Funciones de orden superior y polimorfismo

Parcialización

$$
\begin{aligned}
&inc \quad :: Integer \rightarrow Integer\\
&inc\ x\ = x + 1
\end{aligned}
$$

Puede escribirse como:

$$
\begin{aligned}
&inc :: Integer \rightarrow Integer\\
&inc = \lambda\, x \rightarrow x + 1
\end{aligned}
$$

$inc\ 10$
$\Longrightarrow$! por la definición de inc
$(\lambda\, x \rightarrow x + 1)\ 10$
$\Longrightarrow$! sustituyendo x por 10 en el cuerpo de la λ-expresión
$(10 + 1)$
$\Longrightarrow$! por el operador $(+)$
11

Regla de λ-abstracciones

$$\lambda\ x_1\ x_2 \ldots x_n \rightarrow e \rightsquigarrow \lambda x_1 \rightarrow (\lambda x_2 \rightarrow (\cdots \rightarrow (\lambda x_n \rightarrow e)))$$

$$\textit{Asociatividad a la derecha de } (\rightarrow)$$

$$t_1 \rightarrow t_2 \rightarrow \ldots t_n \rightsquigarrow (t_1 \rightarrow (t_2 \rightarrow (\cdots \rightarrow t_n)))$$

$sumaCuadrados :: Integer \rightarrow (Integer \rightarrow Integer)$
$sumaCuadrados = \lambda\, x \rightarrow (\lambda\, y \rightarrow x * x + y * y)$

$$\textit{Asociatividad izquierda de la aplicación de funciones}$$

$$f\ \ a_1\ \ a_2 \ldots a_n \rightsquigarrow (((f\ \ a_1)\ \ a_2) \ldots\ \ a_n)$$

$sumaCuadrados\ 2\ 3$
$\rightsquigarrow$! la aplicación es asociativa a la izquierda
$(sumaCuadrados\ 2)\ 3$
$\Longrightarrow$! definición de $sumaCuadrados$
$((\lambda\, x \rightarrow (\lambda\, y \rightarrow x * x + y * y))\ 2)\ 3$
$\Longrightarrow$! sustituyendo x por 2 en el cuerpo de la λ-expresión
$(\lambda\, y \rightarrow 2 * 2 + y * y)\ 3$
$\Longrightarrow$! sustituyendo y por 3 en el cuerpo de la λ-expresión
$2 * 2 + 3 * 3$
$\Longrightarrow$! por el operador $(*)$
$4 + 3 * 3$
$\Longrightarrow$! por el operador $(*)$
$4 + 9$
$\Longrightarrow$! por el operador $(+)$
13

Aplicación parcial

- Las funciones de más de un argumento se pueden interpretar como funciones que toman un único argumento y devuelven como resultado otra función con un argumento menos.

$$
\begin{array}{ll}
\textit{múltiploDe} & :: \ \textit{Integer} \ \rightarrow \ \textit{Integer} \ \rightarrow \ \textit{Bool} \\
\textit{múltiploDe} \ p \ n & = n \ \text{`}\textit{mod}\text{`} \ p == 0 \\
\textit{esPar} & :: \ \textit{Integer} \ \rightarrow \ \textit{Bool} \\
\textit{esPar} & = \textit{múltiploDe} \ 2
\end{array}
$$

$$
\begin{array}{ll}
\text{MAIN}> & \textit{múltiploDe} \ 3 \ 15 \\
\textit{True} \ :: & \textit{Bool} \\
\text{MAIN}> & \textit{esPar} \ 18 \\
\textit{True} \ :: & \textit{Bool}
\end{array}
$$

$\quad \textit{esPar}$

$\equiv$! definición de $\textit{esPar}$

$\quad \textit{múltiploDe} \ 2$

$\equiv$! definición de $\textit{múltiploDe}$

$\quad (\lambda \ p \ \rightarrow \ (\lambda \ n \ \rightarrow \ n \ \text{`}\textit{mod}\text{`} \ p == 0 \ ) \ ) \ 2$

