Retomando…

Ejercicio 1

Ejercicio 1

Demostrar que la siguiente función $f : \mathbb{N} \longrightarrow \mathbb{N}$ es primitiva recursiva:

$f(0) = 0$

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

Ejercicio 1

Demostrar que la siguiente función f : $\mathbb{N} \longrightarrow \mathbb{N}$ es **primitiva recursiva**:

$f(0) = 0$

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

$$f(0) = 0$$

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

Funciones primitivas recursivas

Una función es primitiva recursiva (p.r.) si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de composición y recursión primitiva.

Funciones primitivas recursivas

Una función es primitiva recursiva (p.r.) si se puede obtener a partir de las **funciones iniciales** por un número finito de aplicaciones de **composición** y **recursión primitiva**.

Funciones iniciales

Las siguientes funciones se llaman iniciales:

- $s(x) = x + 1$
- $n(x) = 0$
- proyecciones: $u_i^n(x_1, \ldots, x_n) = x_i$ para $i \in \{1, \ldots, n\}$

Composición y recursión primitiva

Sea $f : \mathbb{N}^{\square} \to \mathbb{N}$ y $g_1, \ldots, g_{\square} : \mathbb{N}^n \to \mathbb{N}$. $h : \mathbb{N}^n \to \mathbb{N}$ se obtiene a partir de f y $g_1, \ldots, g_{\square}$ por composición si:

$$h(x_1, \ldots, x_{\square}) = f(g_1(x_1, \ldots, x_{\square}), \ldots, g_{\square}(x_1, \ldots, x_{\square}))$$

$h : \mathbb{N}^{n+1} \to \mathbb{N}$ se obtiene a partir de $g : \mathbb{N}^{n+2} \to \mathbb{N}$ y $f : \mathbb{N}^n \to \mathbb{N}$ por recursión primitiva si:

$$h(x_1, \ldots, x_{\square}, 0) = f(x_1, \ldots, x_{\square})$$

$$h(x_1, \ldots, x_{\square}, t+1) = g(h(x_1, \ldots, x_{\square}, t), x_1, \ldots, x_{\square}, t)$$

Ejercicio 1

Demostrar que la siguiente función f : $\mathbb{N} \longrightarrow \mathbb{N}$ es primitiva recursiva:

$f(0) = 0$

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

¿Se adapta al esquema de recursión primitiva?

Sí

Además, $f(x) = n(x)$, que es inicial

Ejercicio 1

Demostrar que la siguiente función $f : \mathbb{N} \longrightarrow \mathbb{N}$ es primitiva recursiva:

$f(0) = 1$

$f(1) = 2$

$f(2) = 4$

$f(n+3) = f(n) + f(n+1)^2 + f(n+2)^3$

Ejercicio 1

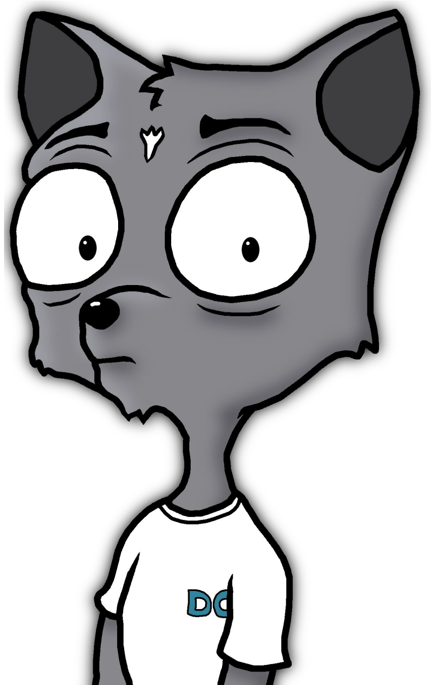
Demostrar que la siguiente función $f : \mathbb{N} \rightarrow \mathbb{N}$ es primitiva recursiva:

$f(0) = 1$

$f(1) = 2$

$f(2) = 4$

$f(n+3) = f(\mathbf{n}) + f(\mathbf{n+1})^2 + f(\mathbf{n+2})^3$



$f(0) = 1$

$f(1) = 2$

$f(2) = 4$

$f(n+3) = f(n) + f(n+1)^2 + f(n+2)^3$

A pensar...

