Ejercicio 2.

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

qvq la clase de funciones P.C. está cerrada por composición y después ver lo mismo para recursión. Aclarar al principio que las funciones iniciales son P.C.

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

$*(x,y)$ se puede obtener usando composición y recursión de unas PR. $\Rightarrow *(x,y)$ es PR.

$\Rightarrow *(x,y)$ es COMPUTABLE

↑

PR $\Rightarrow$ COMPUTABLE

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

Existe.

Clausura por composición

Teorema
Si h se obtiene a partir de las funciones (parciales) computables $f, g_1, \ldots, g_k$ por composición entonces h es (parcial) computable.

Demostración.
El siguiente programa computa h:

$$Z_1 \leftarrow g_1(X_1, \ldots, X_n)$$
$$\vdots$$
$$Z_k \leftarrow g_k(X_1, \ldots, X_n)$$
$$Y \leftarrow f(Z_1, \ldots, Z_k)$$

Si $f, g_1, \ldots, g_k$ son totales entonces h es total. □

Clausura por recursión primitiva

Teorema
Si h se obtiene a partir de g por recursión primitiva y g es computable entonces h es computable.

Demostración.
El siguiente programa computa h:

$Y \leftarrow k$ (es una macro, se puede hacer fácil)
$[A]$ IF $X = 0$ GOTO E (otra macro, condición del IF por $=$)
$Y \leftarrow g(Z, Y)$
$Z \leftarrow Z + 1$
$X \leftarrow X - 1$
GOTO A

Si g es total entonces h es total. □

Las funciones computables forman una clase PRC

Teorema
La clase de funciones computables es una clase PRC.

Demostración.
Ya vimos que la clase de funciones computables está cerrada por composición (p. 68) y recursión primitiva (p. 69). Veamos que las funciones iniciales son computables:

- $s(x) = x + 1$ se computa con el programa
$$Y \leftarrow X + 1$$
- $n(x) = 0$ se computa con el programa vacío
- $u_i^n(x_1, \ldots, x_n) = x_i$ se computa con el programa
$$Y \leftarrow X_i$$

□

Corolario
Toda función primitiva recursiva es computable.

Ejercicio 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.

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

Si un programa no es autocontenido => tiene un salto que va a una etiqueta inexistente. Cuando eso pasa el programa termina con el valor que Y tuviera al momento del salto. Podemos agarrar P y agregarle la instruccion Y <- Y bajo esa etiqueta que antes se salia del programa. Asi el programa P' quedaria autocontenido y su comportamiento seria igual al de P.

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

- IF $r(\bar{V})$ GOTO [L]

  $Z_1 \leftarrow r(\bar{V})$

  if $Z_1 = 1$ goto [L]

- IF $r(\bar{V})$ THEN P ELSE Q

  if $r(\bar{V})$ goto [P]

  goto [Q]

- WHILE $r(\bar{V})$ DO P

[X] if $r(\bar{V})$ goto [P]

En P tengo que definir el salto a X.

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

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

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

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

if $X_1 \geq 5 \vee X_1 = 3$ goto A

$Y \leftarrow g(X_1)$

goto F

[A] $Y \leftarrow f(X_1)$

goto F

Ejercicio 5.

a. Demostrar que si $p : \mathbb{N}^{n+1} \to \{0, 1\}$ es un predicado $\mathcal{S}$-computable (total), entonces es $\mathcal{S}$-parcial computable:

$$\text{minimoNA}_p(x_1, \ldots, x_n, y) = \begin{cases} \text{mín}\{t \mid y \leq t \wedge p(x_1, \ldots, x_n, t)\} & \text{si existe algún tal } t \\ \uparrow & \text{en otro caso} \end{cases}$$

$Z_1 \leftarrow X_{n+1}$

[A] if $P(\bar{X}, Z_1)$ goto F

$Z_1 \leftarrow Z_1 + 1$

goto A

[F] $y \leftarrow Z_1$

$f(x) = y \qquad f^{-1}(y) = x$

b. Mostrar, usando el resultado anterior, que si $f : \mathbb{N} \to \mathbb{N}$ es biyectiva y $\mathcal{S}$-computable (total), entonces también lo es su inversa, f^{-1}.

$f^{-1}(x) = \min_t \left[f^{-1}(t) = x \right]$ Vale solo por biyectividad y S-COMPUTABLE TOTAL

Ejercicio 6. Un programa P en el lenguaje $\mathcal{S}$ con instrucciones $I_1, I_2, ..., I_n$ se dice *optimista* si $\forall\, i = 1, ..., n$, si I_i es la instrucción IF $V \neq 0$ GOTO L entonces L no aparece como etiqueta de ninguna instrucción I_j con $j \leq i$.

