín}_{x\leq z}\,((\exists y)_{\leq z}\; z = \langle x, y\rangle)$
- $r(z) = \text{mín}_{y\leq z}\,((\exists x)_{\leq z}\; z = \langle x, y\rangle)$

□

Por ejemplo,

- $\langle 2, 5\rangle = 2^2(2 \cdot 5 + 1)\dot{-}1 = 43$
- $l(43) = 2$
- $r(43) = 5$

Codificación de secuencias

El número de Gödel de la secuencia

$$a_1, \ldots, a_n$$

es el número

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

Por ejemplo el número de Gödel de la secuencia

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

es

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

Propiedades de la codificación de secuencias

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

Demostración.
Por la factorización única en primos. □

Observar que

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

pero

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

Observadores de secuencias

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

- $x[i] = a_i$
- $|x|$ = longitud de x

Proposición

Los observadores de secuencias son primitivas recursivas.

Demostración.

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

□

Por ejemplo,

- $[1, 3, 3, 2, 2][2] = 4 = 40020750[2]$
- $[1, 3, 3, 2, 2][6] = 0 = 40020750[6]$
- $|[1, 3, 3, 2, 2]| = 5 = |40020750|$
- $|[1, 3, 3, 2, 2, 0]| = |[1, 3, 3, 2, 2, 0, 0]| = 5 = |40020750|$
- $x[0] = 0$ para todo x
- $0[i] = 0$ para todo i

En resumen

Teorema (Codificación de pares)

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

Teorema (Codificación de secuencias)

- $[a_1, \ldots, a_n][i] = \begin{cases} a_i & \text{si } 1 \leq i \leq n \\ 0 & \text{sino} \end{cases}$
- *si* $n \geq |x|$ *entonces* $[x[1], \ldots, x[n]] = x$
- *la codificación y observadores de secuencias son p.r.*

Codificación de programas en $\mathscr{S}$

Recordemos que las instrucciones de $\mathscr{S}$ 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 $\mathscr{S}$

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 $\mathscr{S}$

Codificamos a la instrucción I con

$$\#(I) = \langle a, \langle b, c \rangle \rangle$$

donde

1. si I tiene etiqueta L, entonces $a = \#(L)$; sino $a = 0$
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 $\mathscr{S}$

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

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

Ambigüedades

Dijimos que P

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

tiene número $[21, 46]$. 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.

Conjuntos numerables y no numerables

Los números naturales son numerables.

Teorema (Cantor)

El conjunto de los números reales en $[0,1]$ *no es numerable.*

Demostración.

Supongamos que lo fuera. Los enumero:

$$\begin{array}{lllllll} r_1 = & 0, & r_{11} & r_{12} & r_{13} & r_{14} & \dots \\ r_2 = & 0, & r_{21} & r_{22} & r_{23} & r_{24} & \dots \\ r_3 = & 0, & r_{31} & r_{32} & r_{33} & r_{34} & \dots \\ r_3 = & 0, & r_{41} & r_{42} & r_{43} & r_{44} & \dots \\ \vdots & & & & & & \\ r_k = & 0, & r_{k1} & r_{k2} & r_{k3} & r_{k4} & \dots \\ \vdots & & & & & & \ddots \end{array}$$

Defino el siguiente número x

$$x = \quad 0, \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad \dots$$

con $x_i = (r_{ii} + 2) \mod 10$. Entonces x no es un real. □

Hay funciones no computables

Hay tantas funciones totales $f : \mathbb{N} \to \{0, \dots, 9\}$ como números reales en $[0,1]$.
Se puede codificar la función f como

$$0, \quad f(0) \quad f(1) \quad f(2) \quad f(3) \quad \dots$$

- toda función se representa con un único real en $[0,1]$
- todo real en $[0,1]$ representa una única función

En general (hablando informalmente),

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

El problema de la detención (halting problem)

HALT(x,y) es verdadero sii el programa con número y y entrada x no se indefine, i.e.

