

# Tema 7: Razonamiento sobre programas

## Programación declarativa (2009–10)

José A. Alonso Jiménez

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

---

IM Tema 7: Razonamiento sobre programas

---

## Tema 7: Razonamiento sobre programas

1. Razonamiento ecuacional
  - Cálculo con longitud
  - Propiedad de intercambia
  - Inversa de listas unitarias
  - Razonamiento ecuacional con análisis de casos
2. Razonamiento por inducción sobre los naturales
  - Esquema de inducción sobre los naturales
  - Ejemplo de inducción sobre los naturales
3. Razonamiento por inducción sobre listas
  - Esquema de inducción sobre listas
  - Asociatividad de ++
  - [] es la identidad para ++ por la derecha
  - Relación entre length y ++
  - Relación entre take y drop
  - La concatenación de listas vacías es vacía
4. Equivalencia de funciones

## Tema 7: Razonamiento sobre programas

### 1. Razonamiento ecuacional

Cálculo con longitud

Propiedad de intercambia

Inversa de listas unitarias

Razonamiento ecuacional con análisis de casos

### 2. Razonamiento por inducción sobre los naturales

### 3. Razonamiento por inducción sobre listas

### 4. Equivalencia de funciones

3 / 37

## Cálculo con longitud

### ► Programa:

```
longitud []      = 0                -- longitud.1
longitud (_:xs) = 1 + longitud xs  -- longitud.2
```

### ► Propiedad: $\text{longitud } [2,3,1] = 3$

### ► Demostración:

```
longitud [2,3,1]
= 1 + longitud [2,3]                [por longitud.2]
= 1 + (1 + longitud [3])            [por longitud.2]
= 1 + (1 + (1 + longitud []))      [por longitud.2]
= 1 + (1 + (1 + 0))                [por longitud.1]
= 3
```

4 / 37

## Propiedad de intercambia

- ▶ Programa:

---

```
intercambia :: (a,b) -> (b,a)
intercambia (x,y) = (y,x)      -- intercambia
```

---

- ▶ Propiedad:  $\text{intercambia} (\text{intercambia} (x,y)) = (x,y)$ .

- ▶ Demostración:

```
intercambia (intercambia (x,y))
= intercambia (y,x)           [por intercambia]
= (x,y)                       [por intercambia]
```

## Propiedad de intercambia

### Comprobación con QuickCheck

- ▶ Propiedad:

---

```
prop_intercambia :: Eq a => a -> a -> Bool
prop_intercambia x y =
    intercambia (intercambia (x,y)) == (x,y)
```

---

- ▶ Comprobación:

```
*Main> quickCheck prop_intercambia
+++ OK, passed 100 tests.
```

## Inversa de listas unitarias

- ▶ Inversa de una lista:

---

```

inversa :: [a] -> [a]
inversa []      = []           -- inversa.1
inversa (x:xs) = inversa xs ++ [x] -- inversa.2

```

---

- ▶ Prop.:  $\text{inversa } [x] = [x]$ 

```

inversa [x]
= inversa (x:[])           [notación de lista]
= (inversa []) ++ [x]     [inversa.2]
= [] ++ [x]               [inversa.1]
= [x]                     [def. de ++]

```

## Inversa de listas unitarias

### Comprobación con QuickCheck

- ▶ Propiedad:

---

```

prop_inversa_unitaria :: (Eq a) => a -> Bool
prop_inversa_unitaria x =
  inversa [x] == [x]

```

---

- ▶ Comprobación:

```

*Main> quickCheck prop_inversa_unitaria
+++ OK, passed 100 tests.

```

## Razonamiento ecuacional con análisis de casos

- ▶ Negación lógica:

---

```
Prelude
```

---

```
not :: Bool -> Bool
not False = True
not True  = False
```

---

- ▶ Prop.:  $\text{not} (\text{not } x) = x$

- ▶ Demostración por casos:

- ▶ Caso 1:  $x = \text{True}$ :
 

$\text{not} (\text{not True})$	$= \text{not False}$	[not.2]
	$= \text{True}$	[not.1]
- ▶ Caso 2:  $x = \text{False}$ :
 

$\text{not} (\text{not False})$	$= \text{not True}$	[not.1]
	$= \text{False}$	[not.2]

