# Práctico 2: Especificación, Derivación y Verificación de Programas Funcionales

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

En esta guía se proponen ejercicios que requieren generalización, ya sea reemplazo de constantes por variables o por abstracción. Recordemos que descubrimos la necesidad de generalizar una especificación porque cuando estamos derivando el programa nos encontramos con que la hipótesis inductiva es demasiado rígida para aplicarla.

1. A partir de las siguientes especificaciones expresar en lenguaje natural qué devuelven las funciones, agregarles su tipo y derivarlas:

   *a*) $\text{psum}.xs = \langle\, \forall\, i \;:\; 0 \le i \le \#xs :\; sum.(xs{\uparrow}i) \ge 0 \,\rangle$

   *b*) $\text{sum\_ant}.xs = \langle\, \exists\, i \;:\; 0 \le i < \#xs :\; xs\,!\,i = sum.(xs{\uparrow}i) \,\rangle$

   **Reflexión:** Cada uno de los ítems de este punto ilustra las dos clases de generalización que usamos; para el primero alcanza con reemplazar una constante por una variable, mientras que el segundo requiere generalización por abstracción.

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

   *a*) esCuad : $Nat \to Bool$, dado un natural determina si es el cuadrado de un número.

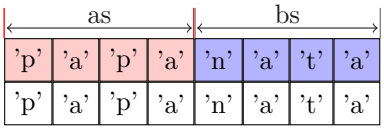
   *b*) sumanOcho : $[Num] \to Nat$, dada una lista cuenta la cantidad de prefijos que suman 8.

   **Reflexión:** La primera parte de este ejercicio no parece tener que ver con generalización: efectivamente las especificaciones formales que surgirán serán las habituales. Sin embargo, cuando intentemos derivar los programas nos encontraremos con la necesidad de generalizar. Está bien que descubramos la necesidad de generalizar al derivar y que no intentemos anticiparnos.

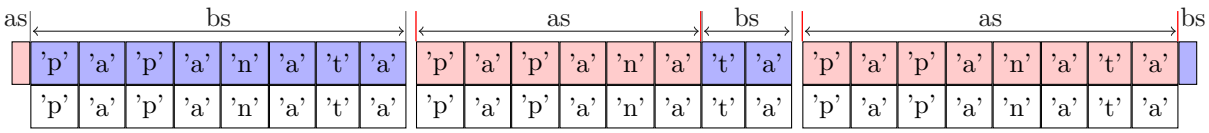
   Recuerden que es una buena práctica *testear* las especificaciones que hicieron con ejemplos concretos. Por ejemplo, esCuad.4 $\equiv True$, esCuad.5 $\equiv False$. Para el segundo: sumanOcho.$[8, 0, -1, 1] = 3$, pero sumanOcho.$[8, 0, -1] = 2$. Para cerciorarse que las especificaciones son correctas, *evaluamos* las especificaciones en esos argumentos; es decir, manipulamos las expresiones hasta encontrar el valor final.

## Segmentos de lista

Hasta ahora las especificaciones sobre listas han sido usando cuantificaciones sobre índices de la lista. De esa forma son todos los ejercicios anteriores de este mismo práctico. Una manera alternativa es cuantificar sobre segmentos de listas; un segmento es una sucesión contigua de elementos de la lista. Consideremos, por ejemplo, la lista $xs$ =“papanata”, que es la forma cómoda de escribir la lista `['p','a','p','a','n','a','t','a']`; a $xs$ la podemos partir en dos partes cuya concatenación reproduce la lista original: $as$ =“papa” y $bs$ =“nata”.

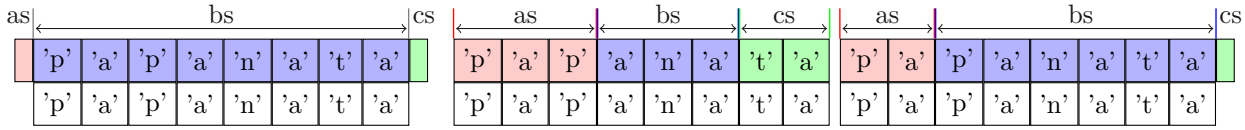



También podemos dividirla en dos segmentos de distintas longitudes, alguno de los cuales puede ser la lista vacía.




Claramente, si la lista original no es vacía no podemos dividirla en segmentos vacíos. Pero si la lista es vacía, entonces sólo podemos dividirla en segmentos vacíos.

Dividir en dos segmentos nos da un prefijo ($as$ en los ejemplos de más arriba) y un sufijo ($bs$). Eso es suficiente en algunas situaciones, pero a veces queremos poder referirnos a un segmento intermedio (como antes, dos de los tres segmentos podrían ser vacíos y si la lista fuera vacía los tres deben ser la lista vacía).