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

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

HALT no es computable

Teorema

HALT *no es computable.*

Demostración.

Supongamos que lo fuera. Construimos el siguiente programa P:

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

Es claro que

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

Supongamos que $\#(P) = e$. Por definición de HALT,

$$\text{HALT}(x,e) \quad \text{sii} \quad P(x) \text{ termina} \quad \text{sii} \quad \neg\text{HALT}(x,x)$$

e está fijo; x es variable. En particular, para $x = e$:

$$\text{HALT}(e,e) \quad \text{sii} \quad P(e) \text{ termina} \quad \text{sii} \quad \neg\text{HALT}(e,e)$$

□

Tesis de Church

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

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

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

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

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

Teorema
Para cada $n > 0$ la función $\Phi^{(n)}$ es parcialmente 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

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

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 $x_1, \ldots, x_n$
  - 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

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

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 $K = 0$)
// sino, 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
// $Z = [i_1, \ldots, i_m]$, $i_K = \langle a, \langle b, c\rangle\rangle$, $U = \langle b, c\rangle$
// P es el primo para la variable V que aparece en i_K
IF $I(U) = 0$ GOTO N
// si se trata de una instrucción $V \leftarrow V$ salta a N
IF $I(U) = 1$ GOTO S
// si se trata de una instrucción $V \leftarrow V + 1$ salta a S
// sino, es de la forma $V \leftarrow V - 1$ o IF $V \neq 0$ GOTO L
IF $\neg(P|S)$ GOTO N
// si P no divide a S (i.e. V=0), salta a N
IF $I(U) = 2$ GOTO R
// $V \neq 0$ y se trata de una instrucción $V \leftarrow V - 1$ salta a R

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

// S codifica el estado, K es el número de instrucción
// $Z = [i_1, \ldots, i_m]$, $i_K = \langle a, \langle b, c\rangle\rangle$, $U = \langle b, c\rangle$
// P es el primo para la variable V que aparece en i_K
// $V \neq 0$ 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
$K \leftarrow \min_{j \leq |Z|} (I(Z[j]) + 2 = I(U))$
// 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

$Z \leftarrow X_{n+1} + 1$
$S \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i}$
$K \leftarrow 1$
$[C]$ IF $K = |Z| + 1 \vee K = 0$ GOTO F
$U \leftarrow r(Z[k])$
$P \leftarrow p_{r(U)+1}$
IF $l(U) = 0$ GOTO N
IF $l(U) = 1$ GOTO S
IF $\neg(P|S)$ GOTO N
IF $l(U) = 2$ GOTO R
$K \leftarrow \min_{i \leq |Z|} (l(Z[i]) + 2 = l(U))$
GOTO C

$[R]$ $S \leftarrow S$ div P
GOTO N
$[S]$ $S \leftarrow S \cdot P$
GOTO N
$[N]$ $K \leftarrow K + 1$
GOTO C
$[F]$ $Y \leftarrow S[1]$

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

$\mathrm{STP}^{(n)}(x_1, \ldots, x_n, e, t)$ 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 $\leq t + 1$, comenzando con la entrada $x_1, \ldots, x_n$

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

Snapshot

Definimos

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

La configuración instantánea se representa como

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

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

Una función computable que no es primitiva recursiva

- se pueden codificar los programas de $\mathscr{S}$ con constructores y observadores p.r.
- se pueden codificar las definiciones de funciones p.r. con constructores y observadores p.r.
- existe $\Phi_e^{(n)}(x_1, \ldots, x_n)$ parcialmente 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 $x_1, \ldots, x_n$.

Definamos $f : \mathbb{N} \to \mathbb{N}, f(x) = \tilde{\Phi}_x(x) + 1$

- f es computable porque $\tilde{\Phi}$ lo es
- f no es p.r. porque si lo fuera,
  - existiría 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 $x = e$, $\tilde{\Phi}_e(e) = f(e) = \tilde{\Phi}_e(e) + 1$
- esta misma demostración también prueba que $\tilde{\Phi}$ no es p.r.

