

Introducción a los Algoritmos - 2do. cuatrimestre 2015

Guía 5: Funciones, listas, recursión e inducción

Funciones recursivas

Una **función recursiva** es una función tal que en su definición puede aparecer su propio nombre. Una buena pregunta sería ¿Cómo lograr que no sea una definición circular? La clave está en el principio de inducción: en primer lugar hay que definir la función para el (los) caso(s) más “pequeño(s)”, que llamaremos **caso base** y luego definir el caso general en términos de algo más “chico”, que llamaremos **caso inductivo**. El caso base no debe aparecer el nombre de la función que se está definiendo. El caso inductivo es donde aparece el nombre de la función que se está definiendo, y debe garantizarse que el (los) argumento(s) al cual se aplica en la definición es más “chico” (para alguna definición de más chico) que el valor para la que se está definiendo.

1. Una función de **filter** es aquella que dada una lista devuelve otra lista cuyos elementos son los elementos de la primera que cumplan una determinada condición, en el mismo orden y con las mismas repeticiones (si las hubiere). Por ejemplo: $\text{soloPares} : [Int] \rightarrow [Int]$ devuelve aquellos elementos de la lista que son pares.

Definí recursivamente las siguientes funciones filter.

- a) $\text{soloPares} : [Int] \rightarrow [Int]$, que dada una lista de enteros xs devuelve una lista sólo con los números pares contenidos en xs , en el mismo orden y con las mismas repeticiones (si las hubiera).
Por ejemplo: $\text{soloPares}.[3, 0, -2, 12] = [0, -2]$
- b) $\text{mayoresQue10} : [Int] \rightarrow [Int]$, que dada una lista de enteros xs devuelve una lista sólo con los números mayores que 10 contenidos en xs ,
Por ejemplo: $\text{mayoresQue10}.[3, 0, -2, 12] = [12]$
- c) $\text{mayoresQue} : Int \rightarrow [Int] \rightarrow [Int]$, que dado un entero n y una lista de enteros xs devuelve una lista sólo con los números mayores que n contenidos en xs ,
Por ejemplo: $\text{mayoresQue}.2.[3, 0, -2, 12] = [3, 12]$

Preguntas:

- a) ¿Se te ocurre algún otro ejemplo de una función de este tipo?
 - b) ¿Cómo describirías una regla general para este tipo de funciones? **Ayuda:** Considerá que la condición que se testea se representa con una condición booleana p .
2. Una función de **map** es aquella que dada una lista devuelve otra lista cuyos elementos son los que se obtienen de aplicar una función a cada elemento de la primera en el mismo orden y con las mismas repeticiones (si las hubiere). Por ejemplo: $\text{duplica} : [Int] \rightarrow [Int]$ devuelve cada elemento de la lista multiplicado por 2.

Definí recursivamente las siguientes funciones de map.

- a) $\text{sumar1} : [Int] \rightarrow [Int]$, que dada una lista de enteros le suma uno a cada uno de sus elementos.
Por ejemplo: $\text{sumar1}.[3, 0, -2] = [4, 1, -1]$
- b) $\text{duplica} : [Int] \rightarrow [Int]$, que dada una lista de enteros duplica cada uno de sus elementos.
Por ejemplo: $\text{duplica}.[3, 0, -2] = [6, 0, -4]$
- c) $\text{multiplica} : Int \rightarrow [Int] \rightarrow [Int]$, que dado un número n y una lista, multiplica cada uno de los elementos por n .
Por ejemplo: $\text{multiplica}.3.[3, 0, -2] = [9, 0, -6]$

Preguntas:

- a) ¿Se te ocurre algún otro ejemplo de una función de este tipo?

