Teoremas del parámetro y de la recursión

Lógica y Computabilidad

Franco Frizzo

13 de septiembre de 2017

1. Teorema del parámetro

Ejercicio 1

(a) Sea $h : \mathbb{N} \to \mathbb{N}$ una función parcialmente computable.

i. Demostrar que, para todo $u \in \mathbb{N}$ fijo, existe P_u tal que

$$\Psi^{(1)}_{P_u}(x) = h(\Phi^{(1)}_u(x)).$$

ii. Demostrar que existe $f : \mathbb{N} \to \mathbb{N}$ primitiva recursiva tal que, para todo $u \in \mathbb{N}$,

$$\Phi^{(1)}_{f(u)}(x) = h(\Phi^{(1)}_u(x)).$$

(b) Demostrar que existe $f' : \mathbb{N}^2 \to \mathbb{N}$ primitiva recursiva tal que, para todos $u, v \in \mathbb{N}$,

$$\Phi^{(1)}_{f'(u,v)}(x) = \left(\Phi^{(1)}_v \circ \Phi^{(1)}_u\right)(x).$$

Resolución

(a) i. Esta parte es sencilla de resolver; dado un programa fijo cuyo número es u, queremos construir otro programa que le aplique la función h a cada una de sus salidas. Como sabemos que tanto el intérprete de programas como h son funciones parcialmente computables, asumiremos que contamos con macros para computar ambos. Así, podemos obtener el siguiente programa, que cumple con lo pedido.

$$\begin{aligned} P_u\text{:} \qquad & Y \leftarrow \Phi^{(1)}_u(X_1) \\ & Y \leftarrow h(Y) \end{aligned}$$

ii. Observando con atención lo que pide el enunciado podemos notar que, al parecer, lo que debe hacer la función f es "generar", para cada entrada u, el número del programa P_u correspondiente. Así, un primer intento de definir f podría ser el siguiente:

$$f(u) = \#(P_u).$$

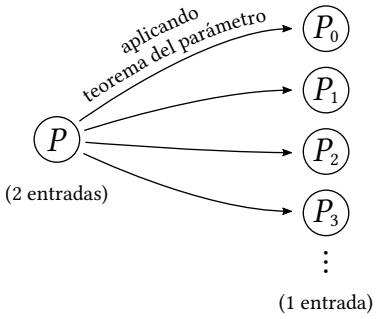
El problema con esta definición es que no tenemos forma de saber si f efectivamente es primitiva recursiva. Así que tendremos que ser un poco más astutos.

El hecho de que nos pidan hallar una función primitiva recursiva cuya salida se va a usar como número de programa sugiere que el teorema del parámetro podría ser útil. Recordemos que este

teorema nos garantiza que, de manera primitiva recursiva, podemos transformar un programa en otro donde algunas de las entradas hayan sido fijadas en valores abritrarios.

El primer paso para resolver este ejercicio será construir un programa P que generalice el comportamiento de todos los P_u. P recibirá una segunda entrada, a través de la cual le indicaremos el valor de u deseado; por lo demás, su comportamiento será idéntico al del P_u correspondiente.

A partir de este programa P definiremos nuestra función f: su tarea será recibir como argumento un valor de u y, mediante el teorema del parámetro, fijar en este valor la segunda entrada de P, obteniendo así un programa equivalente al P_u deseado.



Para construir el programa P, basta con tomar P_u y reemplazar todas las apariciones de u por la variable X_2.

$$\begin{aligned} P\colon \quad & Y \leftarrow \Phi^{(1)}_{X_2}(X_1) \\ & Y \leftarrow h(Y) \end{aligned}$$

De esta forma,

$$\Psi^{(2)}_P(x, u) = \Psi^{(1)}_{P_u}(x) = h(\Phi^{(1)}_u(x)),$$

o, tomando $e = \#(P)$,

$$\Phi^{(2)}_e(x, u) = h(\Phi^{(1)}_u(x)).$$

Por el teorema del parámetro, sabemos que existe una función primitiva recursiva $S : \mathbb{N}^2 \to \mathbb{N}$ tal que

$$\Phi^{(1)}_{S(u,e)}(x) = \Phi^{(2)}_e(x, u).$$

Vale la pena fijarse, una vez más, en lo que hace esta función S: a partir de un valor u y un programa con número e, genera un *nuevo programa* que, al ser ejecutado con x como única entrada, se comporta exactamente igual que e cuando es ejecutado con las dos entradas x y u.