Pistas

- Si sólo podemos acceder al "valor anterior", necesitaríamos guardar en el "valor anterior", toda la información necesaria para generar el valor actual.
- ¿Cómo podemos "codificar" varios valores en uno sólo?
- Quizás con una función auxiliar que guarde la información que necesito…

Solución

- Armar una auxiliar F que "codifique" la información necesaria (los 3 pasos anteriores)
- Probar que F es p.r.
- Reescribir f como composición de funciones pr (usando F)
- Probar que f es p.r.

Solución

Sea $F : \mathbb{N} \longrightarrow \mathbb{N}$ definida por $F(n) = [f(n), f(n+1), f(n+2)]$.

F se ajusta al esquema de recursión primitiva:

$F(0) = [f(0), f(1), f(2)] = [1,2,4]$ (una constante, es p.r.)

$F(n+1) = [f(n+1), f(n+2), f(n+3)]$ (por definición de F)

$= [f(n+1), f(n+2), \mathbf{f(n) + f(n+1)^2 + f(n+2)^3}]$ (por definición de f)

$= [\mathbf{F(n)}[2], \mathbf{F(n)}[3], \mathbf{F(n)}[1] + (\mathbf{F(n)}[2])^2 + (\mathbf{F(n)}[3])^3]$ (por definición de F)

(composición, sumas, potencias, crear y observar listas y todo sobre F(n))

Solución

- Sabemos que F es p.r.
- Reescribimos f:
  - f(n) = F(n)[1]
- f es composición de funciones p.r. (F y [.])
- Por lo tanto, f es p.r.

□

Conclusiones

- Armamos una auxiliar, F(n) = [f(n),f(n+1),f(n+2)]
  - Vimos que F es p.r.
  - Reescribimos f en función de F: f(n) = F(n)[1]
  - Vimos que f es p.r.
- ¿Alcanza con decir que esta definición es p.r.? ¿La otra no lo era?
  - Recordemos el primer ejemplo f(0)=0 ; f(n+1) = f(n)
  - No son dos funciones distintas. Son dos formas de describir a la misma función. Que a primera vista no se ajuste al esquema no quiere decir que no sea p.r.
    - f(n) = g(n) - g(n) es pr (es 0) aunque g no sea p.r.
  - Pensemos a las funciones como objetos abstractos. Una misma función se puede describir de varias formas

Conclusiones

- ¿Se les ocurre cómo resolverlo con otra cantidad de pasos anteriores?
    - ¿Y con una cantidad variable?
        - Ejercicio 14 de la práctica
- Supongamos que a la definición recursiva de f le agrego "+ g(n)"
    - ¿Puedo probar que f es pr?
        - No, porque no sé nada sobre g
    - ¿Y si tengo que g $\in$ C, una clase PRC?
        - Tampoco
        - Lo que sí puedo probar es que pertenece a la misma clase que g
        - ¿Por qué? f es composición/recursión de funciones en C y C está cerrada por composición y recursión => f está en C
        - **¿Cómo cambia la demostración?**

Ejercicio 2

Ejercicio 2 - a

Sea $h : \mathbb{N} \to \mathbb{N}$ una función primitiva recursiva. Probar que también es primitiva recursiva la función $map_h : \mathbb{N} \to \mathbb{N}$ que interpreta la entrada como una lista (sin ceros al final) y devuelve la lista con el resultado de aplicar h a cada uno de sus elementos. Por ejemplo, si x2(n) = 2*n, map_{x2} ([1, 2, 3, 5]) = [2, 4, 6, 10].

A pensar...

Solución

- Abusando de la notación, map□(L) = [h(L[1]), h(L[2]), …, h(L[|L|])]
- Recordar: Las listas **son** números

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} p_i^{a_i},$$

- Siendo p_i el i-ésimo primo

$$map_f(l) = \prod_{i=1}^{|l|} p_i^{f(l[i])}$$

$$h(y, x_1, \ldots, x_n) = \prod_{t=0}^{y} f(t, x_1, \ldots, x_n)$$

¿Intervalo?