- b) ¿Cómo describirías una regla general para este tipo de funciones? **Ayuda:** considerará que la función a aplicar a cada elemento se llama f .
3. Una función de **fold** es aquella que dada una lista devuelve un valor resultante de combinar los elementos de la lista. Por ejemplo: $sum : [Int] \rightarrow Int$ devuelve la sumatoria de los elementos de la lista. Definí recursivamente las siguientes funciones fold.
- a) $todosMenores10 : [Int] \rightarrow Bool$, que dada una lista devuelve $True$ si ésta consiste sólo de números menores que 10.
- b) $hay0 : [Int] \rightarrow Bool$, que dada una lista decide si existe algún 0 en ella.
- c) $sum : [Int] \rightarrow Int$, que dada una lista devuelve la suma de todos sus elementos.

Preguntas:

- a) ¿Se te ocurre algún otro ejemplo de una función de este tipo?
- b) ¿Cómo describirías una regla general para este tipo de funciones?
4. Una función de tipo **zip** es aquella que dadas dos listas devuelve una lista de pares donde el primer elemento de cada par se corresponde con la primera lista, y el segundo elemento de cada par se corresponde con la segunda lista. Por ejemplo: $repartir : [String] \rightarrow [String] \rightarrow [(String, String)]$ donde los elementos de la primera lista son nombres de personas y los de la segunda lista son cartas españolas es una función que devuelve una lista de pares que le asigna a cada persona una carta.

Ej:

$$repartir.[\text{"Juan"}, \text{"Maria"}, \text{"Damian"}].[\text{"1deCopa"}, \text{"3deOro"}, \text{"7deEspada"}, \text{"10deOro"}, \text{"2deBasto"}] =$$

$$[(\text{"Juan"}, \text{"1deCopa"}), (\text{"Maria"}, \text{"3deOro"}), (\text{"Damian"}, \text{"7deEspada"})]$$

Defina la función recursivamente.

5. Una función de tipo **unzip** es aquella que dada una lista de tuplas devuelve una lista de alguna de las proyecciones de la tupla. Por ejemplo, si tenemos una lista de ternas donde el primer elemento representa el nombre de un alumno, el segundo el apellido, y el tercero la edad, la función que devuelve la lista de todos los apellidos de los alumnos en una de tipo `unzip`.

Definir la función $apellidos : [(String, String, Int)] \rightarrow [String]$

Ej:

$$apellidos.[(\text{"Juan"}, \text{"Dominguez"}, 22), (\text{"Maria"}, \text{"Gutierrez"}, 19), (\text{"Damian"}, \text{"Rojas"}, 18)] =$$

$$[\text{"Dominguez"}, \text{"Gutierrez"}, \text{"Rojas"}]$$

Defina la función recursivamente.

Funciones generales

En las funciones vistas hasta el momento hemos trabajado con tipos específicos de funciones sobre listas. A continuación trabajaremos con funciones recursivas en términos generales. También analizaremos el tipo de las funciones.

6. Definí recursivamente los operadores básicos de listas: $\#$, $!$, \leftarrow , \uparrow , \downarrow , $\#$. Para los operadores \uparrow , \downarrow y \cdot deberás hacer recursión en ambos parámetros, en el parámetro lista y en el parámetro numérico.
7. (i) Definí funciones por recursión para cada una de las siguientes descripciones. (ii) Evaluá los ejemplos manualmente (iii) Identificá si las funciones son de algún tipo ya conocido (filter, map, fold). (iv) Programálas en `haskell` y verificá los resultados obtenidos.

