Trabajo Práctico 2 Derivación Programas Funcionales

1. Dado el **programa**

$$\begin{array}{|l}
\text{sum} : [Num] \to Num \\
\hline
\text{sum}.[\,] \doteq 0 \\
\text{sum}.(x \triangleright xs) \doteq x + sum.xs
\end{array}$$

   a) ¿Qué hace esta función? Escriba en lenguaje natural el **problema** que resuelve.

   b) Escriba una **especificación** de la función con una expresión cuantificada.

   c) **Verifique** que esta especificación vale para toda lista xs.

   d) Ahora **derive** la definición de la función a partir de su especificación.
   ¿Esta derivación es parecida a la demostración en el punto $1c$?

a) El problema es sumar todos los elementos de una lista.

b) sum.xs ≐ ⟨ Σ i : 0 ≤ i < #xs : xs ! i ⟩

c) La verificación es por inducción.

<u>Caso base</u><u>, xs = []</u>:

sum.[ ] = ⟨ Σ i : 0 ≤ i < <u>#xs</u> : xs ! i ⟩

≡ { #xs = 0 }

sum.[ ] = ⟨ Σ i : <u>0 ≤ i < 0</u> : xs ! i ⟩

≡ { lógica }

sum.[ ] = <u>⟨ Σ i : False : xs ! i ⟩</u>

≡ { rango vacío }

<u>sum.[ ]</u> = 0

≡ { definición de sum }

0 = 0

≡ { lógica }

True

<u>Hipótesis Inductiva:</u>

sum.xs = ⟨ Σ i : 0 ≤ i < #xs : xs ! i ⟩

<u>Caso inductivo:</u>

sum.(x▹xs) = ⟨ Σ i : 0 ≤ i < <u>#(x▹xs)</u> : (x▹xs) ! i ⟩

≡ { #(x▹xs) = #xs + 1 }

sum.(x▹xs) = <u>⟨ Σ i : 0 ≤ i < #xs + 1 : (x▹xs) ! i ⟩</u>

$\equiv$ { separación de un término }

$sum.(x \triangleright xs) = \underline{(x \triangleright xs) ! 0} + \langle \Sigma i : 0 < i < \#xs+1 : (x \triangleright xs) ! i \rangle$

$\equiv$ { evaluación de índice }

$sum.(x \triangleright xs) = x + \langle \Sigma i : \underline{0 < i < \#xs+1} : (x \triangleright xs) ! i \rangle$

$\equiv$ { reescribo el rango }

$sum.(x \triangleright xs) = x + \underline{\langle \Sigma i : 1 \leq i < \#xs+1 : (x \triangleright xs) ! i \rangle}$

$\equiv$ { cambio de variable, f.j = j+1 }

$sum.(x \triangleright xs) = x + \langle \Sigma j : \underline{1 \leq j+1 < \#xs+1} : \underline{(x \triangleright xs) ! (j+1)} \rangle$

$\equiv$ { resto 1 en la desigualdad, reescribo el índice }

$sum.(x \triangleright xs) = x + \underline{\langle \Sigma j : 0 \leq j < \#xs : xs ! j \rangle}$

$\equiv$ { Hipótesis Inductiva }

$\underline{sum.(x \triangleright xs)} = x + sum.xs$

$\equiv$ { definición de sum }

$\underline{x + sum.xs = x + sum.xs}$

$\equiv$ { lógica }

True

d)

$sum.xs \doteq \langle \Sigma i : 0 \leq i < \#xs : xs ! i \rangle$

Caso base:

$\underline{sum.[\,]}$

$\equiv$ { definición de sum }

$\langle \Sigma i : 0 \leq i < \underline{\#[\,]} : [\,] ! i \rangle$

$\equiv$ { definición de cardinal }

$\langle \Sigma i : \underline{0 \leq i < 0} : [\,] ! i \rangle$

$\equiv$ { lógica }

$\underline{\langle \Sigma i : False : [\,] ! i \rangle}$

$\equiv$ { rango vacío }

0

Hipótesis inductiva:

$\mathbf{sum.xs \doteq \langle \Sigma i : 0 \leq i < \#xs : xs ! i \rangle}$

Paso inductivo:

$\underline{sum.(x \triangleright xs)}$

≡ { definición de sum }

$\langle\ \Sigma\ i : \underline{0 \le i < \#(x \triangleright xs)} : (x \triangleright xs) \,!\, i\ \rangle$

≡ { reescribo el rango }

$\underline{\langle\ \Sigma\ i : i = 0\ \vee\ 1 \le i < \#(x \triangleright xs) : (x \triangleright xs) \,!\, i\ \rangle}$

≡ { partición de rango (los rangos son disjuntos) }

$\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle + \langle\ \Sigma\ i : 1 \le i < \underline{\#(x \triangleright xs)} : (x \triangleright xs) \,!\, i\ \rangle$

≡ { definición de cardinal }

$\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle + \underline{\langle\ \Sigma\ i : 1 \le i < \#xs + 1 : (x \triangleright xs) \,!\, i\ \rangle}$

≡ { cambio de variable, f.j = j + 1 }

$\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle + \langle\ \Sigma\ j : \underline{1 \le j + 1 < \#xs + 1} : (x \triangleright xs) \,!\, j + 1\ \rangle$

≡ { álgebra }

$\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle + \langle\ \Sigma\ j : 0 \le j < \#xs : \underline{(x \triangleright xs) \,!\, j + 1}\ \rangle$

≡ { definición de índice }

$\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle + \underline{\langle\ \Sigma\ j : 0 \le j < \#xs : xs \,!\, j\ \rangle}$

≡ { Hipótesis Inductiva }

$\underline{\langle\ \Sigma\ i : i = 0 : (x \triangleright xs) \,!\, i\ \rangle} + sum.xs$

≡ { rango unitario }

$\underline{(x \triangleright xs) \,!\, 0} + sum.xs$

≡ { evalúo el índice }

$x + sum.xs$

El programa resulta ser:

sum :: [Int] → Int

sum.[ ] ≐ 0

sum.(x▹xs) ≐ x + sum.xs

2. A partir de las siguientes especificaciones, dar el tipo de cada función y *derivar* las soluciones algorítmicas correspondientes.

   a) $\text{sum\_cuad}.xs = \langle \sum i\ :\ 0 \le i < \#xs\ :\ xs\,!\,i * xs\,!\,i \rangle$
   b) $\text{iga}.e.xs = \langle \forall i\ : 0 \le i < \#xs :\ xs\,!\,i = e\,\rangle$
   c) $\text{exp}.x.n = x^n$
   d) $\text{sum\_par}.n = \langle \sum i\ :\ 0 \le i \le n \wedge par.i\ :\ i \rangle$
   donde $par.i \doteq i \bmod 2 = 0$.
   e) $\text{cuántos}.p.xs = \langle \text{N}\ i\ :\ 0 \le i < \#xs\ :\ p.(xs\,!\,i) \rangle$
   f) $\text{busca}.e.xs = \langle \text{Min}\ \ i\ : 0 \le i < \#xs \wedge xs\,!\,i = e :\ i \rangle$

a)

sum.cuad xs = 〈 ∑ i : 0 ≤ i ≤ #xs : xs ! i * xs ! i 〉

Caso Base:

sum.cuad [ ]

≡ {Aplico especificación}

〈 ∑ i : 0 ≤ i ≤ **#[ ]** : [ ] ! i * [ ] ! i 〉

≡ {Cardinal de [ ]}

〈 ∑ i : **0 ≤ i ≤ 0**: [ ] ! i * [ ] ! i 〉

≡ {Lógica}

〈 ∑ i :**False** : [ ] ! i * [ ] ! i 〉

≡ {Rango vacío}

0

Caso inductivo:

