

Programación declarativa (2007–08)

Tema 2: Números y funciones

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 2: Números y funciones (I)

1. Operadores

Operadores como funciones y viceversa

Precedencias

Asociatividad

Definición de operadores

2. Currificación

Instanciación parcial

Reducción de paréntesis

Secciones de operadores

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

Tema 2: Números y funciones (II)

- 4. Problemas de cálculo numérico
 - Cálculo con números enteros
 - Diferenciación numérica
 - Cálculo de la raíz cuadrada
 - Ceros de una función
 - Funciones inversas

Tema 2: Números y funciones

- 1. Operadores
 - Operadores como funciones y viceversa
 - Precedencias
 - Asociatividad
 - Definición de operadores
- 2. Currificación
- 3. Funciones como parámetros
- 4. Problemas de cálculo numérico

Operadores como funciones

▶ Ejemplo:

$$\left| \begin{array}{l} 2+3 \quad \rightsquigarrow 5 \\ (+) 2 3 \quad \rightsquigarrow 5 \end{array} \right.$$

- ▶ Regla: un operador entre paréntesis se comporta como la función correspondiente.

Funciones como operadores

▶ Ejemplo:

$$\left| \begin{array}{l} \max 2 3 \quad \rightsquigarrow 3 \\ 2 \text{ 'max' } 3 \quad \rightsquigarrow 3 \end{array} \right.$$

- ▶ Regla: una función entre comillas inversas se comporta como el operador correspondiente.

Tema 2: Números y funciones

1. Operadores

Operadores como funciones y viceversa

Precedencias

Asociatividad

Definición de operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Ejemplos de precedencias

► Ejemplos:

$$2*3+4*5 \rightsquigarrow 26$$

$$2*3<7 \rightsquigarrow \text{True}$$

* tiene mayor precedencia que +

< tiene mayor precedencia que *

► Pueden usarse paréntesis para saltarse las precedencia. Por ejemplo,

$$2*3+4*5 \rightsquigarrow 26$$

$$2*(3+4)*5 \rightsquigarrow 70$$

Precedencia de los operadores del preludio

nivel 9	., !!
nivel 8	^
nivel 7	*, /, 'quot', 'rem', 'div', 'mod'
nivel 6	+, -
nivel 5	:, ++
nivel 4	==, /=, <, <=, >=, >, 'elem', 'notElem'
nivel 3	&&
nivel 2	

Precedencia de la llamada a funciones

- ▶ La llamada a funciones (el operador invisible entre f y x en $f\ x$) tiene la máxima precedencia. Por ejemplo,

$$\begin{array}{l} \text{cuadrado } 3 + 4 \rightsquigarrow 13 \\ \text{cuadrado } (3 + 4) \rightsquigarrow 49 \end{array}$$

- ▶ En la definición de funciones por análisis de patrones la llamada a función tiene la precedencia más alta. Por ejemplo,

$$\begin{array}{l} \text{fac } 0 = 1 \\ \text{fac } (n+1) = (n+1) * \text{fac } n \end{array}$$

los paréntesis en la segunda ecuación son necesarios.

Tema 2: Números y funciones

1. Operadores

Operadores como funciones y viceversa

Precedencias

Asociatividad

Definición de operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Operadores

- ▶ asocian por la izquierda:
+, -, *, /, 'quot', 'rem', 'div', 'mod', '&&', '||', '!!'
- ▶ asocian por la derecha:
^, :, ++, .
- ▶ no asocian:
==, /=, <, <=, >=, >, 'elem', 'notElem'

Tema 2: Números y funciones

1. Operadores

Operadores como funciones y viceversa

Precedencias

Asociatividad

Definición de operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Definición de operadores

- ▶ Ejemplo de declaración de operadores del prelude:

```

_____ Prelude _____
infixr 8  ^
infixl 6  +, -
infix  4  ==, /=, <, <=, >=, >, 'elem', 'notElem'
    
```

- ▶ Ejemplo de definición de operador:

$x \sim= y$ se verifica si $|x - y| < 0,0001$. Por ejemplo,

```

| 3.00001 ~= 3.00002 ~> True
| 3.1 ~= 3.2           ~> False
    
```

```

_____
infix 4 ~=
(~=)  :: Float -> Float -> Bool
x ~= y = abs(x-y) < 0.0001
    
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

Instanciación parcial

Reducción de paréntesis

Secciones de operadores

3. Funciones como parámetros

4. Problemas de cálculo numérico

Ejemplo de instanciación parcial

Definición de siguiente mediante instanciación parcial de suma:

- ▶ suma x y es la suma de x e y:

