

Programación Funcional

Curso 2019-20

DEFINICIÓN DE FUNCIONES EN HASKELL

Programas Haskell

Programación funcional

- Programas \equiv definiciones de funciones
- Cómputos \equiv evaluación de expresiones

Un programa Haskell `file.hs` consta de:

- Definiciones de funciones
- Definiciones acerca de tipos:
 - Nuevos tipos de datos (*data*)
 - Nuevas clases de tipos (*class*)
 - Declaraciones de instancia de tipos (*instance*)
 - Alias de tipo (*type*) y tipos isomorfos (*newtype*)
- Nuevos operadores infijos (*infix*)
- Declaraciones relativas a módulos (*module, import, ...*)

Definición de nuevas funciones

Las funciones son definidas mediante **ecuaciones**

```
doblo x = x + x
factorial n = product [1..n]
sandwich xs ys = let  us = xs++xs
                   vs = ys++ys
                   in  us++vs++us
```

- ★ Notación currificada
- ★ xs,ys,us,vs nombres típicos para listas

Definición de nuevas funciones

Las funciones son definidas mediante **ecuaciones**


```
doble x = x + x
factorial n = product [1..n]
sandwich xs ys = let  us = xs++xs
                   vs = ys++ys
                   in  us++vs++us
```

- ★ Notación currificada
- ★ xs,ys,us,vs nombres típicos para listas
- ★ ¿No hay marcadores de fin de ecuación?

Definición de nuevas funciones

Las funciones son definidas mediante **ecuaciones**

```
doblex = x + x ;  
factorial n = product [1..n] ;  
sandwich xs ys = let  us = xs++xs ;  
                   vs = ys++ys  
                   in  us++vs++us
```

- ★ Notación currificada
- ★ xs,ys,us,vs nombres típicos para listas
- ★ ¿No hay marcadores de fin de ecuación?
Sí, hay un ; implícito entre ecuaciones
 **regla de indentación**

Inciso: regla de indentación

```
doble x = x + x
  factorial n = product [1..n]
sandwich xs ys = let us = xs++xs
                  vs = ys++ys in us ++ vs ++ us
```

Error de sintaxis!

Lo siguiente sí es correcto (pero no recomendado)!

```
{doble x = x + x ;   factorial n = product [1..n]
  ;
  sandwich xs ys = ...}
```

- En una secuencia de definiciones ecuacionales, hay un ; implícito cada vez que una línea comienza en la misma columna que la definición anterior
- Se aplica también a las secuencias de definiciones locales *let* y *where* (y *do*, que veremos más adelante)

Definición de nuevas funciones (II)

Distinciones de casos por expresiones condicionales

```
factorial n = if n==0  then 1
              else n*factorial (n-1)
```

```
f x y =  if x==0  then True else
         if y==0  then False
         else f (x-1) (y-1)
```

- ★ Estas definiciones hay que escribirlas en un *file.hs*
- ★ No usar tabuladores!

Definición de nuevas funciones (III)

Distinciones de casos por **ajuste de patrones**

```
factorial 0 = 1  
factorial n = n*factorial (n-1)
```

```
f 0 y = True  
f x 0 = False  
f x y = f (x-1) (y-1)
```

En el lado izquierdo de cada ecuación se indica a qué valores de los argumentos resulta aplicable la ecuación

- ★ Puede haber más de una ecuación por función
- ★ Puede haber *solapamiento* de patrones entre ecuaciones
- ★ Una ecuación se aplica solo si las anteriores no son aplicables

⇒ El orden de las ecuaciones importa

Para bien: ecuaciones más simples (condiciones implícitas)

Para mal: cada ecuación suelta no tiene valor declarativo

- ★ La aparición de patrones de ajuste en las distintas ecuaciones guía la evaluación perezosa de la función

Definición de nuevas funciones (IV)

Distinciones de casos por **ecuaciones guardadas**

```
factorial n
| n==0 = 1   Guarda
| True  = n*factorial (n-1)
```

Las guardas pueden solaparse

Uso secuencial de las ecuaciones guardadas

```
f x y
| x==0 = True
| y==0 = False
| True  = f (x-1) (y-1)
```

$f \ t_1 \ \dots t_n$

| $b_1 = e_1$

...

| $b_m = e_m$

t_i patrones *lineales* (sin variables repetidas)

b_i expresiones booleanas

e_i expresiones (del mismo tipo)

Ojo: | $b_i = e_i$ **no** es una expresión

Variaciones sobre el mismo tema

```
factorial n
| n==0  = 1
| n>0   = n*factorial (n-1)
```

Las guardas pueden no ser exhaustivas

```
> factorial (-2)
```

Exception: Non-exhaustive patterns

Lo mismo ocurre con el ajuste de patrones

Variaciones sobre el mismo tema

```
factorial n
| n==0  = 1
| n>0   = n*factorial (n-1)
| n<0   = error "argumento negativo"
```

```
> factorial (-2)
Exception: argumento negativo
```

error string

Genera un error con mensaje asociado *string*

Variaciones sobre el mismo tema

```
factorial n
| n==0  = 1
| n>0   = n*factorial (n-1)
| n<0   = error "el argumento "++show n++" es negativo"
```

```
> factorial (-2)
```

```
Exception:  el argumento -2 es negativo
```

`show x`

Devuelve un string (lista de caracteres) que representa a `x`

`x` de cualquier tipo de la clase *Show*

Variaciones sobre el mismo tema

```
factorial n
| n==0      = 1
| n>0      = n*factorial (n-1)
| otherwise = error "argumento negativo"
```

otherwise

Función de aridad 0 definida en Prelude

- `otherwise = True`
- Por tanto, $\llbracket \text{otherwise} \rrbracket = \text{True}$