H.I: **〈 ∑ i : 0 ≤ i ≤ #xs : xs ! i * xs ! i 〉**

sum.cuad (x:xs)

≡ {Aplico especificación}

〈 ∑ i : 0 ≤ i ≤ **#(x : xs)** : (x :xs) ! i * (x :xs) ! i 〉

≡ {Cardinal de x:xs}

〈 ∑ i : **0 ≤ i ≤ #xs + 1**: (x :xs) ! i * (x :xs) ! i 〉

≡ {Lógica}

〈 ∑ i :**i = 0 ∨ 1 ≤ i ≤ #xs + 1** : (x :xs) ! i * (x :xs) ! i 〉

≡ {Partición de Rango}

〈∑ i :i = 0 : (x :xs) ! i * (x :xs) ! i 〉+ 〈 ∑ i : 1 ≤ i ≤ #xs + 1 : (x :xs) ! i * (x :xs) ! i 〉

≡ {Rango Unitario}

(x :xs) ! 0 * (x :xs) ! 0 +〈 ∑ i : 1 ≤ i ≤ #xs + 1 : (x :xs) ! i * (x :xs) ! i 〉

≡ { Def. de Cardinal}

x * x +〈 ∑ i : 1 ≤ i ≤ #xs + 1 : (x :xs) ! i * (x :xs) ! i〉

≡ { Cambio de variable i = i + 1}

x * x +〈 ∑ i : **1 ≤ i + 1 ≤ #xs + 1** : (x :xs) ! (i + 1) * (x :xs) ! (i + 1) 〉

≡ {resto 1 en los términos del rango}

$x * x + \langle \sum i : 0 \leq i \leq \#xs : \underline{\mathbf{(x :xs) \,!\, (i + 1) * (x :xs) \,!\, (i + 1)}} \rangle$

≡ {indexar}

$x * x + \underline{\mathbf{\langle \sum i : 0 \leq i \leq \#xs : xs \,!\, i * xs \,!\, i \rangle}}$

≡ { HI}

x * x + sum.cuad.xs

Finalmente:

sum.cuad: [Int] -> Int

sum.cuad [ ] = 0

sum.cuad xs = (x^2) + sum.cuad xs

b)

iga e xs = $\langle \forall \mathbf{i: 0 \leq i < \#xs : xs \,!\, i = e} \rangle$

iga: Int -> [Int] -> Bool

Caso base:

iga.e.[]

≡ {aplicó especificación}

$\langle \forall i: 0 \leq i < \underline{\mathbf{\#[\ ]}} : [\ ] \,!\, i = e \rangle$

≡ {Cardinal de [ ]}

$\langle \forall i: \underline{\mathbf{0 \leq i < 0}} : [\ ] \,!\, i = e \rangle$

≡{Lógica}

$\langle \forall i: \underline{\mathbf{false}} : [\ ] \,!\, i = e \rangle$

≡{Rango vacío}

<u>True</u>

Caso Inductivo:

HI: iga e xs = $\langle \forall \mathbf{i: 0 \leq i < \#xs : xs \,!\, i = e} \rangle$

$\underline{\mathbf{iga.e.(x \blacktriangleright xs)}}$

≡ {Especificación de iga}

$\langle \forall i: 0 \leq i < \underline{\mathbf{\#(x \blacktriangleright xs)}} : (x \blacktriangleright xs)! \ i = e \rangle$

≡ { Def de #}

$\langle \forall i: 0 \leq i < \underline{\mathbf{1 + \#xs}} : (x \blacktriangleright xs)! \ i = e \rangle$

≡ {Lógica}

$\langle \forall i: \underline{\mathbf{0 = i \lor 1 \leq i < 1 + \#xs}} : (x \blacktriangleright xs)! \ i = e \rangle$

≡ {Partición del rango}

$\underline{\mathbf{\langle \forall i: 0 = i : (x \blacktriangleright xs)! \ i = e \rangle}} \land \langle \forall i: 1 \leq i < 1 + \#xs : (x \blacktriangleright xs)! \ i = e \rangle$

≡ {Rango unitario}

(x▶xs)! 0 = e ∧⟨∀i: 1 ≤ i < 1 + #xs : (x▶xs)! i = e⟩

≡ {Def de !}

(x = e) ∧⟨∀i: 1 ≤ **i** < 1 + #xs : (x▶xs)! **i** = e⟩

≡ {Cambio de variable i = i+1}

(x = e) ∧⟨∀i: **1 ≤ i + 1 < 1 + #xs** : (x▶xs)!(i+1) = e⟩

≡ {Restamos -1 en cada término del rango}

(x = e) ∧⟨∀i: 0 ≤ i < #xs : **(x▶xs)! (i+1)** = e⟩

≡ {Def de !}

(x = e) ∧ **⟨∀i: 0 ≤ i < #xs : (xs)! i = e⟩**

≡ {H}

(x = e) ∧ (iga.e.xs)

Finalmente:

iga e xs : Int -> [Int] -> Bool

iga e [ ] = True

iga e (x:xs) = (x = e) Λ iga e xs

c)

exp.x.n = x^n

Tipo : Int -> Int -> Int

Caso base

exp.x.0

≡ { especificación }

x^0

= { aritmética }

1

Paso inductivo

HI: exp.x.n = x^n

exp.x.(n+1)

= { especificación }

x^{n+1}

= { propiedad de exponenciación }

x * $\underline{\mathbf{x^n}}$

= { HI }

x * exp.x.n

Finalmente

exp :: Int -> Int -> Int

exp.x.0 = 1

exp.x.(n+1) = x * exp.x.(n)

d)

sumPar.n = ⟨ Σi : 0 ≤ i ≤ n ∧ par.i : i⟩ donde **par.i = i mod 2 = 0**.

Tipo: Int -> Int

Caso base

sumPar.0

≡{especificación}

⟨ ∑i : **0 ≤ i ≤ 0** ∧ par.i : i⟩

≡{lógica}

⟨ ∑i : **i=0 ∧ par.i** : i⟩

≡{Leibniz 2}

⟨ ∑i : i=0 ∧ **par.0** : i⟩

≡{def de par (**par.0 = 0 mod 2 = = 0 = True**)}

⟨ ∑i : i=0 ∧ **True** : i⟩

≡{Neutro conj.}

⟨ ∑i : i=0 : i⟩

≡{Rango unitario}

0

Paso inductivo,

HI: sumPar.n = ⟨ Σi : 0 ≤ i ≤ n ∧ par.i : i⟩

sumPar.(n+1)

≡ { especificación }

⟨ Σi : **0 ≤ i ≤ (n+1)** ∧ par.i : i⟩

≡ { 0 ≤ i ≤ (n+1) ≡ 0 ≤ i ≤ n ∨ i = n+1 }

⟨ Σi : **(0 ≤ i ≤ n ∨ i = n+1) ∧ par.i** : i⟩

≡ { distribución de $\wedge$ respecto con $\vee$ }

$\langle$ Σi : (**(0 ≤ i ≤ n) ∧ par.i) ∨ ((i = n+1) ∧ par.i)** : i$\rangle$

≡ { partición de rango }

$\langle$ Σi : (0 ≤ i ≤ n) ∧ par.i : i$\rangle$ + $\langle$ Σi : ((i = n+1) ∧ par.i) : i$\rangle$

