

Tipos, currificación y alto orden

Programación funcional

La clase pasada vimos:

- ▶ Expresiones
- ▶ Reducción

Declaración de tipos:

```
isWeekend :: Day -> Bool
```

```
not :: Bool -> Bool
```

```
inc :: Nat -> Nat
```

```
type String = [Char]
```

Visión denotacional:

- ▶ **Valores:** elementos abstractos.
- ▶ **Expresiones:** construcciones sintácticas (correctas) que denotan valores.
- ▶ **Tipos:** conjuntos de valores con propiedades comunes.

Tipos básicos:

- ▶ Datos: A
- ▶ Funciones: $A \rightarrow B$

Ejemplo: `isWeekend :: Day -> Bool`

Ventajas:

- ▶ Detección de errores.
- ▶ Especificación rudimentaria.
- ▶ Optimización durante la compilación.

Polimorfismo paramétrico

¿Qué tipo tiene la función `id` $x = x$?

```
id :: a -> a
```

Donde `a` es un parámetro de tipo.

Bottom

¿Qué tipo tiene `bottom`?

```
bottom = bottom
```

Aplicando las siguientes reglas podemos hacer chequeo e inferencia:

$$\frac{\begin{array}{l} f :: A \rightarrow B \\ x :: A \end{array}}{f\ x :: B}$$
$$\frac{\begin{array}{l} C_1, \dots, C_n :: A \\ T_1, \dots, T_n :: B \end{array}}{\text{case } C_i \text{ of } \{C_1 \rightarrow T_1; \dots; C_n \rightarrow T_n\} :: B}^1$$

! Una expresión que no tipa es inválida.

¹Abuso del lenguaje entre expresiones y patrones.

¿Qué tipo tiene el operador `(/=)`?

```
x /= y = not (x == y)
```

Tenemos que pedir que `x` e `y` sean comparables por `(==)`.

```
class Eq a where  
  (==) :: a -> a -> Bool
```

Entonces:

```
(/=) :: Eq a => a -> a -> Bool
```

Funciones como valores

! Las funciones son valores, al igual que los números o los booleanos.

Funciones como valores

! Las funciones son valores, al igual que los números o los booleanos.

¿Qué implica?

- ▶ Pueden ser argumento de otras funciones.
- ▶ Pueden ser resultado de otras funciones.
- ▶ Pueden almacenarse en estructuras de datos.
- ▶ Pueden representar (estructuras de) datos.

Funciones como valores

! Las funciones son valores, al igual que los números o los booleanos.

¿Qué implica?

- ▶ Pueden ser argumento de otras funciones.
- ▶ Pueden ser resultado de otras funciones.
- ▶ Pueden almacenarse en estructuras de datos.
- ▶ Pueden representar (estructuras de) datos.

Llamamos funciones de alto orden a las funciones que:

- ▶ toman funciones como parámetro; o
- ▶ retornan funciones como resultado.

Funciones como valores

! Las funciones son valores, al igual que los números o los booleanos.

¿Qué implica?

- ▶ Pueden ser argumento de otras funciones.
- ▶ Pueden ser resultado de otras funciones.
- ▶ Pueden almacenarse en estructuras de datos.
- ▶ Pueden representar (estructuras de) datos.

Llamamos funciones de alto orden a las funciones que:

- ▶ toman funciones como parámetro; o
- ▶ retornan funciones como resultado.

Ejemplos:

```
const  :: a -> (b -> a)
twice  :: (a -> a) -> (a -> a)
flip   :: (a -> b -> c) -> (b -> a -> c)
```



¿Cómo definimos funciones que toman múltiples parámetros?

Haskell Brooks Curry

Con funciones que retornan funciones:

```
const :: a -> (b -> a)
const x = \y -> x
```

```
apply :: (a -> b) -> (a -> b)
apply f = \x -> f x
```

```
(.) :: (a -> b) -> (b -> c) -> (a -> c)
(.) f = \g -> \x -> f (g x)
```

¿Podemos ahorrarnos paréntesis?

La aplicación de funciones asocia a izquierda:

```
(const False) True ≡ const False True
```

El tipo de las funciones asocia a derecha:

```
const :: a -> (b -> a) ≡ const :: a -> b -> a
```

Ejemplos:

```
twice  :: (a -> a) -> a -> a
flip   :: (a -> b -> c) -> b -> a -> c
apply  :: (a -> b) -> a -> b
(.)     :: (a -> b) -> (b -> c) -> a -> c
```

Consideramos la siguiente equivalencia:

$$(\backslash x \rightarrow f\ x) \equiv f$$

Ejemplo:

$$\begin{aligned} \text{inc} = \backslash n \rightarrow 1 + n &\equiv \text{inc} = (+)\ 1 \\ &\equiv \text{inc} = (1+) \quad \text{-- seccion de operadores} \end{aligned}$$

Currificar o no currificar esa es la cuestión

Desafío

Defina las siguientes funciones:

```
curry :: ((a,b) -> c) -> a -> b -> c
```

```
uncurry :: (a -> b -> c) -> (a,b) -> c
```