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 (que están en $\mathcal{C}_c$) 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 existen $f, g_1, \ldots, g_m \in \mathcal{C}_c$ que cumplen con la propiedad. Como la única operación en $\mathcal{C}_c$ es la composición, esa es la única forma de generar nuevas funciones. Probemos entonces que la función generada por composición $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) \in \mathcal{C}_c$ también cumple con la propiedad.

Miremos todos los posibles casos para f que ya sabemos que cumple con la propiedad por hipótesis inductiva.

- $f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) = k \Rightarrow h(x_1, \ldots, x_n) = k$
- $f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) = g_j(x_1, \ldots, x_n) + k_0$ para algún $j \mid 1 \leqslant j \leqslant m$.

  Por hipótesis inductiva, $g_j \in \mathcal{C}_c$ cumple con la propiedad. Por lo tanto $g_j(x_1, \ldots, x_n) = k_j$ o bien $g_j(x_1, \ldots, x_n) = x_i + k_j$ para algún $i \mid 1 \leqslant i \leqslant n$. Observar que necesitamos identificar k_j con un subíndice ya que cada g_j puede tener constantes arbitrarias. Además, necesitamos usar un subíndice distinto para el x_i ya que sino estaríamos limitando qué parámetro de entrada puede usar cada g_j.

  - $g_j(x_1, \ldots, x_n) = k_j \Rightarrow h(x_1, \ldots, x_n) = k_j + k_0 = k$
  - $g_j(x_1, \ldots, x_n) = x_i + k_j \Rightarrow h(x_1, \ldots, x_n) = x_i + k_j + k_0 = x_i + k$

Probamos entonces que cualquier función $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) \in \mathcal{C}_c$ cumple con la propiedad. Es decir, vale que si $h \in \mathcal{C}_c \Rightarrow h(x_1, \ldots, x_n) = k$ o bien $h(x_1, \ldots, x_n) = x_i + k$.

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 $f \in \mathcal{C}_c$.

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

b)

En el ejercicio 2 vimos que la función $\text{suma}(x, y) = x + y$ es p.r. pero $\text{suma} \notin \mathcal{C}_c$ 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.

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

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.

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

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

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

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 $\overline{x} = x_1, \ldots, x_n$ para simplificar la escritura. Ya vimos en la teórica que los operadores acotados están necesariamente acotados por arriba. La idea de este ejercicio es mostrar que también podemos acotarlos por abajo.

- $\text{cantidad}_p(\overline{x}, y, z) = \sum_{t=0}^{z} p(\overline{x}, t) \cdot (t \geqslant y) = \sum_{t=y}^{z} p(\overline{x}, t)$
- $\text{todos}_p(\overline{x}, y, z) = (\forall t)_{\leqslant z}\ (p(\overline{x}, t) \cdot (t \geqslant y) + (t < y)) = (\forall t)_{y \leqslant t \leqslant z}\ p(\overline{x}, t)$
  Otra forma: $\text{todos}_p(\overline{x}, y, z) = \text{cantidad}_p(\overline{x}, y, z) = z - y$
- $\text{alguno}_p(\overline{x}, y, z) = (\exists t)_{\leqslant z}\ (p(\overline{x}, t) \cdot (t \geqslant y) + (t < y)) = (\exists t)_{y \leqslant t \leqslant z}\ p(\overline{x}, t)$
  Otra forma: $\text{alguno}_p(\overline{x}, y, z) = \text{cantidad}_p(\overline{x}, y, z) > 0$
- Recordemos la definición de la minimización acotada: $\min_{t \leqslant y} p(\overline{x}, t) = \sum_{u=0}^{y} \prod_{t=0}^{u} \alpha(p(\overline{x}, t))$
  $\text{mínimo}_p(\overline{x}, y, z) = \min_{t \leqslant z}\ (p(\overline{x}, t) \cdot (t \geqslant y)) = \min_{y \leqslant t \leqslant z}\ p(\overline{x}, t)$
- Podemos 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

- $\text{cociente}(x, y) = \min_{t \leqslant x}\ ((t \cdot y + r = x) \wedge (0 \leqslant r < y))$
- $\text{resto}(x, y) = x - \text{cociente}(x, y) \cdot y$
- $\text{divide}(x, y) = \text{resto}(x, y) = 0$
- $\text{primo}(x) = \neg(\exists t)_{1 < t < x}\ \text{divide}(x, t)$
- $\text{raíz}(x, y) = \min_{t \leqslant y}\ (t+1)^x > y$
- $\text{nprimo}(0) = 0$
  $\text{nprimo}(n+1) = \min_{t \leqslant c}\ (\text{primo}(t) \wedge t > \text{nprimo}(n))$
  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

Sea $\langle x, y \rangle = 2^x(2y + 1) \dot{-} 1$ la codificación de pares de naturales. Esta función es p.r. pues es una composición de otras funciones p.r. como la potencia, multiplicación y suma.

Esta codificación es una biyección, pues hay una única solución (x, y) a la ecuación $z = \langle x, y \rangle$. Notemos que para garantizar esto es necesario restar 1 en la codificación, pues sino el par $\langle 0, 0 \rangle = 1$ y luego sucedería que $\langle x, y \rangle \geqslant 1 \ \forall x, y \in \mathbb{N}$.

$z = \langle x, y \rangle = 2^x(2y + 1) \dot{-} 1 \iff z + 1 = 2^x(2y + 1)$

Observemos que 2^x es siempre un número par o 1 (en el caso $x = 0$), mientras que $2y + 1$ es siempre un número impar. Por lo tanto, dado z podemos volver a obtener el par (x, y) realizando 2 pasos.