$\equiv$! sustituyendo p por 2 en el cuerpo de la λ-expresión

$\quad \lambda \ n \ \rightarrow \ n \ \text{`}\textit{mod}\text{`} \ 2 == 0$

Tipificado de la aplicación de funciones
si $f :: \ t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n$ y $x_1 :: \ t_1$, entonces $f \ x_1 :: \ t_2 \rightarrow \cdots \rightarrow t_n$

- Hay dos modos de representar funciones de varios argumentos en HASKELL:

$f \;::\; t_1 \to t_2 \to \ldots t_n \to t_r$ $\qquad$ $f' \;::\; (t_1,\, t_2,\, \ldots,\, t_n) \to t_r$

$f\; x_1\; x_2\; \ldots\; x_n \;=\; \ldots \quad$ -- Forma parcializada $\qquad$ $f'\; (x_1,\, x_2,\, \ldots,\, x_n) \;=\; \ldots \quad$ -- Forma no parcializada

Solo la primera puede parcializarse:

Si $v_1 \;::\; t_1$ entonces $f\; v_1 \;::\; t_2 \to \ldots t_n \to t_r$

Si $v_1 \;::\; t_1,\; v_2 \;::\; t_2$ entonces $f\; v_1\; v_2 \;::\; t_3 \to \ldots t_n \to t_r$

$\ldots$

Si $v_1 \;::\; t_1,\; v_2 \;::\; t_2,\; \ldots\; v_n \;::\; t_n$ entonces $f\; v_1\; v_2\; \ldots\; v_n \;::\; t_r$

η-reducción

$\lambda x \to expr\; x \qquad \equiv \qquad expr$ $\quad$ si x no aparece libre en $expr$

$g\; x\; y \;= x + 4 + y$

$f\; x \;= g\; 5\; x$

La función f puede definirse como $f = g\; 5$. En cambio la siguiente no puede parcializarse:

$g\; x\; y \;= x + 4 + y$

$f\; x \;= g\; (x + 1)\; x$

- $curry$ y $uncurry$ permiten convertir una función de dos argumentos no parcializada a la forma estándar y viceversa

Secciones

- Es posible aplicar a un operador menos de dos argumentos.
  ✓ Una expresión así se la denomina *sección*.

$\textsc{Prelude}>\ 5 \uparrow 2$
$25 :: \ \mathit{Integer}$

$\textsc{Prelude}>\ (\uparrow)\, 5\, 2$
$25 :: \ \mathit{Integer}$

- Para el operador potencia, tres posibles secciones son:

  $(5\ \uparrow)$ la función que toma un argumento y eleva el valor 5 a éste

  $(\uparrow\ 2)$ la función que toma un argumento y eleva éste a 2

  $(\uparrow)$ la función que toma dos argumentos y eleva el primero al segundo

Los paréntesis son obligatorios.

Secciones de operadores

$$
\begin{array}{ccl}
(x \bigotimes) & \rightsquigarrow & (\lambda y \rightarrow x \bigotimes y) \\
(\bigotimes y) & \rightsquigarrow & (\lambda x \rightarrow x \bigotimes y) \\
(\bigotimes) & \rightsquigarrow & (\lambda x\ y \rightarrow x \bigotimes y)
\end{array}
$$

- Una excepción a la reglas de las secciones es el operador $(-)$.
  ✓ Una expresión con la forma $(-\ e)$ no es una sección sino que representa la negación de e.

$\mathit{inc} :: \mathit{Integer} \rightarrow \mathit{Integer}$
$\mathit{inc} = (+1)$

$\mathit{alCubo} :: \mathit{Integer} \rightarrow \mathit{Integer}$
$\mathit{alCubo} = (\uparrow\ 3)$

