Segunda parte (ex.3)

De EjerciciosLMF2014

```
header {* Examen 3 *}

theory ex3_sol

imports Main
begin

text {*
  Reglas básicas de deducción natural de la lógica proposicional,
  de los cuantificadores y de la igualdad:
  · conjI:      ⟦P; Q⟧ ⟹ P ∧ Q
  · conjunct1:  P ∧ Q ⟹ P
  · conjunct2:  P ∧ Q ⟹ Q
  · notnotD:    ¬¬ P ⟹ P
  · mp:         ⟦P ⟶ Q; P⟧ ⟹ Q
  · impI:       (P ⟹ Q) ⟹ P ⟶ Q
  · disjI1:     P ⟹ P ∨ Q
  · disjI2:     Q ⟹ P ∨ Q
  · disjE:      ⟦P ∨ Q; P ⟹ R; Q ⟹ R⟧ ⟹ R
  · FalseE:     False ⟹ P
  · notE:       ⟦¬P; P⟧ ⟹ R
  · notI:       (P ⟹ False) ⟹ ¬P
  · iffI:       ⟦P ⟹ Q; Q ⟹ P⟧ ⟹ P = Q
  · iffD1:      ⟦Q = P; Q⟧ ⟹ P
  · iffD2:      ⟦P = