PROGRAMACIÓN FUNCIONAL

Tipos de Datos: Tipos Algebraicos

Tipos de Datos

- Noción de tipo de datos
- Formas de definición de tipos de datos
- Tipos de datos algebraicos
- Pattern matching

Tipos de Datos

- Un tipo de datos se compone de:
  - un conjunto de *elementos* con ciertas características comunes
  - un conjunto de *operaciones* para manipular dichos elementos
- ¿Cómo definimos tipos de datos?
- ¿Cómo utilizamos tipos de datos?

Definición de Tipos 1

- Para definir un tipo de datos podemos:
  - establecer qué *forma* tendrá cada *elemento*, y
  - dar un *mecanismo único* para *inspeccionar* cada elemento
  - entonces: TIPO ALGEBRAICO

ó

  - determinar cuáles serán las *operaciones* que manipularán los elementos, SIN decir cuál será la forma exacta de éstos o aquéllas
  - entonces: TIPO ABSTRACTO

Definición de Tipos 2

- Tipos Algebraicos
  - dar la forma de los elementos
  - dar un mecanismo único de acceso
- Tipos Abstractos
  - dar sólo las operaciones
  - NO dar la forma de elementos ni operaciones
- Tipos predefinidos
  - Int, Float, a->b
    - tipos abstractos con sintaxis especial
  - Char, Bool, (a,b), [a]
    - tipos algebraicos con sintaxis especial

Tipos Algebraicos

- ¿Cómo damos en Haskell la forma de un elemento de un tipo algebraico?
  - Mediante **constantes** llamadas *constructores*
    - nombres con mayúsculas
    - no tienen asociada una regla de reducción
    - pueden tener argumentos
- Ejemplos:

```
False :: Bool        True :: Bool
```

Tipos Algebraicos

- La cláusula data
  - introduce un nuevo tipo algebraico
  - introduce los nombres de los constructores
  - define los tipos de los argumentos de los constructores
- Ejemplos:

```
data Sensacion = Frio | Calor

data Shape = Circle Float | Rectangle Float Float
```

Tipos Algebraicos

- data Shape = Circle Float | Rectangle Float Float

Ejemplos de elementos:

c1 = Circle 1.0

c2 = Circle (4.0-3.0)

circulo x = Circle (x+1.0)

r1 = Rectangle 2.5 3.0

cuadrado x = Rectangle x x

Pattern Matching

- ¿Cuál es el mecanismo único de acceso?
  - *Pattern matching* (correspondencia de patrones)
- Pattern: expresión especial
  - sólo con constructores y variables sin repetir
  - argumento en el lado izquierdo de una ecuación
- Matching: operación asociada a un pattern
  - inspecciona el valor de una expresión
  - puede fallar o tener éxito
  - si tiene éxito, liga las variables del pattern

Pattern Matching

- Ejemplo:

  ```
  area :: Shape -> Float
  area (Circle radio) = pi * radio^2
  area (Rectangle base altura) = base * altura
  ```

  - Al evaluar (area (circulo 2.0))
    - primero se reduce (circulo 2.0) a (Circle 3.0)
    - luego se verifica cada ecuación, para hacer el matching
    - si lo hace, la variable toma el valor correspondiente
  - radio se liga a 3.0, y la expresión retorna 28.2743
- ¿Cuánto valdrá (area (cuadrado 2.5))?

Tuplas

- Son tipos algebraicos con sintaxis especial

```
fst :: (a,b) -> a
fst (x,y) = x

snd :: (a,b) -> b
snd (x,y) = y

distance :: (Float, Float) -> Float
distance (x,y) = sqrt (x^2 + y^2)
```

- ¿Cómo definir distance sin usar pattern matching?

```
distance p = sqrt ((fst p)^2 + (snd p)^2)
```

Tipos Algebraicos

- Pueden tener argumentos de tipo
- Ejemplo:

```
data Maybe a = Nothing | Just a
```

- ¿Qué elementos tiene (Maybe Int)?
- En general:
  - tiene los mismos elementos que el tipo a (pero con Just adelante) más uno adicional (Nothing)

Tipos Algebraicos

- ¿Para qué se usa el tipo Maybe?
- Ejemplo:

```
buscar :: clave -> [(clave,valor)] -> valor
buscar k [] = error "La clave no se encontró"
buscar k ((k',v):kvs) = if k==k'
                        then v
                        else buscar k kvs
```

- ¿La función buscar es total o parcial?

