Estructuras de datos

Clases teóricas

por Pablo E. “Fidel” Martínez López

4. Repaso 1

Repaso

Tipos algebraicos

- Los tipos algebraicos son tipos nuevos
  - Se definen a través de ***constructores***
    - Los constructores pueden llevar argumentos
    - Cada constructor expresa a un grupo de elementos
  - Se acceden mediante ***pattern matching***
    - Los constructores se usan para preguntar

Tipos algebraicos

- Los tipos algebraicos se clasifican en
  - enumerativos
    - varios constructores sin argumentos (e.g. **`Direccion`**)
  - registros (o productos)
    - un único constructor con varios argumentos (e.g. **`Persona`**)
  - sumas (o variantes)
    - varios constructores con argumentos (e.g. **`Helado`**)
  - recursivos
    - suma que usa el mismo tipo como argumento (e.g. **`Listas`**)

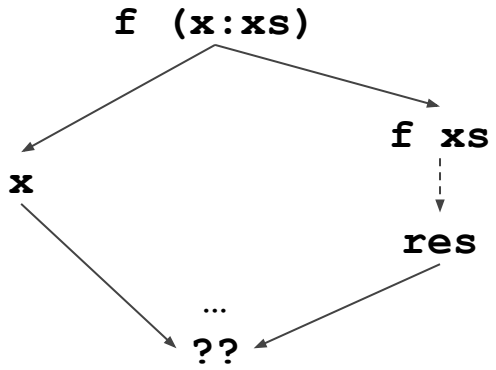
Listas

- Las listas son un tipo algebraico predefinido
  - Tiene sintaxis especial
  - Los constructores son
    - `[] :: [a]` `-- Nil`
    - `(:) :: a -> [a] -> [a]` `-- Cons`
  - Se usa notación especial para simplificar
    - `[10,20,30]` es en realidad `(10:20:30:[])`
  - Para definir funciones hace falta recursión estructural

Recursión estructural sobre listas

- Las definiciones recursivas estructurales sobre listas
  - Tienen un caso por cada constructor
  - En el recursivo, usan *la misma función* en la parte recursiva
  - Solo hace falta pensar cómo agregar lo que falta

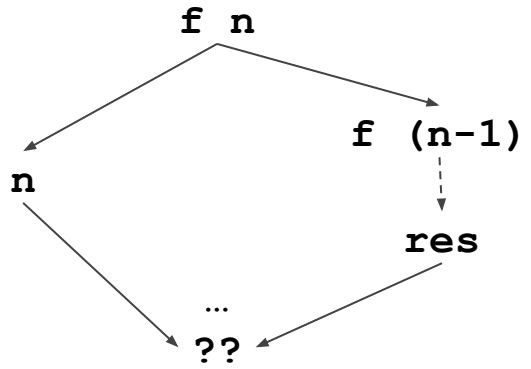
```
f :: [a] -> b
f []     = …
f (x:xs) = … x … f xs …
```



Recursión estructural sobre números

- Las definiciones recursivas estructurales sobre números
  - Tienen un caso base (0) y un caso recursivo (>0)
  - En el recursivo, usan *la misma función* en el anterior
  - No sirve para negativos

```
f :: Int -> b
f 0 = …
f n = … n … f (n-1) …
```

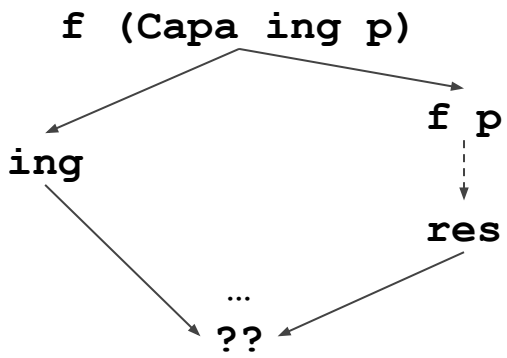


Otros tipos recursivos lineales

- Tipos algebraicos recursivos
  - Se utiliza como argumento el tipo que se define
    - Solamente en *algunos* de los constructores
    - La recursión sigue la estructura

```
data Pizza = Prepizza | Capa Ingrediente Pizza

f :: Pizza -> a
f Prepizza      = …
f (Capa ing p) = … ing … f p …
```



Árboles

- Tipos algebraicos recursivos: árboles
  - Se utiliza como argumento el tipo que se define
    - Solamente en *algunos* de los constructores
    - La recursión sigue la estructura

```
data Dungeon = Armario
             | Habitacion Objeto Dungeon Dungeon

f :: Dungeon -> a
f Armario                    = …
f (Habitacion obj d1 d2) = … obj … f d1 … f d2 …
```

Árboles

- Tipos algebraicos recursivos: árboles
  - Se utiliza como argumento el tipo que se define
    - Solamente en *algunos* de los constructores
    - La recursión sigue la estructura

```
data Tree a = EmptyT
            | NodeT a (Tree a) (Tree a)

f :: Tree a -> b
f EmptyT           = …
f (NodeT x t1 t2) = … x … f t1 … f t2 …
```