En nuestro caso, ¡$S(u, e)$ ya es el programa que necesitamos! Basta con definir

$$f(u) = S(u, e)$$

(que claramente es primitiva recursiva, porque S lo es y e está fijo) para obtener

$$\Phi^{(1)}_{f(u)}(x) = \Phi^{(1)}_{S(u,e)}(x) = \Phi^{(2)}_e(x, u) = h(\Phi^{(1)}_u(x)).$$

Nota: Un detalle a considerar es que, definida de esta forma, no es necesariamente cierto que $f(u) = \#(P_u)$, porque no estamos teniendo en cuenta *qué tipo* de transformación realiza S sobre el programa con número e. Sin embargo, sí sabemos (y con eso es suficiente) que son programas equivalentes, es decir, que $\Phi^{(1)}_{f(u)} = \Psi^{(1)}_{P_u}$.

(b) Al igual que en el caso anterior, si u y v están fijos, es sencillo construir un programa de una entrada que componga las funciones computadas por ambos (basta con correrlos uno detrás del otro). La idea será, de nuevo, ver a u y v como argumentos de un programa más general, de tres entradas, que podemos definir como sigue.

$$Q: \quad Y \leftarrow \Phi^{(1)}_{X_2}(X_1)$$
$$Y \leftarrow \Phi^{(1)}_{X_3}(Y)$$

No es difícil verificar que

$$\Psi^{(3)}_Q(x, u, v) = \Phi^{(1)}_v\left(\Phi^{(1)}_u(x)\right) = \left(\Phi^{(1)}_v \circ \Phi^{(1)}_u\right)(x).$$

Esta vez, nos interesa fijar dos de los argumentos de $\Psi^{(3)}_Q$ (u y v), en lugar de solo uno. Esto quiere decir que tendremos que utilizar la versión más general del teorema del parámetro. En particular, nos servirá saber que existe una función $S^2_1 : \mathbb{N}^3 \to \mathbb{N}$ tal que

$$\Phi^{(1)}_{S^2_1(u,v,d)}(x) = \Phi^{(3)}_d(x, u, v),$$

donde $d = \#(Q)$.

Definiendo, por último

$$f'(u, v) = S^2_1(u, v, d)$$

tenemos que la función primitiva recursiva f' cumple con lo pedido, ya que

$$\Phi^{(1)}_{f'(u,v)}(x) = \Phi^{(1)}_{S^2_1(u,v,d)}(x) = \Phi^{(3)}_d(x, u, v) = \left(\Phi^{(1)}_v \circ \Phi^{(1)}_u\right)(x).$$

Nota: En adelante, y para simplificar la notación, llamaremos indistintamente S a todas las funciones S^n_m del teorema del parámetro, siempre que la aridad de las mismas pueda inferirse claramente del contexto.

Ejercicio 2

Demostrar que la función

$$f_1(x) = \begin{cases} 1 & \text{si } \Phi^{(1)}_x(74) \downarrow \\ 0 & \text{en otro caso} \end{cases}$$

no es computable, usando el teorema del parámetro para reducir la función no computable

$$g_1(x, y) = \begin{cases} 1 & \text{si } \Phi^{(1)}_x(y) \downarrow \text{ y } \Phi^{(1)}_x(y) \text{ es par} \\ 0 & \text{en otro caso.} \end{cases}$$

Resolución

Recordemos de qué se trataban las reducciones: la idea es hacer una demostración por el absurdo mostrando que, si f_1 fuera computable, podríamos también computar g_1, que ya mostramos no computable en la clase anterior.[1]

Es decir, queremos computar g_1 "usando" f_1. El problema es que ¡f_1 nos da menos información que g_1! Mientras que g_1 habla de la paridad de la salida de un programa para una entrada cualquiera, f_1 solo es capaz de decirnos si el programa termina cuando la entrada es 74. Incluso sin tener en cuenta esto, todo indica que, si queremos usar f_1 para computar g_1, tendremos que encontrar una forma de pasar los dos argumentos que recibe g_1 a través del único parámetro de f_1.

[1] En realidad probamos la no computabilidad de una variante que tenía intercambiados el 0 y el 1, pero de allí se sigue inmediatamente que g_1 tampoco puede ser computable.

Como vamos a trabajar con dos funciones, usar los mismos nombres para los argumentos de ambas puede resultar muy confuso. Por eso, en adelante, para los argumentos de g_1 (la función a reducir) vamos a usar las letras u y v, en lugar de x e y.

Supongamos, entonces, que f_1 es computable; y dados ciertos $u, v \in \mathbb{N}$, intentemos encontrar una forma de computar $g_1(u, v)$. Por ahora, para empezar a encaminar la solución, consideraremos que u y v están fijos. Si bien, como dijimos antes, f_1 recibe un solo argumento, tenemos la ventaja de que lo interpreta como un número de programa. Esto nos da la posibilidad de construir un programa a medida de nuestras necesidades, y luego pasarle su número a f_1. En este caso:

- para un programa cualquiera, podemos saber si termina o no cuando su entrada es 74, y
- nos interesa averiguar si $\Phi_u^{(1)}(v)$ está definido y es par.

Por lo tanto, deberíamos encontrar un programa que termine para la entrada 74 exactamente cuando $\Phi_u^{(1)}(v)$ esté definido y sea un número par. Dicho de otro modo, buscamos un programa cuyo número sea, digamos, $e_{u,v}$, y que cumpla

$$\Phi_{e_{u,v}}^{(1)}(74) \downarrow \quad \Leftrightarrow \quad \Phi_u^{(1)}(v) \downarrow \text{ y } \Phi_u^{(1)}(v) \text{ es par.}$$

Por ejemplo, podemos definir $e_{u,v}$ de forma tal que

$$\Phi_{e_{u,v}}^{(1)}(x) = \begin{cases} 1 & \text{si } \Phi_u^{(1)}(v) \downarrow \text{ y } \Phi_u^{(1)}(v) \text{ es par} \\ \uparrow & \text{en otro caso.} \end{cases}$$

No estamos dando explícitamente el programa, pero resulta claro que la función recién definida es parcialmente computable, y por lo tanto, que $e_{u,v}$ efectivamente existe (este detalle es *fundamental* para que la demostración sea válida).

Como el programa con número $e_{u,v}$ ignora por completo su entrada, tomando el caso particular $x = 74$, vemos que efectivamente cumple con la propiedad que buscábamos. De allí podemos deducir que, para todos $u, v \in \mathbb{N}$,

$$g_1(u, v) = f_1(e_{u,v}).$$

¡Casi! Si bien pudimos "computar" g_1 a partir de f_1, está faltando tener en cuenta algo esencial: la reducción tiene que poder hacerse de forma computable. En este caso, hay un paso que no está claro que sea computable: obtener $e_{u,v}$ a partir de los valores de u y v. Afortunadamente, contamos con una herramienta muy potente a la hora de generar programas de manera computable: el teorema del parámetro.

La idea es, igual que en el ejercicio anterior, definir un programa e "más general" que cada uno de los $e_{u,v}$, que reciba los valores de u y v como entradas adicionales. Luego, podremos usar el teorema del parámetro para fijar los valores de dos estas entradas según nos convenga.

