

Implementación de Relaciones en Haskell: Listas por Comprensión

El objetivo de este material, así como el de los sucesivos que se irán proporcionando a lo largo del curso, es introducir en cada tema aplicaciones de los mismos en Haskell, así como comenzar a implementar eficazmente dichos conceptos a través de este lenguaje de programación funcional.

Haskell permite definir **Listas** o **Secuencias** (conjunto definido por inducción, concepto que abordaremos en el curso más adelante) mediante una notación similar a la utilizada para escribir **conjuntos por comprensión** en matemática.

Por ejemplo:

$P = \{x \in \mathbb{N} / x \text{ es par}\}$, denota el conjunto de los x tales que x es un número natural par.

En Haskell podemos usar la notación de las listas por comprensión para obtener el mismo resultado:

```
Hugs> [x | x <- [0..], even x]
```

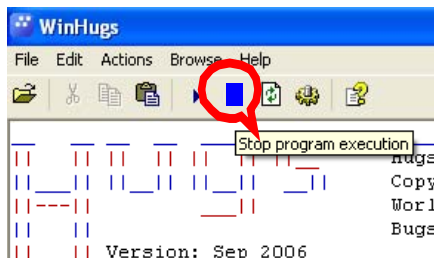
El conjunto $[0..]$ es el conjunto de los números naturales, \mathbb{N} .

Los operadores `even`, `odd :: Int -> Bool`, comprueban la naturaleza par o impar de un número entero respectivamente.

```
Main> even 5  
False
```

```
Main> even 6  
True
```

Claro, si se intenta usar en Haskell una lista infinita, el programa llenará la pantalla de números.... lo cual seguramente hará que el usuario se ponga muy nervioso, intentando cerrar el programa!!! Para no cerrarlo y parar un proceso que no termina se utiliza el botón **Stop program execution**.



La condición `{Interrupted!}` se utiliza cuando el proceso no termina debido a la naturaleza infinita del conjunto (en este caso la lista o secuencia).

Se utiliza el botón **Stop program execution**. Probarlo !!

Para implementar dicha construcción del conjunto en Haskell, y poder visualizar el resultado en forma correcta, se sugiere utilizar `[0..100]` en lugar de `[0..]` debido a la naturaleza infinita de este último.

Ejercicio: definir en Haskell el conjunto $I = \{x \in \mathbb{N} / x \text{ es impar}\}$.

La notación de listas por comprensión permite declarar de forma concisa una gran cantidad de iteraciones sobre listas. Esta notación está adaptada de la teoría de conjuntos de Zermelo- Fraenkel. Sin embargo en Haskell se trabaja con listas, no con conjuntos. El formato básico de la definición de una lista por comprensión es:

```
[ <expr> | <qualif_1>, <qualif_2> . . . <qualif_n> ]
```

Es decir que, la **sintaxis** de una lista por comprensión en Haskell es la siguiente:

```
[expresión | cualificador1, cualificador2, ..., cualificadorn]
```

Donde cada cualificador puede ser:

- Un **generador**: expresión que genera una lista.

Generadores: Un cualificador de la forma `pat<-exp` es utilizado para extraer cada elemento que encaje con el patrón `pat` de la lista `exp` en el orden en que aparecen los elementos de la lista. Un ejemplo simple sería la expresión:

```
Hugs> [x*x | x <- [1..10]]  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- Una **guarda o filtro**: Una expresión de valor booleano, el significado de una lista por comprensión con un único filtro podría definirse como:

```
[e | condicion ] = if condition then [e] else []
```

Esta forma de lista por comprensión resulta útil en combinación con generadores, Ejemplo:

```
Hugs> [x*x | x<-[1..10], even x ]  
[4, 16, 36, 64, 100]
```

- Una **definición local**: se usan para definir elementos locales para la expresión.

En el ejemplo:

```
Hugs> [2*x | x <- [1..5]]  
[2, 4, 6, 8, 10] :: Integer
```

El único cualificador es un generador. La variable `x` se instancia a cada uno de los elementos de la lista `[1..5]` y, para cada instancia, un elemento de la forma $2*x$ se añade a la lista resultado. El ejemplo muestra que la expresión a la izquierda del símbolo `|` puede depender de las variables introducidas en los generadores.

Una guarda puede ser utilizada para seleccionar los elementos de un generador que cumplen cierta condición:

```
Hugs> [2*x | x <- [1..5], odd x]  
[2, 6, 10]
```

Sólo los elementos impares de la lista `[1..5]` generan un elemento en la lista resultado. El ejemplo muestra que un cualificador puede utilizar una variable introducida en uno previo (más a la izquierda).

Si aparecen varios cualificadores hay que tener en cuenta que:

- Las variables generadas por los cualificadores posteriores varían más rápidamente que las generadas por los cualificadores anteriores:

```
Hugs> [ (x,y) | x<-[1..3], y<-[1..2] ]  
[(1,1), (1,2), (2,1), (2,2), (3,1), (3,2)]
```

- Los cualificadores posteriores podrían utilizar los valores generados por los anteriores:

```
Hugs> [ (x,y) | x<-[1..3], y<-[1..x]]  
[(1,1), (2,1), (2,2), (3,1), (3,2), (3,3)] ¿Es este un producto cartesiano?
```

- Las variables definidas en cualificadores posteriores ocultan las variables definidas por cualificadores anteriores. Las siguientes expresiones son listas definidas por comprensión de

forma válida. Sin embargo, no es una buena costumbre reutilizar los nombres de las variables ya que dificulta la comprensión de los programas.

