

Práctico 2: Especificación, Derivación y Verificación de Programas Funcionales

Algoritmos y Estructuras de Datos I
2^{do} cuatrimestre 2021

Esta guía tiene como objetivo obtener las habilidades necesarias para llevar adelante un proceso de derivación o verificación de programas recursivos a partir de especificaciones formales.

Los ejercicios de cálculo de programas tienen una dificultad creciente: en los primeros, la derivación o verificación se obtiene de manera *directa* a través de una demostración inductiva. Los ejercicios sucesivos son más complejos y requieren el uso de modularización. En la siguiente entrega del práctico veremos ejemplos que necesitan generalización.

1. Dado el **programa**

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

- ¿Qué hace esta función? Escriba en lenguaje natural el **problema** que resuelve.
 - Escriba una **especificación** de la función con una expresión cuantificada.
 - Verifique** que esta especificación vale para toda lista xs .
 - Ahora **derive** la definición de la función a partir de su especificación.
¿Esta derivación es parecida a la demostración en el punto [1c](#)?
2. A partir de las siguientes especificaciones, dar el tipo de cada función y *derivar* las soluciones algorítmicas correspondientes.
- $\text{sum_cuad}.xs = \langle \sum i : 0 \leq i < \#xs : xs!i * xs!i \rangle$
 - $\text{iga}.e.xs = \langle \forall i : 0 \leq i < \#xs : xs!i = e \rangle$
 - $\text{exp}.x.n = x^n$
 - $\text{sum_par}.n = \langle \sum i : 0 \leq i \leq n \wedge \text{par}.i : i \rangle$
donde $\text{par}.i \doteq i \bmod 2 = 0$.
 - $\text{cuántos}.p.xs = \langle \text{N } i : 0 \leq i < \#xs : p.(xs!i) \rangle$
 - $\text{busca}.e.xs = \langle \text{Min } i : 0 \leq i < \#xs \wedge xs!i = e : i \rangle$
3. Para todos los ítems 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.
4. Derivar las siguientes funciones.

- a) $\text{sum_pot} : \text{Num} \rightarrow \text{Nat} \rightarrow \text{Num}$ 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) $\text{cos}' : \text{Nat} \rightarrow \text{Num} \rightarrow \text{Num}$ computa la aproximación del coseno del segundo argumento.

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

Ayuda: Modularizar dos veces. La segunda con la función *exp* del ejercicio [2c](#) y factorial.

- c) $\text{cubo} : \text{Nat} \rightarrow \text{Nat}$ 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) $\text{prod_suf} : [\text{Num}] \rightarrow \text{Bool}$ decide si existe un elemento igual al producto de los elementos que le siguen:

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

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

- a) $\text{iguales} : [A] \rightarrow \text{Bool}$, 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) $\text{minimo} : [\text{Int}] \rightarrow \text{Int}$, que calcula el mínimo elemento de una lista **no vacía** de enteros.

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

- c) $\text{creciente} : [\text{Int}] \rightarrow \text{Bool}$, que determina si los elementos de una lista de enteros están ordenados en forma creciente.
- d) $\text{prod} : [\text{Num}] \rightarrow [\text{Num}] \rightarrow \text{Num}$, 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 la más chica.

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

- a) $\text{punto} : (\text{Num} \rightarrow \text{Num}) \rightarrow \text{Num} \rightarrow \text{Punto}$ que dada una función y un valor nos da el punto correspondiente a $(x, f x)$.
- b) $\text{dist} : (\text{Num} \rightarrow \text{Num}) \rightarrow (\text{Num} \rightarrow \text{Num}) \rightarrow \text{Num} \rightarrow \text{Seg}$ que dadas dos funciones f, g y un valor nos da el segmento de recta dados por los puntos en f y g .
- c) $\text{curva} : (\text{Num} \rightarrow \text{Num}) \rightarrow [\text{Num}] \rightarrow [\text{Punto}]$ 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) $\text{angulos} : \text{Nat} \rightarrow [\text{Num}]$ que en el argumento n devuelve la lista de $2n+1$ ángulos entre $-2*\pi$ y $2*\pi$: $[-2\pi, -2\pi \frac{n-1}{n}, \dots, 0, 2\pi \frac{1}{n}, \dots, 2\pi \frac{n-1}{n}, 2\pi]$.

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