```
suma :: Int -> Int -> Int
```

```
suma x y = x+y
```

- ▶ siguiente x es el siguiente de x. Por ejemplo,

```
| siguiente 3  ~> 4
```

```
siguiente :: Int -> Int
```

```
siguiente = suma 1
```

Instancias parciales

- ▶ Ejemplo de uso de una instancia parcial como parámetro:

```
| map (suma 5) [1,2,3] ~> [6,7,8]
```

- ▶ Tipos de instancias parciales:

```
| Main> :type suma  
suma :: Int -> Int -> Int  
Main> :type suma 1  
suma 1 :: Int -> Int  
Main> :type suma 1 3  
suma 1 3 :: Int
```

Regla: \rightarrow asocia por la derecha.

- ▶ Estrategia de currificación: describir las funciones de más de un parámetro por funciones de un parámetro que devuelven otra función.

Tema 2: Números y funciones

1. Operadores

2. Currificación

Instanciación parcial

Reducción de paréntesis

Secciones de operadores

3. Funciones como parámetros

4. Problemas de cálculo numérico

Ejemplos de reducción de paréntesis

- ▶ Se ha elegido la asociatividad de \rightarrow y la aplicación de función de tal manera que no se ve la currificación: la aplicación de función asocia por la izquierda y \rightarrow asocia por la derecha. Por ejemplo,

```
sumaCuatro :: Int -> Int -> Int -> Int -> Int
sumaCuatro a b c d = a+b+c+d
```

entonces

```
| sumaCuatro 2 3 4 5 ~> 14
```

- ▶ La regla es: si no están escritos los paréntesis, entonces están escritos implícitamente de tal manera que funciona la currificación.

Ejemplos en los que se necesitan paréntesis

- ▶ en declaraciones de tipo:

```
map :: (a -> b) -> [a] -> [b]
```

- ▶ en expresiones:

```
| cuadrado (3 + 4)
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

Instanciación parcial

Reducción de paréntesis

Secciones de operadores

3. Funciones como parámetros

4. Problemas de cálculo numérico

Secciones de operadores

Si \oplus es un operador, entonces

▶ $(\oplus x)$ es la función definida por $(\oplus x)y = y \oplus x$.

▶ $(x \oplus)$ es la función definida por $(x \oplus)y = x \oplus y$.

Ejemplos de definiciones mediante secciones:

▶ `succ x` es el siguiente de `x`. Por ejemplo, `succ 3` \rightsquigarrow 4

_____ Prelude _____
`succ = (+1)`

▶ `doble x` es el doble de `x`. Por ejemplo, `doble 3` \rightsquigarrow 6

_____ `doble = (2*)` _____

▶ `mitad x` es la mitad de `x`. Por ejemplo, `mitad 5` \rightsquigarrow 2.5

_____ `mitad = (/2)` _____

Secciones de operadores

- ▶ `inverso x` es el inverso de `x`. Por ejemplo, `inverso 2` \rightsquigarrow `0.5`

```
inverso = (1/)
```

- ▶ `cuadrado x` es el cuadrado de `x`. Por ejemplo, `cuadrado 3` \rightsquigarrow `9`

```
cuadrado = (^2)
```

- ▶ `dosElevadoA x` es 2^x . Por ejemplo, `dosElevadoA 3` \rightsquigarrow `8`

```
dosElevadoA = (2^)
```

- ▶ `esPositivo` se verifica si `x` es positivo. Por ejemplo,

```
esPositivo 3       $\rightsquigarrow$  True
esPositivo (-3)   $\rightsquigarrow$  False
```

```
esPositivo = (>0)
```

Secciones como parámetros

Ejemplos de uso de secciones como parámetros:

```
Main> map (2*) [1,2,3]
[2,4,6]
Main> map (2/) [1,2,4,8]
[2.0,1.0,0.5,0.25]
Main> map (/2) [1,2,4,8]
[0.5,1.0,2.0,4.0]
Main> map (2^) [1,2,4,8]
[2,4,16,256]
Main> map (^2) [1,2,4,8]
[1,4,16,64]
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

4. Problemas de cálculo numérico

Funciones de orden superior

Las **funciones de orden superior** son las que tienen funciones como parámetros.

- ▶ **map** f l es la lista obtenida aplicando f a cada elemento de l .

Por ejemplo,

```
| map (2*) [1,2,3] ~> [2,4,6]
```

- ▶ **filter** p l es la lista de los elementos de l que cumplen la propiedad p . Por ejemplo,