Sea, entonces, e tal que

$$\Phi_e^{(3)}(x, u, v) = \begin{cases} 1 & \text{si } \Phi_u^{(1)}(v) \downarrow \text{ y } \Phi_u^{(1)}(v) \text{ es par} \\ \uparrow & \text{en otro caso.} \end{cases}$$

Por el teorema del parámetro, existe S primitiva recursiva que cumple

$$\Phi_{S(u,v,e)}^{(1)}(x) = \Phi_e^{(3)}(x, u, v).$$

Veamos que podemos escribir $g_1(u, v)$ como $f_1(S(u, v, e))$.

$$\begin{aligned}
f_1(S(u,v,e)) &= \begin{cases} 1 & \text{si } \Phi^{(1)}_{S(u,v,e)}(74) \downarrow \\ 0 & \text{en otro caso} \end{cases} && \text{(por definición de } f_1) \\
&= \begin{cases} 1 & \text{si } \Phi^{(3)}_{e}(74,u,v) \downarrow \\ 0 & \text{en otro caso} \end{cases} && \text{(por t. del parámetro)} \\
&= \begin{cases} 1 & \text{si } \Phi^{(1)}_{u}(v) \downarrow \text{ y } \Phi^{(1)}_{u}(v) \text{ es par} \\ 0 & \text{en otro caso} \end{cases} && \text{(por definición de } e) \\
&= g_1(u,v) && \text{(por definición de } g_1)
\end{aligned}$$

Acabamos de escribir a g_1 como el resultado de componer dos funciones computables, f_1 y S; por lo tanto, g_1 también es computable. Esto es un absurdo que se originó a partir de nuestra suposición inicial: que f_1 es computable. Así, queda demostrado que f_1 no puede ser computable.

Ejercicio 3

Demostrar que la siguiente función no es computable.

$$f_2(x) = \begin{cases} 1 & \text{si } \Phi^{(1)}_{x} \text{ es la función identidad} \\ 0 & \text{en otro caso} \end{cases}$$

Resolución

Como esta vez no nos indican desde qué función hacer la reducción, intentaremos hacerla desde la función no computable por excelencia: el *halting problem*. Es decir, supondremos que f_2 es computable, e intentaremos computar a partir de ella la función no computable

$$g_2(u) = \begin{cases} 1 & \text{si } \Phi^{(1)}_{u}(u) \downarrow \\ 0 & \text{en otro caso.} \end{cases}$$

De nuevo, comencemos por considerar un valor de $u \in \mathbb{N}$ e intentemos constuir un programa a medida, con número e_u, que podamos usar como entrada de f_2. Esta vez, nos será útil que e_u compute la función identidad si y solo sí $\Phi^{(1)}_{u}(u)$ está definido. Supongamos, entonces, que f_2 es computable, y consideremos e_u tal que

$$\Phi^{(1)}_{e_u}(x) = \begin{cases} x & \text{si } \Phi^{(1)}_{u}(u) \downarrow \\ \uparrow & \text{en otro caso.} \end{cases}$$

No es difícil verificar que la función recién presentada es parcialmente computable, es decir, que e_u está bien definido. Dado cualquier número $u \in \mathbb{N}$, se cumple que

$$g_2(u) = f_2(e_u).$$

Usando el teorema del parámetro, veremos que para cada valor de u podemos generar el e_u correspondiente (o un programa equivalente) de manera computable. Para hacerlo, definimos un programa e "más general", de dos entradas, que acepte como segundo argumento el valor de u.

$$\Phi^{(2)}_{e}(x, u) = \begin{cases} x & \text{si } \Phi^{(1)}_{u}(u) \downarrow \\ \uparrow & \text{en otro caso.} \end{cases}$$

Por el teorema del parámetro, existe S primitiva recursiva que cumple

$$\Phi^{(1)}_{S(u,e)}(x) = \Phi^{(2)}_e(x, u).$$

Veamos que podemos escribir $g_2(u)$ como $f_2(S(u, e))$.

$$\begin{aligned}
f_2(S(u,e)) &= \begin{cases} 1 & \text{si } \Phi^{(1)}_{S(u,e)}(x) = x \text{ para todo } x \\ 0 & \text{en otro caso} \end{cases} && \text{(por definición de } f_2) \\
&= \begin{cases} 1 & \text{si } \Phi^{(2)}_e(x,u) = x \text{ para todo } x \\ 0 & \text{en otro caso} \end{cases} && \text{(por t. del parámetro)} \\
&= \begin{cases} 1 & \text{si } \Phi^{(1)}_u(u) \downarrow \\ 0 & \text{en otro caso} \end{cases} && \text{(por definición de } e) \\
&= g_2(u) && \text{(por definición de } g_2)
\end{aligned}$$

Dado que podemos escribir a g_2 como la composición de dos funciones computables, resulta que g_2 también lo es. Este absurdo demuestra que nuestra suposición de que f_2 es computable debe ser falsa.