- a) $maximo : [Int] \rightarrow Int$, que calcula el máximo elemento de una lista de enteros.
 Por ejemplo: $maximo.[2, 5, 1, 7, 3] = 7$
Ayuda: Ir tomando de a dos elementos de la lista y ‘quedarse’ con el mayor.
- b) $sumaPares : [(Num, Num)] \rightarrow Num$, que dada una lista de pares de números, devuelve la sumatoria de todos los números de todos los pares.
 Por ejemplo: $sumaPares.[(1, 2)(7, 8)(11, 0)] = 29$
- c) $todos0y1 : [Int] \rightarrow Bool$, que dada una lista devuelve $True$ si ésta consiste sólo de 0s y 1s.
 Por ejemplo: $todos0y1.[1, 0, 1, 2, 0, 1] = False$, $todos0y1.[1, 0, 1, 0, 0, 1] = True$
- d) $quitar0s : [Int] \rightarrow [Int]$ que dada una lista de enteros devuelve la lista pero quitando todos los ceros.
 Por ejemplo $quitar0s.[2, 0, 3, 4] = [2, 3, 4]$
- e) $ultimo : [A] \rightarrow A$, que devuelve el último elemento de una lista.
 Por ejemplo: $ultimo.[10, 5, 3, 1] = 1$
- f) $listasIguales : [A] \rightarrow [A] \rightarrow Bool$, que determina si dos listas son iguales, es decir, contienen los mismos elementos en las mismas posiciones respectivamente.
 Por ejemplo: $listasIguales.[1, 2, 3].[1, 2, 3] = True$, $listasIguales.[1, 2, 3, 4].[1, 3, 2, 4] = False$

Interpretación y formalización de enunciados en lógica de predicados

8. Expresá el significado de cada una de las siguientes fórmulas en lenguaje natural. Por ejemplo, en la fórmula

$$\langle \forall i : 0 \leq i < \#xs : xs ! i = 0 \rangle$$

debemos suponer que la variable xs es de tipo lista, para que el tipado sea correcto. Las siguientes sentencias son todas traducciones de esta fórmula al lenguaje natural:

“Todos los elementos de la lista xs son ceros”
 “La lista xs está compuesta únicamente por ceros”
 “La lista xs no tiene otro valor que cero”...

- a) $\langle \forall x : x \in Num : \langle \exists y : y \in Int : x < y \rangle \rangle$
 b) $\langle \exists x : x \in Num : \langle \forall y : y \in Int : x < y \rangle \rangle$ ¿Es lo mismo que el anterior?
 c) $\langle \forall x, z : x, z \in Num \wedge x < z : \langle \exists y : y \in Num : x < y < z \rangle \rangle$.
 d) $\langle \forall p, q : 0 \leq p \wedge 0 \leq q \wedge p + q = \#xs - 1 : xs ! p = xs ! q \rangle$.
 e) $N \leq \#xs \wedge \langle \exists i : 0 \leq i < N : xs ! i = 0 \rangle$
 f) $\langle \exists N : N \leq \#xs : \langle \forall i : 0 \leq i < N : xs ! i \geq 0 \rangle \rangle$
 g) $\#xs = \#ys \Rightarrow \langle \exists i : 0 \leq i < \#xs \text{ mín } \#ys : xs ! i \neq ys ! i \rangle$.
9. Formalizá las siguientes sentencias escritas en lenguaje natural, utilizando cuantificadores y predicados arbitrarios para aquellas propiedades elementales. Por ejemplo, para la sentencia “Hay enteros pares” puede formalizarse con la fórmula:

$$\langle \exists x : entero.x : mod.x.2 = 0 \rangle$$

- a) Todo entero es par o impar.
 b) El producto de dos impares es impar.
 c) Dados dos números enteros positivos, existe un tercer entero tal que el primer entero multiplicado por el tercer entero es mayor que el segundo entero.
 d) x está en la lista xs .
 e) La lista xs consiste sólo de 0's y 1's.
 f) Si el 1 está en xs , entonces también el 0 está.

- g)* Todos los elementos de xs son iguales.
- h)* Todos los elementos de la lista son distintos.
- i)* La lista xs es capicúa.
- j)* La lista xs está ordenada de manera decreciente.
- k)* Las listas xs e ys tienen los mismos elementos.
- l)* Todos los elementos de xs tienen al menos un elemento. ¿Cuál debe ser el tipo de xs ?