```
Hugs> [ x | x<-[[1,2],[3,4]], x<-x ]  
[1,2,3,4]
```

```
Hugs> [ x | x<-[1,2], x<-[3,4] ]  
[3,4,3,4]
```

- Un cambio en el orden de los cualificadores puede tener un efecto directo en la eficiencia. Los siguientes ejemplos producen el mismo resultado, pero el primero utiliza más reducciones debido a que repite la evaluación de "even x" por cada valor posible de "y".

```
Hugs> [ (x,y) | x<-[1..3], y<-[1..2], even x ]  
[(2,1),(2,2)]
```

```
Hugs> [ (x,y) | x<-[1..3], even x, y<-[1..2] ]  
[(2,1),(2,2)]
```

Si se introduce más de un generador, los que aparecen a la derecha cambian más rápido:

Ejemplo: Se define el producto cartesiano $A \times B$ de los $A=\{1,2,3\}$ y $B=\{a,b\}$

```
Hugs> [(x,y) | x <- [1..3], y <- ['a','b']]  
[(1,'a'),(1,'b'),(2,'a'),(2,'b'),(3,'a'),(3,'b')]
```

Ejercicio: Definir en Haskell los productos cartesianos indicados en el siguiente ejercicio. Dados $A = \{2,3,4\}$ y $B = \{4,5\}$, determinar: $A \times B$, $B \times A$, $B \times B$, $A \times A$

Ejemplo: (Ejercicio 14 – Práctico 3) Se define la relación binaria R sobre $A / A=\{1,2,3,4,5\}$ tal que dos elementos x e y están relacionados sí y sólo si, la suma de los mismos da como resultado un múltiplo de 3.

```
Hugs> [(x,y) | x <- [1..5], y <- [1..5], mod(x+y)3==0]  
[(1,2),(1,5),(2,1),(2,4),(3,3),(4,2),(4,5),(5,1),(5,4)]
```

Los operadores **div** y **mod** devuelven el cociente y el resto respectivamente de la división entera entre sus argumentos. Ejemplos:

```
Main> mod 15 4  
3  
Main> mod 27 3  
0  
Main> div 27 3  
9  
Main> div 16 4  
4
```

¿Que esperas obtener con `div (-28) 3`? ¿Cómo lo explicas?

Ejercicios: Definir en Haskell las siguientes relaciones:

1. $A = \{1,2,3\}$, $R \subseteq A \times A / x R y \Leftrightarrow 2|(x + y)$
2. $A = \{0,1,2,\dots,9\}$, $B = \{a,b,c\}$, $R \subseteq A \times B / x$ es múltiplo de 3
3. $X = \mathbb{N}$, $R \subseteq \mathbb{N} \times \mathbb{N} / x R y \Leftrightarrow x$ divide a y (es decir si y es múltiplo de x)

Escribir sus propios ejemplos en Haskell utilizando relaciones binarias, y en cada uno de los casos que se defina, investigue si la relación es de equivalencia.

Algunos ejemplos de Relaciones en Haskell:

1. Definir en Haskell una relación binaria R sobre el conjunto $A=\{1,3,7,8\}$ de la siguiente forma: $R:A \rightarrow A / x R y \Leftrightarrow x < y$

```
Hugs> [(x,y) | x<-[1,3,7,8], y<-[1,3,7,8], x<y]
[(1,3), (1,7), (1,8), (3,7), (3,8), (7,8)]
```

2. Definir en Haskell la relación de orden \leq sobre el conjunto de los números naturales menores que 10.

```
Hugs> [(x,y) | x<-[0..9], y<-[0..9], x<=y]
```

3. La relación R "ser divisor de" definida en un conjunto numérico A, es una relación de orden Parcial (Probarlo). Definirla en Haskell, sobre el conjunto de los Naturales menores que 100.

4. Se consideran las siguientes relaciones en el conjunto de los números naturales, se pide que sean definidas en Haskell:

- $R_1=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a \leq b\}$
- $R_2=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a > b\}$
- $R_3=\{(a,b) \in \mathbb{Z} \times \mathbb{Z} / a=b \text{ ó } a=-b\}$
- $R_4=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a=b\}$
- $R_5=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a=b+1\}$
- $R_6=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a+b \leq 3\}$
- $R_7=\{(a,b) \in \mathbb{N} \times \mathbb{N} / a \neq b\}$

Obs.: para visualizar en Haskell lo que se pretende conceptualizar, es conveniente, en lugar de definir N como $[0..]$, utilizar un conjunto finito, como por ejemplo $[0..n]$ con n elegido a conveniencia. Asimismo para trabajar con el conjunto de los Enteros, y con el mismo objetivo utilizaremos $[-n..n]$

Ejemplo:

$R_3=\{(a,b) \in \mathbb{Z} \times \mathbb{Z} / a=b \text{ ó } a=-b\}$

```
Hugs> [(x,y) | y<-[-9..9], x<-[-9..9], abs(x)==abs(y)]
```

5. Enumera en Haskell los pares ordenados de la relación R de $A=\{1,2,3,4\}$ en $B=\{1,2,3\}$ donde $(a,b) \in R$ si y sólo si:

- i) $a=b$
- ii) $a+b=4$
- iii) $a > b$
- iv) $a | b$ ("a" divide a "b") Por ejemplo $3 | 12$
- v) $\text{mcd}(a,b)=1$

Ejemplo: 5 v) $\text{mcd}(a,b)=1$ (en inglés, máximo común divisor es **gcd**)

```
Main> [(x,y) | y<-[1,2,3,4], x<-[1,2,3], gcd y x ==1]
[(1,1), (2,1), (3,1), (1,2), (3,2), (1,3), (2,3), (1,4), (3,4)]
```

6. Enumera en Haskell los pares ordenados de la relación R de $A=\{1,2,3,4\}$ en $B=\{1,2,3\}$ donde $(a,b) \in R$ si y sólo si $a+b > 8$