$\textsc{Main}>\ \mathit{inc}\ 10$
$11 :: \ \mathit{Integer}$

$\textsc{Main}>\ \mathit{alCubo}\ 5$
$125 :: \ \mathit{Integer}$

Funciones de orden superior

- Las funciones son datos de *primera clase*.
  - ✓ Una función puede aparecer en cualquier lugar donde pueda aparecer un dato de otro tipo

$$
\begin{array}{l}
\mathit{lista} :: [\mathit{Integer} \rightarrow \mathit{Integer}] \\
\mathit{lista} = [\, (\lambda\, x \rightarrow x + 1),\ (+1),\ \mathit{dec},\ (\uparrow\ 2)\,] \\
\quad \textbf{where} \\
\qquad \mathit{dec}\ x = x - 1
\end{array}
$$

$$
\begin{array}{l}
\mathit{dosVeces} \quad :: (\mathit{Integer} \rightarrow \mathit{Integer}) \rightarrow \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{dosVeces}\ f\ x = f\ (f\ x)
\end{array}
$$

$$
\begin{array}{l}
\text{MAIN>}\ \mathit{dosVeces}\ (\lambda\, x \rightarrow x + 1)\ 10 \\
12 :: \mathit{Integer} \\
\text{MAIN>}\ \mathit{dosVeces}\ (*2)\ 10 \\
40 :: \mathit{Integer} \\
\text{MAIN>}\ \mathit{dosVeces}\ \mathit{inc}\ 10 \\
12 :: \mathit{Integer}
\end{array}
$$

los paréntesis son obligatorios en el tipo de la función.

Funciones de orden superior sobre enteros

- Muchas de las funciones recursivas que se definen sobre enteros siguen el siguiente esquema inductivo:

▶ **Caso Base** Resultado de la función cuando el argumento es cero.

▶ **Paso Inductivo** Resultado de la función cuando el argumento es $n+1$ en función del resultado cuando el argumento es n.

✓ Este tipo de definiciones se llaman inductivas.

$$\begin{array}{ll} \mathit{factorial} & :: \mathit{Integer} \to \mathit{Integer} \\ \mathit{factorial}\ 0 & = 1 \\ \mathit{factorial}\ m@(n+1) & = (*)\ m\ (\mathit{factorial}\ n) \end{array}$$

$$\begin{array}{ll} \mathit{sumatorio} & :: \mathit{Integer} \to \mathit{Integer} \\ \mathit{sumatorio}\ 0 & = 0 \\ \mathit{sumatorio}\ m@(n+1) & = (+)\ m\ (\mathit{sumatorio}\ n) \end{array}$$

- Ambas funciones siguen la siguiente plantilla:

$$\begin{array}{ll} \mathit{fun} & :: \mathit{Integer} \to \mathit{Integer} \\ \mathit{fun}\ 0 & = \boxed{e} \\ \mathit{fun}\ m@(n+1) & = \boxed{op}\ m\ (\mathit{fun}\ n) \end{array}$$

- Es posible definir una función de orden superior que tome como argumentos la operación a aplicar en el caso recursivo y el valor a devolver en el caso base:

$$\begin{array}{ll} \mathit{iter} & :: (\mathit{Integer} \to \mathit{Integer} \to \mathit{Integer}) \to \\ & \quad \mathit{Integer} \to \\ & \quad \mathit{Integer} \to \\ & \quad \mathit{Integer} \\ \mathit{iter}\ \ op\ e\ 0 & = e \\ \mathit{iter}\ \ op\ e\ m@(n+1) & = op\ m\ (\mathit{iter}\ op\ e\ n) \end{array}$$

las funciones anteriores se escriben como:

$$\begin{array}{l} \mathit{factorial}' :: \mathit{Integer} \to \mathit{Integer} \\ \mathit{factorial}' = \mathit{iter}\ (*)\ 1 \end{array}$$

$$\begin{array}{l} \mathit{sumatorio}' :: \mathit{Integer} \to \mathit{Integer} \\ \mathit{sumatorio}' = \mathit{iter}\ (+)\ 0 \end{array}$$

$$\begin{array}{ll} & \mathit{factorial}'\ 2 \\ \Longrightarrow & \text{! por definición de } \mathit{factorial}' \\ & \mathit{iter}\ (*)\ 1\ 2 \\ \Longrightarrow & \text{! por segunda ecuación de } \mathit{iter} \\ & (*)\ 2\ (\mathit{iter}\ (*)\ 1\ 1) \\ \Longrightarrow & \text{! por segunda ecuación de } \mathit{iter} \\ & \ldots \\ & 2 \end{array}$$