10. Implementa las especificaciones anteriores en Haskell

Inducción

Una técnica poderosa para demostrar propiedades sobre un dominio inductivo, como son los naturales o las listas, es usar el **principio de inducción**. La idea que rige este principio consiste en demostrar dos cosas. Por un lado verificar que la propiedad se satisface para los elementos “más chicos” del dominio (por ejemplo, la lista []). Por otro lado demostrar para un elemento arbitrario del dominio (por ejemplo, la lista $x \triangleright xs$) que si suponemos que la propiedad es cierta para todos los elementos más chicos que él (por ejemplo xs), entonces la propiedad también es satisfecha por ese elemento. Dado que todo elemento de un dominio inductivo puede ser “construido” a partir de elementos más simples, este procedimiento demuestra que la propiedad es satisfecha por todos los elementos del dominio, y por lo tanto es válida.

11. Considerando las definiciones de los ejercicios anteriores demostrará por inducción sobre xs las siguientes propiedades:

a)

$$\text{sum.}(\text{sumar1}.xs) = \text{sum}.xs + \#xs$$

b)

$$\text{sum.}(\text{duplica}.xs) = 2 * \text{sum}.xs$$

c)

$$\#(\text{duplica}.xs) = \#xs$$

12. Demostrará por inducción las siguientes propiedades. **Ayuda:** Recordá la definición de cada uno de los operadores implicados en cada expresión.

a) $xs \uparrow [] = xs$ (la lista vacía es el elemento neutro de la concatenación)

b) $\#xs \geq 0$

c) $xs \uparrow (ys \uparrow zs) = (xs \uparrow ys) \uparrow zs$ (la concatenación es asociativa)

d) $(xs \uparrow ys) \uparrow \#xs = xs$

e) $(xs \uparrow ys) \downarrow \#xs = ys$

f) $xs \uparrow (y \triangleright ys) = (xs \triangleleft y) \uparrow ys$

g) $xs \uparrow (ys \triangleleft y) = (xs \uparrow ys) \triangleleft y$

13. Considerando la función $\text{sum} : [\text{Num}] \rightarrow \text{Num}$ que toma una lista de números y devuelve la suma de ellos, definí sum y demostrará que:

$$\text{sum.}(xs \uparrow ys) = \text{sum}.xs + \text{sum}.ys$$

14. Considerando la función $\text{repetir} : \text{Nat} \rightarrow \text{Num} \rightarrow [\text{Num}]$, que construye una lista de un mismo número repetido cierta cantidad de veces, definida recursivamente como:

$$\begin{aligned} \text{repetir}.0.x &\doteq [] \\ \text{repetir.}(n+1).x &\doteq x \triangleright \text{repetir}.n.x \end{aligned}$$

demostrará que $\#\text{repetir}.n.x = n$.

15. Considerando las funciones sum y duplica de los ejercicios 13 y 2b, respectivamente, demostrará que:

$$\text{sum.}(\text{duplica}.xs) = 2 * \text{sum}.xs$$

16. Considerando la función $concat : [[A]] \rightarrow [A]$ que toma una lista de listas y devuelve la concatenación de todas ellas, definida recursivamente como:

$$\begin{aligned} concat.[] &\doteq [] \\ concat.(xs \triangleright xss) &\doteq xs \uparrow\uparrow concat.xss \end{aligned}$$

demostrá que $concat.(xss \uparrow\uparrow yss) = concat.xss \uparrow\uparrow concat.yss$

17. Considerando la función $rev : [A] \rightarrow [A]$ que toma una lista y devuelve una lista con los mismos elementos pero en orden inverso, definida recursivamente como:

$$\begin{aligned} rev.[] &\doteq [] \\ rev.(x \triangleright xs) &\doteq rev.xs \triangleleft x \end{aligned}$$

demostrá que $rev.(xs \uparrow\uparrow ys) = rev.ys \uparrow\uparrow rev.xs$

18. Considerando las definiciones dadas en cada caso. Demuestre por inducción sobre n las siguientes propiedades:

a) $f.n = 2 * n$, donde

$$\begin{aligned} f.0 &= 0 \\ f.(n+1) &= 2 + f.n \end{aligned}$$

b) $g.n = n$, donde

$$\begin{aligned} g.0 &= 0 \\ g.(n+1) &= 1 + g.n \end{aligned}$$

c) $f.n = g.n$, donde

$$\begin{aligned} g.0 &= 0 & f.n &= n * (n+1)/2 \\ g.(n+1) &= n+1 + g.n \end{aligned}$$