Recordemos que el rango de una expresión con más de una variable cuantificada lo expresamos como un conjunto de tuplas; por ejemplo, dada la expresión $\langle \mathrm{N}\, as, bs \;:\; xs = as +\!\!+ bs :\; \#as = \#bs \,\rangle$, el rango para $xs =$"popa" es (nos ahorramos las comillas por comodidad):

$$(as, bs) \in \{(\mathrm{popa}, []), (\mathrm{pop}, \mathrm{a}), (\mathrm{po}, \mathrm{pa}), (\mathrm{p}, \mathrm{opa}), ([], \mathrm{popa})\}$$

Una manera de encontrar ese rango es comenzando con la tupla que tiene toda la lista en el primer componente y la lista vacía en el segundo: (popa, []) y luego ir pasando el último elemento de la primera lista a la segunda lista. Cuando tenemos un rango como $xs = as +\!\!+ bs +\!\!+ cs$ podemos realizar el siguiente procedimiento: primero calculamos el rango para $xs = as +\!\!+ bs'$ y luego dividimos $bs' = bs +\!\!+ cs$. En las siguientes tablas hacemos ese procedimiento (no escribimos las comillas por comodidad); en la primera tabla consideramos $xs = as +\!\!+ bs'$ y luego, en las siguientes dos tablas, vamos pasando de $bs$ a $cs$.

| $as$ | $bs'$ |
|---|---|
| popa | [] |
| pop | a |
| po | pa |
| p | opa |
| [] | popa |

| $as$ | $bs$ | $cs$ |
|---|---|---|
| pop | [] | a |
| po | p | a |
| po | [] | pa |
| p | op | a |
| p | o | pa |

| $as$ | $bs$ | $cs$ |
|---|---|---|
| [] | pop | a |
| [] | popa | pa |
| [] | popa | opa |
| [] | [] | popa |

Recomendamos tener a mano el digesto de propiedades de listas para los ejercicios que siguen.

3. Expresar en lenguaje natural cada una de las siguientes expresiones; para ello primero calcular los rangos, ya sea como conjunto de tuplas o una tabla, y evaluar las expresiones para $xs = [9, -5, 1, -3]$ e $ys = [9, -5, 3]$.

   *a*) $\langle \forall\, as, bs \;:\; xs = as +\!\!+ bs :\; sum.as \geq 0 \,\rangle$

   *b*) $\langle \mathrm{Min}\;\; as, bs, cs \;:\; xs = as +\!\!+ bs +\!\!+ cs :\; sum.bs \,\rangle$

   *c*) $\langle \mathrm{N}\, as, b, bs \;:\; xs = as +\!\!+ (b \triangleright bs) :\; b > sum.bs \,\rangle$

   *d*) $\langle \mathrm{Max}\;\; as, bs, cs \;:\; xs = as +\!\!+ bs \wedge ys = as +\!\!+ cs :\; \#as \,\rangle$

   **Reflexión:** En los rangos que dividen la lista en dos o tres partes es conveniente pensar que la primera parte es un *prefijo* y la última un *sufijo*. Muchos problemas interesantes tienen que ver con encontrar el prefijo (o sufijo) de una lista con cierta propiedad. El último punto nos muestra que con segmentos es fácil especificar que dos listas tengan un segmento en común.

4. Expresar utilizando cuantificadores las siguientes sentencias del lenguaje natural; entre paréntesis está el nombre de la función.

   *a*) La lista $xs$ es un segmento inicial de la lista $ys$ (prefijo.$xs$.$ys$).

   *b*) La lista $xs$ es un segmento de la lista $ys$ (seg.$xs$.$ys$).

   *c*) La lista $xs$ es un segmento final de la lista $ys$ (sufijo.$xs$.$ys$).

   *d*) Las listas $xs$ e $ys$ tienen en común un segmento no vacío (segComun.$xs$.$ys$).

   *e*) La lista $xs$ posee un segmento que no es ni **prefijo** ni **sufijo** y cuyo mínimo es mayor a los valores del prefijo y del sufijo (hayMeseta.$xs$).

   *f*) La lista $xs$ de numeros enteros tiene la misma cantidad de elementos pares e impares (balanceada.$xs$).

   **Reflexión:** El último punto es de una naturaleza completamente distinta a los anteriores: valores pares e impares pueden aparecer dispersos en la lista, entonces los segmentos no serán útiles para especificarlo.