Ejercicio 2 - b

Sea $p : \mathbb{N} \rightarrow \{0,1\}$ un predicado primitivo recursivo. Probar que también es primitiva recursiva la función $\text{indice}_p : \mathbb{N}^2 \rightarrow \mathbb{N}$ tal que $\text{indice}_p(l, i) = n$ si n es la posición del $i+1$-ésimo elemento de la lista (sin ceros al final) codificada por l que cumple con P. En caso de que no exista un $i+1$-ésimo elemento tal, $f(l, i)$ devuelve 0 (incluso si $l = 0$ que no codifica ninguna lista). Por ejemplo, si $par(x)$ determina si x es par, entonces:

$\text{indice}_{par}([1,8,3,6,6], \mathbf{0}) = 2$
(el **primer** par (8) está en la posición 2)

$\text{indice}_{par}([1,8,3,6,6], \mathbf{1}) = 4$
(el **segundo** par (6) está en la posición 4)

$\text{indice}_{par}([1,8,3,6,6], 2) = 5$
(el **tercer** par (6) está en la posición 5)

$\text{indice}_{par}([1,8,3,6,6], 3) = 0$
(no hay **cuarto** par)

A pensar...

Pistas

- Pensarlo recursivamente
- ¿Qué necesitamos para el caso base?
  - El “primer” elemento que cumple p(x) … ¿les suena?

Minimización acotada

Sea $p : \mathbb{N}^{n+1} \longrightarrow \{0, 1\}$ un predicado de una clase PRC C. También está en C la función

$$g(y, x_1, \ldots, x_n) = \min_{t \leq y} p(t, x_1, \ldots, x_n)$$

A pensar...

- Pensarlo recursivamente
- ¿Qué necesitamos para el caso base?
    - El “primer” elemento que cumple p(x) … ¿les suena?
- ¿Qué necesitamos para el caso recursivo?
    - Recordar que tenemos “gratis” el resultado para n
        - Es decir, podemos asumir que conocemos el índice del “anterior” que cumple p

No me interesa

[…, 1, 0, 0, …, 0, 1, …]

i-ésimo que cumple
(ya lo tengo)

Ninguno cumple

i+1-ésimo que cumple
(el que quiero obtener)

¡Atención!

- No confundir índice en la lista con i-ésimo que cumple
  - Los índices en la lista van de 1 a n inclusive
  - Al hablar del i-ésimo que cumple, el **primero** es el **0**
    - El rango está incluido en [0, .., n)

lista	[	6	5	8	6	1	]
p(x)		1	0	1	1	0	
índice en la lista		1	2	3	4	5	
i-ésimo que cumple		0		1	2		
		(primero)		(segundo)	(tercero)		

Solución

- Caso base: el mínimo índice i tal que p(l[i])
    - $\text{indice}\square(l, 0) = \min_{1 \leq t \leq |l|} p(\, l[t] \,)$
- Caso recursivo (teniendo el índice i del anterior): el mínimo índice t tal que p(l[t]) **y además** t > i
    - $\text{indice}\square(l, i + 1) = \min_{1 \leq t \leq |l|} p(\, l[t] \,) \land t > \text{indice}\square(l, i)$
- La minimización acotada, los observadores de listas, p y la conjunción son todas p.r.
- Esquema de recursión primitiva a partir de funciones p.r.
    - Entonces, es p.r.

□

Ejercicio 2 - c

Sea $p : \mathbb{N} \rightarrow \{0,1\}$ un predicado primitivo recursivo. Probar que también es primitiva recursiva la función filter□ : $\mathbb{N} \rightarrow \mathbb{N}$ que interpreta la entrada como una lista (sin ceros al final) y devuelve la lista con los elementos de la misma que cumplen con p (en el mismo orden que estaban en la lista original). filter(0) se define como 1. Por ejemplo, si par(x) determina si x es par, entonces filter□ar([1,8,3,6,6]) = [8,6,6] y filter□ar([1,3,3,7,9]) = [].

A pensar...