Polimorfismo

- *Funciones polimórficas* = Tienen sentido para más de un tipo.
- El ejemplo más simple de función polimórfica es la *identidad*.

$$identidad\ x = x$$

$\textsc{Main}>$ $identidad\ 'd'$
$'d' :: Char$

$\textsc{Main}>$ $identidad\ True$
$True :: Bool$

$\textsc{Main}>$ $identidad\ [1,2,3]$
$[1,2,3] :: [Integer]$

✓ El tipo de la función identidad es:

$$identidad :: a \to a$$

donde a es una *variable de tipo*

✓ Debe leerse como $identidad :: \forall\, a\ .\ a \to a$.

✓ $(Char \to Char, Bool \to Bool, \ldots)$ constituyen un uso válido de la función $identidad$.

$$unaVez\ f\ x = f\ x$$

$\textsc{Main}> :t\ unaVez$
$unaVez :: (a \to b) \to a \to b$

$\textsc{Main}>$ $unaVez\ (+1)\ 7$
$8 :: Integer$

$\textsc{Main}>$ $unaVez\ (==0)\ 9$
$False :: Bool$

$\textsc{Main}>$ $unaVez\ inc\ 'p'$

$ERROR$	: $Type\ error\ in\ application$
$***\ Expression$	: $unaVez\ inc\ 'p'$
$***\ Term$	: inc
$***\ Type$	: $Integer \to Integer$
$***\ Does\ not\ match$	: $Char \to Integer$

- ¿Cuál es el tipo más general de la función $dosVeces$?

$$dosVeces\ f\ x = f\ (f\ x)$$

La composición de funciones

$$(g \circ f)(x) = g(f(x))$$

$\textbf{infixr}\ 9\ .$

$(.) \quad :: (b \to c) \to (a \to b) \to (a \to c)$
$g\ .\ f \ = \lambda\, x \to g\ (f\ x)$

$sumayMult :: Integer \to Integer$
$sumayMult \ = (*10)\ .\ (+1)$

$\textsc{Main}\!> \ sumayMult\ 5$
$60 :: Integer$

$esPar,\ esImpar :: Integer \to Bool$
$esPar\ x \quad = (x\ \text{`}mod\text{`}\ 2 == 0)$
$esImpar \quad = not\ .\ esPar$

$f :: Integer \to Integer$
$f \ = (+1)\ .\ (*3)\ .\ (\uparrow\ 2)$

es equivalente a

$f \ = (+1)\ .\ ((*3)\ .\ (\uparrow 2))$

- El operador $(\$)$.

$\textbf{infixr}\ 0\ \$$
$f\ \$\ x \ = f\ x$

$\textsc{Prelude}\!> \ ((+1)\ .\ (*3)\ .\ (\uparrow\ 2))\ 10$
$301 :: Integer$
$\textsc{Prelude}\!> \ (+1)\ .\ (*3)\ .\ (\uparrow 2)\ \$\ 10$
$301 :: Integer$

$dosVeces\ f \ = f\ .\ f$

- Algunas definiciones alternativas para $unaVez$

$unaVez \quad :: (a \to b) \to a \to b$
$unaVez\ f\ x \ = f\ x$

$unaVez \quad :: t \to t$
$unaVez\ f \ = f$

$unaVez :: t \to t$
$unaVez \ = id$

- Al parcializar se puede perder información:

Ecuación	Tipo
$fun\ f\ x\ y \ = f\ x\ y$	$(a \to b \to c) \to a \to b \to c$
$fun\ f\ x \ = f\ x$	$(a \to b) \to a \to b$
$fun\ f \ = f$	$a \to a$

Otras funciones polimórficas

$$flip :: (a \to b \to c) \to (b \to a \to c)$$
$$flip\ f\ x\ y = f\ y\ x$$

- Un operador sobre listas que concatene la primera tras la segunda.

