Clase práctica 3 (computabilidad)
Lógica y Computabilidad

Facundo Carreiro

1. Funciones Primitivas Recursivas

Vamos a definir varias funciones y probar que son p.r., podemos suponer definidas y p.r. todas las funciones vistas en las teóricas y probadas en las clases prácticas anteriores

1.1. Ejercicio 1

Enunciado: Probar que la función

$$\begin{aligned} f(0,x) &= g(x) \\ f(n+1,x) &= f(n,f(n,x)) \end{aligned}$$

es p.r. si g es p.r.

Resolución: Podemos ver que f, definida como está no cumple con el esquema de recursión primitiva. Tampoco queda claro *qué* es lo que la función realiza así que primero vamos a probar algunos pasos particulares para ver su funcionamiento:

$$\begin{aligned} f(0,x) &= g(x) \\ f(0+1,x) &= f(0,f(0,x)) = g(g(x)) = g^2(x) \\ f(1+1,x) &= f(1,f(1,x)) = f(1,g(g(x))) = g(g(g(g(x)))) = g^4(x) \\ &\vdots \end{aligned}$$

pareciera que el comportamiento es $f(n,x) = g^{2^n}(x)$, debemos probarlo por inducción (sobre n).

Caso base:

$$f(0,n) = g(x) = g^1(x) = g^{2^0}(x)$$

Caso inductivo: Suponiendo $f(n,x) = g^{2^n}(x)$ queremos ver que vale $f(n+1,x) = g^{2^{(n+1)}}(x)$

$$f(n+1,x) = f(n,f(n,x))$$

usando la hipótesis inductiva sobre el término de más adentro nos queda

$$f(n+1,x) = f(n,g^{2^n}(x))$$

aplicando hipótesis inductiva nuevamente queda

$$f(n+1,x) = g^{2^n}(g^{2^n}(x)) = g^{2*2^n}(x) = g^{2^{(n+1)}}(x)$$

Redefinición: Ahora que probamos que la función se comporta como esperábamos podemos definirla

$$f(n, x) = aplica_g(2^n, x)$$

dónde $aplica_g(n, x)$ es la función que aplica n veces la función g. Esta función ya fue probada primitiva recursiva y como la composición y la potencia también lo son finalmente f resulta primitiva recursiva. □

1.2. Ejercicio 2

Enunciado: Dado un k fijo y $g, g_0, \ldots, g_k$ funciones p.r. probar que el siguiente esquema de recursión es primitivo recursivo

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= g_0(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, 1) &= g_1(x_1, \ldots, x_n) \\ &\vdots \\ h(x_1, \ldots, x_n, k) &= g_k(x1, \ldots, xn) \\ h(x_1, \ldots, x_n, t+1) &= g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n) \quad \text{si } t+1 > k \end{aligned}$$

Resolución: Primero intentamos definir h respetando el esquema de recursión primitiva para luego instanciar correctamente las funciones y que h resulte equivalente a lo pedido

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= g_0(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, t+1) &= g'(x_1, \ldots, x_n, h(x_1, \ldots, x_n, t), t) \end{aligned}$$

También vamos a necesitar una función de selección, la definimos de la siguiente manera

$$multip_k(t, y_0, \ldots, y_k) = \begin{cases} y_0 & \text{si } t = 0 \\ y_1 & \text{si } t = 1 \\ \vdots & \\ y_{k-1} & \text{si } t = k-1 \\ y_k & \text{en otro caso} \end{cases}$$

Podemos ver que $multip_k$ es p.r. ya que está definida por casos y

- Los casos son finitos (k)
- Los casos son disjuntos
- Las guardas son funciones p.r.
- Los resultados son funciones p.r. (son las respectivas proyecciones de los parámetros de entrada)
- Se usa fuertemente la composición

