Lógica y Computabilidad

Práctica 1: Funciones primitivas recursivas y clases PRC

2do cuatrimestre 2022

Ejercicio 1

Para construir una constante k, aplicamos la función s (sucesor) unas k veces, partiendo inicialmente de la función n que nos devuelve el 0.

$f(x) = k = (\underbrace{s \circ \cdots \circ s}_{k \text{ veces}} \circ n)(x) = s^k(n(x))$

Ejercicio 2

- $f_1(x, y) = \text{suma}(x, y) = x + y$
  $\text{suma}(x, 0) = u_1^1(x) = x$
  $\text{suma}(x, y + 1) = g(\text{suma}(x, y), x, y)$ donde $g(x_1, x_2, x_3) = s(u_1^3(x_1, x_2, x_3))$
  $\Rightarrow \text{suma}(x, y + 1) = s(\text{suma}(x, y))$

- $f_2(x, y) = \text{prod}(x, y) = x \cdot y$
  $\text{prod}(x, 0) = n(x) = 0$
  $\text{prod}(x, y + 1) = g(\text{prod}(x, y), x, y)$ donde $g(x_1, x_2, x_3) = \text{suma}(u_1^3(x_1, x_2, x_3), u_2^3(x_1, x_2, x_3))$
  $\Rightarrow \text{prod}(x, y + 1) = \text{suma}(\text{prod}(x, y), x)$

- $f_3(x, y) = \text{pot}(x, y) = x^y$
  $\text{pot}(x, 0) = s(n(x)) = 1$
  $\text{pot}(x, y + 1) = g(\text{pot}(x, y), x, y)$ donde $g(x_1, x_2, x_3) = \text{prod}(u_1^3(x_1, x_2, x_3), u_2^3(x_1, x_2, x_3))$
  $\Rightarrow \text{pot}(x, y + 1) = \text{prod}(\text{pot}(x, y), x)$

- $f_4(x, y) = \underbrace{x^{x^{\cdot^{\cdot^{x}}}}}_{y \text{ veces}}$
  $f_4(x, 0) = 1$
  $f_4(x, y + 1) = g(f_4(x, y), x, y)$ donde $g(x_1, x_2, x_3) = \text{pot}(u_2^3(x_1, x_2, x_3), u_1^3(x_1, x_2, x_3))$
  $\Rightarrow f_4(x, y + 1) = \text{pot}(x, f_4(x, y))$
  Esta función a veces se la llama "Power Tower" (Wikipedia)

- $g_1(x) = \text{pred}(x) = x \dot{-} 1$
  $\text{pred}(0) = n() = 0$ Permitimos utilizar la función nula n sin parámetros.
  $\text{pred}(x + 1) = g(\text{pred}(x), x)$ donde $g(x_1, x_2) = u_2^2(x_1, x_2) = x_2$
  $\Rightarrow \text{pred}(x + 1) = x$

- $g_2(x, y) = \text{resta}(x, y) = x \dot{-} y$
  $\text{resta}(x, 0) = u_1^1(x) = x$
  $\text{resta}(x, y + 1) = g(\text{resta}(x, y), x, y)$ donde $g(x_1, x_2, x_3) = \text{pred}(u_1^3(x_1, x_2, x_3))$
  $\Rightarrow \text{resta}(x, y + 1) = \text{pred}(\text{resta}(x, y))$

- $g_3(x, y) = \max\{x, y\}$

  $g_3(x, y) = \text{suma}(\text{resta}(x, y), y) = (x \dot{-} y) + y$

  Si $x \geqslant y$, entonces g_3 simplemente resta y suma y a un x que es más grande, y en efecto terminamos con x que era el máximo. En el otro caso $x < y$, al hacer la resta en $\mathbb{N}$ obtenemos $x \dot{-} y = 0$, luego al sumar y obtenemos nuevamente y que era el máximo.

