Programación Funcional

Curso 2019-20

INTRODUCIENDO LOS TIPOS EN HASKELL

Tipos: cuestiones básicas (I)

`>:t` *expresion* $\rightsquigarrow$ muestra el tipo de *expresion*, sin evaluarla

`e::τ` $\rightsquigarrow$ indica que e tiene tipo τ

```
>:t True
True::Bool
```

```
>:t 'a'
'a'::Char
```

```
>:t ['a','b','c']
['a','b','c']::[Char]
```

```
>:t 1 == 2
1 == 2::Bool
```

```
>:t "h"++"ola"
"h"++"ola"::[Char]
```

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

$\rightsquigarrow$ Tipo funcional

```
>:t (&&)
(&&)::Bool -> Bool -> Bool
```

Tipos: cuestiones básicas (II)

Los tipos no intervienen en los cómputos

- `:t` es un comando del intérprete, no una función (no sirve para formar expresiones)
- `Bool`, `Char`, ..., son tipos, no valores (datos). No se puede formar expresiones con ellos.

```
> Bool == Char
Error:  Bool, Char, not in scope
```

Tipos: cuestiones básicas (III)

Tipos básicos

`Char` `Bool` `Int` `Integer` `Float` `Double` ...

Tipo(s) tupla

`(Char,Int)` `(Int,Int,Int)` `()` ...

Tipo(s) lista

`[Bool]` `[(Char,Int)]` `[[Int]]` ... ~~`[Int,Int,Int]`~~ 

Tipos funcionales

`Bool -> Bool` `Int -> Char` Funciones de aridad 1

`Bool -> Bool -> Bool` Función de aridad 2

☞Todos estos son tipos monomórficos

Tipos: cuestiones básicas (IV)

Tipos polimórficos (polimorfismo *paramétrico*)

- Contienen variables de tipo (parámetros del tipo), que representan tipos cualesquiera.
- Cada valor concreto de los parámetros determina una instancia concreta del tipo polimórfico.
- El usuario podrá definir nuevos tipos (polimórficos o no)

```
>:t head
head::[a] -> a
```

```
>:t fst
fst::(a,b) -> a
```

```
>:t []
[]::[a]
```

- Expresiones polimórficas
- `head` puede aplicarse a una expresión e si e`::[`τ`]`, siendo τ un tipo cualquiera, y (`head` e) tendrá entonces el tipo τ

Tipos: cuestiones básicas (V)

Tipos cualificados (polimorfismo *ad hoc* $\rightsquigarrow$ clases de tipos)

- Contienen variables de tipo restringidas a pertenecer a una (o varias) cierta familia de tipos (*clase de tipos*)

```
>:t (+)
(+)::Num a => a -> a -> a
```

- + puede aplicarse a e_1 y e_2 si $e_1 :: \tau$ y $e_2 :: \tau$, siendo τ un tipo cualquiera que satisfaga la restricción `Num` τ, es decir, que τ sea un tipo de la clase de tipos `Num` (que es una clase de tipos predefinida).
- ¿Qué tipos están en (o son instancia de) la clase `Num`?
  `Int, Integer, Float, Double, ...`
- El usuario podrá definir sus propias clases de tipos, e instancias de ellas, así como nuevas instancias de clases ya existentes.

Tipos: cuestiones básicas (VI)

Otros ejemplos de tipos cualificados

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

- `Eq` es una clase de tipos (no un tipo)
- En `Eq` están casi todos los tipos

```
>:t (<=)
(<=)::Ord a => a -> a -> Bool
```

- `Ord` es una clase de tipos (no un tipo)
- En `Ord` están casi todos los tipos

```
>:t succ
succ::Enum a => a -> a
```

En GHCi: `>:info` *nombre* muestra información de *nombre*