## Razonamiento ecuacional con análisis de casos

### Comprobación con QuickCheck

- ▶ Propiedad:

---

```
prop_doble_negacion :: Bool -> Bool
prop_doble_negacion x =
  not (not x) == x
```

---

- ▶ Comprobación:

```
*Main> quickCheck prop_doble_negacion
+++ OK, passed 100 tests.
```

## Tema 7: Razonamiento sobre programas

1. Razonamiento ecuacional
2. Razonamiento por inducción sobre los naturales
  - Esquema de inducción sobre los naturales
  - Ejemplo de inducción sobre los naturales
3. Razonamiento por inducción sobre listas
4. Equivalencia de funciones

## Esquema de inducción sobre los números naturales

Para demostrar que todos los números naturales tienen una propiedad  $P$  basta probar:

1. Caso base  $n=0$ :  
 $P(0)$ .
2. Caso inductivo  $n=(m+1)$ :  
Suponiendo  $P(m)$  demostrar  $P(m+1)$ .

En el caso inductivo, la propiedad  $P(n)$  se llama la hipótesis de inducción.

## Ejemplo de inducción sobre los naturales: Propiedad

- ▶ `(replicate n x)` es la lista formada por  $n$  elementos iguales a  $x$ .

Por ejemplo,

`replicate 3 5`  $\rightsquigarrow$  `[5,5,5]`

---

```
Prelude
replicate :: Int -> a -> [a]
replicate 0 _ = []
replicate (n+1) x = x : replicate n x
```

---

- ▶ Prop.: `length (replicate n xs) = n`

## Ejemplo de inducción sobre los naturales: Demostración

- ▶ Caso base ( $n=0$ ):

$$\begin{aligned} \text{length (replicate 0 xs)} & \\ = \text{length []} & \quad \text{[por replicate.1]} \\ = 0 & \quad \text{[por def. length]} \end{aligned}$$

- ▶ Caso inductivo ( $n=m+1$ ):

$$\begin{aligned} \text{length (replicate (m+1) xs)} & \\ = \text{length (x:(replicate m xs))} & \quad \text{[por replicate.2]} \\ = 1 + \text{length (replicate m xs)} & \quad \text{[por def. length]} \\ = 1 + m & \quad \text{[por hip. ind.]} \\ = m + 1 & \quad \text{[por conmutativa de +]} \end{aligned}$$

## Ejemplo de inducción sobre los naturales: Verificación

Verificación con QuickCheck:

- ▶ Especificación de la propiedad:

---

```
prop_length_replicate :: Int -> Int -> Bool
prop_length_replicate n xs =
  length (replicate m xs) == m
  where m = abs n
```

---

- ▶ Comprobación de la propiedad:

```
*Main> quickCheck prop_length_replicate
OK, passed 100 tests.
```

## Tema 7: Razonamiento sobre programas

1. Razonamiento ecuacional
2. Razonamiento por inducción sobre los naturales
3. Razonamiento por inducción sobre listas
  - Esquema de inducción sobre listas
  - Asociatividad de ++
  - [] es la identidad para ++ por la derecha
  - Relación entre length y ++
  - Relación entre take y drop
  - La concatenación de listas vacías es vacía
4. Equivalencia de funciones



## Esquema de inducción sobre listas

Para demostrar que todas las listas finitas tienen una propiedad  $P$  basta probar:

1. Caso base  $xs=[]$ :  
 $P([])$ .
2. Caso inductivo  $xs=(y:ys)$ :  
Suponiendo  $P(ys)$  demostrar  $P(y:ys)$ .

En el caso inductivo, la propiedad  $P(ys)$  se llama la hipótesis de inducción.

## Asociatividad de ++

- ▶ Programa:

```

_____ Prelude _____
(++ ) :: [a] -> [a] -> [a]
[]      ++ ys = ys          -- ++.1
(x:xs) ++ ys = x : (xs ++ ys) -- ++.2

```

- ▶ Propiedad:  $xs++(ys++zs)=(xs++ys)++zs$
- ▶ Comprobación con QuickCheck:

```

prop_asociativa_conc :: [Int] -> [Int] -> [Int] -> Bool
prop_asociativa_conc xs ys zs =
  xs++(ys++zs)==(xs++ys)++zs

```

```

Main> quickCheck prop_asociatividad_conc
OK, passed 100 tests.

```

## Asociatividad de ++

- ▶ Demostración por inducción en xs:

- ▶ Caso base  $xs=[]$ : Reduciendo el lado izquierdo

$$\begin{aligned} & xs++(ys++zs) \\ &= []++(ys++zs) && \text{[por hipótesis]} \\ &= ys++zs && \text{[por ++.1]} \end{aligned}$$

y reduciendo el lado derecho

$$\begin{aligned} & (xs++ys)++zs \\ &= ([]++ys)++zs && \text{[por hipótesis]} \\ &= ys++zs && \text{[por ++.1]} \end{aligned}$$

Luego,  $xs++(ys++zs)=(xs++ys)++zs$

## Asociatividad de ++

- ▶ Demostración por inducción en xs:

- ▶ Caso inductivo  $xs=a:as$ : Suponiendo la hipótesis de inducción

$$as++(ys++zs)=(as++ys)++zs$$

hay que demostrar que

$$\begin{aligned} & (a:as)++(ys++zs)=((a:as)++ys)++zs \\ & (a:as)++(ys++zs) \\ &= a:(as++(ys++zs)) && \text{[por ++.2]} \\ &= a:((as++ys)++zs) && \text{[por hip. ind.]} \\ &= (a:(as++ys))++zs && \text{[por ++.2]} \\ &= ((a:as)++ys)++zs && \text{[por ++.2]} \end{aligned}$$

## [] es la identidad para ++ por la derecha

- ▶ Propiedad:  $xs++[] = xs$
- ▶ Comprobación con QuickCheck:

```
prop_identidad_concatenacion :: [Int] -> Bool  
prop_identidad_concatenacion xs = xs++[] == xs
```

```
Main> quickCheck prop_identidad_concatenacion  
OK, passed 100 tests.
```

## [] es la identidad para ++ por la derecha

- ▶ Demostración por inducción en  $xs$ :
  - ▶ Caso base  $xs=[]$ :  
 $= []++[]$   
 $= []$  [por ++.1]
  - ▶ Caso inductivo  $xs=(a:as)$ : Suponiendo la hipótesis de inducción  
 $as++[] = as$   
hay que demostrar que  
 $(a:as)++[] = (a:as)$   
 $(a:as)++[]$   
 $= a:(as++[])$  [por ++.2]  
 $= a:as$  [por hip. ind.]

## Relación entre length y ++

► Programas:

---

```
length :: [a] -> Int
length []      = 0                -- length.1
length (x:xs) = 1 + n_length xs  -- length.2
```

```
(++) :: [a] -> [a] -> [a]
[]    ++ ys = ys                -- ++.1
(x:xs) ++ ys = x : (xs ++ ys)  -- ++.2
```

---

► Propiedad:  $\text{length}(xs++ys) = (\text{length } xs) + (\text{length } ys)$

## Relación entre length y ++

► Comprobación con QuickCheck:

---

```
prop_length_append :: [Int] -> [Int] -> Bool
prop_length_append xs ys = length(xs++ys) == (length xs) + (length ys)
```

---

```
|Main> quickCheck prop_length_append
|OK, passed 100 tests.
```

► Demostración por inducción en xs:

- Caso base  $xs=[]$ :
  - $\text{length}([]++ys)$
  - $= \text{length } ys$  [por ++.1]
  - $= 0 + (\text{length } ys)$  [por aritmética]
  - $= (\text{length } []) + (\text{length } ys)$  [por length.1]

## Relación entre length y ++

► Demostración por inducción en xs:

► Caso inductivo  $xs=(a:as)$ : Suponiendo la hipótesis de inducción

$$\text{length}(as++ys) = (\text{length } as) + (\text{length } ys)$$

hay que demostrar que

$$\text{length}((a:as)++ys) = (\text{length } (a:as)) + (\text{length } ys)$$

$$\begin{aligned} & \text{length}((a:as)++ys) \\ &= \text{length}(a:(as++ys)) && \text{[por ++.2]} \\ &= 1 + \text{length}(as++ys) && \text{[por length.2]} \\ &= 1 + ((\text{length } as) + (\text{length } ys)) && \text{[por hip. ind.]} \\ &= (1 + (\text{length } as)) + (\text{length } ys) && \text{[por aritmética]} \\ &= (\text{length } (a:as)) + (\text{length } ys) && \text{[por length.2]} \end{aligned}$$