1. $x = \max_{x \leqslant z} 2^x | (z + 1)$
2. $y = (\frac{z+1}{2^x} - 1)/2$

Definimos los observadores l (left) y r (right) del par $z = \langle x, y \rangle$ tal que:

- $l(z) = \min_{x \leqslant z} ((\exists y)_{\leqslant z} \ z = \langle x, y \rangle)$
- $r(z) = \min_{y \leqslant z} ((\exists x)_{\leqslant z} \ z = \langle x, y \rangle)$

Como la codificación de pares y los observadores son p.r. podemos concluir que toda clase PRC los contiene.

Ejercicio 10

Para demostrar que la función de Fibonacci está en toda clase PRC vamos a mostrar que podemos definirla como una función primitiva recursiva.

Sea $f : \mathbb{N} \to \mathbb{N}$ la función de Fibonacci tal que:

- $f(0) = 0$
- $f(1) = 1$
- $f(n + 2) = f(n + 1) + f(n)$

Esta definición difiere del esquema de recursión primitiva por 2 razones: tenemos 2 casos bases y necesitamos mirar los 2 resultados anteriores en el paso recursivo.

¿Acaso no tenemos ya una forma de codificar 2 valores como un único número? Intentemos definir Fibonacci con una nueva función g utilizando la codificación de pares. Para cualquier $n \in \mathbb{N}$, $g(n)$ nos devuelve un par codificado con los términos n y $n + 1$ de Fibonacci.

- $g(0) = \langle 0, 1 \rangle$
- $g(n + 1) = \langle r(g(n)), l(g(n)) + r(g(n)) \rangle$

La función g es p.r. pues sigue el esquema de recursión primitiva y es composición de funciones p.r.

Finalmente, podemos redefinir f en función de g de la siguiente forma: $f(n) = l(g(n))$. Por lo tanto, f es p.r. y consecuentemente pertenece a cualquier clase PRC.

Ejercicio 11

Definimos una función H que va a realizar la recursión mutua codificándola en un par.

$H(\overline{x}, t) = \langle h_1(\overline{x}, t), h_2(\overline{x}, t) \rangle$

Planteamos el esquema de recursión primitiva.

$H(\overline{x}, 0) = \langle h_1(\overline{x}, 0), h_2(\overline{x}, 0)\rangle = \langle f_1(\overline{x}), f_2(\overline{x})\rangle$

$$\begin{aligned} H(\overline{x}, t+1) &= \langle h_1(\overline{x}, t+1), h_2(\overline{x}, t+1)\rangle \\ &= \langle g_1(h_1(\overline{x}, t), h_2(\overline{x}, t), \overline{x}, t), g_2(h_2(\overline{x}, t), h_1(\overline{x}, t), \overline{x}, t)\rangle \\ &= \langle g_1(l(H(\overline{x}, t)), r(H(\overline{x}, t)), \overline{x}, t), g_2(r(H(\overline{x}, t)), l(H(\overline{x}, t)), \overline{x}, t)\rangle \end{aligned}$$

H sigue el esquema de recursión primitiva y es composiciones de funciones p.r. y de $f_1, f_2, g_1, g_2 \in \mathcal{C} \Rightarrow H \in \mathcal{C}$.

Como $H \in \mathcal{C}$ está codificando a h_1 y h_2 de la siguiente forma:

$h_1(\overline{x}, t) = l(H(\overline{x}, t))$

$h_2(\overline{x}, t) = r(H(\overline{x}, t))$

$\Rightarrow h_1, h_2 \in \mathcal{C}$.

Ejercicio 12

Pendiente

Ejercicio 13

a)

Por el Teorema Fundamental de la Aritmética, todos los números naturales mayores a 1 tienen una única factorización como producto de números primos. La codificación de una secuencia es simplemente interpretar todos los elementos de ella como las potencias de los números primos de la factorización de algún número natural.

Como el TFA nos dice que esta factorización es única, y como solo consideramos secuencias finitas que no terminan en cero, efectivamente tenemos una biyección entre los números naturales y las secuencias al utilizar esta codificación.

Notemos que no podemos permitir que haya 0s al final de la secuencia pues sino rompemos la biyección, ya que tendríamos infinitas secuencias que codifican al mismo número.

Nos quedaron afuera el 0 y el 1 (el TFA solo vale para números naturales mayores a 1). No existe secuencia que codifique a 0, pero esto no es problema pues el enunciado pide una biyección con los naturales mayores a 0. Y por otro lado, veamos que la secuencia vacía [] codifica a 1 pues resulta una productoria vacía.

b)

La secuencia vacía codifica al 1. Si n es el tamaño de la secuencia, entonces la secuencia vacía [] tiene $n = 0$, y por lo tanto:

$[] = \prod_{i=1}^{n} \text{nprimo}(i)^{a_i} = \prod_{i=1}^{0} \text{nprimo}(i)^{a_i} = 1$

Definimos las funciones a partir de otras funciones que ya mostramos que están en toda clase PRC:

- $|s| = \max_{i \leqslant s} \text{divide}(\text{nprimo}(i), s)$
- $s[i] = \max_{j \leqslant s} \text{divide}(\text{nprimo}(i)^j, s)$
- $[x] = \text{nprimo}(1)^x = 2^x$
- $r \circ s = r \cdot \prod_{i=1}^{|s|} \text{nprimo}(|r| + i)^{s[i]}$
- $\text{sub}(s, i, j) = \prod_{k=i}^{j} \text{nprimo}(k - i)^{s[i]}$

c)

Ejercicio 14

Pendiente

Ejercicio 15

Pendiente