La función de Ackermann (1928)

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

- $A_0(y,z) = A(0,y,z) = y + z$
- $A_1(y,z) = A(1,y,z) = y \cdot z$
- $A_2(y,z) = A(2,y,z) = y \uparrow z$
- $A_3(y,z) = A(3,y,z) = y \uparrow\uparrow z$
- ...

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

Versión de Robinson & Peter (1948)

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

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

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

A y B crecen más rápido que cualquier función p.r.

Teorema de la forma normal

Teorema

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

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

Demostración.

Sea e el número de algún programa para $f(x_1, \ldots, x_n)$.
Recordar que la configuración instantánea se representa como

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

El siguiente predicado $R(x_1, \ldots, x_n, z)$ es el buscado:

$$l(z) = \underbrace{\underbrace{r\left(\text{SNAP}^{(n)}(x_1, \ldots, x_n, e, r(z))\right)}_{\text{estado final de } e \text{ con entrada } x_1, \ldots, x_n}}_{\text{valor de la variable } Y \text{ en ese estado final}} \; [1]$$

$$\text{STP}^{(n)}(x_1, \ldots, x_n, e, r(z)) \;\wedge$$

Otra caracterización de funciones computables

Teorema

Una función es parcialmente 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*
  (del tipo $\min_t q(x_1, \ldots, x_n, t)$ donde siempre existe al menos un t tal que $q(x_1, \ldots, x_n, t)$ es verdadero)

Eliminando variables de entrada

Consideremos un programa P que usa la entrada X_1 y X_2:

INSTRUCCIÓN 1	$\#(I_1)$
$\vdots$	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f : \mathbb{N}^2 \to \mathbb{N}$

$$f(x, y) = \Psi_P^{(2)}(x, y)$$

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

Busco número de programa P_0 para $f_0 : \mathbb{N} \to \mathbb{N}, f_0(x) = f(x, 0)$

[A] $X_2 \leftarrow X_2 - 1$	109
IF $X_2 \neq 0$ GOTO A	110
INSTRUCCIÓN 1	$\#(I_1)$
$\vdots$	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f_0 : \mathbb{N} \to \mathbb{N}$

$$f_0(x) = \Psi_{P_0}^{(1)}(x)$$

$$\#(P_0) = [109, 110, \#(I_1), \ldots, \#(I_k)] - 1$$

Eliminando variables de entrada

Busco número de programa P_1 para $f_1 : \mathbb{N} \to \mathbb{N}, f_1(x) = f(x, 1)$

[A] $X_2 \leftarrow X_2 - 1$	109
IF $X_2 \neq 0$ GOTO A	110
$X_2 \leftarrow X_2 + 1$	26
INSTRUCCIÓN 1	$\#(I_1)$
$\vdots$	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f_1 : \mathbb{N} \to \mathbb{N}$

$$f_1(x) = \Psi_{P_1}^{(1)}(x)$$

$$\#(P_1) = [109, 110, 26, \#(I_1), \ldots, \#(I_k)] - 1$$

Busco número de programa P_2 para $f_2 : \mathbb{N} \to \mathbb{N}, f_2(x) = f(x, 2)$

[A] $X_2 \leftarrow X_2 - 1$	109
IF $X_2 \neq 0$ GOTO A	110
$X_2 \leftarrow X_2 + 1$	26
$X_2 \leftarrow X_2 + 1$	26
INSTRUCCIÓN 1	$\#(I_1)$
$\vdots$	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f_2 : \mathbb{N} \to \mathbb{N}$

$$f_2(x) = \Psi_{P_2}^{(1)}(x)$$

$$\#(P_2) = [109, 110, 26, 26, \#(I_1), \ldots, \#(I_k)] - 1$$

Teorema del Parámetro

Hay un programa P_i para la función $f_i(x) = f(x, i)$

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

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

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

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

Programas autoreferentes

- en la demostración del Halting Problem construimos un programa P que, cuando se ejecuta con su mismo número de programa (i.e. #(P)), evidencia una contradicción.
- en general, los programas pueden dar por supuesto que conocen su mismo número de programa
- pero si un programa P conoce su número de programa, podría, por ejemplo, devolver su mismo número, i.e. $\#(P)$

Teorema de la Recursión

Teorema

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

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

Demostración.