## Relación entre take y drop

► Programas:

---

```
take :: Int -> [a] -> [a]
take 0 _      = []           -- take.1
take _ []    = []           -- take.2
take n (x:xs) = x : take (n-1) xs -- take.3
```

```
drop :: Int -> [a] -> [a]
drop 0 xs    = xs           -- drop.1
drop _ []    = []           -- drop.2
drop n (_:xs) = drop (n-1) xs -- drop.3
```

```
(++) :: [a] -> [a] -> [a]
[]    ++ ys = ys           -- ++.1
(x:xs) ++ ys = x : (xs ++ ys) -- ++.2
```

---

## Relación entre take y drop

- ▶ Propiedad: `take n xs ++ drop n xs = xs`
- ▶ Comprobación con QuickCheck:

```
prop_take_drop :: Int -> [Int] -> Property
prop_take_drop n xs =
  n >= 0 ==> take n xs ++ drop n xs == xs
```

```
|Main> quickCheck prop_take_drop
|OK, passed 100 tests.
```

## Relación entre take y drop

- ▶ Demostración por inducción en  $n$ :
  - ▶ Caso base  $n=0$ :
 
$$\begin{aligned} & \text{take } 0 \text{ xs ++ drop } 0 \text{ xs} \\ &= [] \text{ ++ xs} && \text{[por take.1 y drop.1]} \\ &= \text{xs} && \text{[por ++.1]} \end{aligned}$$
  - ▶ Caso inductivo  $n=m+1$ : Suponiendo la hipótesis de inducción 1
 
$$(\forall \text{xs} :: [a]) \text{take } m \text{ xs ++ drop } m \text{ xs} = \text{xs}$$
 hay que demostrar que
 
$$(\forall \text{xs} :: [a]) \text{take } (m+1) \text{ xs ++ drop } (m+1) \text{ xs} = \text{xs}$$

## Relación entre take y drop

Lo demostraremos por inducción en xs:

- ▶ Caso base  $xs = []$ :

```
take (m+1) [] ++ drop (m+1) []
= [] ++ []                               [por take.2 y drop.2]
= []                                     [por ++.1]
```

- ▶ Caso inductivo  $xs = (a:as)$ : Suponiendo la hip. de inducción 2

$take (m+1) as ++ drop (m+1) as = as$

hay que demostrar que

```
take (m+1) (a:as) ++ drop (m+1) (a:as) = (a:as)
take (m+1) (a:as) ++ drop (m+1) (a:as)
= (a:(take m as)) ++ (drop m as)         [take.3 y drop.3]
= (a:((take m as) ++ (drop m as)))      [por ++.2]
= a:as                                   [por hip. de ind. 1]
```

## La concatenación de listas vacías es vacía

- ▶ Programas:

```
----- Prelude -----
null :: [a] -> Bool
null []           = True           -- null.1
null (_:_)       = False          -- null.2

(++) :: [a] -> [a] -> [a]
[] ++ ys         = ys              -- (++).1
(x:xs) ++ ys     = x : (xs ++ ys) -- (++).2
```

- ▶ Propiedad:  $null\ xs = null\ (xs\ ++\ xs)$ .

## La concatenación de listas vacías es vacía

► Demostración por inducción en  $xs$ :

► Caso 1:  $xs = []$ : Reduciendo el lado izquierdo

$$\begin{aligned} & \text{null } xs \\ &= \text{null } [] && \text{[por hipótesis]} \\ &= \text{True} && \text{[por null.1]} \end{aligned}$$

y reduciendo el lado derecho

$$\begin{aligned} & \text{null } (xs ++ xs) \\ &= \text{null } ([] ++ []) && \text{[por hipótesis]} \\ &= \text{null } [] && \text{[por (++).1]} \\ &= \text{True} && \text{[por null.1]} \end{aligned}$$

Luego,  $\text{null } xs = \text{null } (xs ++ xs)$ .