≡ { HI }

sumPar.n + **$\langle$ Σi : (i = n+1) ∧ par.i : i$\rangle$**

≡ { rango unitario y condición }

sumPar.n + |par.(n+1) => n+1

|¬par.(n+1) => 0

sumPar :: Int -> Int

sumPar.0 = 0

sumPar.(n+1)

| (n+1) mod 2 == 0 = n + 1 + sumPar.n

| (n+1) mod 2 ≠ 0 = sumPar.n

e)

cuantos.p.xs = $\langle$N i : 0 ≤ i < #xs : p.(xs !i)$\rangle$

Tipo : (a -> Bool) -> [a] -> Int

Caso Base

cuantos.p.[ ]

≡{ especificación}

$\langle$N i : 0 ≤ i < **#[ ]** : p.([] !i)$\rangle$

≡ {def # }

$\langle$N i : **0 ≤ i < 0** : p.([] !i)$\rangle$

≡{ lógica }

$\langle$N i : **False** : p.([] !i)$\rangle$

≡ { rango vacío }

0

Paso inductivo:

HI : cuantos.p.xs = $\langle$N i : 0 ≤ i < #xs : p.(xs !i)$\rangle$

cuantos.p.(x:xs)

≡ { especificación }

⟨N i : **0 ≤ i < #(x:xs)** : p.((x:xs) !i)⟩

≡{ 0 ≤ i < #(x:xs) ≡ **i=0 ∨ 1 ≤ i < #xs+1**}

⟨N i : **i=0 ∨ 1 ≤ i < #xs+1** : p.((x:xs) !i)⟩

≡{ separación de rango }

⟨N i : i=0 : p.((x:xs) !i)⟩ + **⟨N i :1 ≤ i < #xs+1 : p.((x:xs) !i)⟩**

≡ { cambio de variable i = i +1 }

⟨N i : i=0 : p.((x:xs) !i)⟩ + ⟨N j :1 ≤ i +1 < #xs+1 : **p.((x:xs) ! (i+1)**⟩

≡ { def de ! }

⟨N i : i=0 : p.((x:xs) !i)⟩ + ⟨N j :**1 ≤ i +1 < #xs+1** : p.(xs!j)⟩

≡ { Resto 1 a cada término del rango }

⟨N i : i=0 : p.((x:xs) !i)⟩ + **⟨N j :0 ≤ j < #xs : p.(xs!j)⟩**

≡ { HI }

⟨N i : i=0 : p.((x:xs) !i)⟩ + cuantos.p.xs

≡ { Rango unitario de Conteo }

cuantos.p.xs + |p (x:xs)!0 -> 1

|□¬p (x:xs)!0 -> 0

≡ { def. de !}

|p.x -> 1 + cuantos.p.xs

|¬p.x-> 0 +cuantos.p.xs

```
cuantos :: (a -> Bool) -> [a] -> Int
cuantos.p.[ ] = 0
cuantos.p.(x:xs)
                 | p.x = 1 + cuantos.p.xs
                 | ¬ p.x = cuantos.p.xs
```

f)

busca.e.xs = ⟨Min i : 0 ≤ i < #xs ∧ xs !i = e : i ⟩

Tipo: Int -> [Int] -> Int

Caso base

$busca.e.[\,]$

$\equiv$ { especificación }

$\langle Min\ i : 0 \le i < \underline{\#[\,]} \wedge [\,]\,!i = e : i \rangle$

$\equiv$ { def # }

$\langle Min\ i : \underline{0 \le i < 0 \wedge [\,]\,!i = e} : i \rangle$

$\equiv$ { lógica }

$\underline{\langle Min\ i : False : i \rangle}$

{Rango vacio}

+inf

Paso inductivo :

HP : $busca.e.xs = \langle Min\ i : 0 \le i < \#xs \wedge xs\,!i = e : i \rangle$

$\underline{busca.e.(x:xs)}$

= { especificación }

$\langle Min\ i : \underline{0 \le i < \#(x : xs)} \wedge (x : xs)\,!i = e : i \rangle$

= { $0 \le i < \#(x:xs) \equiv i=0 \vee 1 \le i < \#xs+1$ }

$\langle Min\ i : \underline{(i=0 \vee 1 \le i < \#xs+1) \wedge (x:xs)\,!i} = e : i \rangle$

= { distribución de $\wedge$ respecto con $\vee$ }

$\langle Min\ i : \underline{(i=0 \wedge (x:xs)\,!i = e) \vee (1 \le i < \#xs+1 \wedge (x:xs)\,!i = e)} : i \rangle$

= { partición de rango }

$\langle Min\ i : i=0 \wedge (x:xs)\,!i = e : i \rangle\ min\ \underline{\langle Min\ i : 1 \le i < \#xs+1 \wedge (x:xs)\,!\,i = e : i \rangle}$

= { cambio de variable $i = j + 1$ }

$\langle Min\ i : i=0 \wedge (x:xs)\,!i = e : i \rangle\ min\ \langle Min\ j : \underline{1 \le j+1 < \#xs+1 \wedge (x:xs)\,!\,(j+1) = e} : (j+1) \rangle$

= { lógica y def de ! }

$\langle Min\ i : i=0 \wedge (x:xs)\,!i = e : i \rangle\ min\ \underline{\langle Min\ j : 0 \le j < \#xs \wedge xs\,!\,j = e : (j+1) \rangle}$

= { distributividad, el neutro de min (+inf) es absorbente en + }

$\langle Min\ i : i=0 \wedge (x:xs)\,!i = e : i \rangle\ min\ (\underline{\langle Min\ j : 0 \le j < \#xs \wedge xs!j = e : j \rangle} +1)$

= { HI }

$\underline{\langle Min\ i : i=0 \wedge (x:xs)\,!i = e : i \rangle}\ min\ (busca.e.xs + 1)$

= { rango unitario y condición y def !}

|x = e => 0

|¬x = e => +inf

busca :: Int -> [Int] -> Int

busca.e.[ ] = maxBound

busca.e.(x:xs)

| x == e = 0

| x /= e = busca.e.xs +1

3. Para todos los items del ejercicio anterior, dar un ejemplo de uso de la función, es decir: elegir valores concretos para los parámetros y calcular el resultado usando la solución algorítmica obtenida. Las listas deben tener por lo menos tres elementos.

a)
sum.cuad: [Int] -> Int
sum.cuad [ ] = 0
sum.cuad xs = (x^2) + sum.cuad xs
Ejemplo: [2,3,0,4]
sum.cuad [2,3,0,4] =
(2^2) + sum.cuad [3,0,4]
4 + (3^2)+ sum.cuad [0,4]
4 + 9 + (0^2) + sum.cuad [4]
4 + 9 + 0 + (4^2) + sum.cuad [ ]
4 + 9 + 0 + 16 + sum.cuad [ ]
4 + 9 + 0 + 16 + 0 = 29

b)
iga.e.xs : Int -> [Int] -> Bool
iga .e.[ ] = True
iga.e.(x:xs) = (x = e) Λ iga e xs
Ejemplo:[4,4,5,2] e=4
iga 4 [4,4,5,2] =
(4 = 4) Λ iga 4 [4,5,2]
True Λ (4 = 4) Λ iga 4 [5,2]
True Λ True Λ (5 = 4) Λ iga 4 [2]
True Λ True Λ False Λ (2= 4) Λ iga 4 [ ]
True Λ True Λ False Λ False Λ iga 4 [ ]
True Λ True Λ False Λ False Λ True
=False

c)
exp.x.n : Int -> Int -> Int (siendo el primer Int ≠ 0)
exp x.0 = 1
exp.x.n = x * exp x (n-1)
Ejemplo 2^5
exp.2.5 =
2 * exp 2 (5-1)