Problemas más complejos

Los siguientes problemas son un poco más complejos que los que se vieron, especialmente porque se parecen más a los problemas a los que nos enfrentamos en la vida real. Se resuelven desarrollando programas funcionales; es decir, se pueden plantear como la búsqueda de un resultado a partir de ciertos datos (argumentos).

Para ello, será necesario en primer lugar descubrir los tipos de los argumentos y del resultado que necesitamos. Luego combinaremos la mayoría de las técnicas estudiadas hasta ahora: **modularización** (dividir un problema complejo en varias tareas intermedias), **análisis por casos**, e identificar qué clase de funciones de lista (cuando corresponda) son las que necesitamos: **map**, **filter** o bien **fold**.

19. Películas

Contamos con una base de datos de películas representada con una lista de tuplas. Cada tupla contiene la siguiente información:

$((\text{Nombre de la película}), (\text{Año de estreno}), (\text{Duración de la película}), (\text{Nombre del director}))$

Observamos entonces que el tipo de la tupla que representa cada película es $(String, Int, Int, String)$.

- Definí la función `verTodas` : $[(String, Int, Int, String)]$ que dada una lista de películas devuelva el tiempo que tardaría en verlas a todas.
- Definí la función `estrenos` : $[(String, Int, Int, String)] \rightarrow [String]$ que dada una lista de películas devuelva el listado de películas que estrenaron en 2015.
- Definí la función `filmografia` : $[(String, Int, Int, String)] \rightarrow String \rightarrow [String]$ que dada una lista de películas y el nombre de un director, devuelva el listado de películas de ese director.
- Definí la función `duracion` : $[(String, Int, Int, String)] \rightarrow Int$ que dada una lista de películas y el nombre de una película, devuelva la duración de esa película.
- Definí otras funciones que te parezcan interesantes.

20. Otros problemas similares

Alumnos de años anteriores presentaron los siguientes problemas a resolver.

- `mejorNota`, que selecciona la nota más alta de cada alumno. Por ejemplo,
 $mejorNota.[("matias", 7, 7, 8), ("Juan", 10, 6, 9), ("Lucas", 2, 10, 10)] = [("matias", 8), ("Juan", 10), ("Lucas", 10)].$
(C.L.)
- Dadas las listas de los alumnos de las cuatro comisiones, la función que da el apellido del primer alumno de cada lista. (S.K.)
- Dada una lista de estudiantes con apellido, nombre y nota, necesito el nombre del estudiante con nota más alta. (F.S.)

21. Problemas de finales

Definir las siguientes funciones y evaluarlas manualmente sobre los ejemplos dados:

- `incPrim` : $[(Int, Int)] \rightarrow [(Int, Int)]$, que dada una lista de pares de enteros, le suma 1 al primer número de cada par.
Ejemplos: $incPrim.[(20, 5), (50, 9)] = [(21, 5), (51, 9)]$, $incPrim.[(4, 11), (3, 0)] = [(5, 11), (4, 0)]$.
- `conElMayor` : $([Int], [Int]) \rightarrow [(Int, Int)]$, que dado un par de listas de enteros, empareja los elementos de la primera con el mayor de la segunda.
Ejemplos: $conElMayor.([20, 5], [5, 9, 0]) = [(20, 9), (5, 9)]$, $conElMayor.([4, 11], [-3, 0]) = [(4, 0), (11, 0)]$.
- `expandir` : $String \rightarrow String$, pone espacios entre cada letra de una palabra.
Ejemplo: $expandir."hola" = "h o l a"$ (¡sin espacio al final!).