- $g_4(x, y) = \min\{x, y\}$

  $g_4(x, y) = \text{resta}(\text{suma}(x, y), \max\{x, y\}) = x + y - \max\{x, y\}$

Ejercicio 3

a)

Para la ida ($\Rightarrow$) hacemos una demostración por inducción estructural. Primero probamos que todas las funciones iniciales cumplen la propiedad.

- Función nula

  $f(x) = n(x) = 0$. La función nula cae en el caso $f(x) = k$ donde $k = 0$.

- Función sucesor

  $f(x) = s(x) = x + 1$. La función sucesor cae en el caso $f(x) = x + k$ donde $k = 1$.

- Función proyector

  $f(x_1, \ldots, x_n) = u_i^n(x_1, \ldots, x_n) = x_i$. La función proyector cae en el caso $f(x_1, \ldots, x_n) = x_i + k$ donde $k = 0$.

Paso inductivo. Supongamos que existe $h_m \in \mathcal{C}_c$ generada a partir de m composiciones, tal que $h_m(x_1, \ldots, x_n) = k$ o bien $h_m(x_1, \ldots, x_n) = x_i + k$. Queremos ver si cualquier $h_{m+1} \in \mathcal{C}_c$ también cumple la propiedad. Para generar h_{m+1} componemos h_m con alguna función $f \in \mathcal{C}_c$.

- Caso $f(x) = n(x)$

  $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = n(h_m(x_1, \ldots, x_n)) = 0$.

  No importa la forma de h_m pues $n(x) = 0$ para cualquier x.

- Caso $f(x) = s(x)$
  - Caso $h_m(x_1, \ldots, x_n) = x_i + q$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = s(x_i + q) = x_i + q + 1 = x_i + k$ donde $k = q + 1$.
  - Caso $h_m(x_1, \ldots, x_n) = q$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = s(q) = q + 1 = k$ donde $k = q + 1$.

- Caso $f(x) = u_i^n(x)$

  Como $h_m : \mathbb{N}^n \to \mathbb{N}$, necesariamente $f(x) = u_1^1(x)$ para poder componer $f \circ h_m$.
  - Caso $h_m(x_1, \ldots, x_n) = x_i + k$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = u_1^1(x_i + k) = x_i + k$.
  - Caso $h_m(x_1, \ldots, x_n) = k$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = u_1^1(k) = k$.

- Caso $f(x) = x + r$
  - Caso $h_m(x_1, \ldots, x_n) = x_i + q$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = x_i + q + r = x_i + k$ donde $k = q + r$.
  - Caso $h_m(x_1, \ldots, x_n) = q$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = q + r = k$ donde $k = q + r$.

- Caso $f(x) = k$
  - Caso $h_m(x_1, \ldots, x_n) = x_i + q$

    $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = f(x_i + q) = k$.

- Caso $h_m(x_1, \ldots, x_n) = q$
  $h_{m+1} = f(h_m(x_1, \ldots, x_n)) = f(q) = k$.

Por lo tanto, partiendo de una función $h_m \in \mathcal{C}_c$, vemos que al realizar una composición con alguna función $f \in \mathcal{C}_c$ obtenemos una función $h_{m+1} \in \mathcal{C}_c$ (pues $\mathcal{C}_c$ es cerrado por composición) que mantiene la propiedad enunciada.

Para la vuelta ($\Leftarrow$) mostramos que podemos construir cualquier $f(x_1, \ldots, x_n) = k$ o $f(x_1, \ldots, x_n) = x_i + k$ a partir de composición de las funciones iniciales, y por lo tanto $f \in \mathcal{C}_c$.

- $f(x_1, \ldots, x_n) = k = s^k(n(x_1, \ldots, x_n))$
- $f(x_1, \ldots, x_n) = x_i + k = s^k(u_i^n(x_1, \ldots, x_n))$

b)

En el ejercicio 2 vimos que la función suma$(x, y) = x + y$ es P.R. pero suma $\notin \mathcal{C}_c$ pues no cumple con la propiedad.