= 2 * exp 2 (4)
2 * 2 * exp x (4-1)
= 2 * 2 * exp x (3)
2 * 2 * 2 * exp x (3-1)
=2 * 2 * 2 * exp x (2)
2 * 2 * 2 *2 * exp x (2-1)
=2 * 2 * 2 *2 * exp x (1)
2 * 2 * 2 *2 * 2 *exp x (1-1)
=2 * 2 * 2 *2 * 2 *exp x (0)
2 * 2 * 2 *2 * 2 *1
== 32

d)
sum par n :: Int -> Int
sum par 0 = 0
sum par (n+1) | n mod 2 == 0 = n + sumpares (n-1)
| n mod 2 /= 0 = sumpares (n-1)
Ejemplo: 4
4 mod 2 ==0 =True
4 + sumapares (4-1)
3 mod 2 ==0 =False
4 + sumapares (3-1)
2 mod 2 == 0 = True
2 + 4 + sumapares (2-1)
1 mod 2 ==0 = False
2 + 4 + sumapares (1-1)
{Caso base}
0+2+4
=6

e)
cuantos p xs :: [Int] -> Int
cuantos p [ ] = 0
cuantos p (x:xs)
| p = = x = 1 + cuantos p xs
| p ≠ x = cuantos p xs
p.x ≐ x mod 2 = 0
xs = [2,4,5]

cuantos.(x mod 2 = 0).[2,4,5] → 1 + cuantos.(x mod 2 = 0).[4,5]
→ 1 + 1 + cuantos.(x mod 2 = 0).[5]
→ 1 + 1 + cuantos.(x mod 2 = 0).[ ]
→ 1 + 1 + 0
→ 2

f)

busca :: Int -> [Int] -> Int
busca.e.[ ] = maxBound
busca.e.(x:xs)
| x == e = 0
| x /= e = busca.e.xs +1

e = 5
xs = [1,2,3,4,5]

busca.5.[1,2,3,4,5] → busca.5.[2,3,4,5] + 1
→ busca.5.[3,4,5] + 1 + 1
→ busca.5.[4,5] + 1 + 1 + 1
→ busca.5.[5] + 1 + 1 + 1 + 1
→ 0 + 1 + 1 + 1 + 1
→ 4

4. Derivar las siguientes funciones.

a) sum_pot : $Num \to Nat \to Num$ computa la suma de potencias de un número, esto es

$$\text{sum\_pot}.x.n = \langle \sum i \ : 0 \leq i < n : \ x^i \rangle \ .$$

b) cos' : $Nat \to Num \to Num$ computa la aproximación del coseno del segundo argumento.

$$cos'.n.x \doteq \left\langle \sum i \ : 0 \leq i \leq n : \ (-1)^i * \frac{x^{2*i}}{(2*i)!} \right\rangle$$

Ayuda: Modularizar dos veces. La segunda con la función *exp* del ejercicio 2*c* y factorial.

c) cubo : $Nat \to Nat$ computa el cubo ($\text{cubo}.x = x^3$) de un número natural x utilizando únicamente sumas.

Ayuda: Usar inducción y modularización una o más veces.

d) prod_suf: $[Num] \to Bool$ decide si existe un elemento igual al producto de los elementos que le siguen:

$$\text{prod\_suf}.xs = \left\langle \exists i \ : 0 < i \leq \#xs : \ \left\langle \prod j \ : 0 \leq j < \#(xs{\downarrow}i) : \ (xs{\downarrow}i)\,!\,j \right\rangle = xs\,!\,(i-1) \right\rangle$$

•

a)

sum_pot : Num → Nat → Num

sum_por.x.n = 〈 ∑ i: 0 ≤ i ≤ n : x^i 〉

Caso Base:

sum_pot.x.0

≡ {especificacion}

〈 ∑ i : **<u>0 ≤ i ≤ 0</u>** : x^i〉

≡ { logica y rango vacio}

0

Paso Inductivo:

H.I = sum_pot.x.n = 〈 ∑ i : 0 ≤ i ≤ n : x^i 〉

sum_pot.x.(n +1)

≡ { especificacion}

〈 ∑ i : **0 ≤ i ≤ (n +1)** : x^i 〉

≡ {logica}

〈 ∑ i :**0 ≤ i < n** ∨ **i = n** : x^i 〉

≡ {Particion de rango porque son disjuntos}

〈 ∑ i :0 ≤ i < n : x^i 〉+ 〈 ∑ i : i = n : x^i 〉

≡ {Hipótesis Inductiva}

sum_pot.x.n + **〈 ∑ i : i = n : x^i 〉**

≡ {Rango Unitario}

sum_pot.x.n + **x^n**

≡ { x^n no es programable, así que MODULARIZAR, exp.x.n = x^n}

sum_pot.x.n + exp.x.n

⇒

sum_pot.x.0 = 0

sum_pot.x.(n+1) = sum_pot.x.n + exp.x.n

Ahora hay que derivar la función exp.x.n = x^n

Caso Base:

exp.x.0

≡ {especificación}

x^0

≡ {Aritmética}

1

Caso Inductivo:

H.I = exp.x.n = x^n

exp.x.(n+1)

≡{especificacion}

x^(n+1)

≡{Propiedad de la exponenciacion}

x^n * x

≡{Hipótesis Inductiva}

exp.x.n * x

⇒

exp.x.0 = 1

exp.x.(n+1) = exp.x.n * x

Finalmente:

sum_pot.x.0 = 0

sum_pot.x.(n+1) = sum_pot.x.n + exp.x.n

exp.x.0 = 1

exp.x.(n+1) = exp.x.n * x

b)

cos’ :: Nat -> Num -> Num

cos’n.x = 〈 ∑ i : 0 ≤ i ≤ n : (-1)^i * x^(2*i) / (2 *i)! 〉

Inducción en n

caso base n=0

cos’.0.x

≡ { especificación }

〈 ∑ i : **0 ≤ i ≤ 0** : (-1)^i * x^(2*i) / (2 *i)! 〉

≡ { lógica }

〈 ∑ i : **i=0** : (-1)^i * x^(2*i) / (2 *i)! 〉

≡{ rango unitario }

(-1)^0 * x^(2*0) / (2 *0)!

≡ { aritmética }

1 * x^0 / 0!

≡ { aritmética }

1

Caso recursivo

HI = cos’n.x = 〈 ∑ i : 0 ≤ i ≤ n : (-1)^i * x^(2*i) / (2 *i)! 〉

cos’.(n+1).x

≡ { especificación }
$\langle \sum i : \underline{\mathbf{0 \le i \le n+1}} : (-1)^i * x^{(2*i)} / (2*i)! \rangle$
≡ {lógica}
$\langle \sum i : \underline{\mathbf{0 \le i \le n \lor i = n+1}} : (-1)^i * (x^{(2*i)})/(2*i)! \rangle$
≡ {partición de rango}
$\langle \sum i : 0 \le i \le n : (-1)^i * (x^{(2*i)})/(2*i)! \rangle + \underline{\langle \sum \mathbf{i : i = n+1 : (-1)^i * (x^{(2*i)})/(2*i)!} \rangle}$
≡ {Rango Unitario}
$\underline{\langle \sum \mathbf{i : 0 \le i \le n : (-1)^i * x^{(2*i)} / (2*i)!} \rangle} + (-1)^{(n+1)} * x^{(2*(n+1))} / (2*(n+1))!$
= { HI }
cos'n.x + $\underline{\mathbf{(-1)^{(n+1)} * x^{(2*(n+1))}}} / (2*(n+1))!$
= { **exp.x.n** definida en 2c }
cos'n.x + exp.(-1).(n+1) * exp.x.(2n+2) / **__(2*(n+1))!__**
= {modularización fac especificada por **fac.n = n!**}
cos'n.x + exp.(-1).(n+1) * exp.x.(2n+2) / fac.(2n+2)

cos' :: Nat -> Num -> Num
cos'.0.x = 1
cos.(n+1).x = cos.n.x + exp.(-1).(n+1) * exp.x.(2n+2) / fac.(2n+2)

Modularizar Fac.n = n!