Sea S_n^1 la función del Teorema del Parámetro:

$$\Phi_y^{(n+1)}(x_1, \ldots, x_n, u) = \Phi_{S_n^1(u,y)}^{(n)}(x_1, \ldots, x_n).$$

La función $g(S_n^1(v, v), x_1, \ldots, x_n)$ es parcialmente computable, de modo que existe d tal que

$$\begin{aligned} g(S_n^1(v, v), x_1, \ldots, x_n) &= \Phi_d^{(n+1)}(x_1, \ldots, x_n, v) \\ &= \Phi_{S_n^1(d,v)}^{(n)}(x_1, \ldots, x_n) \end{aligned}$$

d está fijo; v es variable. Elegimos $v = d$ y $e = S_n^1(d, d)$.

□

Aplicaciones del Teorema de la Recursión

Corolario

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

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

Demostración.

En la demostración del teorema anterior, existen infinitos d tal que

$$\Phi_d^{(n+1)} = g(S_n^1(v, v), x_1, \ldots, x_n).$$

$S_n^1(v, v)$ es inyectiva de modo que existen infinitos

$$e = S_n^1(d, d).$$

□

Quines

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

Por ejemplo:

```
char*f="char*f=%c%s%c;main()
{printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```


Quines

¿Existe e tal que $\Phi_e(x) = e$?

Sí, el programa vacío tiene numero 0 y computa la función constante 0, i.e. $\Phi_0(x) = 0$.

Corolario

Hay infinitos e *tal que* $\Phi_e(x) = e$.

Demostración.

Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}, g(z, x) = z$.
Aplicando el Teorema de la Recursión, existen infinitos e tal que

$$\Phi_e(x) = g(e, x) = e.$$

□

Teorema del punto fijo

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

Demostración.
Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}, g(z, x) = \Phi_{f(z)}(x)$. Aplicando el Teorema de la Recursión, existe un e tal que

$$\Phi_e(x) = g(e, x) = \Phi_{f(e)}(x)$$

□

Ejercicio

Probar que $f : \mathbb{N} \to \mathbb{N}$,

$$f(x) = \begin{cases} 1 & \Phi_x \text{ es total} \\ 0 & \text{sino} \end{cases}$$

no es computable.

Supongamos f computable. Puedo definir

$$[A] \qquad \text{IF } f(X) = 1 \text{ GOTO } A$$

y por lo tanto

$$g(x, y) = \begin{cases} \uparrow & \Phi_x \text{ es total} \\ 0 & \text{sino} \end{cases}$$

es parcialmente computable. Por el Teorema de la Recursión, sea e tal que $\Phi_e(y) = g(e, y)$.

- Φ_e es total $\Rightarrow g(e, y) \uparrow$ para todo $y \Rightarrow \Phi_e(y) \uparrow$ para todo y $\Rightarrow \Phi_e$ no es total
- Φ_e no es total $\Rightarrow g(e, y) = 0$ para todo $y \Rightarrow \Phi_e(y) = 0$ para todo $y \Rightarrow \Phi_e(y) = 0$ es total

Conjuntos en teoría de la computabilidad

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

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

Así, un conjunto puede ser:

- computable
- primitivo recursivo

Teorema
Sean A, B conjuntos de una clase PRC $\mathcal{C}$. Entonces $A \cup B$, $A \cap B$ y $\overline{A}$ están en $\mathcal{C}$.

Conjuntos recursivamente enumerables

Igual que con las funciones

- hay conjuntos computables, por ejemplo
  - $\emptyset$
  - $\mathbb{N}$
  - $\{p : p \text{ es primo}\}$
- hay conjuntos no computables, por ejemplo