Más ejemplos en listas

Ejemplos

- Más ejemplos para recursión estructural

```
tomarHasta :: Int -> [a] -> [a]          -- PRECOND: i>=0
  -- tomarHasta 3 [10,20,30,40,50] = [10,20,30]
  -- tomarHasta 3 [10,20] = [10,20]

tomarDesde :: Int -> [a] -> [a]          -- PRECOND: i>=0
  -- tomarDesde 3 [10,20,30,40,50] = [40,50]
  -- tomarDesde 3 [10,20] = []

tomarEntre :: Int -> Int -> [a] -> [a]   -- PRECOND: j>=i
  -- tomarEntre 1 3 [10,20,30,40,50] = [20,30,40]
  -- tomarEntre 1 3 [10,20] = [20]
```

Ejemplos

- Más ejemplos para recursión estructural

```
apariciones :: Eq a => [ a ] -> [ (a,Int) ]
  -- apariciones "acbaac" = [ ('a',3), ('b',1), ('c',2) ]

indexar :: [a] -> [(Int,a)]
  -- indexar [17,42,99] = [(0,17), (1,42), (2,99)]

indexarDesde :: Int -> [a] -> [(Int,a)]
  -- indexarDesde 3 [17,42,99] = [(3,17), (4,42), (5,99)]

indexarDesdeHasta :: Int -> Int -> [a] -> [(Int,a)]
  -- PRECOND: j>=i
  -- indexarDesde 2 4 [10,20,30,40,50]
  --                        = [(2,10), (3,20), (4,30)]
```

Ejemplos

- Más ejemplos para recursión estructural

```
ordenar :: Ord a => [a] -> [a]
  -- ordenar [99,17,666,42,21] = [17,21,42,99,666]

pertenece :: Eq a => a -> [a] -> Bool
  -- pertenece 17 [42,99,17,21] = True
  -- pertenece 17 [42,99,666,21] = False

prefijosPropios :: [a] -> [[a]]
  -- prefijosPropios [1,2,3] = [[1],[1,2],[1,2,3]]
  -- prefijosPropios [] = []
```

Ejemplos

```
data Persona =
     P String Int  String
    -- Nombre Edad DNI
```

- Más ejemplos para recursión estructural

```
hayMayorDeEdad :: [Persona] -> Bool
  -- Indica si hay alguna persona mayor de edad

hayMayorDeEdadEn :: Int -> [Persona] -> Bool
  -- Indica si la persona en posición i es mayor de edad

cantidadHastaMayorDeEdad :: [Persona] -> Int
  -- Indica cuántas personas hay hasta un mayor de edad
  -- PRECOND: hay al menos un mayor de edad

sumaDeEdadesEntre :: Int -> Int -> [Persona] -> Int
  -- Suma las edades de las personas entre las posiciones dadas
  -- PRECOND: i<=j
```

Más ejemplos en árboles

Ejemplos

```
data Tree a = EmptyT
            | NodeT a (Tree a) (Tree a)
```

- Ejemplos para recursión estructural en árboles

```
levelN :: Int -> Tree a -> [a]
  -- Lista los nodos del nivel a la profundidad dada

listPerLevel :: Tree a -> [[a]]
  -- Lista todos los niveles, como lista de elementos por nivel

todosLosCaminos :: Tree a -> [[a]]
  -- Lista todos los caminos desde la raíz a alguna hoja
  --  (incluyendo caminos más cortos)
```

Ejemplos

```
data Tree a = EmptyT
            | NodeT a (Tree a) (Tree a)

data Opcion   = Izq | Der
type Posicion = [ Opcion ]
```

- Ejemplos para recursión estructural en árboles

```
elementoEn :: Posicion -> Tree a -> a
  -- Describe el elemento en la posición dada
  -- PRECOND: la posición es válida dentro del árbol

posicionesDe :: Eq a => a -> Tree a -> [ Posicion ]
  -- Describe las posiciones donde está el elemento dado
```

```
data Tree a = EmptyT
            | NodeT a (Tree a) (Tree a)
```

Ejemplos

- Ejemplos para recursión estructural en otros tipos

```
data Componente = LanzaTorpedos | Motor Int | Almacen [Barril]
data Sector     = S SectorId [Componente] [Tripulante]
data Nave       = N (Tree Sector)

sectores :: Nave -> [ SectorId ]
  -- Lista los ids de todos los sectores

poderDePropulsion :: Nave -> Int
  -- Capacidad de propulsión de todos los motores de la nave
```

Ejemplos

- Ejemplos para recursión estructural en otros tipos

```
data Lobo   = Cazador Nombre [Presa] Lobo Lobo Lobo
            | Explorador Nombre [Territorio] Lobo Lobo
            | Cria Nombre
data Manada = M Lobo

buenaCaza :: Manada -> Bool
  -- Indica si la cantidad de alimento es mayor que las crías

elAlfa :: Manada -> (Nombre, Int)
  -- El nombre del lobo que más cazó, con la cantidad cazada

exploradoresPorTerritorio :: Manada -> [(Territorio, [Nombre])]
  -- Por cada territorio, qué exploradores lo tienen
```

Resumen

Resumen

- Tipos algebraicos recursivos
  - Sumas con algún constructor con el mismo tipo como argumento
  - Se utilizan *las mismas técnicas* que con otros tipos algebraicos (subtareas, estructura, etc.)
  - Recursión estructural
    - Una ecuación por cada constructor
    - La misma función en las partes recursivas
    - Solo hace falta pensar cómo agregar lo que falta