Caso base n=0
__fac.0__
= { especificación }
__0!__
= { prop factorial }
1

caso inductivo
HI : n! = fac.n
__fac.(n+1)__
= { especificación }
__(n+1)!__
= { prop factorial }
n+1 * **__n!__**
= { HI }

n+1 * fac.n

fac :: Nat -> Nat

fac.0 = 0

fac.(n+1) = (n+1) * fac.n

Finalmente:

cos' : Nat → Num → Num
cos'.0.x ≐ -exp.x.2
cos'.(n+1).x ≐ cos'.n.x + exp.(-1).(n+1) ∗ exp.x.(2∗(n+1))/factorial.(2∗(n+1))

exp :: Num → Nat → Num
exp.x.0 ≐ 1
exp.x.(n+1) ≐ x ∗ exp.x.n

fac : Nat → Nat
fac.0 ≐ 1
fac.(n+1) ≐ (n+1) ∗ fac.n

c)
cubo : Nat -> Nat
cubo.x = x^3

Hago inducción en x
Caso base x = 0
<u>cubo.0</u>
= { especificación }
<u>0^3</u>
= { aritmetica }
0

Caso inductivo
HI : cubo.x = x^3

<u>cubo.(x+1)</u>
≡ { especificación }
<u>(x+1)^3</u>
≡ { aritmética (binomio al cubo) }
(x+1) * (x+1) * (x+1) =
(x^2 + 2x + 1) * (x + 1) =
x^3 + 2x^2 + x + x^2 + 2x + 1 =
<u>x^3</u> + 3x^2 + 3x + 1
= { HI }

cubo.x + **<u>3x^2</u>** + 3x + 1
= { busco **modularización cuadrado** con **especificación <u>cuadrado.x = x^2</u>** (pero solo usando sumas) }
cubo.x + cuadrado.x + cuadrado.x + cuadrado.x + x + x + x + 1

=>

cubo :: Nat -> Nat
cubo.0 = 0
cubo.(x+1) = cubo.x + cuadrado.x + cuadrado.x + cuadrado.x + x + x + x + 1

Modularización cuadrado.x
Hago inducción en x

caso base x = 0
<u>cuadrado.0</u>
= { especificación }
<u>0^2</u>
= { aritmética }
0

Caso inductivo
H.I : cuadrado.x = x^2

<u>cuadrado.(x+1)</u>
= { especificación }
<u>(x+1) ^2</u>
= { aritmética (cuadrado binomial) }
<u>x^2</u> + 2x + 1
= { HI }
cuadrado.x + 2x + 1

=>

cuadrado :: Nat -> Nat
cuadrado.0 = 0
cuadrado.(x+1) = cuadrado.x + x + x + 1

Finalmente:

cubo :: Nat -> Nat
cubo.0 = 0
cubo.(x+1) = cubo.x + cuadrado.x + cuadrado.x + cuadrado.x + x + x + x + 1

cuadrado :: Nat -> Nat
cuadrado.0 = 0
cuadrado.(x+1) = cuadrado.x + x + x + 1

d)

prodSuf :: [Num] -> Bool
prodSuf.xs $\equiv \langle \exists i : 0 < i \leq \#xs : \langle \prod j : 0 \leq j < \#(xs{\downarrow}i) : (xs{\downarrow}i) ! j \rangle = xs ! (i - 1) \rangle$

Hago inducción en lista

Caso Base xs = [ ]

prodSuf.[ ]

$\equiv$ { especificación }

$\langle \exists i : 0 < i \leq \underline{\#[\,]} : \langle \prod j : 0 \leq j < \#([\,]{\downarrow}i) : ([\,]{\downarrow}i) ! j \rangle = [\,] ! (i - 1) \rangle$

$\equiv$ { def de # }

$\langle \exists i : \underline{0 < i \leq 0} : \langle \prod j : 0 \leq j < \#([\,]{\downarrow}i) : ([\,]{\downarrow}i) ! j \rangle = [\,] ! (i - 1) \rangle$

$\equiv$ { logica }

$\underline{\langle \exists i : False : \langle \prod j : 0 \leq j < \#([\,]{\downarrow}i) : ([\,]{\downarrow}i) ! j \rangle = [\,] ! (i - 1) \rangle}$

$\equiv$ { rango vacío }

False

Caso recursivo :

HI = prodSuf.xs $\equiv \langle \exists i : 0 < i \leq \#xs : \langle \prod j : 0 \leq j < \#(xs{\downarrow}i) : (xs{\downarrow}i) ! j \rangle = xs ! (i - 1) \rangle$

prodSuf.(x:xs)

$\equiv$ { especificación }

$\langle \exists i : 0 < i \leq \underline{\#(x:xs)} : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { def # }

$\langle \exists i : \underline{0 < i \leq \#xs + 1} : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { $0 < i \leq \#xs + 1 \equiv 1 < i \leq \#xs+1 \lor i = 1$ }

$\langle \exists i : \underline{1 < i \leq \#xs+1 \lor i = 1} : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { separación de rango }

$\underline{\langle \exists i : 1 < i \leq \#xs+1 : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle}$

$\lor \langle \exists i : i = 1 : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { cambio de variable **i = i+1** }

$\langle \exists i : \underline{1 < i+1 \leq \#xs+1} : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i+1) : ((x:xs){\downarrow}i+1) ! j \rangle = (x:xs) ! (i+1 - 1) \rangle$

$\lor \langle \exists i : i = 1 : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { aritmética }

$\langle \exists i : 0 < i \leq \#xs : \langle \prod j : 0 \leq j < \#(\underline{(x:xs){\downarrow}i+1}) : (\underline{(x:xs){\downarrow}i+1}) ! j \rangle = (x:xs) ! (i+1 - 1) \rangle$

$\lor \langle \exists i : i = 1 : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { definición de $\downarrow$ }

$\langle \exists i : 0 < i \leq \#xs : \langle \prod j : 0 \leq j < \#(xs{\downarrow}i) : (xs{\downarrow}i) ! j \rangle = \underline{(x:xs) ! (i+1 - 1)} \rangle$

$\lor \langle \exists i : i = 1 : \langle \prod j : 0 \leq j < \#((x:xs){\downarrow}i) : ((x:xs){\downarrow}i) ! j \rangle = (x:xs) ! (i - 1) \rangle$

$\equiv$ { definición de ! }

$\underline{\langle \exists i : 0 < i \leq \#xs : \langle \prod j : 0 \leq j < \#(xs{\downarrow}i) : (xs{\downarrow}i) ! j \rangle = xs ! (i - 1) \rangle}$

∨ ⟨∃ i : i = 1 : ⟨∏ j : 0 ≤ j < #((x:xs)↓i) : ((x:xs)↓i) ! j ⟩ = (x:xs) ! (i − 1) ⟩

≡ { HI }

prodSuf.xs ∨ **⟨∃ i : i = 1 : ⟨∏ j : 0 ≤ j < #((x:xs)↓i) : ((x:xs)↓i) ! j ⟩ = (x:xs) ! (i − 1) ⟩**

