

# Tema 13

De EjerciciosLMF2014

```
header {* Tema 13: Razonamiento sobre programas en Isabelle *}

theory T13
imports Main
begin

text {*
  En este tema se demuestra con Isabelle las propiedades de los
  programas funcionales como se expone en el tema 8 del curso
  "Informática" que puede leerse en
  http://www.cs.us.es/~jalonso/cursos/ilm/temas/tema-8t.pdf
*}

section {* Razonamiento ecuacional *}

text {* -----
  Ejercicio 1. Definir, por recursión, la función
  longitud :: "'a list ⇒ nat" where
  tal que (longitud xs) es la longitud de la listas xs. Por ejemplo,
  longitud [4,2,5] = 3
  ----- *}

fun longitud :: "'a list ⇒ nat" where
  "longitud [] = 0"
| "longitud (x#xs) = 1 + longitud xs"

value "longitud [4,2,5]" -- "= 3"

text {* -----
  Ejercicio 2. Demostrar que
  longitud [4,2,5] = 3
  ----- *}

lemma "longitud [4,2,5] = 3"
by simp

text {* -----
  Ejercicio 3. Definir la función
  fun intercambia :: "'a × 'b ⇒ 'b × 'a"
  tal que (intercambia p) es el par obtenido intercambiando las
  componentes del par p. Por ejemplo,
  "intercambia (2,3) = (3,2)"
  ----- *}

fun intercambia :: "'a × 'b ⇒ 'b × 'a" where
  "intercambia (x,y) = (y,x)"

value "intercambia (2,3)" -- "= (3,2)"

text {* -----
  Ejercicio 4. Demostrar que
  intercambia (intercambia (x,y)) = (x,y)
  ----- *}

lemma "intercambia (intercambia (x,y)) = (x,y)"
by simp

text {* -----
  Ejercicio 5. Definir, por recursión, la función
  inversa :: "'a list ⇒ 'a list"
  tal que (inversa xs) es la lista obtenida invirtiendo el orden de los
  elementos de xs. Por ejemplo,
  ----- *}

lemma "inversa [x] = [x]"
by simp

text {* -----
  Ejercicio 6. Definir la función
  ----- *}


```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE

LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS

CALL OR WHATSAPP:689 45 44 70

www.Cartagena99.com no se hace responsable de la información contenida en el presente documento en virtud al

Artículo 17,1 de la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico, de 11 de julio de 2002.

Si la información contenida en el documento es ilícita o lesiona bienes o derechos de un tercero háganoslo saber y será retirada.

Ejercicio 7. Definir la función

```

repite :: "nat => 'a => 'a list" where
tal que (repite n x) es la lista obtenida repitiendo n veces el
elemento x. Por ejemplo,
  repite 3 5 = [5,5,5]
----- *}

```

```

fun repite :: "nat => 'a => 'a list" where
  "repite 0 x = []"
| "repite (Suc n) x = x # (repite n x)"

```

```

value "repite 3 5" -- "= [5,5,5]"

```

```

text {* -----
Ejercicio 8. Demostrar que
  longitud (repite n x) = n
----- *}

```

```

lemma "longitud (repite n x) = n"
by (induct n) auto

```

```

lemma longitud_repite:
  "longitud (repite n x) = n"
proof (induct n)
  show "longitud (repite 0 x) = 0" by simp
next
  fix n
  assume hi: "longitud (repite n x) = n"
  show "longitud (repite (Suc n) x) = Suc n"
  proof -
    have "longitud (repite (Suc n) x) = longitud (x # (repite n x))" by simp
    also have "... = 1 + longitud (repite n x)" by simp
    also have "... = 1 + n" using hi by simp
    also have "... = Suc n" by simp
    finally show ?thesis .
  qed
qed

```

```

text {* -----
Ejercicio 9. Definir la función
  fun conc :: "'a list => 'a list => 'a list"
tal que (conc xs ys) es la concatenación de las listas xs e ys. Por
ejemplo,
  conc [2,3] [4,3,5] = [2,3,4,3,5]
----- *}