## La concatenación de listas vacías es vacía

► Demostración por inducción en  $xs$ :

► Caso  $xs = (y:ys)$ : Reduciendo el lado izquierdo

$$\begin{aligned} & \text{null } xs \\ &= \text{null } (y:ys) && \text{[por hipótesis]} \\ &= \text{False} && \text{[por null.2]} \end{aligned}$$

y reduciendo el lado derecho

$$\begin{aligned} & \text{null } (xs ++ xs) \\ &= \text{null } ((y:ys) ++ (y:ys)) && \text{[por hipótesis]} \\ &= \text{null } (y:(ys ++ (y:ys))) && \text{[por (++).2]} \\ &= \text{False} && \text{[por null.2]} \end{aligned}$$

Luego,  $\text{null } xs = \text{null } (xs ++ xs)$ .



## Tema 7: Razonamiento sobre programas

1. Razonamiento ecuacional
2. Razonamiento por inducción sobre los naturales
3. Razonamiento por inducción sobre listas
4. Equivalencia de funciones

## Equivalencia de funciones

- ▶ Programas:

---

```

inversa1, inversa2 :: [a] -> [a]
inversa1 []        = []
inversa1 (x:xs)    = inversa1 xs ++ [x]

inversa2 xs = inversa2Aux xs []
  where inversa2Aux []      ys = ys
        inversa2Aux (x:xs) ys = inversa2Aux xs (x:ys)

```

---

- ▶ Propiedad: `inversa1 xs = inversa2 xs`
- ▶ Comprobación con QuickCheck:

---

```

prop_equiv_inversa :: [Int] -> Bool
prop_equiv_inversa xs = inversa1 xs == inversa2 xs

```

---

## Equivalencia de funciones

- Demostración: Es consecuencia del siguiente lema:

$$\text{inversa1 } xs \text{ ++ } ys = \text{inversa2Aux } xs \text{ } ys$$

En efecto,

$$\begin{aligned} & \text{inversa1 } xs \\ &= \text{inversa1 } xs \text{ ++ } [] && \text{[por identidad de ++]} \\ &= \text{inversa2Aux } xs \text{ ++ } [] && \text{[por el lema]} \\ &= \text{inversa2 } xs && \text{[por el inversa2.1]} \end{aligned}$$

## Equivalencia de funciones

- Demostración del lema: Por inducción en  $xs$ :

- Caso base  $xs=[]$ :

$$\begin{aligned} & \text{inversa1 } [] \text{ ++ } ys \\ &= [] \text{ ++ } ys && \text{[por inversa1.1]} \\ &= ys && \text{[por ++.1]} \\ &= \text{inversa2Aux } [] \text{ ++ } ys && \text{[por inversa2Aux.1]} \end{aligned}$$

- Caso inductivo  $xs=(a:as)$ : La hipótesis de inducción es

$$(\forall ys :: [a]) \text{inversa1 } as \text{ ++ } ys = \text{inversa2Aux } as \text{ } ys$$

Por tanto,

$$\begin{aligned} & \text{inversa1 } (a:as) \text{ ++ } ys \\ &= (\text{inversa1 } as \text{ ++ } [a]) \text{ ++ } ys && \text{[por inversa1.2]} \\ &= (\text{inversa1 } as) \text{ ++ } ([a] \text{ ++ } ys) && \text{[por asociativa de ++]} \\ &= (\text{inversa1 } as) \text{ ++ } (a:ys) && \text{[por ley unitaria]} \\ &= (\text{inversa2Aux } as \text{ } (a:ys)) && \text{[por hip. de inducción]} \\ &= (\text{inversa2Aux } (a:as) \text{ } ys) && \text{[por inversa2Aux.2]} \end{aligned}$$

## Bibliografía

1. H. C. Cunningham (2007) *Notes on Functional Programming with Haskell*.
2. J. Fokker (1996) *Programación funcional*.
3. G. Hutton *Programming in Haskell*. Cambridge University Press, 2007.
  - ▶ Cap. 13: Reasoning about programs.
4. B.C. Ruiz, F. Gutiérrez, P. Guerrero y J.E. Gallardo. *Razonando con Haskell*. Thompson, 2004.
  - ▶ Cap. 6: Programación con listas.
5. S. Thompson. *Haskell: The Craft of Functional Programming*, Second Edition. Addison-Wesley, 1999.
  - ▶ Cap. 8: Reasoning about programs.
6. E.P. Wentworth (1994) *Introduction to Funcional Programming*.