Ahora definimos a g' como

$$g'(x_1, \ldots, x_n, r, t) = multip_k(t, \underbrace{g_1(x_1, \ldots, x_n)}_{y_0}, \ldots, \underbrace{g_k(x_1, \ldots, x_n)}_{y_{k-1}}, \underbrace{g(x_1, \ldots, x_n, r, t)}_{y_k}))$$

Que también resulta p.r. ya que cada función utilizada es p.r. por hipótesis y la composición preserva p.r. □

Se puede ver que con esta manera de definir h si la evaluamos con un $t \leq k$ entonces la función selectora se encarga de retornar el "caso base" correspondiente. Si $t > k$ entonces se produce el llamado recursivo. También es interesante notar que, si bien este esquema pareciera tener muchos casos bases en los que se puede "caer" en realidad si evaluamos la función con un $t > k$ solamente podemos llegar al caso base g_k ya que t se decremente siempre de a uno.

1.3. Ejercicio 3

Enunciado: Dado un predicado $P : \mathbb{N}^n \to \mathbb{N}$ p.r., $g_0 : \mathbb{N}^n \to \mathbb{N}$ p.r., ver que el siguiente esquema recursivo es p.r.

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= g_0(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, t+1) &= g'(x_1, \ldots, x_n, h(x_1, \ldots, x_n, F(t)), t) \end{aligned}$$

donde

$$F(t) = \begin{cases} \max\limits_{0 \leq c \leq t} P(c) & \text{si existe} \\ 0 & \text{sino} \end{cases}$$

Este esquema se puede ver como una generalización del esquema primitivo recursivo en el que la recursión se va haciendo "de a saltos". En el llamado recursivo se cuenta con una función que puede elegir el tamaño del salto según un predicado fijo.

Resolución: Nos gustaría una función que se comportara de la siguiente manera

$$h\_en\_ft(x_1, \ldots, x_n, t) = h(x_1, \ldots, x_n, F(t)) \tag{1}$$

si h_en_ft fuera p.r. entonces h sería primitiva recursiva porque en tal caso podríamos definirla de la siguiente manera

$$h(x_1, \ldots, x_n, t) := \begin{cases} g_0(x_1, \ldots, x_n) & \text{si } t = 0 \\ g(x_1, \ldots, x_n, h\_en\_ft(x_1, \ldots, x_n, t-1), t-1) & \text{sino} \end{cases}$$

Notar que *esta no es una definición recursiva*! Estaríamos definiendo a h usando división por casos, composición, y funciones que ya vimos que son p.r. (suponiendo a h_en_ft p.r.) por lo tanto vemos que h *podría definirse* de una manera p.r. Nos falta ver si h_en_ft es p.r o no.

Observacion 1: Según la definición dada en (1) el comportamiento debería ser el siguiente

$$h\_en\_ft(x_1, \ldots, x_n, 0) = h(x_1, \ldots, x_n, F(0)) = h(x_1, \ldots, x_n, 0) = g_0(x_1, \ldots, x_n)$$

Observacion 2: Mirando la definición

$$h\_en\_ft(x_1, \ldots, x_n, t+1) = h(x_1, \ldots, x_n, F(t+1))$$

si vale $\neg P(t+1)$ (o sea si el predicado no vale en ese punto) entonces podemos concluir que la función tomará el mismo valor que tenía en el punto "anterior". Esto se desprende del comportamiento de $F(t)$

$$h\_en\_ft(x_1, \ldots, x_n, F(t+1)) = h(x_1, \ldots, x_n, F(t))$$

que por definición es

$$h(x_1, \ldots, x_n, F(t)) = h\_en\_ft(x_1, \ldots, x_n, t)$$

entonces

$$h\_en\_ft(x_1, \ldots, x_n, t+1) = h\_en\_ft(x_1, \ldots, x_n, t)$$

Observacion 3: En el otro caso, suponiendo verdadero $P(t+1)$

$$h\_en\_ft(x_1, \ldots, x_n, t+1) = h(x_1, \ldots, x_n, F(t+1)) = h(x_1, \ldots, x_n, t+1)$$

pero por definicion de h

$$h\_en\_ft(x_1, \ldots, x_n, t+1) = g(x_1, \ldots, x_n, h(x_1, \ldots, x_n, F(t)), t)$$

y usando (1) dentro de la función g queda

$$h\_en\_ft(x_1, \ldots, x_n, t+1) = g(x_1, \ldots, x_n, h\_en\_ft(x_1, \ldots, x_n, t), t)$$

