Resueltos Lógica y Computabilidad

Ignacio E. Losiggio

February 15, 2019

1 Práctica 2 — Funciones $\mathcal{S}$–computables

1.1

1.1.1 Definir *macros* para las siguientes pseudo-instrucciones (con su interpretación natural) e indicar en cada caso qué etiquetas se asumen "frescas"

- $V_i \leftarrow k$

$$
\begin{array}{ll}
[R] & V_i \leftarrow V_i - 1 \\
 & \text{IF } V_i \neq 0 \text{ GOTO } R \\
 & V_i \leftarrow V_i + 1 \\
 & \quad \vdots \qquad k \text{ veces} \\
 & V_i \leftarrow V_i + 1
\end{array}
$$

Se toma sólo la etiqueta R cómo fresca.

- $V_i \leftarrow V_j + k$

$$
\begin{array}{ll}
 & V_i \leftarrow k \\
 & Z_a \leftarrow Z_a + 1 \\
 & \text{IF } Z_a \neq 0 \text{ GOTO } C \\
[S] & V_j \leftarrow V_j - 1 \\
 & V_i \leftarrow V_i + 1 \\
 & Z_a \leftarrow Z_a + 1 \\
[C] & \text{IF } V_j \neq 0 \text{ GOTO } S \\
 & \text{IF } Z_a \neq 0 \text{ GOTO } F \\
[L] & V_j \leftarrow V_j + 1 \\
[F] & Z_a \leftarrow Z_a - 1 \\
 & \text{IF } Z_a \neq 0 \text{ GOTO } L
\end{array}
$$

Se toman las etiquetas S, C, L, F y la variable Z_a cómo frescas.

- IF $V_i = 0$ GOTO L

$$\begin{array}{ll} & \text{IF } V_i \neq 0 \text{ GOTO } C \\ & Z_a \leftarrow Z_a + 1 \\ & \text{IF } Z_a \neq 0 \text{ GOTO } L \\ [C] & Z_a \leftarrow Z_a + 1 \end{array}$$

Se toman la etiqueta C y la variable Z_a cómo frescas.

- GOTO L

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

Se toma sólo la variable Z_a cómo fresca.

1.1.2 Definir dos pseudo-programas distintos en el lenguaje $\mathcal{S}$ (usando las macros convenientes del punto anterior) que computen la función de dos variables $f(x_1, x_y) = x_1 + x_2$. Par aalguno de los dos, expandir las macros utilizadas prestando atención a la instanciación de variables y etiquetas frescas.

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

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

Vamos a expandir la segunda de las formulaciones (por ser la que tiene macros más simples).

$[A]$ IF $X_1 \neq 0$ GOTO Y
$Z_1 \leftarrow Z_1 + 1$
IF $Z_1 \neq 0$ GOTO B
$[Y]$ $Z_1 \leftarrow Z_1 + 1$
$Y \leftarrow Y + 1$
$X_1 \leftarrow X_1 - 1$
$Z_2 \leftarrow Z_2 + 1$
IF $Z_2 \neq 0$ GOTO A
$[B]$ IF $X_2 \neq 0$ GOTO Z
$Z_3 \leftarrow Z_3 + 1$
IF $Z_3 \neq 0$ GOTO E
$[Z]$ $Z_3 \leftarrow Z_3 + 1$
$Y \leftarrow Y + 1$
$X_2 \leftarrow X_2 - 1$
$Z_4 \leftarrow Z_4 + 1$
IF $Z_4 \neq 0$ GOTO B

1.1.3 Sea P el programa en $\mathcal{S}$ que resulta de expandir todas las macros en aguno de los códigos del punto anterior. Determinar cuál es la función computada en cada caso:

- $\Psi_P^{(1)} : \mathbb{N} \to \mathbb{N}$

  $f(x) = x$, se puede ver fácil desde planteo del ejercicio anterior, los parámetros no inicializados son ceros por lo que la función pedida se instancia como $f(x_1, 0) = x + 0$ y se transforma nuestra suma en la función identidad.

- $\Psi_P^{(2)} : \mathbb{N}^2 \to \mathbb{N}$

  $f(x, y) = x + y$, la función pedida el ejercicio anterior.

- $\Psi_P^{(3)} : \mathbb{N}^3 \to \mathbb{N}$

  $f(x, y, z) = x + y$, dado que ignoramos el tercer parámetro en ambas formulaciones del programa.

1.2

1.2.1 Sea $\mathcal{C}_{\mathcal{S}} = \{\Psi_P^{(n)} \mid P \text{ es un programa en } S, n \geq 1\}$ la clase de funciones $\mathcal{S}$-parciales computables. Mostrar que $\mathcal{C}_{\mathcal{S}}$ es una clase PRC

Para mostrar que es PRC necesito dar un programa para las iniciales y demostrar que $\mathcal{C}_{\mathcal{S}}$ está cerrado por composición y recursión primitiva.

Vamos primero por las iniciales:

$$\begin{array}{lll} n(x) = 0 & s(x) = x + 1 & u_i^n(x_1, \ldots, x_n) = x_i \\ Z_1 \leftarrow Z_1 + 1 & Y \leftarrow X_1 + 1 & Y \leftarrow X_i + 0 \end{array}$$

Y ahora veamos cómo resolver la composición, tomo $h : \mathbb{N}^k \to \mathbb{N}$ y $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$ que pertenezcan a $\mathcal{C}_{\mathcal{S}}$ (y por lo tanto tengan programas que podamos usar cómo macros):

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

$$\begin{array}{l} Z_1 \leftarrow g_1(X_1, \ldots, X_n) \\ \quad\vdots \\ Z_k \leftarrow g_k(X_1, \ldots, X_n) \\ Y \leftarrow h(Z_1, \ldots, Z_k) \end{array}$$

Para que esto funcione tenemos que restringir a cada macro a dejar sus variables de entrada intactas al finalizar su ejecución. Una forma de mecanizarlo es que cada macro copie todas sus variables de entrada a variables temporales "frescas" (que no se hayan usado ni se vayan a usar) y que cada macro designe como variable de salida una variable temporal "fresca". La única excepción a esto es el macro de asignación $V_i \leftarrow \mathit{Expr}$ que aunque el resultado de Expr esté en una variable "fresca" debe modificar V_i para cumplir con su tarea.