≡ { rango unitario }

prodSuf.xs ∨ ⟨∏ j : 0 ≤ j < #(**(x:xs)↓(0+1)**) : (**(x:xs)↓(0+1)** ! j ⟩ = (x:xs) ! 0

≡ { def de ↓ }

prodSuf.xs ∨ ⟨∏ j : 0 ≤ j < #(xs↓0) : (xs↓0) ! j ⟩ = **(x:xs) ! 0**

≡ { def de ! }

prodSuf.xs ∨ ⟨∏ j : 0 ≤ j < #(xs↓0) : (xs↓0) ! j ⟩ = x

≡ { def de ↓ }

prodSuf.xs ∨ **⟨∏ j : 0 ≤ j < #xs : xs ! j ⟩** = x

≡ { **modularizo** prod, con especificación **prod.xs =** ⟨∏ **j : 0 ≤ j < #xs : xs ! j** ⟩ }

prodSuf.xs ∨ **prod.xs = x**

=>

prodSuf :: [Num] -> Bool

prodSuf [ ] = False

prodSuf (x:xs) = prodSuf.xs ∨ **prod.xs = x**

Modularizo **prod.xs =** ⟨∏ **j : 0 ≤ j < #xs : xs ! j** ⟩

Caso Base xs = [ ]

prod.[ ]

= { especificación }

⟨∏ j : 0 ≤ j < **#[ ]** : x[ ] ! j ⟩

= { def # }

⟨∏ j : **0 ≤ j < 0** : x[ ] ! j ⟩

= { lógica }

⟨∏ j : **False** : x[ ] ! j ⟩

= { rango vacío }

1

Caso recursivo

HI : prod.xs = ⟨∏ **j : 0 ≤ j < #xs : xs ! j** ⟩

prod(x:xs)

= { especificación }

⟨∏ j : 0 ≤ j < **#(x:xs)** : (x:xs) ! j ⟩

≡ {Definicion de #}

⟨∏ j : **<u>0 ≤ j < #xs + 1</u>** : (x:xs) ! j ⟩

≡ { logica}

⟨∏ j : **<u>j = 0</u>** ∨ **<u>1 ≤ j < #xs + 1</u>** : (x:xs) ! j ⟩

≡ { separación de término }

⟨∏ j : **<u>j = 0</u>** : (x:xs) ! j ⟩ * ⟨∏ j : 1 ≤ j < #xs + 1 : (x:xs) ! j ⟩

≡ {Rango Unitario}

<u>(x:xs) ! 0</u> * ⟨∏ j : 1 ≤ j < #xs + 1 : (x:xs) ! j ⟩

≡ {def. de indexar}

x * ⟨∏ j : **<u>1 ≤ j < #xs + 1 : (x:xs) ! j</u>** ⟩

≡ {cambio de variable j = j + 1}

x * ⟨∏ j : **<u>1 ≤ j + 1 < #xs + 1 : (x:xs) ! j + 1</u>** ⟩

≡ {resto 1 a cada termino de el rango}

x * ⟨∏ j : 0 ≤ j < #xs : **<u>(x:xs) ! j + 1</u>** ⟩

≡ {Def. de indexar}

x * **<u>⟨∏ j : 0 ≤ j < #xs : xs ! j ⟩</u>**

≡ {Hipótesis Inductiva}

x * prod.xs

=>

```
prod :: [Num] -> Num
prod.[ ] = 1
prod.(x:xs) = x * prod.xs
```

Finalmente:

```
prodSuf :: [Num] -> Bool
prodSuf [ ] = False
prodSuf (x:xs) = prodSuf.xs ∨ prod.xs = x

prod :: [Num] -> Num
prod.[ ] = 1
prod.(x:xs) = x * prod.xs
```

5. Especificar formalmente utilizando cuantificadores cada una de las siguientes funciones descriptas informalmente. Luego, *derivar* soluciones algorítmicas para cada una.

   a) iguales : $[A] \to Bool$, que determina si los elementos de una lista de tipo A son todos iguales entre sí. Suponga que el operador $=$ es la igualdad para el tipo A.

   b) minimo : $[Int] \to Int$, que calcula el mínimo elemento de una lista **no vacía** de enteros.

   Nota: 1 *La función no debe devolver* $\pm\infty$.

   c) creciente : $[Int] \to Bool$, que determina si los elementos de una lista de enteros están ordenados en forma creciente.

   d) prod : $[Num] \to [Num] \to Num$, que calcula el producto entre pares de elementos en iguales posiciones de las listas y suma estos resultados (producto punto). Si las listas tienen distinto tamaño se opera hasta la última posición de las más chica.

a)

iguales.xs = 〈 ∀ i,j : 0 ≤ i < j < #xs : xs ! i = xs ! j 〉

Derivación:

Caso base

iguales.[ ]

≡ { especificación }

〈 ∀ i,j : 0 ≤ i < j < **#[ ]** : [ ] ! i = [ ] ! j 〉

≡ { definición de # }

〈 ∀ i,j : **0 ≤ i < j < 0** : [ ] ! i = [ ] ! j 〉

≡ { lógica }

〈 ∀ i,j : **False** : [ ] ! i = [ ] ! j 〉

≡ { rango vacío }

True

Caso recursivo

Caso xs = ([x])

iguales.([x])

≡ { especificación }

〈 ∀ i,j : 0 ≤ i < j < **#[x])** : [x] ! i = [x]! j 〉

≡ { definición de indexación }

〈 ∀ i,j : **0 ≤ i < j < 1** : [x] ! i = [x] ! j 〉

≡ { evalúo el rango }

〈 ∀ i,j : **False** : [x] ! i = [x] ! j 〉

≡ { rango vacío }

True

H.I.: $iguales.xs = \langle \forall\ i,j : 0 \le i < j < \#xs : xs\ !\ i = xs\ !\ j \rangle$
Paso inductivo: probarlo para $(x \triangleright (y \triangleright xs))$

$iguales.(x \triangleright xs)$

$\equiv$ { especificación }

$\langle \forall\ i,j : 0 \le i < j < \underline{\#(x \triangleright (y \triangleright ys))} : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { definición de # }

$\langle \forall\ i,j : \underline{0 \le i < j < \#(y \triangleright ys)+1} : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { reescribo la desigualdad }

$\langle \forall\ i,j : \underline{(i = 0 \wedge j = 1) \vee 1 \le i < j < \#(y \triangleright ys)+1} : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { partición de rango }

$\underline{\langle \forall\ i,j : i = 0 \wedge j = 1 : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle} \wedge \langle \forall\ i,j : 1 \le i < j < \#(y \triangleright ys)+1 : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { Eliminación de variable (T2) }

$\underline{\langle \forall\ j : j = 1 : (x \triangleright (y \triangleright ys))\ !\ 0 = (x \triangleright (y \triangleright ys))\ !\ j \rangle} \wedge \langle \forall\ i,j : 1 \le i < j < \#(y \triangleright ys)+1 : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { rango unitario }

$\underline{(x \triangleright (y \triangleright ys))\ !\ 0 = (x \triangleright (y \triangleright ys))\ !\ 1} \wedge \langle \forall\ i,j : 1 \le i < j < \#(y \triangleright ys)+1 : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle$

$\equiv$ { definición de indexación }

$x = y \wedge \underline{\langle \forall\ i,j : 1 \le i < j < \#(y \triangleright ys)+1 : (x \triangleright (y \triangleright ys))\ !\ i = (x \triangleright (y \triangleright ys))\ !\ j \rangle}$