```

```

fun conc :: "'a list => 'a list => 'a list" where
  "conc [] ys = ys"
| "conc (x#xs) ys = x # (conc xs ys)"

```

```

value "conc [2,3] [4,3,5]" -- "= [2,3,4,3,5]"

```

```

text {* -----
Ejercicio 10. Demostrar que
  conc xs (conc ys zs) = (conc xs ys) zs
----- *}

```

```

lemma "conc xs (conc ys zs) = conc (conc xs ys) zs"
by (induct xs) auto

```

```

lemma conc_asociativa:
  "conc xs (conc ys zs) = conc (conc xs ys) zs"
proof (induct xs)
  show "conc [] (conc ys zs) = conc (conc [] ys) zs" by simp
next
  fix x xs
  assume hi: "conc xs (conc ys zs) = conc (conc xs ys) zs"
  show "conc (x # xs) (conc ys zs) = conc (conc (x # xs) ys) zs"
  proof -
    have "conc (x # xs) (conc ys zs) = x # (conc xs (conc ys zs))" by simp

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE

LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS

CALL OR WHATSAPP:689 45 44 70

```

lemma "conc xs ys = conc ys xs"

```

```

quickcheck

```

www.cartagena99.com no se hace responsable de la información contenida en el presente documento en virtud al

Artículo 17.1 de la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico, de 11 de julio de 2002.

Si la información contenida en el documento es ilícita o lesiona bienes o derechos de un tercero háganoslo saber y será retirada.

```

text {* Encuentra el contraejemplo.
  xs = [a2]

```

```

ys = [a1] *}

text {* -----
Ejercicio 12. Demostrar que
  conc xs [] = xs
----- *}

lemma "conc xs [] = xs"
by (induct xs) auto

text {* -----
Ejercicio 13. Demostrar que
  longitud (conc xs ys) = longitud xs + longitud ys
----- *}

lemma "longitud (conc xs ys) = longitud xs + longitud ys"
by (induct xs) auto

lemma long_conc:
  "longitud (conc xs ys) = longitud xs + longitud ys"
proof (induct xs)
  show "longitud (conc [] ys) = longitud [] + longitud ys" by simp
next
  fix x xs
  assume hi: "longitud (conc xs ys) = longitud xs + longitud ys"
  show "longitud (conc (x # xs) ys) = longitud (x # xs) + longitud ys"
  proof -
    have "longitud (conc (x # xs) ys) = longitud (x # (conc xs ys))" by simp
    also have "... = 1 + longitud (conc xs ys)" by simp
    also have "... = 1 + (longitud xs + longitud ys)" using hi by simp
    also have "... = (1 + longitud xs) + longitud ys" by simp
    also have "... = longitud (x # xs) + longitud ys" by simp
    finally show ?thesis .
  qed
qed

text {* -----
Ejercicio 14. Definir la función
  coge :: "nat ⇒ 'a list ⇒ 'a list"
tal que (coge n xs) es la lista de los n primeros elementos de xs. Por
ejemplo,
  coge 2 [3,7,5,4] = [3,7]
----- *}

fun coge :: "nat ⇒ 'a list ⇒ 'a list" where
  "coge n []          = []"
| "coge 0 xs         = []"
| "coge (Suc n) (x#xs) = x # (coge n xs)"

value "coge 2 [3,7,5,4]" -- "= [3,7]"

text {* -----
Ejercicio 15. Definir la función
  elimina :: "nat ⇒ 'a list ⇒ 'a list"
tal que (elimina n xs) es la lista obtenida eliminando los n primeros
elementos de xs. Por ejemplo,
  elimina 2 [3,7,5,4] = [5,4]
----- *}

fun elimina :: "nat ⇒ 'a list ⇒ 'a list" where
  "elimina n []          = []"
| "elimina 0 xs         = xs"
| "elimina (Suc n) (x#xs) = elimina n xs"

value "elimina 2 [3,7,5,4]" -- "= [5,4]"

text {* -----
Ejercicio 16. Demostrar que
  conc (coge n xs) (elimina n xs) = xs
----- *}

lemma conc_coge_elimina:
  "conc (coge n xs) (elimina n xs) = xs"
proof (induct n)
  show "conc (coge 0 xs) (elimina 0 xs) = xs" by simp
  next
    show "∀v va. conc (coge 0 (v # va)) (elimina 0 (v # va)) = v # va" by simp