Tipos Algebraicos

- ¿Para qué se usa el tipo Maybe?
- Ejemplo:

```
lookup :: clave -> [(clave,valor)] -> Maybe valor
lookup k [] = Nothing
lookup k ((k',v):kvs) = if k==k'
                        then Just v
                        else lookup k kvs
```

- ¿La función lookup es total o parcial?

Tipos Algebraicos

- El tipo Maybe
  - permite expresar la posibilidad de que el resultado sea erróneo, sin necesidad de usar 'casos especiales'
  - evita el uso de ⊥ hasta que el programador decida, permitiendo controlar los errores

```
sueldo :: Nombre -> [Empleado] -> Int
sueldo nombre empleados =
    case (lookup nombre empleados) of
      Nothing -> error "No pertenece a la empresa!"
      Just s    -> s
```

Tipos Algebraicos

- Otro ejemplo:

```
data Either a b = Left a | Right b
```

- ¿Qué elementos tiene (Either Int Bool)?
- En general:
  - representa la unión disjunta de dos conjuntos (los elementos de uno se identifican con Left y los del otro con Right)

Tipos Algebraicos

- ¿Para qué sirve Either?
- Para mantener el tipado fuerte y poder devolver elementos de distintos tipos
  - Ejemplo: [Left 1, Right True] :: [Either Int Bool]
- Para representar el origen de un valor
  - Ejemplo: lectora de temperaturas

```
mostrar :: Either Int Int -> String
mostrar (Left t) = show t ++ " Celsius"
mostrar (Right t) = show t ++ " Fahrenheit"
```

Tipos Algebraicos

- ¿Por qué se llaman tipos algebraicos?
- Por sus características:
  - toda combinación válida de constructores y valores es elemento de un tipo algebraico (y sólo ellas lo son)
  - dos elementos de un tipo algebraico son iguales si y sólo si están construídos utilizando los mismos constructores aplicados a los mismos valores

Tipos Algebraicos

- Expresividad: números complejos
  - Toda combinación de dos flotantes es un complejo
  - Dos complejos son iguales si tienen las mismas partes real e imaginaria

```
data Complex = C Float Float

realPart, imagePart :: Complex -> Float
realPart (C r i) = r
imagePart (C r i) = i

mkPolar :: Float -> Float -> Complex
mkPolar r theta = C (r * cos theta) (r * sin theta)
```

Tipos Algebraicos

- Expresividad: números racionales
  - No todo par de enteros es un número racional (R 1 0)
  - Hay racionales iguales con distinto numerador y denominador (R 4 2 = R 2 1)

```
data Racional = R Int Int
numerador, denominador :: Racional -> Int
numerador (R n d) = n
denominador (R n d) = d
```

- ¡No se puede representar a los racionales como tipo algebraico!

Tipos Algebraicos

- Podemos clasificarlos en:
  - Enumerativos (Sensacion, Bool)
    - Sólo constructores sin argumentos
  - Productos (Complex, Tuplas)
    - Un único constructor con varios argumentos
  - Sumas (Shape, Maybe, Either)
    - Varios constructores con argumentos
  - Recursivos (Listas)
    - Utilizan el tipo definido como argumento

Resumen

- Formas de definición de tipos de datos
- Tipos algebraicos
- Pattern matching
- Expresividad de los tipos algebraicos
- Ejemplos
  - Maybe, Either
  - Listas