  - $\{\langle x, y\rangle : \text{HALT}(x, y)\}$
  - $\{\langle x, \langle y, z\rangle\rangle : \Phi_x(y) = z\}$

Un conjunto A es recursivamente enumerable (r.e.) cuando existe una función parcialmente 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 pertenece 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 r.e.

Teorema
Si A es computable entonces A es r.e.

Demostración.
Sea P_A un programa para [la función característica de] A.
Consideremos el siguiente programa P:

$$[C] \quad \text{IF } P_A(X) = 0 \text{ GOTO } C$$

Tenemos

$$\Psi_P(x) = \begin{cases} 0 & \text{si } x \in A \\ \uparrow & \text{sino} \end{cases}$$

y por lo tanto

$$A = \{x : \Psi_P(x) \downarrow\}$$

□

Propiedades de los conjuntos r.e.

Teorema
Si A y B son r.e. entonces $A \cup B$ y $A \cap B$ también son r.e.

Demostración.
Sean $A = \{x : \Phi_p(x) \downarrow\}$, $B = \{x : \Phi_q(x) \downarrow\}$

$(A \cap B)$ El siguiente programa R computa $A \cap B$:

$$Y \leftarrow \Phi_p(x)$$
$$Y \leftarrow \Phi_q(x)$$

En efecto, $\Psi_R(x) \downarrow$ sii $\Phi_p(x) \downarrow$ y $\Phi_q(x) \downarrow$.

$(A \cup B)$ El siguiente programa R' computa $A \cup B$:

$$\begin{aligned} [C] \quad & \text{IF STP}^{(1)}(X, p, T) = 1 \text{ GOTO } E \\ & \text{IF STP}^{(1)}(X, q, T) = 1 \text{ GOTO } E \\ & T \leftarrow T + 1 \\ & \text{GOTO } C \end{aligned}$$

En efecto, $\Psi_{R'}(x) \downarrow$ sii $\Phi_p(x) \downarrow$ o $\Phi_q(x) \downarrow$.

□

Propiedades de los conjuntos r.e.

Teorema
A es computable sii A y $\overline{A}$ son r.e.

Demostración.

$(\Rightarrow)$ si A es computable entonces $\overline{A}$ es computable

$(\Leftarrow)$ supongamos que A y $\overline{A}$ son r.e.

$$A = \{x : \Phi_p(x) \downarrow\} \quad , \quad \overline{A} = \{x : \Phi_q(x) \downarrow\}$$

Consideremos P:

$$\begin{aligned} [C] \quad & \text{IF STP}^{(1)}(X, p, T) = 1 \text{ GOTO } F \\ & \text{IF STP}^{(1)}(X, q, T) = 1 \text{ GOTO } E \\ & T \leftarrow T + 1 \\ & \text{GOTO } C \\ [F] \quad & Y \leftarrow 1 \end{aligned}$$

Para cada x, $x \in A$ o bien $x \in \overline{A}$. Entonces Ψ_P computa A.

□

Teorema de la enumeración

Definimos

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

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

Existe una enumeración de todos los conjuntos r.e.

$$W_0, W_1, W_2, \ldots$$

Problema de la detención (visto como conjunto)

Recordar que

$$W_n = \{x : \Phi_n(x) \downarrow\}$$

Definimos

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

Observar que

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

Teorema
K es r.e. pero no computable.

Demostración.

- la función $\Phi(n, n)$ es parcialmente computable, de modo que K es r.e.
- supongamos que K fuera computable. Entonces $\overline{K}$ también lo sería. Luego existe un e tal que $\overline{K} = W_e$. Por lo tanto

$$e \in K \quad \text{sii} \quad e \in W_e \quad \text{sii} \quad e \in \overline{K}$$

□