Dicho todo esto, vamos por la recursión primitiva, la cuál es más simple de lo que parece, tomo $h : \mathbb{N}^n \to \mathbb{N}$ y $g : \mathbb{N}^{n+2} \to \mathbb{N}$ que pertenezcan a $\mathcal{C}_{\mathcal{S}}$ (¡Osea que tenemos programas!):

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

$Y \leftarrow h(X_1, \ldots, X_n)$
$[L]$ IF $X_{n+1} = 0$ GOTO E
$Z_1 \leftarrow Z_1 + 1$
$X_{n+1} \leftarrow X_{n+1} - 1$
$Y \leftarrow g(Y, X_1, \ldots, X_n, Z_1)$
GOTO L

1.2.2 Demostrar (sin definir un programa en $\mathcal{S}$) que la función $* : \mathbb{N}^2 \rightarrow \mathbb{N}$ definida por $*(x, y) = x \cdot y$ es $\mathcal{S}$-computable.

En la práctica anterior la demostramos primitiva recursiva y sabemos que $pr \subseteq PRC$ para cualquier conjunto PRC. En el punto anterior demostramos que $\mathcal{C}_{\mathcal{S}}$ era PRC, por lo que todas las primitivas recursivas están allí, entre ellas $*$.

1.2.3 Si $f : \mathbb{N}^n \rightarrow \mathbb{N}$ es una función primitiva recursiva ¿Qué podemos decir acerca de la existencia de un programa en el lenguaje $\mathcal{S}$ que la compute?

Que existe (por las mismas razones del punto anterior).

1.3 Decimos que un programa P es *autocontenido* si en cada instrucción IF $V \neq 0$ GOTO L que ocurre en P, L es una etiqueta definida en P.

1.3.1 Demostrar que todo programa P tiene un programa autocontenido P' equivalente (P y P' son programs equivalentes si $\Psi_P^{(n)} = \Psi_{P'}^{(n)} \forall n \geq 1$).

Supongamos P un programa no autocontenido con una cantidad k de etiquetas "libres" $L_1, \ldots, L_k$ (etiquetas en uso pero sin lugar hacia dónde saltar). Cómo el programa P es finito usa una cantida finita de variables temporales $Z_1, \ldots, Z_n$ y tiene una cantidad finita de instrucciones podemos construir P' agregándole las siguientes instrucciones al final de P:

$[L_1]$ $Z_{n+1} \leftarrow Z_{n+1} + 1$
$\vdots$
$[L_k]$ $Z_{n+1} \leftarrow Z_{n+1} + 1$

Necesitamos sólo una variable "fresca" y cómo nunca modificamos a Y el resultado del programa P no se modifica.

1.3.2 Sean P y Q dos programas autocontenidos con etiquetas disjuntas y sea $r : \mathbb{N}^n \to \{0,1\}$ un predicado primitivo recursivo. Definir macros para las siguientes pseudo-instrucciones (con su interpretación natural)

- IF $r(V_1, \ldots, V_n)$ GOTO L

  $Z_a \leftarrow r(V_1, \ldots, V_n)$
  IF $Z_a \neq 0$ GOTO L

- IF $r(V_1, \ldots, V_n)$ THEN P ELSE Q

  IF $r(V_1, \ldots, V_n)$ GOTO P
  ; Acá van los contenidos de Q
  GOTO E
  $[P]$; Acá van los contenidos de P

- WHILE $r(V_1, \ldots, V_n)$ P

  $[L]$ IF $\alpha(r(V_1, \ldots, V_n))$ GOTO E
  ; Acá van los contenidos de P
  GOTO L

1.3.3 Dadas las funciones $f, g : \mathbb{N} \to \mathbb{N}$ definidas por:

$$f(x) = \begin{cases} 1 & \text{si } x = 3 \\ \uparrow & \text{en otro caso} \end{cases} \qquad\qquad g(x) = 2x$$

Demostrar que es $\mathcal{S}$-parcial computable la función

$$h(x) = \begin{cases} f(x) & \text{si } x \geq 5 \lor x = 3 \\ g(x) & \text{en otro caso} \end{cases}$$

Tengo que las operaciones $a = b$, $a \geq b$, $a \lor b$ y $a \cdot b$ son p.r., por lo que sé que existen programas en $\mathcal{S}$ que las computan. También tengo que la división por casos en una clase PRC genera una función dentro de la misma clase. Dado todo esto ya puedo asegurar que $g(x)$ es $\mathcal{S}$-parcial computable y que si $f(x)$ lo fuera entonces $h(x)$ también.

Me queda demostrar que en $\mathcal{S}$ los programas (a veces) se cuelgan. Defino $j(x) = \uparrow \forall x$ y reescribo $f(x)$ cómo una composición con $j(x)$.

$$f(x) = \begin{cases} 1 & \text{si } x = 3 \\ j(x) & \text{en otro caso} \end{cases}$$

Y sólo me queda ofrecer el código de $j(x)$:

$[A]$ GOTO A