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE

LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS

CALL OR WHATSAPP:689 45 44 70

Cartagena99

www.cartagena99.com no se hace responsable de la información contenida en el presente documento en virtud al artículo 1.º de la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico, de 11 de julio de 2002. Si la información contenida en el documento es ilícita o lesiona bienes o derechos de un tercero háganoslo saber y será retirada.

```

next
fix x xs n
assume hi: "conc (coge n xs) (elimina n xs) = xs"
show "conc (coge (Suc n) (x # xs)) (elimina (Suc n) (x # xs)) = x # xs"
proof -
  have "conc (coge (Suc n) (x # xs)) (elimina (Suc n) (x # xs)) = conc (coge (Suc n) (x # xs)) (elimina n xs)" by simp
  also have "... = conc (x# (coge n xs)) (elimina n xs)" by simp
  also have "... = x#(conc (coge n xs) (elimina n xs))" by simp
  also have "... = (x#xs)" using hi by simp
  finally show ?thesis .
qed
qed

```

```

text {* -----
Ejercicio 17. Definir la función
  esVacía :: "'a list ⇒ bool"
tal que (esVacía xs) se verifica si xs es la lista vacía. Por ejemplo,
  esVacía [] = True
  esVacía [1] = False
----- *}

```

```

fun esVacía :: "'a list ⇒ bool" where
  "esVacía [] = True"
| "esVacía (x#xs) = False"

```

```

value "esVacía []" -- "= True"
value "esVacía [1]" -- "= False"

```

```

text {* -----
Ejercicio 18. Demostrar que
  esVacía xs = esVacía (conc xs xs)
----- *}

```

```

lemma "esVacía xs = esVacía (conc xs xs)"
by (induct xs) auto

```

```

lemma vacía_conc:
  "esVacía xs = esVacía (conc xs xs)"
proof (induct xs)
  show "esVacía [] = esVacía (conc [] [])" by simp
  next
    fix x xs
    assume hi: "esVacía xs = esVacía (conc xs xs)"
    show "esVacía (x # xs) = esVacía (conc (x # xs) (x # xs))"
    proof -
      have "esVacía (conc (x # xs) (x # xs)) = esVacía (x# (conc xs (x#xs)))" by simp
      also have "... = esVacía (x # xs)" by simp
      finally show ?thesis by simp
    qed
  qed

```

```

lemma vacía_conc':
  "esVacía xs = esVacía (conc xs xs)"
proof (cases xs)
  case Nil thus ?thesis by simp
next
  case Cons thus ?thesis by simp
qed

```

```

lemma vacía_conc'':
  "esVacía xs = esVacía (conc xs xs)"
by (cases xs) simp_all

```

```

text {* -----
Ejercicio 19. Definir la función
  inversaAc :: "'a list ⇒ 'a list"

```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

```

value inversaAc [3,4,5] -- - [5,4,3]"

```

```

text {* -----
www.cartagena99.com no se hace responsable de la información contenida en el presente documento en virtud al
Artículo 17 de la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico, de 11 de julio de 2002.
Si la información contenida en el documento es ilícita o lesiona bienes o derechos de un tercero háganoslo saber y será retirada.
----- *}

