# Tema 4. Funciones de orden superior

# 4.1 Funciones de orden superior

- ✓ Una función tal que alguno de sus argumentos es una función o que devuelve una función como resultado.

- ✓ Son útiles porque permiten capturar esquemas de cómputo generales (*abstracción*).

- ✓ Son más útiles que las funciones normales (parte del comportamiento se especifica al usarlas).

Ejemplo:

$$\begin{array}{lcl} dosVeces & :: & (Integer \to Integer) \to Integer \to Integer \\ dosVeces\ f\ x & = & f\ (f\ x) \end{array}$$

$$\begin{array}{lcl} inc & :: & Integer \to Integer \\ inc\ x & = & x + 1 \end{array}$$

$$\begin{array}{lcl} dec & :: & Integer \to Integer \\ dec\ x & = & x - 1 \end{array}$$

- ✓ El primer argumento de $dosVeces$ debe ser una función con tipo $Integer \to Integer$.

- ✓ Los paréntesis en el tipo de $dosVeces$ son **obligatorios**.

Uso

**?** $dosVeces\ inc$ 10
12 $::$ $Integer$

**?** $dosVeces\ dec$ 10
8 $::$ $Integer$

# 4.2 Expresiones lambda

✓ Permiten definir funciones *anónimas* (sin nombre).

Ejemplo:

$\lambda\ x \rightarrow x\ +\ 1$ denota en Haskell la función que toma un argumento ($x$) y lo devuelve incrementado.

? $\lambda\ x \rightarrow x\ +\ 1$
$\ll function \gg\ ::\ Integer \rightarrow Integer$

? $(\lambda\ x \rightarrow x\ +\ 1)\ 10$
$11\ ::\ Integer$

Paso a paso:

$(\lambda\ x \rightarrow x\ +\ 1)\ 10$
$\Longrightarrow$ {Sustitutyendo el argumento $x$ por 10}
$10\ +\ 1$
$\Longrightarrow$ {por $(+)$}
$11$

✓ Funciones de más de un argumento con la notación lambda:

? $(\lambda\ x\ y \rightarrow x\ +\ y)$
$\ll function \gg\ ::\ Integer \rightarrow Integer \rightarrow Integer$

? $(\lambda\ x\ y \rightarrow x\ +\ y)\ 5\ 7$
$12\ ::\ Integer$

✓ Son útiles como argumentos de funciones de orden superior:

? $dosVeces\ (\lambda\ x \rightarrow x\ +\ 1)\ 10$
$12\ ::\ Integer$

? $dosVeces\ (\lambda\ x \rightarrow x\ -\ 1)\ 10$
$8\ ::\ Integer$

? $dosVeces\ (\lambda\ x \rightarrow x\ *\ 2)\ 10$
$40\ ::\ Integer$

# 4.3 Aplicación parcial

✓ Permite aplicar a una función menos argumentos de los que tiene para obtener una nueva función

**Aplicación parcial** o **parcialización**: Si $f$ es una función de $n$ argumentos y se le aplican $k \leq n$ argumentos con los tipos adecuados, se obtiene como resultado una nueva función que espera los $n - k$ argumentos restantes.

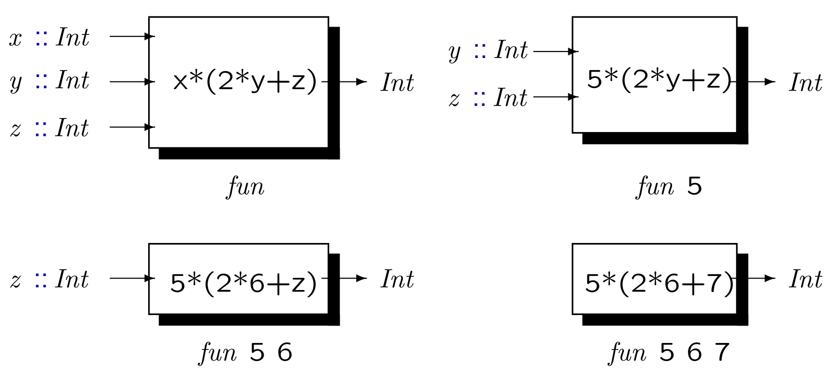
Regla de la cancelación

| | |
|---|---|
| si | $f :: t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n \rightarrow t_r$ |
| y | $e_1 :: t_1,\ e_2 :: t_2 \ldots, e_k :: t_k$ con $(k \leq n)$ |
| entonces | $f\ e_1\ e_2\ \ldots\ e_k :: t_{k+1} \rightarrow t_{k+2} \ldots \rightarrow t_n \rightarrow t_r$ |

Ejemplo: A la siguiente función de tres argumentos:

$$
\begin{array}{lcl}
fun & :: & Int \rightarrow Int \rightarrow Int \rightarrow Int \\
fun\ x\ y\ z & = & x * (2 * y + z)
\end{array}
$$

es posible aplicarle uno, dos o tres argumentos. En cada caso obtenemos una función con un argumento menos.