```
| filter even [1,3,5,4,2,6,1] ~> [4,2,6]
```

```
| filter (>3) [1,3,5,4,2,6,1] ~> [5,4,6]
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

4. Problemas de cálculo numérico

27 / 54

La función de iteración `until`

- `until p f x` aplica la `f` a `x` el menor número posible de veces, hasta alcanzar un valor que satisface el predicado `p`. Por ejemplo,

```
| until (>1000) (2*) 1 ~> 1024
```

_____ Prelude _____

```
until :: (a -> Bool) -> (a -> a) -> a -> a
```

```
until p f x = if p x
```

```
              then x
```

```
              else until p f (f x)
```

28 / 54

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

4. Problemas de cálculo numérico

29 / 54

Composición de funciones

- ▶ $f . g$ es la composición de las funciones f y g ; es decir, la función que aplica x en $f(g(x))$. Por ejemplo,

```
| (cuadrado . succ) 2 ~> 9  
| (succ . cuadrado) 2 ~> 5
```

----- Prelude -----

```
(.) :: (b -> c) -> (a -> b) -> (a -> c)  
(f . g) x = f (g x)
```

- ▶ Ejemplo de definición mediante composición:

```
impar = not . even
```

- ▶ Ejemplo de uso de composición como parámetro:

```
| Main> filter (not . even) [1,3,5,4,2,6,1]  
| [1,3,5,1]
```

30 / 54

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

4. Problemas de cálculo numérico

Intercambio de argumentos

- ▶ `flip f` intercambia el orden de los argumentos de `f`. Por ejemplo,

```
(-) 5 2      ~> 3  
flip (-) 5 2 ~> -3  
(/) 5 2      ~> 2.5  
flip (/) 5 2 ~> 0.4
```

_____ Prelude _____

```
flip :: (a -> b -> c) -> b -> a -> c  
flip f x y = f y x
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

Funciones sobre listas

Iteración

Composición

Intercambio de argumentos

Funciones anónimas

4. Problemas de cálculo numérico

Funciones anónimas

La expresión

`\x1 ... xn -> cuerpo`

representa una función anónima. Por ejemplo,

```
Main> map (\x -> 2*x+1) [1..5]
[3,5,7,9,11]
Main> filter (\x -> 2*x > 7) [1..5]
[4,5]
Main> filter (\x -> odd x) [1..5]
[1,3,5]
Main> filter (\x -> 2*x > 7 && odd x) [1..5]
[5]
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Cálculo con números enteros

Diferenciación numérica

Cálculo de la raíz cuadrada

Ceros de una función

Funciones inversas

Cociente y resto

- ▶ $\text{div } x \ y$ es el cociente entero de x entre y . Por ejemplo,
 $| 7 \text{ 'div' } 2 \rightsquigarrow 3$
- ▶ $\text{rem } x \ y$ es el resto de dividir x por y . Por ejemplo,
 $| 7 \text{ 'rem' } 2 \rightsquigarrow 1$
- ▶ Aplicaciones de las funciones cociente y resto:
 - ▶ Calcular horas: Si son las 9 ahora, entonces 33 horas más tarde serán las $(9+33) \text{ 'rem' } 24 = 20$.
 - ▶ Calcular días: Si se codifican los días con $0=\text{domingo}$, $1=\text{lunes}$, ..., $6=\text{sábado}$ y hoy es el día 3 (miércoles), entonces dentro de 40 días será $(3+40) \text{ 'rem' } 7 = 1$ (lunes).
 - ▶ Determinar si un número es divisible: Un número es divisible por n si el resto de la división por n es igual a cero.
 - ▶ Determinar las cifras de un número:
La última cifra de un número x es $x \text{ 'rem' } 10$.
La penúltima cifra es $(x \text{ 'div' } 10) \text{ 'rem' } 10$.
La antepenúltima cifra es $(x \text{ 'div' } 100) \text{ 'rem' } 10$, etc.

Diseño ascendente: Cálculo de una lista de primos

- ▶ divisible x y se verifica si x es divisible por y . Por ejemplo,

```
| divisible 9 3 ~> True  
| divisible 9 2 ~> False
```

```
divisible :: Int -> Int -> Bool  
divisible x y = x `rem` y == 0
```

- ▶ divisores x es la lista de los divisores de x . Por ejemplo,

```
| divisores 12 ~> [1,2,3,4,6,12]
```

```
divisores :: Int -> [Int]  
divisores x = filter (divisible x) [1..x]
```

37 / 54

Diseño ascendente: Cálculo de una lista de primos

- ▶ primo x se verifica si x es primo. Por ejemplo,

```
| primo 5 ~> True  
| primo 6 ~> False
```

```
primo :: Int -> Bool  
primo x = divisores x == [1,x]
```