```

```

lemma inversaAcAux_es_inversa:
  "inversaAcAux xs ys = (inversa xs)@ys"
by (induct xs arbitrary: ys) auto

lemma inversaAcAux_es_inversa_b:
  "inversaAcAux xs ys = (inversa xs)@ys"
proof (induct xs arbitrary: ys)
  show " $\wedge$ ys. inversaAcAux [] ys = inversa [] @ ys" by simp
next
  fix x xs zs
  assume hi: " $\wedge$ ys. inversaAcAux xs ys = inversa xs @ ys"
  show "inversaAcAux (x#xs) zs = inversa (x#xs) @ zs"
proof -
  have "inversaAcAux (x#xs) zs = inversaAcAux xs (x#zs)" by simp
  also have "... = inversa xs @ (x#zs)" using hi by simp
  also have "... = inversa xs @ [x] @ zs" by simp
  also have "... = inversa (x#xs) @ zs" by simp
  finally show ?thesis .
qed
qed

text {* -----
Ejercicio 21. Demostrar que
  inversaAc xs = inversa xs
----- *}

corollary "inversaAc xs = inversa xs"
by (simp add: inversaAcAux_es_inversa)

text {* -----
Ejercicio 22. Definir la función
  sum :: "int list  $\Rightarrow$  int"
tal que (sum xs) es la suma de los elementos de xs. Por ejemplo,
  sum [3,2,5] = 10
----- *}

fun sum :: "int list  $\Rightarrow$  int" where
  "sum [] = 0"
| "sum (x#xs) = x + sum xs"

value "sum [3,2,5]" -- "= 10"

text {* -----
Ejercicio 23. Definir la función
  map :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a list  $\Rightarrow$  'b list
tal que (map f xs) es la lista obtenida aplicando la función f a los
elementos de xs. Por ejemplo,
  map ( $\lambda$ x. 2*x) [3,2,5] = [6,4,10]
----- *}

fun map :: "('a  $\Rightarrow$  'b)  $\Rightarrow$  'a list  $\Rightarrow$  'b list" where
  "map f [] = []"
| "map f (x#xs) = (f x) # map f xs"

value "map ( $\lambda$ x. 2*x) [3::int,2,5]" -- "= [6,4,10]"
value "map ( $\lambda$ x. 6*x) [3::int,2,5]" -- "= [18,12,30]"

text {* -----
Ejercicio 24. Demostrar que
  sum (map ( $\lambda$ x. 2*x) xs) = 2 * (sum xs)
----- *}

lemma "sum (map ( $\lambda$ x. 2*x) xs) = 2 * (sum xs)"
by (induct xs) auto

lemma sum_map:
  "sum (map ( $\lambda$ x. 2*x) xs) = 2 * (sum xs)"

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE

LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS

CALL OR WHATSAPP:689 45 44 70

finally show thesis .

qed

qed

www.cartagena99.com no se hace responsable de la información contenida en el presente documento en virtud al

Artículo 17.1 de la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico, de 11 de julio de 2002.

Si la información contenida en el documento es ilícita o lesiona bienes o derechos de un tercero háganoslo saber y será retirada.

Ejercicio 25. Demostrar que

Cartagena99

```

longitud (map f xs) = longitud xs
----- *}

lemma "longitud (map f xs) = longitud xs"
by (induct xs) auto

lemma long_map:
  "longitud (map f xs) = longitud xs"
proof (induct xs)
  show "longitud (map f []) = longitud []" by simp
next
  fix x xs
  assume hi: "longitud (map f xs) = longitud xs"
  show "longitud (map f (x#xs)) = longitud (x#xs)"
  proof -
    have "longitud (map f (x#xs)) = longitud ((f x)#(map f xs))" by simp
    also have "... = 1 + longitud (map f xs)" by simp
    also have "... = 1 + longitud xs" using hi by simp
    also have "... = longitud (x#xs)" by simp
    finally show ?thesis .
  qed
qed

end

```

Obtenido de "[http://www.glc.us.es/~jalonso/ejerciciosLMF2014/index.php5/Tema\\_13](http://www.glc.us.es/~jalonso/ejerciciosLMF2014/index.php5/Tema_13)"



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE

LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS

CALL OR WHATSAPP:689 45 44 70