Más propiedades de los conjuntos r.e.

Teorema
Si A es r.e., existe un predicado p.r. $R : \mathbb{N}^2 \to \mathbb{N}$ tal que

$$A = \{x : (\exists t)\ R(x, t)\}$$

Demostración.
Sea $A = W_e$. Es decir,

$$A = \{x : \Phi_e(x) \downarrow\}.$$

Entonces $x \in A$ cuando en algún tiempo t, el programa e con entrada x termina, i.e.

$$A = \{x : (\exists t)\ \underbrace{\text{STP}^{(1)}(x, e, t)}_{R(x,t)}\}$$

□

Más propiedades de los conjuntos r.e.

Teorema
Si $A \neq \emptyset$ es r.e., existe una función p.r. $f : \mathbb{N} \to \mathbb{N}$ tal que

$$A = \{f(0), f(1), f(2), \dots\}$$

Demostración.
Por el teorema anterior, existe P p.r. tal que

$$A = \{x : (\exists t)\ P(x, t)\}.$$

Sea $a \in A$ y definimos

$$f(u) = \begin{cases} l(u) & \text{si } P(l(u), r(u)) \\ a & \text{sino} \end{cases}$$

- $x \in A \Rightarrow$ existe t tal que $P(x, t) \Rightarrow f(\langle x, t\rangle) = x$
- sea x tal que $f(u) = x$ para algún u. Entonces $x = a$ o bien u es de la forma $u = \langle x, t\rangle$, con $P(x, t)$. Luego $x \in A$.

□

Más propiedades de los conjuntos r.e.

Teorema
Si $f : \mathbb{N} \to \mathbb{N}$ es parcialmente computable, $A = \{f(x) : f(x) \downarrow\}$ es r.e.

Demostración.
Sea $\Phi_p = f$. Definamos P'

[A] IF $\text{STP}^{(1)}(Z, p, T) = 0$ GOTO B
IF $\Phi_p(Z) = X$ GOTO E
[B] $Z \leftarrow Z + 1$
IF $Z \leq T$ GOTO A
$T \leftarrow T + 1$
$Z \leftarrow 0$
GOTO A

Notar que $\Psi_{P'}(X) \downarrow$ si existen Z, T tal que

- $Z \leq T$
- $\text{STP}^{(1)}(Z, p, T)$ es verdadero (i.e. el programa para f termina en T o menos pasos con entrada Z)
- $X = f(Z)$

$$\Psi_{P'}(x) = \begin{cases} 0 & \text{si } x \in A \\ \uparrow & \text{sino} \end{cases}$$

Luego A es r.e.

□

Caracterizaciones de los conjuntos r.e.

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

1. *A es r.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*

Demostración.

$(1 \Rightarrow 2)$ *Teorema de hoja 99*
$(2 \Rightarrow 3)$ *Trivial*
$(3 \Rightarrow 4)$ *Trivial*
$(4 \Rightarrow 1)$ *Teorema de hoja 100*

□

Teorema de Rice

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

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

Demostración.
Supongamos $\mathcal{C}$ tal que $A = \{x : \Phi_x \in \mathcal{C}\}$ computable. Sean $f \in \mathcal{C}$ y $g \notin \mathcal{C}$ funciones parcialmente computables.
Sea $h : \mathbb{N}^2 \to \mathbb{N}$ la función parcialmente computable:

$$h(t,x) = \begin{cases} g(x) & \text{si } t \in A \\ f(x) & \text{sino} \end{cases}$$

Por el Teorema de la Recursión, existe e tal que $\Phi_e(x) = h(e,x)$.

- $e \in A \Rightarrow \Phi_e = g \Rightarrow \Phi_e \notin \mathcal{C} \Rightarrow e \notin A$
- $e \notin A \Rightarrow \Phi_e = f \Rightarrow \Phi_e \in \mathcal{C} \Rightarrow e \in A$

□

Aplicaciones del Teorema de Rice

El teorema da una fuente de conjuntos no computables:

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

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