- ▶ primos x es la lista de los números primos menores o iguales que x . Por ejemplo,

```
| primos 40 ~> [2,3,5,7,11,13,17,19,23,29,31,37]
```

```
primos :: Int -> [Int]  
primos x = filter primo [1..x]
```

38 / 54

Diseño descendente: Cálculo del día de la semana

- ▶ día d m a es el día de la semana correspondiente al día d del mes m del año a. Por ejemplo,

```
| día 31 12 2007 ~> "lunes"
```

```
día d m a = díaSemana ((númeroDeDías d m a) 'mod' 7)
```

- ▶ númeroDía d m a es el número de días transcurridos desde el 1 de enero del año 0 hasta el día d del mes m del año a. Por ejemplo,

```
| númeroDeDías 31 12 2007 ~> 733041
```

```
númeroDeDías d m a = (a-1)*365  
                    + númeroDeBisiestos a  
                    + sum (take (m-1) (meses a))  
                    + d
```

39 / 54

Diseño descendente: Cálculo del día de la semana

- ▶ númeroDeBisiestos a es el número de años bisiestos antes del año a.

```
númeroDeBisiestos a =  
  length (filter bisiesto [1..a-1])
```

- ▶ bisiesto a se verifica si el año a es bisiesto. La definición de año bisiesto es
 - ▶ un año divisible por 4 es un año bisiesto (por ejemplo 1972);
 - ▶ excepción: si es divisible por 100, entonces no es un año bisiesto
 - ▶ excepción de la excepción: si es divisible por 400, entonces es un año bisiesto (por ejemplo 2000).

```
bisiesto a =  
  divisible a 4  
  && (not(divisible a 100) || divisible a 400)
```

40 / 54

Diseño descendente: Cálculo del día de la semana

- ▶ `meses a` es la lista con el número de días de los meses del año `a`. Por ejemplo,

```
| meses 2000 ~> [31,29,31,30,31,30,31,31,30,31,30,31]
```

```
meses a = [31,feb,31,30,31,30,31,31,30,31,30,31]  
  where feb | bisiesto a = 29  
           | otherwise = 28
```

- ▶ `take n l` es la lista de los `n` primeros elementos de `l`. Por ejemplo,

```
| take 2 [3,5,4,7] ~> [3,5]  
| take 12 [3,5,4,7] ~> [3,5,4,7]
```

Diseño descendente: Cálculo del día de la semana

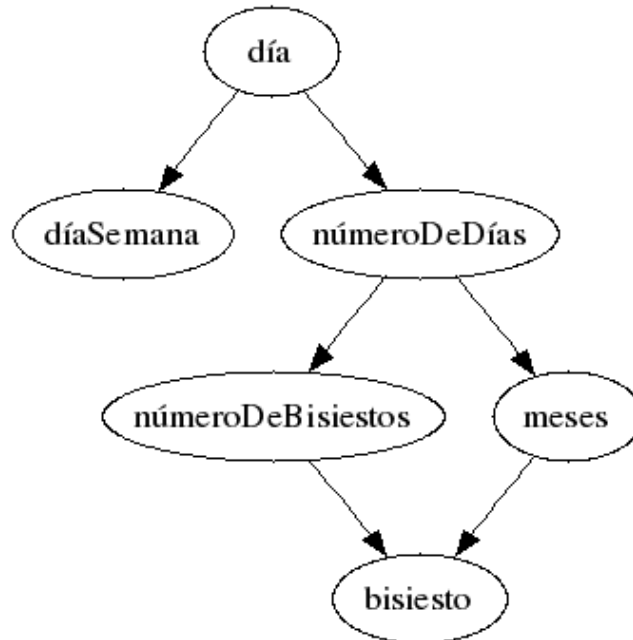
- ▶ `díaSemana n` es el `n`-ésimo día de la semana comenzando con 0 el domingo. Por ejemplo,

```
| díaSemana 2 ~> "martes"
```

```
díaSemana 0 = "domingo"  
díaSemana 1 = "lunes"  
díaSemana 2 = "martes"  
díaSemana 3 = "miércoles"  
díaSemana 4 = "jueves"  
díaSemana 5 = "viernes"  
díaSemana 6 = "sábado"
```

Diseño descendente: Cálculo del día de la semana

► Grafo de funciones



Tema 2: Números y funciones

1. Operadores
2. Currificación
3. Funciones como parámetros
4. Problemas de cálculo numérico
 - Cálculo con números enteros
 - Diferenciación numérica
 - Cálculo de la raíz cuadrada
 - Ceros de una función
 - Funciones inversas

