



PROGRAMACIÓN FUNCIONAL

Tipos de Datos: Esquemas en Árboles



Esquemas de Recursión

- ◆ Generalización a árboles: map y folds
- ◆ Árboles alfa-beta
- ◆ Árboles generales

Esquemas de funciones

◆ ¿Qué ventajas tiene trabajar con esquemas?

Permite

- ◆ definiciones más concisas y modulares
- ◆ reutilizar código
- ◆ demostrar propiedades generales

◆ ¿Qué requiere trabajar con esquemas?

- ◆ Familiaridad con funciones de alto orden
- ◆ Detección de características comunes
(¡ABSTRACCIÓN!)

Esquemas en árboles

- Esquema de map en árboles:

```
data Arbol a = Hoja a | Nodo a (Arbol a) (Arbol a)
```

```
mapArbol :: (a -> b) -> Arbol a -> Arbol b
```

```
mapArbol f (Hoja x) = Hoja (f x)
```

```
mapArbol f (Nodo x t1 t2) =
```

```
    Nodo (f x) (mapArbol f t1) (mapArbol f t2)
```

- ¿Cómo definiría la función que multiplica por 2 cada elemento de un árbol? ¿Y la que los eleva al cuadrado?

Esquemas en árboles

◆ Solución:

`dupArbol :: Arbol Int -> Arbol Int`
`dupArbol = mapArbol (*2)`

`cuadArbol :: Arbol Int -> Arbol Int`
`cuadArbol = mapArbol (^2)`

- ◆ ¿Podría definir, usando `mapArbol`, una función que aplique dos veces una función dada a cada elemento de un árbol? ¿Cómo?

Esquemas en árboles

- ◆ La función foldr expresa el patrón de recursión estructural sobre listas como función de alto orden
- ◆ Todo tipo algebraico recursivo tiene asociado un patrón de recursión estructural
- ◆ ¿Existirá una forma de expresar cada uno de esos patrones como una función de alto orden?
- ◆ ¡Sí, pero los argumentos dependen de los casos de la definición!

Esquemas en árboles

- ◆ Ejemplo:

$\text{foldArbol} :: (a \rightarrow b) \rightarrow (a \rightarrow b \rightarrow b \rightarrow b) \rightarrow \text{Arbol } a \rightarrow b$

$\text{foldArbol } f \ g \ (\text{Hoja } x) = f \ x$

$\text{foldArbol } f \ g \ (\text{Nodo } x \ t1 \ t2) =$
 $\quad g \ x \ (\text{foldArbol } f \ g \ t1) \ (\text{foldArbol } f \ g \ t2)$

- ◆ ¿Cuál es el tipo de los constructores?

$\text{Hoja} :: a \rightarrow \text{Arbol } a$

$\text{Nodo} :: a \rightarrow \text{Arbol } a \rightarrow \text{Arbol } a \rightarrow \text{Arbol } a$

- ◆ ¿Qué similitudes observa con el tipo de `foldArbol`?

Esquemas en árboles

- ◆ Defina una función que sume todos los elementos de un árbol

`sumArbol :: Arbol Int -> Int`

`sumArbol = foldArbol id (\n n1 n2 -> n1 + n + n2)`

- ◆ ¿Podría identificar las llamadas recursivas?
- ◆ ¿Y si expandimos la definición de `foldArbol`?

`sumArbol (Hoja x) = id x`

`sumArbol (Nodo x t1 t2) =`

`sumArbol t1 + x + sumArbol t2`

Esquemas en árboles

- ◆ Defina, usando foldArbol una función que:
 - ◆ cuente el número de elementos de un árbol
 $\text{sizeArbol} = \text{foldArbol } (\lambda x \rightarrow 1) (\lambda x \ s1 \ s2 \rightarrow 1 + s1 + s2)$
 - ◆ cuente el número de hojas de un árbol
 $\text{hojas} = \text{foldArbol } (\text{const } 1) (\lambda x \ h1 \ h2 \rightarrow h1 + h2)$
 - ◆ calcule la altura de un árbol
 $\text{altura} = \text{foldArbol } (\lambda x \rightarrow 0) (\lambda x \ a1 \ a2 \rightarrow 1 + \max a1 \ a2)$
 - ◆ ¿Puede identificar los llamados recursivos?
 - ◆ ¿Por qué el primer argumento es una función?

Árboles alfa-beta

- ◆ Considere la siguiente definición

`data AB a b = Leaf b | Branch a (AB a b) (AB a b)`

- ◆ Defina una función que cuente el número de bifurcaciones de un árbol

`bifs :: AB a b -> Int`

`bifs (Leaf x) = 0`

`bifs (Branch y t1 t2) = 1 + bifs t1 + bifs t2`

- ◆ ¿Cómo sería el esquema de recursión asociado a un árbol AB?

Árboles alfa-beta

- ◆ ¡Utilizamos el esquema de recursión!

$\text{foldAB} :: ??$

$\text{foldAB } f \ g \ (\text{Leaf } x) = f \ x$

$\text{foldAB } f \ g \ (\text{Branch } y \ t1 \ t2) =$
 $g \ y \ (\text{foldAB } f \ g \ t1) \ (\text{foldAB } f \ g \ t2)$

- ◆ ¿Cómo representaría la función bifs?

$\text{bifs}' = \text{foldAB } (\text{const } 0) \ (\backslash x \ n1 \ n2 \rightarrow 1 + n1 + n2)$

- ◆ ¿Puede probar que $\text{bifs}' = \text{bifs}$?