2. Teorema de la recursión

Ejercicio 4

Demostrar que la siguiente función no es computable.

$$f_3(x) = \begin{cases} 1 & \text{si } \Phi^{(1)}_x \text{ tiene algún punto fijo} \\ 0 & \text{en otro caso} \end{cases}$$

Resolución

Para resolver este ejercicio, usaremos el teorema de la recursión, que nos dice (más formalmente) que existen programas que conocen su propio número de programa, y pueden hacer con él "cualquier cosa" parcialmente computable. La demostración será por el absurdo: suponiendo que f_3 es computable, definiremos un programa "rebelde" que compute f_3 para su propio número de programa y luego se comporte de forma opuesta a lo que indica el resultado.

Supongamos que f_3 es computable. Nuestro objetivo será encontrar un programa, con número e, de forma tal que:

- si $f_3(e) = 1 \quad \Rightarrow \quad \Phi^{(1)}_e$ no tiene ningún punto fijo.
- si $f_3(e) = 0 \quad \Rightarrow \quad \Phi^{(1)}_e$ tiene algún punto fijo.

Definir este programa es el único paso "creativo" de la demostración. Podemos, por ejemplo, intentar definir e de forma tal que

$$\Phi^{(1)}_e(x) = \begin{cases} x+1 & \text{si } f_3(e) = 1 \\ x & \text{en otro caso.} \end{cases}$$

Así, si $f_3(e) = 1$, ningún valor será un punto fijo de $\Phi^{(1)}_e$, mientras que por el contrario, si $f_3(e) = 0$, todos los números naturales serán puntos fijos.

El problema de esta definición es que **no** tenemos garantizada la existencia de este e. Si tratamos de pensar cómo sería un programa que se comporte de esa forma, todo parece indicar que en algún momento el programa debería operar con su propio número. Por ejemplo, podríamos comenzar…

$$\begin{aligned} P\text{:} \qquad & Z_1 \leftarrow \#(P) \\ & \vdots \end{aligned}$$

¡No tenemos una macro que nos permita hacer esto! Vale la pena dedicarle unos segundos a pensar lo complicado que sería intentar hacerlo "a mano".

Ahora es cuando el teorema de la recursión viene en nuestra ayuda. La idea es definir una función que, además de las entradas que ya recibe el programa que queremos definir, tenga un argumento adicional, por donde vendrá el número del programa que computará la función. Modificando ligeramente la definición anterior, podemos obtener

$$h(x, u) = \begin{cases} x + 1 & \text{si } f_3(u) = 1 \\ x & \text{en otro caso.} \end{cases}$$

Esta función sí es claramente computable (recordemos que supusimos que f_3 lo es). Aplicando el teorema de la recursión, sabemos que existe $e \in \mathbb{N}$ que computa h con su propio número como última entrada; es decir,

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

Veamos, por casos, qué pasa si computamos $f_3(e)$.

- Si $f_3(e) = 1$, por la definición de f_3, debe ser que $\Phi_e^{(1)}$ tiene algún punto fijo.

  Pero por otra parte,

  $$\begin{aligned} \text{si } f_3(e) = 1 \quad &\Rightarrow \quad h(x, e) = x + 1 \quad \text{para todo } x && \text{(por definición de } h\text{)} \\ &\Rightarrow \quad \Phi_e^{(1)}(x) = x + 1 \quad \text{para todo } x && \text{(por t. de la recursión)} \\ &\Rightarrow \quad \Phi_e^{(1)} \text{ no tiene ningún punto fijo.} \end{aligned}$$

- Si, en cambio, $f_3(e) = 0$, de la definición de f_3 se sigue que $\Phi_e^{(1)}$ no tiene ningún punto fijo.