Diferenciación numérica

- ▶ `derivada a f x` es el valor de la derivada de la función `f` en el punto `x` con aproximación `a`. Por ejemplo,

```
| derivada 0.001 sin pi ~> -0.9999273
| derivada 0.001 cos pi ~> 0.0004768371
```

```
derivada :: Float -> (Float -> Float) -> Float -> Float
derivada a f x = (f(x+a)-f(x))/a
```

- ▶ Tipos de derivadas:

```
derivadaBurda = derivada 0.01
derivadaFina  = derivada 0.0001
derivadaSuper = derivada 0.000001
```

Por ejemplo,

```
| derivadaFina cos pi ~> 0.0
```

Diferenciación numérica

- ▶ `derivadaFinaDelSeno x` es el valor de la derivada fina del seno en `x`. Por ejemplo,

```
| derivadaFinaDelSeno pi ~> -0.9989738
```

```
derivadaFinaDelSeno = derivadaFina sin
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Cálculo con números enteros

Diferenciación numérica

Cálculo de la raíz cuadrada

Ceros de una función

Funciones inversas

Cálculo iterativo de la raíz cuadrada

- ▶ Propiedad: si y es una aproximación de \sqrt{x} , entonces $\frac{1}{2}(y + \frac{x}{y})$ es una aproximación mejor.

- ▶ Propiedad: \sqrt{x} es el límite de la sucesión x_n definida por

$$\begin{cases} x_0 & = 1 \\ x_{n+1} & = \frac{1}{2}(y + \frac{x}{x_n}) \end{cases}$$

- ▶ Programa: `raiz x` es la raíz cuadrada de `x` calculada usando la propiedad anterior. Por ejemplo,

```
| raiz 9 ~> 3.00000000139698
```

```
raiz x = until acceptable mejorar 1
```

```
  where mejorar y = 0.5*(y+x/y)
```

```
        acceptable y = abs(y*y-x) < 0.00001
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Cálculo con números enteros

Diferenciación numérica

Cálculo de la raíz cuadrada

Ceros de una función

Funciones inversas

Método de Newton para calcular los ceros

- ▶ Propiedad: si b es una aproximación para el punto cero de f , entonces $b - \frac{f(b)}{f'(b)}$ es una mejor aproximación.

- ▶ Propiedad: el límite de la sucesión x_n definida por

$$\begin{cases} x_0 & = 1 \\ x_{n+1} & = x_n - \frac{f(x_n)}{f'(x_n)} \end{cases}$$

es un cero de f .

- ▶ Programa: puntoCero f es un cero de la función f calculado usando la propiedad anterior. Por ejemplo,

```
| puntoCero cos ~> 1.570796
```

```
puntoCero f = until acceptable mejorar 1  
  where mejorar b = b - f b / derivadaFina f b  
        acceptable b = abs (f b) < 0.00001
```

Tema 2: Números y funciones

1. Operadores

2. Currificación

3. Funciones como parámetros

4. Problemas de cálculo numérico

Cálculo con números enteros

Diferenciación numérica

Cálculo de la raíz cuadrada

Ceros de una función

Funciones inversas

Funciones inversas

- ▶ Definición de raíces mediante puntoCero:

```
raíz_cuadrada a = puntoCero f
  where f x = x*x-a
```

```
raíz_cúbica a = puntoCero f
  where f x = x*x*x-a
```

- ▶ Definición de funciones inversas mediante puntoCero:

```
arco_seno a = puntoCero f
  where f x = sin x - a
```

```
arco_coseno a = puntoCero f
  where f x = cos x - a
```

Funciones inversas

- ▶ Patrón de función inversa:

```
inversa g a = puntoCero f
  where f x = g x - a
```

- ▶ Redefiniciones con el patrón:

```
arco_seno_1  = inversa sin
arco_coseno_1 = inversa cos
logaritmo    = inversa exp
```

Por ejemplo,

```
|logaritmo (exp 1) ~> 1.0
```

Bibliografía

1. H. C. Cunningham (2007) *Notes on Functional Programming with Haskell*.
2. J. Fokker (1996) *Programación funcional*.
3. B.C. Ruiz, F. Gutiérrez, P. Guerrero y J. Gallardo (2004). *Razonando con Haskell (Un curso sobre programación funcional)*.
4. S. Thompson (1999) *Haskell: The Craft of Functional Programming*.
5. E.P. Wentworth (1994) *Introduction to Funcional Programming*.