$\equiv$ { cambio de variable, **f.i = i+1, f.j = j+1** }

$x = y \wedge \langle \forall\ i,j : \underline{1 \le i+1 < j+1 < \#(y \triangleright ys)+1} : (x \triangleright (y \triangleright ys))\ !\ (i+1) = (x \triangleright (y \triangleright ys))\ !\ (j+1) \rangle$

$\equiv$ { álgebra }

$x = y \wedge \langle \forall\ i,j : 0 \le i < j < \#(y \triangleright ys) : \underline{(x \triangleright (y \triangleright ys))\ !\ (i+1)} = \underline{(x \triangleright (y \triangleright ys))\ !\ (j+1)} \rangle$

$\equiv$ { definición de indexación }

$x = y \wedge \underline{\langle \forall\ i,j : 0 \le i < j < \#(y \triangleright ys) : (y \triangleright ys)\ !\ i = (y \triangleright ys)\ !\ j \rangle}$

$\equiv$ { H.I. }

$x = y \wedge iguales.(y \triangleright ys)$

El programa resulta

$iguales : [A] \rightarrow Bool$
$iguales.[\ ] \doteq True$
$iguales.(x \triangleright [\ ]) \doteq True$
$iguales.(x \triangleright (y \triangleright xs)) \doteq x = y \wedge iguales.(y \triangleright ys)$
iguales.xs = $\langle \forall i, j : 0 \le i < j < \#xs : xs!\,i = xs!\,j \rangle$

b)

minimo :: [Int] -> Int

minimo.xs = ⟨ min i : 0 ≤ i < #xs ∧ xs /= [ ] : xs! i ⟩

Hago inducción en lista

caso base xs = [x] (ya que la lista no es vacia)

<u>minimo.[x]</u>

≡ { especificación }

⟨ Min i : 0 ≤ i < **<u>#[x]</u>** ∧ [x] /= [ ] : [x] ! i ⟩

≡ { #[x] = #(x : [ ]) = 1 + # [ ] = 1 }

⟨ Min i : **<u>0 ≤ i < 1 ∧ [x] ≠ [ ]</u>** : [x] ! i ⟩

≡ { lógica }

<u>⟨ **min i : i = 0 : [x] ! i** ⟩</u>

≡ { rango unitario }

<u>[x] ! 0</u>

≡ { def de ! }

x

HI : mínimo.(y : ys) = ⟨ Min i : 0 ≤ i < #(y ▹ ys) : (y ▹ ys) ! i ⟩

caso inductivo : (x : (y : ys)

<u>minimo.(x ▹ (y ▹ ys))</u>

≡ { especificación }

⟨ Min i : 0 ≤ i < **<u>#(x ▹ (y ▹ ys))</u>** : (x▹(y▹ys)) ! i ⟩

≡ { definición de # }

⟨ Min i : **<u>0 ≤ i < #(y ▹ ys) + 1</u>** : (x▹(y▹ys)) ! i ⟩

≡ { reescribo la desigualdad }

⟨ Min i : **<u>i = 0 ∨ 1 ≤ i < #(y ▹ ys) + 1</u>** : (x▹(y▹ys)) ! i ⟩

≡ { partición de rango }

⟨ Min i : i = 0 : (x▹(y▹ys)) ! i ⟩ min ⟨ Min i : **<u>1 ≤ i < #(y▹ys) + 1 : (x▹(y▹ys)) ! i</u>** ⟩

≡ { cambio de variable, f.i = i+1 }

⟨ Min i : i = 0 : (x▹(y▹ys)) ! i ⟩ min ⟨ Min i : **<u>1 ≤ i + 1 < #(y▹ys) + 1</u>** : **<u>(x ▹ (y ▹ ys)) ! (i + 1)</u>** ⟩

≡ { álgebra, definición de indexación }

⟨ Min i : i = 0 : (x▹(y▹ys)) ! i ⟩ min <u>⟨ **Min i : 0 ≤ i < #(y▹ys) : (y▹ys) ! i** ⟩</u>

≡ { H.I. }

<u>⟨ **Min i : i = 0 : (x▹(y▹ys)) ! i** ⟩</u> min minimo.(y▹ys)

≡ { rango unitario }

<u>(x▹(y▹ys)) ! 0</u> min minimo.(y▹ys)

≡ { definición de indexación }

x min minimo.(y▹ys)

minimo : [Int] → Int

minimo.(x▹[ ]) ≐ x

minimo.(x▹(y▹ys)) ≐ x min minimo.(y▹ys)

c)

creciente.xs = ⟨ ∀ i,j : 0 ≤ i < j < #xs : xs ! i ≤ xs ! j ⟩

Derivación:

Caso base

<u>creciente.[]</u>

≡ { especificación }

⟨ ∀ i,j : 0 ≤ i < j < **<u>#[ ]</u>** : [ ]! i ≤ [] ! j ⟩

≡ { definición de # }

⟨ ∀ i,j : **<u>0 ≤ i < j < 0</u>** : [ ]!i ≤ [ ] ! j ⟩

≡ { lógica }

⟨ ∀ i,j : **<u>False</u>** : [ ] ! i ≤ [ ] ! j ⟩

≡ { rango vacío }

<u>True</u>

<u>Caso recursivo</u>

Caso xs = (x : [ ])

<u>creciente.(x▹[])</u>

≡ { especificación }

⟨ ∀ i,j : 0 ≤ i < j < **<u>#(x▹[])</u>** : (x▹[])!i ≤ (x▹[])!j ⟩

≡ { definición de # }

⟨ ∀ i,j : **<u>0 ≤ i < j < 1</u>** : (x▹[])!i ≤ (x▹[])!j ⟩

≡ { lógica }

⟨ ∀ i,j : **<u>False</u>** : (x▹[])!i ≤ (x▹[])!j ⟩

≡ { Rango vacio }

True

Caso xs = (y▹ys)

Hipótesis Inductiva: creciente.(y▹ys) = ⟨ ∀ i,j : 0 ≤ i < j < #(y▹ys) : (y▹ys)!i ≤ (y▹ys)!j ⟩

creciente.(x▹(y▹ys))

≡ { especificación }

⟨ ∀ i,j : 0 ≤ i < j < **#(x ▹ (y ▹ ys))** : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys)) ! j ⟩

≡ { definición de # }

⟨ ∀ i,j : **0 ≤ i < j < #(y ▹ ys) + 1** : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys)) ! j ⟩

≡ { reescribo el rango }

⟨ ∀ i,j : **i = 0 ∧ j = 1 ∨ 1 ≤ i < j < #(y ▹ ys) + 1** : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys))!j ⟩

≡ { partición de rango }

⟨ ∀ i,j : i = 0 ∧ j = 1 : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys)) ! j ⟩ ∧ ⟨ ∀ i,j : **1 ≤ i < j < #(y ▹ ys) + 1** : **(x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys))!j** ⟩

≡ { cambio de variable, **f.i = i+1, f.j = j+1** }

⟨ ∀ i,j : i = 0 ∧ j = 1 : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys)) ! j ⟩ ∧ ⟨ ∀ i,j : **1 ≤ i+1 < j+1 < #(y▹ys) + 1** : (**x ▹ (y ▹ ys)) ! (i + 1) ≤ (x ▹ (y ▹ ys)) ! (j + 1)** ⟩

≡ { álgebra, definición de indexación }

⟨ ∀ i,j : i = 0 ∧ j = 1 : (x ▹ (y ▹ ys)) ! i ≤ (x ▹ (y ▹ ys)) ! j ⟩ ∧ **⟨ ∀ i,j : 0 ≤ i < j < #(y ▹ ys) : (y ▹ ys) ! i ≤ (y ▹ ys) ! j ⟩**