## Aplicación parcial (2)

Todo esto funciona gracias a los siguientes convenios

> Asociatividad a la derecha de $(\rightarrow)$ (En Tipos)
>
> $t_1 \rightarrow t_2 \rightarrow \ldots t_n \quad \approx\!\approx\!\approx\!> \quad (t_1 \rightarrow (t_2 \rightarrow (\ldots \rightarrow t_n)))$

> Asociatividad izquierda de la aplicación de funciones
>
> $f a_1 a_2 \ldots a_n \quad \approx\!\approx\!\approx\!> \quad (((f\ a_1)\ a_2) \ldots\ a_n)$

Consideremos la función anterior:

$$
\begin{array}{lll}
\mathit{fun} & :: & \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \\
\mathit{fun}\ x\ y\ z & = & x * (2 * y + z)
\end{array}
$$

Por las reglas anteriores la definición es equivalente a:

$$
\begin{array}{lll}
\mathit{fun} & :: & \mathit{Int} \rightarrow (\mathit{Int} \rightarrow (\mathit{Int} \rightarrow \mathit{Int})) \\
\mathit{fun}\ x\ y\ z & = & x * (2 * y + z)
\end{array}
$$

- ✓ La expresión $\mathit{fun}$ 5 tiene tipo $\mathit{Int} \rightarrow (\mathit{Int} \rightarrow \mathit{Int})$
- ✓ La expresión $\mathit{fun}$ 5 6 es equivalente a $(\mathit{fun}$ 5) 6 y tiene tipo $\mathit{Int} \rightarrow \mathit{Int}$
- ✓ La expresión $\mathit{fun}$ 5 6 7 es equivalente a $((\mathit{fun}$ 5) 6) 7 y tiene tipo $\mathit{Int}$

# Secciones

- ✓ Los operadores pueden ser aplicados parcialmente
- ✓ Se obtienen funciones de un argumento

Si $(\bigstar)$ es un operador tenemos las siguientes equivalencias (los paréntesis son obligatorios):

**Secciones de operadores**

$$
\begin{array}{lcl}
(x\ \bigstar) & \rightsquigarrow & \lambda\ y \rightarrow x \bigstar y \\
(\bigstar\ y) & \rightsquigarrow & \lambda\ x \rightarrow x \bigstar y \\
(\bigstar) & \rightsquigarrow & \lambda\ x\ y \rightarrow x \bigstar y
\end{array}
$$

Ejemplos:

- $(2.0/)$ Toma un valor real $x$ y devuelve $2.0/x$.
- $(/2.0)$ Toma un valor real $x$ y devuelve $x/2.0$.
- $(/)$ Toma dos valores reales y devuelve su cociente.
- $(>2)$ Toma un argumento y devuelve *True* si es mayor que 2.
- $(2>)$ Toma un argumento y devuelve *True* si es menor que 2.

**?** (/2.0) 8.0
4.0 :: *Double*

**?** *dosVeces* (+1) 10
12 :: *Integer*

- ✓ Excepción: $(-e)$ donde $e$ es una expresión **NO es una sección**

## 4.4 Ejemplo: una función de orden superior para enteros

Muchas funciones sobre enteros siguen el siguiente esquema:

- Caso base: cuando el argumento es 0
- Paso recursivo: se calcula el resultado para $n+1$ a partir del resultado para $n$

Ejemplos:

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

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

Ambas siguen el esquema:

$$
\begin{array}{lcl}
\mathit{fun} & :: & \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{fun}\ 0 & = & \boxed{e} \\
\mathit{fun}\ m@(n+1) & = & \boxed{f}\ m\ (\mathit{fun}\ n)
\end{array}
$$

Función de orden superior que lo captura:

$$
\begin{array}{lcl}
\mathit{iter} & :: & (\mathit{Integer} \rightarrow \mathit{Integer} \rightarrow \mathit{Integer}) \rightarrow \mathit{Integer} \rightarrow \\
 & & (\mathit{Integer} \rightarrow \mathit{Integer}) \\
\mathit{iter}\ f\ e & = & \mathit{fun}
\end{array}
$$

$$
\begin{array}{l}
\quad \textbf{where} \\
\qquad \begin{array}{lcl}
\mathit{fun} & :: & \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{fun}\ 0 & = & e \\
\mathit{fun}\ m@(n+1) & = & f\ m\ (\mathit{fun}\ n)
\end{array}
\end{array}
$$

Ahora es posible una definición más compacta:

$$
\begin{array}{lcl}
\mathit{factorial}' & :: & \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{factorial}' & = & \mathit{iter}\ (*)\ 1
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{sumatorio}' & :: & \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{sumatorio}' & = & \mathit{iter}\ (+)\ 0
\end{array}
$$

# Objetivos del tema

El alumno debe:

- ✓ Conocer el concepto de función de orden superior
- ✓ Saber definir y utilizar funciones de orden superior
- ✓ Saber utilizar lambda expresiones
- ✓ Conocer el concepto de aplicación parcial y los convenios que hacen que tenga sentido
- ✓ Saber construir nuevas funciones aplicando parcialmente otras
- ✓ Conocer el tipo y el significado de una expresión construída mediante una aplicación parcial
- ✓ Entender que es posible capturar un patrón de cómputo habitual mediante una función de orden superior (*abstracción*) y cómo definir casos concretos de dicho patrón (*concreción*)