  Pero al mismo tiempo,

  $$\begin{aligned} \text{si } f_3(e) = 0 \quad &\Rightarrow \quad h(x, e) = x \quad \text{para todo } x && \text{(por definición de } h\text{)} \\ &\Rightarrow \quad \Phi_e^{(1)}(x) = x \quad \text{para todo } x && \text{(por t. de la recursión)} \\ &\Rightarrow \quad \text{todo } x \in \mathbb{N} \text{ es punto fijo de } \Phi_e^{(1)}. \end{aligned}$$

Esta situación es un absurdo que provino de suponer que f_3 es computable. Así, queda demostrado que la función f_3 no es computable.

Ejercicio 5

Demostrar que la siguiente función no es computable.

$$f_4(x, y) = \begin{cases} 1 & \text{si } x \in \operatorname{Dom} \Phi_y^{(1)} \cap \operatorname{Im} \Phi_y^{(1)} \\ 0 & \text{en otro caso} \end{cases}$$

Resolución

Se trata, de nuevo, de encontrar un programa "rebelde", que haga lo contrario de lo que indica f_4. En este caso, f_4 toma dos argumentos, pero solo el segundo es mirado como número de programa. Por lo tanto, nos concentraremos en el segundo argumento, y dejaremos libre el primero.

Intentaremos encontrar un programa, con número e, tal que:

- si $f_4(x, e) = 1 \quad \Rightarrow \quad x \notin \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}$.
- si $f_4(x, e) = 0 \quad \Rightarrow \quad x \in \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}$.

En particular, para el primer caso, basta con que $x \notin \text{Dom}\, \Phi_e^{(1)}$, mientras que para el segundo, es suficiente que $\Phi_e^{(1)}(x) = x$. Luego, podríamos buscar e de modo tal que se cumpla

$$\Phi_e^{(1)}(x) = \begin{cases} \uparrow & \text{si } f_3(x, e) = 1 \\ x & \text{en otro caso.} \end{cases}$$

El teorema de la recursión nos permite demostrar la existencia de tal programa. Para eso, definimos la siguiente función parcialmente computable:

$$h(x, u) = \begin{cases} \uparrow & \text{si } f_4(x, u) = 1 \\ x & \text{en otro caso.} \end{cases}$$

Por el teorema de la recursión, existe $e \in \mathbb{N}$ tal que

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

Analicemos lo que pasa al computar $f_4(x, e)$, para este e y cualquier $x \in \mathbb{N}$. Hay dos casos posibles.

- Si $f_4(x, e) = 1$, por definición de f_4, vale que $x \in \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}$.

  Pero, por otra parte,

$$\begin{aligned} \text{si } f_4(x, e) = 1 \quad &\Rightarrow \quad h(x, e) \uparrow && \text{(por definición de } h\text{)} \\ &\Rightarrow \quad \Phi_e^{(1)}(x) \uparrow && \text{(por t. de la recursión)} \\ &\Rightarrow \quad x \notin \text{Dom}\, \Phi_e^{(1)} \\ &\Rightarrow \quad x \notin \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}. \end{aligned}$$

- Si, por el contrario, $f_4(x, e) = 0$, por definición de f_4, vale que $x \notin \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}$.

  Sin embargo,

$$\begin{aligned} \text{si } f_4(e, x) = 0 \quad &\Rightarrow \quad h(x, e) = x && \text{(por definición de } h\text{)} \\ &\Rightarrow \quad \Phi_e^{(1)}(x) = x && \text{(por t. de la recursión)} \\ &\Rightarrow \quad x \in \text{Dom}\, \Phi_e^{(1)} \cap \text{Im}\, \Phi_e^{(1)}. \end{aligned}$$

Esto es un absurdo, que demuestra la falsedad de nuestra suposición inicial. Es decir, hemos demostrado que f_4 no es computable.