PROGRAMACIÓN FUNCIONAL

Tipos de Datos: Tipos Abstractos

Tipos de Datos

- Tipos abstractos de datos
- Módulos
- Ejemplos

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 Abstractos

- Al definir un tipo abstracto sólo se establecen las operaciones permitidas:
  - sus nombres
  - sus tipos
  - qué se espera que hagan
- ¿Cómo se utiliza un tipo abstracto?
- A través de las operaciones dadas (y sólo eso)

Tipos Abstractos

- Ejemplo: diccionario

  buscar :: Diccionario -> Palabra -> Maybe Definicion

  agregar :: (Palabra, Definicion) ->
  Diccionario -> Diccionario

  eliminar :: Palabra -> Diccionario -> Diccionario

  castellano :: Diccionario

  - (en general un diccionario se almacena en un medio externo, pero I/O excede a este curso)

Tipos Abstractos

- Existen tres roles en la definición y uso de tipos abstractos de datos:
  - el diseñador
    - decide QUÉ operaciones proveer y sus características
  - el implementador
    - establece CÓMO realizar las operaciones mediante código que las implemente
  - el programador-usuario
    - UTILIZA las operaciones SIN SABER nada de la implementación
- En este curso sólo veremos el tercer rol

Tipos Abstractos

- ¿Cómo saber qué hace cada operación?
- Mediante una especificación
  - informal: descripción en castellano
  - formal: algún lenguaje matemático
    - Ejs: lógica, álgebra, etc.
- Ejemplo de especificación formal:

$\forall p.\forall def.\forall dict.$
agregar (p,def) (agregar (p,def) dict)
= agregar (p,def) dict

Ejemplo: conjuntos

empty :: IntSet

- El conjunto vacío

isEmpty :: IntSet -> Bool

- Determina si el conjunto dado es vacío

belongs :: Int -> IntSet -> Bool

- Determina si un elemento pertenece o no al conjunto

insert :: Int -> IntSet -> IntSet

- Agrega un elemento al conjunto, si no estaba

choose :: IntSet -> (Int, IntSet)

- Elige el menor número y lo quita del conjunto

Ejemplo: conjuntos

- ¿Cómo escribimos la unión?

```
union :: IntSet -> IntSet -> IntSet
union p q | isEmpty p = q
union p q = insert x (union p' q)
                  where (x,p') = choose p
```

- Observar que
  - sólo hace falta conocer las operaciones
  - no hace falta conocer la representación
  - se puede usar recursión
    (¿cuál sería el esquema inductivo asociado?)

Tipos Abstractos

- ¿Cómo expresamos tipos abstractos en Haskell?
  - Mediante MÓDULOS
- Un módulo:
  - permite agrupar definiciones relacionadas, brindando modularización
  - es la unidad de compilación, permitiendo compilación separada (¡no en Hugs!)
  - limita el alcance (scope) de las variables
  - permite exportación e importación explícita de nombres, proveyendo ocultamiento de información

Módulos

- Ejemplo: de agrupación de funciones (modularización)

¡Así, todo el tipo es visible!

```
module Complejos
    (Complex(..), realPart, imagePart, mkPolar)
where

  data Complex = C Float Float

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

Módulos

- Ejemplo: de ocultamiento de información (TAD)

¡Así, el tipo es abstracto!

```
module Racionales
    (Rational, mkR, numerador, denominador)
where
  data Rational = R Int Int
  mkR n d = reduce (n*signo d) (abs d)
  reduce x 0 = error "Racional con denom. 0"
  reduce x y = R (x `quot` d) (y `quot` d)
                  where d = gcd x y
...
```

¡Esta función está oculta para los usuarios!

Módulos

- ¿Cómo se utiliza un módulo?
  - Mediante la cláusula import

```
module Main where
import Complejos
import Racionales (Rational, mkR)
miPar :: (Complex, Rational)
miPar = (C 1 0, mkR 4 2)
-- ¿(numerador (snd miPar)) está permitido?
-- ¿Y (reduce (R 4 2))? ¿Por qué?
```

```
module ConjuntoInt
    (IntSet, empty, isEmpty,
     belongs, insert, choose)
where
 import qualified List (insert)
 data IntSet = Set [ Int ]
 empty = Set [ ]
 isEmpty (Set xs) = null xs
 belongs x (Set xs) = x `elem` xs
 insert x (Set xs) = if x `elem` xs
                     then Set xs
                     else Set (List.insert x xs)
 choose (Set (x:xs)) = (x, Set xs)
```

Resumen

- Tipos abstractos
- Módulos