≡ { H.I. }

⟨ ∀ i , j : **i = 0** ∧ j = 1 : **(x ▹ (y ▹ ys)) ! i** ≤ (x ▹ (y ▹ ys)) ! j ⟩ ∧ creciente.xs

≡ { eliminación de variable, (T2) }

⟨ ∀ j : **j = 1** : (x ▹ (y ▹ ys)) ! 0 ≤ (x ▹ (y ▹ ys)) ! j ⟩ ∧ creciente.xs

≡ { rango unitario }

(x ▹ (y ▹ ys)) ! 0 ≤ (x ▹ (y ▹ ys)) ! 1 ∧ creciente.xs

≡ { evalúo las indexaciones }

x ≤ y ∧ creciente.xs

El programa resulta

creciente : [Int] → Bool

creciente. [ ] ≐ True

creciente. (x ▹ [ ]) ≐ True

creciente. (x ▹ (y ▹ ys)) ≐ x ≤ y ∧ creciente.(y ▹ ys)

d)

$prod : [Num] \to [Num] \to Num$

$prod.xs.ys = \langle \sum i : 0 \le i \le \#xs \wedge i < \#ys : xs\ !\ i * ys\ !\ i \rangle$

Caso base : $xs = [\ ]$

$\underline{prod.[\ ].ys}$

≡ {especificación}

$\langle \sum i : 0 \le i \le \underline{\#[\ ]} \wedge i < \#ys : xs\ !\ i * ys\ !\ i \rangle$

≡ {definición de cardinal}

$\langle \sum i : \underline{0 \le i \le 0} \wedge i < \#ys : xs\ !\ i * ys\ !\ i \rangle$

≡ {lógica}

$\langle \sum i : \underline{False \wedge i < \#ys} : xs\ !\ i * ys\ !\ i \rangle$

≡ { Elemento absorbente del ∧ }

$\underline{\langle \sum i : False : xs\ !\ i * ys\ !\ i \rangle}$

≡ { Rango Vacío}

0

caso base 2 : $ys = [\ ]$

$\underline{prod.xs.[\ ]}$

≡ {especificación}

$\langle \sum i : 0 \le i \le \#xs \wedge i < \underline{\#[\ ]} : xs\ !\ i * ys\ !\ i \rangle$

≡ {definición de cardinal}

$\langle \sum i : \underline{0 \le i \le \#xs \wedge i < 0} : xs\ !\ i * ys\ !\ i \rangle$

≡ {lógica, alguno de los dos es False, despues elemento absorbente del ∧ }

$\underline{\langle \sum i : False: xs\ !\ i * ys\ !\ i \rangle}$

≡ {Rango Vacio}

0

Hipotesis Inductiva : $prod.xs.ys = \langle \sum i : 0 \le i \le \#xs \wedge i < \#ys : xs\ !\ i * ys\ !\ i \rangle$

paso inductivo : $xs = (x \triangleright xs)$ $ys = (y \triangleright ys)$

$prod. (x \triangleright xs). (y \triangleright ys)$

≡ {especificación}

$\langle \sum i : 0 \le i \le \underline{\#(x \triangleright xs)} \wedge i < \underline{\#(y \triangleright ys)} : (x \triangleright xs)\ !\ i * (y \triangleright ys)\ !\ i \rangle$

≡ {def de #}

$\langle \sum i : \underline{0 \le i \le xs + 1} \wedge i < ys + 1 : (x \triangleright xs)\ !\ i * (y \triangleright ys)\ !\ i \rangle$

≡ {logica}

$\langle \sum i : \underline{i = 0 \vee 1 \le i \le xs + 1 \wedge i < ys + 1} : (x \triangleright xs)\ !\ i * (y \triangleright ys)\ !\ i \rangle$

≡ {partición de rango}

⟨∑ i : i = 0 : (x ▹ xs) ! i * (y ▹ ys) ! i ⟩+⟨∑ i : **1 ≤ i ≤ xs + 1 ∧ i < ys + 1 : (x ▹ xs) ! i * (y ▹ ys) ! i** ⟩

≡ {cambio de variable i = i + 1}

⟨∑ i : i = 0 : (x ▹ xs) ! i * (y ▹ ys) ! i ⟩+⟨∑ i : **1 ≤ i + 1 ≤ xs + 1 ∧ i + 1 < ys + 1 : (x ▹ xs) ! (i + 1) * (y ▹ ys) ! (i + 1)** ⟩

≡ {resto 1 a cada término del rango}

⟨∑ i : i = 0 : (x ▹ xs) ! i * (y ▹ ys) ! i ⟩+⟨∑ i : 0 ≤ i ≤ xs ∧ i < ys : **(x ▹ xs) ! (i + 1) * (y ▹ ys) ! (i + 1)** ⟩

≡ {def. de indexar}

⟨∑ i : i = 0 : (x ▹ xs) ! i * (y ▹ ys) ! i ⟩+**⟨∑ i : 0 ≤ i ≤ xs ∧ i < ys : xs ! i * ys ! i ⟩**

≡ {Hipótesis Inductiva}

⟨∑ i : **i = 0** : (x ▹ xs) ! i * (y ▹ ys) ! i ⟩+ prod.xs.ys

≡ {Rango Unitario}

(x ▹ xs) ! 0 * (y ▹ ys) ! 0 + prod.xs.ys

≡ {def. de indexar}

x * y + prod.xs.ys

Finalmente ⇒

prod :: [Num] → [Num] → Num
prod. [ ]. ys = 0
prod. xs. [ ] = 0
prod.(x ▹ xs).(y ▹ ys) = x * y + prod.xs.ys

6. Para complementar el ejercicio 4*b*. Considerando que $Punto = (Num, Num)$ y $Seg = (Punto, Punto)$, defina las siguientes funciones:

a) punto: $(Num \to Num) \to Num \to Punto$ que dada una función y un valor nos da el punto correspondiente a $(x, f\,x)$.

b) dist: $(Num \to Num) \to (Num \to Num) \to Num \to Seg$ que dadas dos funciones f, g y un valor nos da el segmento de recta dados por los puntos en f y g.

c) curva: $(Num \to Num) \to [Num] \to [Punto]$ que dada una función f y una lista xs de valores devuelve la lista de puntos $(x, f\,x)$ para cada elemento x de la lista xs.

d) angulos: $Nat \to [Num]$ que en el argumento n devuelve la lista de $2n+1$ ángulos entre $-2 * \pi$ y $2 * \pi$: $[-2\pi, -2\pi\frac{n-1}{n}, \ldots, 0, 2\pi\frac{1}{n}, \ldots, 2\pi\frac{n-1}{n}, 2\pi]$.

Ayuda: No hay nada difícil y son más del estilo de cosas que hicieron en IntroAlg.

Punto = (Num, Num), Seg = (Punto, Punto)

a)
punto :: (Num -> Num) -> Num -> Punto
punto.f.x = (x , f.x)
b)
dist :: (Num -> Num) -> (Num -> Num) -> Num -> Seg
dist.f.g.x = (punto.f.x,punto.g.x)

c)

```
curva :: (Num -> Num) -> [Num] -> [Punto]
curva.f.[ ] = [ ]
curva.f.(x►xs) = punto.f.x ► curva.f.xs
```

d)

```
angulos :: Nat -> [Num]
angulos.n = anguloFijo.n.0

anguloFijo :: Nat -> Nat -> [Num]
anguloFijo.n.m
| 0<= m <= 2n = -2*pi*(n-m)/n ► anguloFijo.n.(m+1)
| m > n =[ ]
```