Programación Funcional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Polimorfismo, Sobrecarga y Alto Orden

Polimorfismo paramétrico

Polimorfismo Paramétrico

El **polimorfismo paramétrico** es un mecanismo de tipado que permite definir funciones y tipos de datos de forma paramétrica de forma tal que puedan manipular valores de distintos tipos de forma totalmente uniforme.

Polimorfismo Paramétrico

El **polimorfismo paramétrico** es un mecanismo de tipado que permite definir funciones y tipos de datos de forma paramétrica de forma tal que puedan manipular valores de distintos tipos de forma totalmente uniforme.

Ejemplo:

$$head\ (x : xs) = x$$

Polimorfismo Paramétrico

El **polimorfismo paramétrico** es un mecanismo de tipado que permite definir funciones y tipos de datos de forma paramétrica de forma tal que puedan manipular valores de distintos tipos de forma totalmente uniforme.

Ejemplo:

$$head\ (x : xs) = x$$

$$head\ [1, 2, 3, 4] \rightsquigarrow 1$$
$$head\ [False, True, False] \rightsquigarrow False$$
$$head\ [\texttt{'a'}, \texttt{'b'}, \texttt{'c'}, \texttt{'d'}] \rightsquigarrow \texttt{'a'}$$

Polimorfismo Paramétrico

El **polimorfismo paramétrico** es un mecanismo de tipado que permite definir funciones y tipos de datos de forma paramétrica de forma tal que puedan manipular valores de distintos tipos de forma totalmente uniforme.

Ejemplo:

$$head\ (x : xs) = x$$

$$head\ [1, 2, 3, 4] \rightsquigarrow 1$$
$$head\ [False, True, False] \rightsquigarrow False$$
$$head\ [\texttt{'a'}, \texttt{'b'}, \texttt{'c'}, \texttt{'d'}] \rightsquigarrow \texttt{'a'}$$

El tipo de *head* es entonces:

$$head :: [a] \rightarrow a$$

donde a es una **variable de tipo**.

Tipo Más General

- $head :: [a] \rightarrow a$ es el **tipo más general** de *head*.

Tipo Más General

- $head :: [a] \rightarrow a$ es el **tipo más general** de $head$.

- Al reemplazar las variables de tipo por tipos particulares se obtienen **instancias** del tipo:

$$[Int] \rightarrow Int$$
$$[Bool] \rightarrow Bool$$
$$[Char] \rightarrow Char$$

Tipo Más General

- $head :: [a] \rightarrow a$ es el **tipo más general** de $head$.
- Al reemplazar las variables de tipo por tipos particulares se obtienen **instancias** del tipo:

  $[Int] \rightarrow Int$
  $[Bool] \rightarrow Bool$
  $[Char] \rightarrow Char$

- No son instancias:

  $[Int] \rightarrow Char$
  $Bool \rightarrow Bool$
  $[a] \rightarrow Int$

Tipo Más General

- $head :: [a] \rightarrow a$ es el **tipo más general** de *head*.
- Al reemplazar las variables de tipo por tipos particulares se obtienen **instancias** del tipo:

  $[Int] \rightarrow Int$
  $[Bool] \rightarrow Bool$
  $[Char] \rightarrow Char$

- No son instancias:

  $[Int] \rightarrow Char$
  $Bool \rightarrow Bool$
  $[a] \rightarrow Int$

- Haskell es capaz de **inferir** el tipo más general de una función.

Algunas funciones polimórficas sobre listas

Las siguientes son algunas funciones polimórficas sobre listas contenidas en el Prelude:

$$(:) :: a \to [a] \to [a]$$
$$(+\!\!+) :: [a] \to [a] \to [a]$$
$$(!!) :: [a] \to Int \to a$$
$$last :: [a] \to a$$
$$concat :: [[a]] \to [a]$$
$$take :: Int \to [a] \to [a]$$
$$splitAt :: Int \to [a] \to ([a], [a])$$
$$reverse :: [a] \to [a]$$
$$zip :: [a] \to [b] \to [(a, b)]$$
$$unzip :: [(a, b)] \to ([a], [b])$$

Sobrecarga

Sobrecarga

- En Haskell se utiliza el mismo operador (==) para representar la igualdad en más de un tipo.

$$(==) :: Int \to Int \to Bool$$
$$(==) :: Char \to Char \to Bool$$
$$(==) :: Bool \to Bool \to Bool$$

Sobrecarga

- En Haskell se utiliza el mismo operador (==) para representar la igualdad en más de un tipo.

$$(==) :: Int \to Int \to Bool$$
$$(==) :: Char \to Char \to Bool$$
$$(==) :: Bool \to Bool \to Bool$$

- La sobrecarga es otra forma de polimorfismo, dentro del llamado **polimorfismo ad-hoc**.
- Una función sobrecargada puede comportarse de forma diferente (tener una definición diferente) para cada tipo.

Sobrecarga

- En Haskell se utiliza el mismo operador (==) para representar la igualdad en más de un tipo.

$$(==) :: Int \to Int \to Bool$$
$$(==) :: Char \to Char \to Bool$$
$$(==) :: Bool \to Bool \to Bool$$

- La sobrecarga es otra forma de polimorfismo, dentro del llamado **polimorfismo ad-hoc**.
- Una función sobrecargada puede comportarse de forma diferente (tener una definición diferente) para cada tipo.
- En Haskell, la definición de funciones sobrecargadas se hace a través del concepto de **clase**.