Demostrar que el siguiente predicado es primitivo recursivo:

$$r(x) = \begin{cases} 1 & \text{si el programa cuyo número es } x \text{ es optimista} \\ 0 & \text{caso contrario} \end{cases}$$

$\bar{r}(x, y) = \begin{cases} 1 & \text{si la instrucción } y \text{ del programa } x \text{ es optimista} \\ 0 & \text{cc} \end{cases}$

$\bar{r}(x, y) = \left[\text{enIF}(y) \wedge \neg(\exists t)_{t \leq |x|} \left[\text{etiqueta}(t) = \text{goto Etiqueta}(y) \right] \right] \vee \neg \text{enIF}(y)$

$r(x) = (\forall t)_{t \leq |x|} \left[\bar{r}(x, (x+1)[t]) \right]$

Ejercicio 7. Utilizando las funciones primitivas-recursivas $\mathrm{STP}^{(n)}$ y $\mathrm{SNAP}^{(n)} : \mathbb{N}^{n+2} \to \mathbb{N}$ vistas en clase, mostrar que las siguientes son funciones $\mathcal{S}$-parciales computables:

$$f_1(x,y) = \begin{cases} 1 & \text{si } y \in \mathrm{Dom}\, \Phi_x^{(1)} \\ \uparrow & \text{si no} \end{cases} \qquad f_2(x) = \begin{cases} 1 & \text{si } \mathrm{Dom}\, \Phi_x^{(1)} \neq \emptyset \\ \uparrow & \text{si no} \end{cases}$$

$$f_3(x,y) = \begin{cases} 1 & \text{si } y \in \mathrm{Im}\, \Phi_x^{(1)} \\ \uparrow & \text{si no} \end{cases} \qquad f_4(x,y) = \begin{cases} 1 & \text{si } \mathrm{Dom}\, \Phi_x^{(1)} \cap \mathrm{Im}\, \Phi_y^{(1)} \neq \emptyset \\ \uparrow & \text{si no} \end{cases}$$

$f_1(x,y) = (\exists \langle t, x_i \rangle)\left[STP(x_i, x, t) = 1 \wedge r(SNAP(x_i, x, t))[1] = y \right]$

$f_2(x) = (\exists \langle t, x_i \rangle)\left[STP(x_i, x, t) = 1 \right]$

$f_3(x,y) = (\exists t)\left[STP(y, x, t) = 1 \right]$

$f_4(x,y) = (\exists \langle x_i, t, x_j, t' \rangle)\left[STP(x_i, y, t) = 1 \wedge r(SNAP(x_j, x, t'))[1] = x_i \wedge STP(x_j, x, t') = 1 \right]$

más fácil $= (\exists t)\left[f_1(x,t) \wedge f_2(x,t) \right]$

Ejercicio 8. Sea $f : \mathbb{N} \to \mathbb{N}$ una función $\mathcal{S}$-parcial computable en tiempo polinomial (i.e., existe un programa P tal que $\Psi_P^{(1)}(x) = f(x)$ y tal que, para algún polinomio $Q(x)$, P no requiere más que $Q(\lceil \log_2 x \rceil)$ pasos para terminar).

a. Mostrar que f es primitiva recursiva.

b. ¿Sucede lo mismo si la cota es exponencial, doblemente exponencial, etc.?

c. ¿Qué podemos decir, en general, sobre la complejidad temporal de una función computable que no sea primitiva recursiva?

a) $f(x) = r\left(SNAP\left(x, P, Q(\lceil \log_2 x \rceil)\right)\right)[1]$

f es PR porque es composicion de cosas PR. (r y SNAP son funciones que sabemos son PR. Ademas, la cota Q(ceil(log_{2}(x))) la podemos conseguir de forma PR.)

b) Sucede lo mismo ya que estas cotas siguen siendo obtenibles de forma PR y vimos recien que siempre que la cota se pueda conseguir de forma PR, f va a ser PR.

c) Tiene una complejidad temporal tan alta que no es obtenible de forma PR.

Ejercicio 9. Se dice que un programa P en el lenguaje $\mathcal{S}$ *se pisa con n entradas* si para alguna entrada $x_1, x_2, ..., x_n$ y algún tiempo t, la variable de salida Y luego de t pasos de la ejecución de P con entradas $x_1, x_2, ..., x_n$ vale $\#P$.

Demostrar que para cualquier $n \in \mathbb{N}$ es $\mathcal{S}$-parcial computable la función:

$$f_n(x) = \begin{cases} 1 & \text{si el programa cuyo número es } x \text{ se pisa con n entradas} \\ \uparrow & \text{caso contrario} \end{cases}$$

$f_n(x) = (\exists \langle x_1, ..., x_m, t \rangle) [\, r(\text{SNAP}(\bar{x}, x, t))[1] = x \,]$