Pistas

- ¿Podemos usar algo que hayamos definido en un ejercicio anterior?
  - ¿Índice□?
- ¿Hay alguna forma de describir cada elemento en la lista resultante?
  - Definir la lista por extensión, como hicimos con map□

Solución

filter□(L) = [L[indice□(L, 0)], L[indice□(L, 1)], …, L[indice□(L, |L|)]]

- ¿Hay algún problema? ¿Qué longitud tiene el resultado?
  - filter□$_{ar}$([1,2,3]) = [2]
- ¿Entonces? ¿Estoy agregando elementos de más?
- ¿Qué devuelve indice□(x, |L|) si hay menos de x elementos el L que cumplen p?
  - Exacto, 0
  - Tenemos una lista con ceros al final
    - Si sólo hay m (< |L|) que cumplen, tenemos:
    - [L[indice□(L, 0)], …, L[indice□(L, m-1)], 0, …, 0] =
    - [L[indice□(L, 0)], …, L[indice□(L, m-1)]]

Solución

filter□(L) = [L[indice□(L, 0)], L[indice□(L, 1)], …, L[indice□(L, |L|)]]

- Nos “aprovechamos” de la representación
- Finalmente (al igual que como definimos map□),
- Ya demostramos que todas las funciones involucradas son p.r.
  - Luego, filter□ es p.r.

□

Ejercicio 3

Ejercicio 3

El problema subset sum (suma de subconjunto) se define de la siguiente manera: dado un conjunto S de números enteros y un entero k, decidir si existe un subconjunto de S tal que la suma de sus elementos sea exactamente k. Probar que es primitivo recursivo el predicado subsetSum(l, k) : $\mathbb{N} \longrightarrow \{0, 1\}$ que decide si, dada una lista de naturales l y un natural k, existe una sublista de l cuyos elementos sumen exactamente k.

A pensar...

… qué nos está pidiendo.

$$\exists_{l' \subseteq l} \; suma(l') = k$$

(interpretando ⊆ como “sublista de”)

… cómo demostramos que es p.r.

- La suma es p.r.
- La igualdad es p.r.
- El existencial ...

Cuantificadores acotados

Sea $p : \mathbb{N}^{n+1} \longrightarrow \{0, 1\}$ un predicado perteneciente a una clase PRC C.

También está en C el predicado:

$$(\exists t)_{\leq y}\, p(t, x_1, \ldots, x_n).$$

A pensar...

- ¿Podemos expresar $(\exists l') \subseteq l$ como $(\exists t) \leq y$?
  - Recordar (otra vez): Las listas **son** números
- ¿$l' \leq l$?
  - Esto recorre todas las listas que se codifican con números menores al número que codifica a la lista l.
  - ¿Será cierto que $l' \leq l \Leftrightarrow l' \subseteq l$?
  - $l' \leq l \Longleftarrow l' \subseteq l$ es cierto.
    - Demo: Álgebra. Tarea.
  - $l' \leq l \Longrightarrow l' \subseteq l$ **NO** es cierto.
    - Demo: Contraejemplo. Tarea.

A pensar...

- Entonces, esto no sirve:

$$subsetSum(l,k) = \exists_{0 \leq l' \leq l} \; suma(l') = k$$

- ¿O Sí? ¿Podemos arreglarlo?

$$subsetSum(l,k) = \exists_{0 \leq l' \leq l} \; esSublista(l', l) \wedge suma(l') = k$$

- Tarea: Demostrar que esSublista es p.r.

Solución

$$subsetSum(l,k) = \exists_{0 \leq l' \leq l} esSublista(l',l) \wedge suma(l') = k$$

Conclusiones

- Este problema es uno de los tantos considerados “difíciles” pues no se conoce un algoritmo polinomial para resolverlo (Algo 3)
    - Sin embargo, es fácil ver que es primitivo recursivo
    - Ojo con confundir “computable” con “complejo”
- ¿Complejidad?
    - Notar que la cota $l' \leq l$ es bastante grosera
    - ¿Qué complejidad? estamos definiendo funciones matemáticas
    - No nos importa que las cotas sean absurdamente grandes mientras sean cotas y p.r.
    - De vuelta, no estamos programando, estamos definiendo qué puede ser programado

Ya pueden completar la práctica 1

¿Dudas?