Nuevos tipos y Pattern Matching

Taller de Álgebra I

Primer cuatrimestre de 2016

Algoritmo de Euclides

El Algoritmo de Euclides calcula el máximo común divisor entre dos números $a, b \in \mathbb{Z}^+$. Se basa en que si $a, b \in \mathbb{Z}$ y $q \in \mathbb{Z}$ es un número cualquiera, entonces

$$(a \ : \ b) = (a + qb \ : \ b).$$

Si q y r son el cociente y el resto de la división de a por b, tenemos $a = qb + r$, entonces $a - qb = r$. Por lo tanto,

$$(a \ : \ b) = (a - qb \ : \ b) = (r \ : \ b) = (b \ : \ r)$$

Por ejemplo, para calcular $(30 \ : \ 48)$, se hace:

1. $(30 \ : \ 48)$ -- Dividimos 30 por 48, $q = 0$, $r = 30$
2. $= (48 \ : \ 30)$ -- Dividimos 48 por 30, $q = 1$, $r = 18$
3. $= (30 \ : \ 18)$ -- $q = 1$, $r = 12$
4. $= (18 \ : \ 12)$ -- $q = 1$, $r = 6$
5. $= (12 \ : \ 6)$ -- $q = 2$, $r = 0$
6. $= (6 \ : \ 0)$
7. $= 6$

Algoritmo de Euclides

Para hacer

Programar en Haskell

```
mcd :: Integer -> Integer -> Integer
```

que dé el mcd.

- El peor caso del algoritmo se obtiene cuando a y b son dos números consecutivos de la sucesión de Fibonacci (Lamé, 1844).
- El número de divisiones que realiza el algoritmo nunca supera 5 veces el número de dígitos de a y b, con lo cual la cantidad de llamadas recursivas está acotada por $5 \log_{10}(a)$.

Pattern Matching

El **Pattern Matching** consiste en especificar patrones para los datos. Estos datos, cuando pasan el patrón, son deconstruidos en sus componentes para poder utilizarlas por separado.

En Números

```
fact :: Integer -> Integer -> Integer
fact 0 = 1
fact n = n * fact (n - 1)

enIngles :: Integer -> [Char]
enIngles 1 = "One!"
enIngles 2 = "Two!"
enIngles 3 = "Three!"
enIngles 4 = "Four!"
enIngles 5 = "Five!"
enIngles x = "Not sure :s"
```

Pattern Matching: Tuplas

Pattern Matching

El **Pattern Matching** consiste en especificar patrones para los datos. Estos datos, cuando pasan el patrón, son deconstruidos en sus componentes para poder utilizarlas por separado.

En Tuplas

```
sumarVectores :: (Int, Int) -> (Int, Int) -> (Int, Int)
sumarVectores a b = (fst a + fst b, snd a + snd b)
```

```
sumarVectores2 :: (Int, Int) -> (Int, Int) -> (Int, Int)
sumarVectores2 (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)

iguales :: (Int, Int) -> Bool
iguales (x,y) = x ==  y
```

¿Funciona definir la siguiente ecuación `iguales (x, x) = True`?

Pattern Matching: Listas

Pattern Matching

El **Pattern Matching** consiste en especificar patrones para los datos. Estos datos, cuando pasan el patrón, son deconstruidos en sus componentes para poder utilizarlas por separado.

En Listas

```
head :: [a] -> a
head (x:xs) = x

tail :: [a] -> [a]
tail (_:xs) = xs

take  0 _        =  []
take  _ []       =  []
take  n (x:xs)   =  x : take (n-1) xs
```

Pattern Matching: Ejercicios

Completar las siguientes ecuaciones

```
first :: (a, b, c) -> a
first (x, y, z) =

longitud :: [a] -> Integer
longitud [] =
longitud (x:[]) =
longitud (x:y:[]) =
longitud (x:y:z:[]) =
longitud (_:_:_:xs) =

iniciales :: [Char] -> [Char] -> [Char]
iniciales nombre apellido =
    where (n:_) = nombre
          (a:_) = apellido
```

Racionales

Definamos un nuevo tipo

```
type Racional = (Integer, Integer)
```

¿Para qué sirve type?

```
suma :: Racional -> Racional -> Racional
suma (a, b) (c, d) = (a*d + b*c, b*d)

type Punto = (Integer, Integer)
dist :: Punto -> Punto -> Float
```

Implementar las siguientes funciones

- `producto :: Racional -> Racional -> Racional`
- `igual :: Racional -> Racional -> Bool`
- `mayor :: Racional -> Racional -> Bool`

Nuevos Tipos: Enumerados

¿Cómo están definidos los Booleanos?

```
data Bool = False | True
```

- El término **data** significa que estamos definiendo un nuevo tipo de datos. La parte antes del igual denota el tipo
- La parte luego del igual son los constructores. Los constructores especifican los diferentes valores que puede poseer el tipo (La barra vertical denota las distintas opciones de construcción de valores del tipo).

Nuevo tipo: Tipo Dia

```
data Dia = Lunes | Martes | ... | Sabado | Domingo
```

Definir las siguientes funciones

- `esFinde :: Dia -> Bool` que determina si el día es parte del fin de semana.
- `diaHabil :: Dia -> Bool` que determina si el día es un día hábil.
- `soloAlgebra :: [Dia] -> [Dia]` que deja en la lista original solo dias en los que haya cursada de álgebra.

Ejercicios

- implementar la función `tuplas` que dada una `[a]` y una `[b]` calcule la lista resultante de unir los `a` con los `b` para generar una `[(a,b)]`. Si las listas son de distinta longitud se deben usar tantos elementos como se pueda. Nota: en Haskell dicha función es `zip`.
  Ej: `tuplas [1,2,3,4] ['a','b','c']` $\rightsquigarrow$ `[(1,'a'),(2,'b'),(3,'c')]`
- implementar la función `potencia :: Racional -> Integer -> Racional`.

Para los ejercicios que siguen, se define `type Conjunto = [Integer]` y se asume que todos los "conjuntos" son listas sin elementos repetidos. Por otra parte, al construir conjuntos, deben asegurarse de mantener esta propiedad.

- implementar `union :: Conjunto -> Conjunto -> Conjunto`
- implementar `interseccion :: Conjunto -> Conjunto -> Conjunto`
- implementar `incluido :: Conjunto -> Conjunto -> Bool`
- implementar `igual :: Conjunto -> Conjunto -> Bool`
- implementar la función `separar :: Integer -> Conjunto -> (Conjunto,Conjunto)` para que la expresión `separar x c` retorne el par `(c1,c2)` tal que $c1 \cup c2 = c$ y $\forall e \in c1 \cdot e \leq x$ y $\forall e \in c2 \cdot e > x$