$$(\rtimes\!\!\!\ltimes) :: [a] \to [a] \to [a]$$
$$(\rtimes\!\!\!\ltimes) = flip\ (+\!\!+)$$

$$\text{MAIN>}\ [1,2,3] \rtimes\!\!\!\ltimes [5,6]$$
$$[5,6,1,2,3] :: Integer$$

- Otro ejemplo de uso de *flip*

$$fun :: Integer \to Integer \to Integer$$
$$fun\ x\ y = 2 * x + y$$

$$f :: Integer \to Integer$$
$$f = fun\ 3 \quad \text{-- La función que suma 6}$$

$$g :: Integer \to Integer$$
$$g = flip\ fun\ 10 \quad \text{-- La función que duplica y suma 10}$$

- para escribir secciones:

$$\text{PRELUDE>}\ \textbf{let}\ mitad = flip\ (/)\ 2\ \textbf{in}\ mitad\ 10$$
$$5.0 :: Double$$

- Con el operador predefinido $(\$)$ se cumplen las siguientes igualdades

$$(\$)\ f = (f\ \$) = f$$
$$(\$\ x) = flip\ (\$)\ x = \lambda f \to f\ x$$

- Las funciones *curry* y *uncurry* son también polimórficas.

Polimorfismo en listas

$$length :: [a] \to Int$$
$$length\ [\,] = 0$$
$$length\ (x : xs) = 1 + length\ xs$$

$$\text{PRELUDE>}\ length\ [1,2,3]$$
$$3 :: Int$$
$$\text{PRELUDE>}\ length\ [True,\ False,\ False]$$
$$3 :: Int$$
$$\text{PRELUDE>}\ length\ [[1,2,3],\ [4,5]]$$
$$2 :: Int$$
$$\text{PRELUDE>}\ length\ [(+2),\ (*5)]$$
$$2 :: Int$$

✓ Tipo de los constructores de listas

$$(:) :: a \to [a] \to [a]$$
$$[\,] :: [a]$$

- La función map, definida en el PRELUDE

$$\begin{array}{ll} map & :: (a \to b) \to [a] \to [b] \\ map\ f\ [\,] & = [\,] \\ map\ f\ (x : xs) & = f\ x\ :\ map\ f\ xs \end{array}$$

PRELUDE> $map\ (\uparrow\ 2)\ [1,2,3]$
$[1,4,9]\ ::\ [Integer]$
PRELUDE> $map\ toUpper$ "pepe"
"PEPE" $::\ String$

Polimorfismo en tuplas

$$\begin{array}{ll} fst & :: (a, b) \to a \\ fst\ (x, \_) & = x \end{array} \qquad \begin{array}{ll} snd & :: (a, b) \to b \\ snd\ (\_, y) & = y \end{array}$$

- ¿Cuál es el tipo más general de las siguientes funciones?

$$\begin{array}{ll} const\ x\ y & = x \\ subst\ f\ g\ x & = f\ x\ (g\ x) \\ flip\ f\ x\ y & = f\ y\ x \\ curry\ f\ x\ y & = f\ (x,\ y) \\ uncurry\ f\ (x, y) & = f\ x\ y \\ pair\ (f, g)\ x & = (f\ x,\ g\ x) \\ cross\ (f, g)\ (x, y) & = (f\ x,\ g\ y) \end{array}$$

Un iterador polimórfico sobre los naturales

$$\begin{array}{ll} iter & :: (Integer \to a \to a) \to \\ & \quad a \to \\ & \quad Integer \to \\ & \quad a \\ iter\ \ op\ e\ 0 & = e \\ iter\ \ op\ e\ m@(n+1) & = op\ m\ (iter\ op\ e\ n) \end{array}$$

$$\begin{array}{l} listaDecre\ ::\ Integer \to [Integer] \\ listaDecre\ = iter\ (:)\ [\,] \end{array}$$

$$\begin{array}{l} palos\ ::\ Integer \to String \\ palos\ = iter\ (\lambda\ n\ xs\ \to\ 'I' : xs)\ [\,] \end{array}$$

MAIN> $listaDecre\ 5$
$[5,4,3,2,1]\ ::\ [Integer]$
MAIN> $palos\ 3$
"III" $::\ String$