Ejercicio 4

Cualquier clase PRC contiene las funciones iniciales y está cerrada por recursión primitiva y composición. Para mostrar que los predicados están en cualquier clase PRC, es suficiente con mostrar que se pueden construir a partir de las funciones iniciales utilizando recursión primitiva y/o composición.

Para simplificar la escritura vamos a utilizar la función $\alpha(x)$ la cual es PR a partir de las iniciales y por lo tanto pertenece a cualquier clase PRC.

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

Además, podemos definir el predicado $\neg(x) = \alpha(x)$ que niega otro predicado.

$$\leqslant) \;\; p(x,y) = \begin{cases} 1 & \text{si } x \leqslant y \\ 0 & \text{si no} \end{cases} = \alpha(x \dot{-} y)$$

$$\geqslant) \;\; p(x,y) = \begin{cases} 1 & \text{si } x \geqslant y \\ 0 & \text{si no} \end{cases} = \alpha(y \dot{-} x)$$

$$=) \;\; p(x,y) = \begin{cases} 1 & \text{si } x = y \\ 0 & \text{si no} \end{cases} = (x \leqslant y) \cdot (x \geqslant y)$$

$$\neq) \;\; p(x,y) = \begin{cases} 1 & \text{si } x \neq y \\ 0 & \text{si no} \end{cases} = \neg(x = y)$$

$$<) \;\; p(x,y) = \begin{cases} 1 & \text{si } x < y \\ 0 & \text{si no} \end{cases} = \neg(x \geqslant y)$$

$$>) \;\; p(x,y) = \begin{cases} 1 & \text{si } x > y \\ 0 & \text{si no} \end{cases} = \neg(x \leqslant y)$$

Ejercicio 5

Podemos escribir h de la siguiente forma equivalente en donde se puede ver más claramente que es composición de funciones, en particular es composición de funciones en $\mathcal{C}$ pues todas las f_i y g están en $\mathcal{C}$. Como $\mathcal{C}$ es una clase PRC resulta que $h \in \mathcal{C}$.

$h(x_1, \ldots, x_n) = \sum_{i=1}^{k} f_i(x_1, \ldots, x_n) \cdot p_i(x_1, \ldots, x_n) + g(x_1, \ldots, x_n) \cdot \neg(\sum_{i=1}^{k} p_i(x_1, \ldots, x_n))$

También podemos analizarlo por casos:

Caso 1: $\exists! i : \mathbb{N}, 1 \leqslant i \leqslant k$ tal que $p_i(x_1, \ldots, x_n)$ es verdadero.

Observemos que si existe i, tiene que ser único pues todos los predicados $p_1, \ldots, p_k$ son disjuntos. Luego, vale que $h(x_1, \ldots, x_n) = f_i(x_1, \ldots, x_n)$ por definición (el predicado $p_i(x_1, \ldots, x_n)$ es verdadero y por lo tanto "selecciona" el caso de f_i dentro de la definición de h). Como todas las $f_i \in \mathcal{C}$ por hipótesis $\Rightarrow h \in \mathcal{C}$.

Caso 2: $\forall i : \mathbb{N}, 1 \leqslant i \leqslant k \Rightarrow p_i(x_1, \ldots, x_n)$ es falso.

Como no existe predicado p_i que resulte verdadero, por definición h "selecciona" el último caso "si no" y luego resulta $h(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$. Como $g \in \mathcal{C}$ por hipótesis $\Rightarrow h \in \mathcal{C}$.

Ejercicio 6

Una clase de funciones $\mathcal{C}$ es PRC si contiene las funciones iniciales y está cerrada por composición y recursión primitiva.

Podemos afirmar que una función cualquiera va a estar en **toda** clase PRC si podemos demostrar que pertenece a la clase PR. La clase PR es la clase PRC más "chica", son todas las funciones que se pueden construir a partir de las funciones iniciales mediante composición y/o recursión primitiva, y por lo tanto van a pertenecer a cualquier clase PRC.

