Lógica y Computabilidad

2do cuatrimestre 2020 - **A DISTANCIA**

Departamento de Computación - FCEyN - UBA

Computabilidad - clase 4

Lenguaje $\mathcal{S}$, estado, descripción instantánea, cómputo, funciones parciales computables, minimización no acotada

Lenguaje de programación $\mathcal{S}$ (Davis/Sigal/Weyuker)

- resulta igual de poderoso que las máquinas de Turing, pero es más fácil de programar
- imperativo, muy simple

  variables de entrada: $X_1, X_2, \ldots$

  única variable de salida: Y

  variables temporales: $Z_1, Z_2, \ldots$

  } empiezan inicializadas en 0
- las variables almacenan números naturales
- 3 instrucciones (cada una puede o no estar etiquetada):
  1. $V \leftarrow V + 1$
     - la variable V se incrementa en 1
  2. $V \leftarrow V - 1$
     - V se decrementa en 1 si antes era > 0; si no queda en 0
     - es el $V \dot{-} 1$ que ya vimos
  3. IF $V \neq 0$ GOTO A
     - condicional muy primitivo
     - A es una etiqueta que denota una instrucción del programa
     - si el valor de V es distinto de 0, la ejecución sigue con la primera instrucción que tenga etiqueta A
     - si el valor de V es 0, sigue con la próxima instrucción
- programa = sucesión finita de instrucciones

Ejemplo 1

Programa P

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

Ejecución para entrada $X = 3$:

X	Y
3	0
2	1
1	2
0	3

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

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

- siempre deja la variable X en 0

Ejemplo 2

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

- 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

$$
\begin{array}{l}
V \leftarrow V + 1 \\
\text{IF } V \neq 0 \text{ GOTO } L
\end{array}
$$

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

Macros

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

    GOTO L

    en un programa P, lo reemplazamos con

    $$V \leftarrow V + 1$$
    $$\text{IF } V \neq 0 \text{ GOTO } L$$

    donde 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 $\mathcal{S}$, 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

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

Asignación de variables: $Y \leftarrow X$

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

- el primer ciclo copia el valor de X en Y y en Z; deja X en cero
- 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

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
- cambiar A, B, C, D por etiquetas que no aparezcan en P

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 $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$
g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{si } x_1 \geq x_2 \\ \uparrow & \text{si no} \end{cases}
$$

- 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 $j = \text{mín}\{k : k\text{-ésima instrucción de } P \text{ tiene etiqueta } L\}$
   - 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 $i \in \{1, 2, \ldots, k-1\}$
- 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 d_k.

  - en particular, $\Psi_P^{(m)}(r_1, \ldots, r_m)$ está definido (not. $\Psi_P^{(m)}(r_1, \ldots, r_m)\downarrow$)
- 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 $\Psi_P^{(m)}(r_1, \ldots, r_m)$ está indefinido (not. $\Psi_P^{(m)}(r_1, \ldots, r_m)\uparrow$)

Funciones computables

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

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

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 $\mathcal{S}$-computable (o simplemente computable) si es parcial 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 $X_{n+1}, \ldots, X_m$

Minimización no acotada

Recordar la definición de minimización acotada:

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

Definimos la minimización no acotada

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

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 parcial computable.

Demostración.

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

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

□

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.

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. □

Clausura por recursión primitiva

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:

$$Y \leftarrow k \quad \text{(es una macro, se puede hacer fácil)}$$

$$[A] \quad \text{IF } X = 0 \text{ GOTO } E \quad \text{(otra macro, condición del IF por } =)$$

$$Y \leftarrow g(Z, Y)$$

$$Z \leftarrow Z + 1$$

$$X \leftarrow X - 1$$

$$\text{GOTO } A$$

Si g es total entonces h es total. □

Las funciones computables forman una clase PRC

Teorema

La clase de funciones computables es una clase PRC.

Demostración.

Ya vimos que la clase de funciones computables está cerrada por composición (p. 19) y recursión primitiva (p. 20). Veamos que las funciones iniciales son computables:

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

$$Y \leftarrow X + 1$$

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

$$Y \leftarrow X_i$$

□

Corolario

Toda función primitiva recursiva es computable.