Versión final: De las tres observaciones podemos armar una definición primitiva recursiva de la función

$$\begin{aligned} h\_en\_ft(x_1,\ldots,x_n,0) &= g_0(x_1,\ldots,x_n) \\ h\_en\_ft(x_1,\ldots,x_n,t+1) &= \begin{cases} g(x_1,\ldots,x_n,h\_en\_ft(x_1,\ldots,x_n,t),t) & \text{si } P(t+1) \\ h\_en\_ft(x_1,\ldots,x_n,t) & \text{si } \neg P(t+1) \end{cases} \end{aligned}$$

Si bien este esquema no es exactamente igual al esquema p.r. se puede usar una función auxiliar que transforme el caso $h\_en\_ft(x_1,\ldots,x_n,t+1)$. □

1.4. Ejercicio 4

Enunciado: Mostrar que, dado un predicado $P(t,x_1,\ldots,x_n)$ p.r., la cuantificación doblemente acotada es p.r.

$$Forall\_dacot_p(a,b,x_1,\ldots,x_n) = \begin{cases} 1 & \text{si } \forall(t)_{a\leq t\leq b}\ P(t,x_1,\ldots,x_n) \\ 0 & \text{sino} \end{cases}$$

Resolución: Podemos definirla como

$$Forall\_dacot_p(a,b,x_1,\ldots,x_n) = \begin{cases} 1 & \text{si } \forall(t)_{t\leq b}\ Q_p(a,t,x_1,\ldots,x_n) \\ 0 & \text{sino} \end{cases}$$

donde

$$Q_p(a,t,x_1,\ldots,x_n) = (t<a) \vee P(t,x_1,\ldots,x_n)$$

Q es p.r. ya que "P", "$\vee$", "$<$" lo son, la cuantificación universal acotada también lo es junto con la división por casos entonces $Forall\_dacot_p$ resulta p.r.

1.5. Ejercicio 5

Enunciado: Mostrar que, dado un predicado $P(t,x_1,\ldots,x_n)$ p.r., la maximización acotada es p.r.

$$Max\_acot(b,x_1,\ldots,x_n) = \begin{cases} b+1 & \text{si } \forall(y)_{y\leq b}\ \neg P(y,x_1,\ldots,x_n) \\ \text{máx}_{t\leq b}\ P(t,x_1,\ldots,x_n) & \text{sino} \end{cases}$$

Resolución: Lo definimos como

$$Max\_acot(b,x_1,\ldots,x_n) = \min_{t\leq b} Q(t,x_1,\ldots,x_n)$$

donde

$$Q(t,x_1,\ldots,x_n) = P(t,x_1,\ldots,x_n) \wedge \forall(t')_{t+1\leq t'\leq b}\ \neg P(t',x_1,\ldots,x_n)$$

Podemos ver que el predicado Q identifica unívocamente el elemento que "cumple P y todos los siguientes no cumplen P", esto quiere decir el máximo elemento que cumple P. Por lo tanto usando la minimización acotada y fortaleciendo el predicado logramos la maximización acotada. La función es p.r. porque se usan funciones ya demostradas p.r. y composición. □

La minimización acotada es una herramienta **muy** poderosa si se usa bien, siempre que podamos crear un predicado (p.r.) que defina unívocamente a un elemento y dar una cota (p.r.) podemos usar la minimización acotada para obtener el mínimo elemento que lo cumpla que, en este caso, será el *único*.