No obstante, una clase $\mathcal{C}$ puede ser PRC y a su vez contener funciones que no puedan ser construidas a partir de las iniciales. Esto sucede cuando la clase incluye explícitamente alguna función adicional que no pertenece a la clase PR. En esencia, para generar nuevas funciones dentro de esta clase $\mathcal{C}$, además de tener las 3 funciones iniciales, tendríamos esta función adicional.

a)

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

$\text{par}(0) = 1$

$\text{par}(x+1) = g(\text{par}(x), x)$ donde $g(x_1, x_2) = \neg u_1^2(x_1, x_2) \Rightarrow \text{par}(x+1) = \neg\text{par}(x)$

Definimos $\text{impar}(x) = \neg\text{par}(x)$ para usar en los siguientes items.

b)

$f(x) = \text{div2}(x) = \lfloor x/2 \rfloor$

$\text{div2}(0) = 0$

$$\text{div2}(x+1) = \begin{cases} \text{div2}(x) & \text{si par}(x) \\ s(\text{div2}(x)) & \text{si no} \end{cases} = \text{div2}(x) \cdot \text{par}(x) + s(\text{div2}(x)) \cdot \text{impar}(x)$$

c)

$$h(x_1, \ldots, x_n, t) = \begin{cases} f(x_1, \ldots, x_n) & \text{si } t = 0 \\ g_1(x_1, \ldots, x_n, k, h(x_1, \ldots, x_n, t-1)) & \text{si } t = 2 \cdot k + 1 \\ g_2(x_1, \ldots, x_n, k, h(x_1, \ldots, x_n, t-1)) & \text{si } t = 2 \cdot k + 2 \end{cases}$$

Planteamos el caso base del esquema de recursión primitiva para h. Notar que si $t = 0$ siempre entramos en el primer caso de h pues no existe $k \in \mathbb{N}$ para satisfacer los otros 2 casos cuando $t = 0$.

$h(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n)$

Para el caso $t > 0$ primero reescribimos la función por casos colocando $t+1$ en donde antes teníamos t para poder ajustarnos al esquema de recursión primitiva.

$$h(x_1,\dots,x_n,t+1) = \begin{cases} g_1(x_1,\dots,x_n,k,h(x_1,\dots,x_n,t+1-1)) & \text{si } t+1 = 2\cdot k+1 \\ g_2(x_1,\dots,x_n,k,h(x_1,\dots,x_n,t+1-1)) & \text{si } t+1 = 2\cdot k+2 \end{cases}$$
$$= \begin{cases} g_1(x_1,\dots,x_n,k,h(x_1,\dots,x_n,t)) & \text{si } t = 2\cdot k \\ g_2(x_1,\dots,x_n,k,h(x_1,\dots,x_n,t)) & \text{si } t = 2\cdot k+1 \end{cases}$$

Ahora necesitamos despejar k en función de t.

$t = 2\cdot k \Rightarrow k = \lfloor t/2 \rfloor = \text{div2}(t)$

$t = 2\cdot k + 1 \Rightarrow k = \lfloor (t-1)/2 \rfloor = \text{div2}(t-1) = \text{div2}(t)$ pues t es impar y div2 redondea hacia abajo

Reescribimos h eliminando por completo la k.

$$h(x_1,\dots,x_n,t+1) = \begin{cases} g_1(x_1,\dots,x_n,\text{div2}(t),h(x_1,\dots,x_n,t)) & \text{si par}(t) \\ g_2(x_1,\dots,x_n,\text{div2}(t),h(x_1,\dots,x_n,t)) & \text{si impar}(t) \end{cases}$$

También podemos escribir h como suma de cada caso por su predicado.

$h(x_1,\dots,x_n,t+1) = g_1(x_1,\dots,x_n,\text{div2}(t),h(x_1,\dots,x_n,t))\cdot \text{par}(t) + g_2(x_1,\dots,x_n,\text{div2}(t),h(x_1,\dots,x_n,t))\cdot \text{impar}(t)$

Conclusión

- h sigue el esquema de recursión primitiva.
- h compone funciones que están en $\mathcal{C}$ (puntualmente f, g_1 y g_2).
- h compone funciones que están en toda clase PRC (puntualmente div2, par, impar).
- $\mathcal{C}$ es una clase PRC por el enunciado, entonces contiene a las funciones div2, par, impar.