Árboles alfa-beta

- ◆ Ejemplo de uso

```
type AExp = AB BOp Int  
data BOp = Suma | Producto
```

- ◆ ¿Cómo definimos la semántica de AExp usando foldAB?

```
evalAE :: AExp -> Int  
evalAE = foldAB id binOp  
binOp :: BOp -> Int -> Int -> Int  
binOp Suma = (+)  
binOp Producto = (*)
```



Árboles alfa-beta

- ◆ Ejemplo de uso

```
type Decision s a = AB (s->Bool) a
```

- ◆ Definamos una función que dada una situación, decida qué acción tomar basada en el árbol

```
decide :: situation -> Decision situation action -> action
```

```
decide s = foldAB id (\f a1 a2 -> if (f s) then a1 else a2)
```

```
ej = Branch f1 (Leaf "Huya")
```

```
      (Branch f2 (Leaf "Trabaje") (Leaf "Quédese manso"))
```

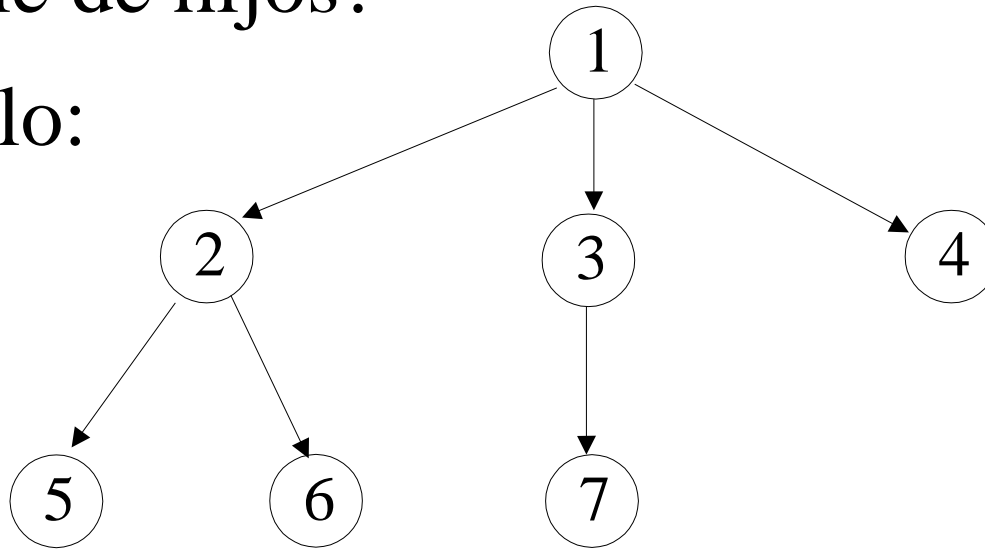
```
      where f1 s = (s==Fuego) || (s==AtaqueExtraterrestre)
```

```
            f2 s = (s==VieneElJefe)
```

Árboles Generales

- ◆ ¿Cómo representar un árbol con un número variable de hijos?

- ◆ Ejemplo:



- ◆ Idea: ¡usar una lista de hijos!

Árboles Generales

- ◆ Ello nos lleva a la siguiente definición:
data AG a = GNode a [AG a]
- ◆ Pero, ¿tiene caso base? ¿cuál?
 - ◆ Un árbol sin hijos...
- ◆ ¡Se basa en el esquema de recursión de listas!
 - ◆ O sea, el caso base es (GNode x []); por ejemplo:
GNode 1 [GNode 2 [GNode 5 [], GNode 6 []]
 , GNode 3 [GNode 7 []]
 , GNode 4 []
]

Árboles Generales

- ◆ Definir una función que sume los elementos

$\text{sumAG} :: \text{AG Int} \rightarrow \text{Int}$

- ◆ ¿Cómo la definimos?

- ◆ ¡Usando funciones sobre listas!

$\text{sumAG} (\text{GNode } x \text{ ts}) = x + \text{sum} (\text{map } \text{sumAG } \text{ts})$

- ◆ Y esto, ¿es estructural?

- ◆ Sí, pues se basa en la estructura de las listas

- ◆ Se ve la utilidad de funciones de alto orden...

Árboles Generales

◆ ¿Cómo sería el esquema de recursión?

◆ Hay dos posibilidades

- ◆ o bien se esperan las partes del esquema de listas

```
foldAG1 :: (a->c->b) -> (b->c->c) -> c -> AG a -> b
foldAG1 g f z (GNode a ts) =
    g a (foldr f z (map (foldAG1 g f z) ts)))
```

- ◆ o bien se espera una función que procese la lista

```
foldAG2 :: (a->c->b) -> ([b]->c) -> AG a -> b
foldAG2 g h (GNode a ts) =
    g a (h (map (foldAG2 g h) ts))
```

Árboles Generales

- ◆ ¿Cómo usar los foldAG?

$\text{sumAG}'1 = \text{foldAG1 } (+) (+) 0$

$\text{sumAG}'2 = \text{foldAG2 } (+) \text{ sum}$

- ◆ ¿Cuál es mejor? Depende del uso y el gusto

- ◆ Una termina para todo árbol, pero no define todas las funciones posibles.

- ◆ Otras funciones sobre árboles generales:

$\text{depthAG} = \text{foldAG2 } (\backslash x \ d \rightarrow 1+d) \text{ maximum}$

$\text{reverseAG} = \text{foldAG2 GNode reverse}$