5. Derivar funciones recursivas para prefijo (4*a*) y seg (4*b*).

   **Reflexión:** Al derivar seg tenga presente la especificación de prefijo. ¿Qué relación hay entre seg y prefijo?

6. El predicado eco.$xs$ dice si la palabra $xs$ es la duplicación de un segmento (pensemos en sílabas).

$$\begin{aligned} &\text{eco}: \mathit{String} \to \mathit{Bool} \\ &\text{eco}.xs = \langle\, \exists\, as, bs \;:\; xs = as +\!\!+ bs \wedge as \neq [] :\; as = bs \,\rangle \end{aligned}$$

Calcular el rango para $xs =$"frufru"; luego decidir si eco."frufru".

7. Especificar y derivar semiEco: $\mathit{String} \to \mathit{Bool}$ que decide si una palabra comienza con eco. Ejemplos que satisfacen semiEco son: "paparulo", "mamado", "tetera", "ananá".

8. Derivar funciones para:

   *a*) Suma mínima de un segmento:

   $$\text{sumaMin}.xs = \langle \text{Min}\;\; as, bs, cs \;:\; xs = as +\!\!+ bs +\!\!+ cs :\;\; sum.bs \,\rangle$$

   *b*) Máxima longitud de elementos iguales a $e$:

   $$\text{maxLongEq}.e.xs = \langle \text{Max}\;\; as, bs, cs \;:\; xs = as +\!\!+ bs +\!\!+ cs \wedge iga.e.bs :\;\; \#bs \,\rangle$$

   donde $iga$ es la función del ejercicio *2.b* del práctico 2.

9. Dar el *tipo* de cada una de las siguientes funciones especificadas formalmente y describir su significado en lenguaje natural.

   *a*) $g.xs = \langle \text{Max}\, p, q \;:\; 0 \leq p < q < \#xs \;:\; xs\,!\,p + xs\,!\,q \rangle$
   *b*) $h.xs = \langle \text{N}\;\; k \;:\; 0 \leq k < \#xs \;:\; \langle \forall i \;:\; 0 \leq i < k \;:\; xs\,!\,i < xs\,!\,k \rangle \rangle$
   *c*) $k.xs = \langle \forall i, j \;:\; 0 \leq i \wedge 0 \leq j \wedge i + j = \#xs - 1 \;:\; xs\,!\,i = xs\,!\,j \rangle$
   *d*) $l.xs = \langle \text{Max}\, p, q \;:\; 0 \leq p \leq q < \#xs \wedge \langle \forall i \;:\; p \leq i < q \;:\; xs\,!\,i \geq 0 \rangle \;:\; q - p \rangle$

   **Reflexión:** Las cuantificaciones que tienen dos índices sobre una lista son complicadas. Recomendamos primero evaluar las especificaciones para listas concretas (y de no más de 5 elementos). ¿Hay alguna de estas especificaciones que escribirías de otra manera?

10. Derivar la función definida en el ejercicio 9*a*.

11. Derivar una definición recursiva para la función f : $[Num] \to [Num] \to Num$, que calcula la mínima distancia entre valores de las listas $xs$ e $ys$, y cuya especificación es la siguiente:

    $$\text{f}.xs.ys = \langle \text{Min}\, i, j \;:\; 0 \leq i < \#xs \wedge 0 \leq j < \#ys \;:\; |xs\,!\,i - ys\,!\,j| \rangle$$

12. Sea $\prec$ el orden en que aparecen las letras en el abecedario; por ejemplo, $a \prec b$ pero $h \not\prec d$ y $a \not\prec a$.

    Dada la especificación para la función lex:

    $$\begin{aligned} &\text{lex}: \mathit{String} \to \mathit{String} \to \mathit{Bool} \\ &\text{lex}.xs.ys = \langle \exists as, bs, c, cs \;:\; xs = as +\!\!+ bs \wedge ys = as +\!\!+ (c \triangleright cs) \;:\; bs = [\,] \vee bs\,!\,0 \prec c \rangle \end{aligned}$$

    Decir en palabras cuándo lex.$xs.ys$ es $True$; luego derivar una versión algorítmica para lex.

    **Reflexión:** Como siempre, puede ser conveniente explorar la especificación evaluando (no necesariamente calculando) ejemplos concretos, por ejemplo para los pares ("fa","fase") y ("mano","pie").

13. (Opcional) Lewis Carroll, el autor de "Alicia en el país de las maravillas", inventó el juego "Word ladder" (escalera de palabras) que consiste en que le primer jugader dice una palabra de cuatro letras y le siguiente dice otra que varíe de la primera en una sola letra. Por ejemplo: "lado", "lodo", "todo", "toro".

    Especificar la función escalera : $[\mathit{String}] \to \mathit{Bool}$ que decide si la secuencia de palabras respeta la condición del juego. Puede ser útil modularizar la decisión de si una palabra es igual a otra excepto en una letra.