Por lo tanto cualquier h que cumpla con este esquema pertenece a $\mathcal{C}$.

Ejercicio 7

Usamos la notación $\overline{x} = x_1,\dots,x_n$ para simplificar la escritura.

- $\text{cantidad}_p(\overline{x},y,z) = \sum_{t=0}^{z} p(\overline{x},t)\cdot(t \geqslant y) = \sum_{t=y}^{z} p(\overline{x},t)$
- $\text{todos}_p(\overline{x},y,z) = \text{cantidad}_p(\overline{x},y,z) = z - y$
- $\text{alguno}_p(\overline{x},y,z) = \text{cantidad}_p(\overline{x},y,z) > 0$
- Recordemos la definición de la minimización acotada.
  $\min_{t\leqslant y} p(\overline{x},t) = \sum_{u=0}^{y}\prod_{t=0}^{u}\alpha(p(\overline{x},t))$
  Usando la minimización acotada definir mínimo_p es trivial.
  $\text{mínimo}_p(\overline{x},y,z) = \min_{t\leqslant z}\,(p(\overline{x},t)\cdot(t\geqslant y)) = \min_{y\leqslant t\leqslant z} p(\overline{x},t)$
- Una opción es reutilizar la misma idea que la minimización acotada, pero buscando el “mínimo” partiendo desde el t más grande yendo hacia la cota inferior. En este contexto, el mínimo es en realidad “el más cercano a la cota superior”, o sea, el máximo, que es lo queremos.
  $\text{máximo}_p(\overline{x},y,z) = z - \sum_{u=y}^{z}\prod_{t=0}^{u}\alpha(p(\overline{x},z-t))$
  Otra opción es definir el máximo utilizando el mínimo, agregando una condición adicional al predicado: que no exista ningún otro $t' > t$ para el cual también valga el predicado p. En esencia, buscamos primero el mínimo real, y si existe un t más grande para el cual también vale el predicado, tenemos que seguir buscando el próximo mínimo a partir del encontrado, y así hasta eventualmente llegar al máximo.
  $\text{máximo}_p(\overline{x},y,z) = \text{mínimo}_q(\overline{x},y,z)$ donde $q(\overline{x},t) = p(\overline{x},t)\cdot\neg(\exists t')_{t<t'\leqslant z}\, p(\overline{x},t')$
- Sea $m = \text{mínimo}_p(\overline{x},y,z)$, $M = \text{máximo}_p(\overline{x},y,z)$
  $\text{único}_p(\overline{x},y,z) = (m = M)\cdot m + (m \neq M)\cdot(z+1)$

Ejercicio 8

- $\text{cociente}(x, y) = \min_{t \leqslant x} ((t \cdot y + r = x) \wedge (0 \leqslant r < y))$
- $\text{resto}(x, y) = x - \text{cociente}(x, y) \cdot y$
- $\text{divide}(x, y) = \text{resto}(x, y) = 0$
- $\text{primo}(x) = \neg(\exists t)_{1<t<x}\ \text{divide}(x, t)$
- $\text{raíz}(x, y) = \min_{t \leqslant y}\ (t + 1)^x > y$
- $\text{nprimo}(0) = 2$
  $\text{nprimo}(n + 1) = \min_{t \leqslant c}\ (\text{primo}(t) \wedge t > \text{nprimo}(n))$
  Donde $c = \text{nprimo}(n)! + 1$ funciona como cota porque $\text{nprimo}(n + 1) \leqslant \text{nprimo}(n)! + 1$ (justificado en la teórica).

Ejercicio 9

Pendiente

Ejercicio 10

Pendiente

Ejercicio 11

Pendiente

Ejercicio 12

Pendiente

Ejercicio 13

Pendiente

Ejercicio 14

Pendiente

Ejercicio 15

Pendiente