Programación Funcional

Curso 2019-20

LISTAS INTENSIONALES

Listas intensionales (*comprehension lists*)

Matemáticas: conjuntos por comprensión

- $\{x^2 \mid x \in \{1, \ldots, 5\}\}$ $\{1, 4, 9, 16, 25\}$
- $\{x + 1 \mid x \in \{1, \ldots, 10\}, x \text{ es primo}\}$ $\{3, 4, 6, 8\}$

Haskell: map, filter, concat

- `map (\x -> x^2) [1..5]`
- `map (\x -> x+1) (filter primo [1..10])`

O también: *listas intensionales*

- `[x^2 | x <- [1..5]]`
- `[x+1 | x <- [1..10] , primo x]`
- `primo x = [y | y <-[2..x-1], mod x y == 0] == []`

Morfología de las listas intensionales

- `[ x+1 | x <- [1..10] , primo x]` expresión principal
- `[x+1 | x <- [1..10] , primo x]` generador
- `[x+1 | x <- [1..10] , primo x ]` filtro

En general, una lista intensional es: `[e | c`$_1$`,...,c`$_n$`]`

- e es una expresión
- c_1 es un generador
- Cada c_i con $i > 1$ es un generador o un filtro
- Un generador es de la forma `p <- l`
  - `l` es una expresión de tipo $[\tau]$
  - `p` es un patrón de tipo τ
- Un filtro es una expresión booleana

Generadores

- Puede haber varios generadores seguidos

  `[(x,y) | x <- [1,2] , y <- ['A','B','C']]`

  `[(1,'A'),(1,'B'),(1,'C'),(2,'A'),(2,'B'),(2,'C')]`

- El orden influye

  `[(x,y) | y <- ['A','B','C'] , x <- [1,2]]`

  `[(1,'A'),(2,'A'),(1,'B'),(2,'B'),(1,'C'),(2,'C')]`

  Generadores múltiples actúan al modo de bucles anidados

- Un generador `p <-l` con `p` no variable tiene un efecto filtro por el ajuste de patrones

  `[x+y|((x,y),0) <-[((2,1),0),((1,3),1),((2,4),0)]]`

  `= [3,6]`

Filtros

- Cada filtro actúa sobre el resultado de los generadores y filtros anteriores
- Si un filtro no produce ningún resultado con éxito, la lista queda vacía

Vinculación de variables y listas intensionales

- Variables vinculadas fuera de la lista pueden usarse en cualquier sitio de la lista. La expresión principal puede usar además variables vinculadas dentro de la lista
  `let u=3 in [u+x | x <- [1..u]]`
- Cada generador `p <- t` vincula las variables de `p`, que pueden usarse en los generadores o filtros posteriores, así como en la expresión principal
  `f n = [(x,y) | x <- [1..n] , y <- [1..x]]`
- El ámbito de una variable vinculada en una lista intensional termina con la lista

Un ejemplo elegante: *quicksort*

```
qsort           ::  Ord a => [a] -> [a]
qsort  []       =   []
qsort  (x:xs)   =   qsort [u | u <- xs, u < x]
                    ++ [x] ++
                    qsort [u | u <- xs, u > x]
```

Ternas pitagóricas: (x, y, z) tales que $x^2 + y^2 = z^2$

```
-- ternasP n = ternas pitagóricas con números <= n
ternasP n = [(x,y,z)| x <- [1..n],
                      y <- [1..n],
                      z <- [1..n],
                      z^2 == x^2+y^2]
```

```
⟦ternasP 10⟧ = [(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
```

```
ternasP' n = [(x,y,z)| z <- [1..n],
                       x <- [1..z-1],
                       y <- [1..x-1],
                       z^2 == x^2+y^2]
```

```
⟦ternasP' 10⟧ = [(4,3,5),(8,6,10)]
```

```
-- ternasP'' = ternas pitagóricas, sin límites
ternasP'' = [(x,y,z)| z <- [1..],
                      x <- [1..z-1],
                      y <- [1..x-1],
                      z^2 == x^2+y^2]
```

Map, *filter*, *concat* y listas intensionales

- `map f xs = [f x | x <- xs]`
- `filter p xs = [x | x <- xs , p x]`
- `concat xxs = [x | xs <- xxs , x <- xs]`
- Recíprocamente toda lista intensional puede expresarse en términos de `map`, `filter` y `concat`.

☞ Las listas intensionales son azúcar sintáctico

```
[x^2 | x <- [1..5]]
```

```
map (\x -> x^2) [1..5]
```

O bien

```
map f [1..5] where f x = x^2
```

En general: `[e | x <- l]` $\equiv$ `map f l where f x = e`

```
[x+1 | x <- [1..10] , primo x]
```

```
map f (filter primo [1..10]) where f x = x+1
```

O bien, más sistemático

```
[x+1 | x <- filter p [1..10]] where p x = primo x
≡ map f (filter p [1..10]) where  p x = primo x
                                  f x = x+1
```

En general:

`[e | x <- l, b]` $\equiv$ `[e | x <- filter p l] where p x = b`

```
[(x,y)| x <- [1,2] , y <- [3,4]]
```

```
[e| x<-l, y<-l',...]
```
$\equiv$
```
concat [[e| y<-l',...]  | x<-l]
```

En el ejemplo:

```
[(x,y)| x<-[1,2] , y <-[3,4]]
```
$\equiv$
```
concat [[(x,y) | y <-[3,4]] | x<-[1,2]]
```
$\equiv$
```
concat (map f [1,2]) where f x = [(x,y) | y<-[3,4]]
```
$\equiv$
```
concat (map f [1,2]) where f x = map (\y->(x,y)) [3,4]
```