  El operador (==) tiene el siguiente tipo:

$$(==) :: Eq\ a \Rightarrow a \to a \to Bool$$

Clases

Una **clase** especifica una colección de tipos a los que se les asocia un conjunto de operaciones (comunmente llamados métodos).

class *Eq a* **where**

$$(==) :: a \to a \to Bool$$
$$(/=) :: a \to a \to Bool$$
$$x \;/= y = not\ (x == y)$$
$$x == y = not\ (x \;/= y)$$

Instancias

Para declarar que un tipo es miembro de una clase hay que definir una **instancia**.

$$\textbf{data}\ T = A \mid B$$

$$\begin{aligned}&\textbf{instance}\ Eq\ T\ \textbf{where}\\ &\quad A == A = True\\ &\quad B == B = True\\ &\quad \_ == \_ = False\end{aligned}$$

Instancias

Para declarar que un tipo es miembro de una clase hay que definir una **instancia**.

```
data T = A | B
instance Eq T where
  A == A = True
  B == B = True
  _ == _ = False
```

Para un número restricto de clases (como *Eq*, *Ord*, *Show*, etc), se puede pedir que Haskell derive la instancia de una clase en forma automática.

```
data T = A | B
  deriving (Eq, Show)
```

La instancia de la clase *Show* permite visualizar los valores del tipo.

Alto Orden

Funciones de Alto Orden

- Una **función de alto orden** es una función que toma una función como parámetro o retorna una función como resultado.
- En PF las funciones son **ciudadanas de primera clase**.

Funciones de Alto Orden

- Una **función de alto orden** es una función que toma una función como parámetro o retorna una función como resultado.
- En PF las funciones son **ciudadanas de primera clase**.
- Ejemplos de funciones de alto orden:

$$map :: (a \to b) \to [a] \to [b]$$
$$filter :: (a \to Bool) \to [a] \to [a]$$
$$zipWith :: (a \to b \to c) \to [a] \to [b] \to [c]$$

Funciones de Alto Orden

- Una **función de alto orden** es una función que toma una función como parámetro o retorna una función como resultado.
- En PF las funciones son **ciudadanas de primera clase**.
- Ejemplos de funciones de alto orden:

$$map :: (a \to b) \to [a] \to [b]$$
$$filter :: (a \to Bool) \to [a] \to [a]$$
$$zipWith :: (a \to b \to c) \to [a] \to [b] \to [c]$$

- Pero también $take :: Int \to [a] \to [a]$
  es de alto orden debido a que su tipo equivale a:

$$take :: Int \to ([a] \to [a])$$

dado que $\to$ asocia a la derecha.

Funciones Currificadas

- Las funciones en Haskell se representan en forma **currificada**: *las funciones tienen sólo un parámetro*.

Funciones Currificadas

- Las funciones en Haskell se representan en forma **currificada**: *las funciones tienen sólo un parámetro*.

- Por ejemplo:

  $$take :: Int \rightarrow ([a] \rightarrow [a])$$

  puede ser aplicado a un entero, retornando la función:

  $$take\ 3 :: [a] \rightarrow [a]$$

Funciones Currificadas

- Las funciones en Haskell se representan en forma **currificada**: *las funciones tienen sólo un parámetro*.

- Por ejemplo:

  $$take :: Int \rightarrow ([a] \rightarrow [a])$$

  puede ser aplicado a un entero, retornando la función:

  $$take\ 3 :: [a] \rightarrow [a]$$

  que luego puede ser aplicada a un lista:

  $$(take\ 3)\ [\texttt{'a'}, \texttt{'b'}, \texttt{'c'}, \texttt{'d'}, \texttt{'e'}]$$

Funciones Currificadas

- Las funciones en Haskell se representan en forma **currificada**: *las funciones tienen sólo un parámetro*.

- Por ejemplo:

  $$take :: Int \rightarrow ([a] \rightarrow [a])$$

  puede ser aplicado a un entero, retornando la función:

  $$take\ 3 :: [a] \rightarrow [a]$$

  que luego puede ser aplicada a un lista:

  $$(take\ 3)\ [\texttt{'a'}, \texttt{'b'}, \texttt{'c'}, \texttt{'d'}, \texttt{'e'}]$$

- Los paréntesis no son necesarios, porque la aplicación asocia a la izquierda:

  $$take\ 3\ [\texttt{'a'}, \texttt{'b'}, \texttt{'c'}, \texttt{'d'}, \texttt{'e'}]$$

Currificadas vs no currificadas

- No currificada

$$add :: Num\ a \Rightarrow (a, a) \rightarrow a$$
$$add\ (x, y) = x + y$$
$$mult :: Num\ a \Rightarrow (a, a, a) \rightarrow a$$
$$mult\ (x, y, z) = x * y * z$$

Currificadas vs no currificadas

- No currificada

$$add :: Num\ a \Rightarrow (a, a) \rightarrow a$$
$$add\ (x, y) = x + y$$
$$mult :: Num\ a \Rightarrow (a, a, a) \rightarrow a$$
$$mult\ (x, y, z) = x * y * z$$

- Currificada

$$addc :: Num\ a \Rightarrow a \rightarrow a \rightarrow a$$
$$addc\ x\ y = x + y$$
$$multc :: Num\ a \Rightarrow a \rightarrow a \rightarrow a \rightarrow a$$
$$multc\ x\ y\ z = x * y * z$$

Currificación

Se puede pasar de representación currificada a no currificada, y viceversa, usando las siguientes funciones:

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

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

Currificación

Se puede pasar de representación currificada a no currificada, y viceversa, usando las siguientes funciones:

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

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

Ejemplo:

$$addc = curry\ add$$

$$add\ \ = uncurry\ addc$$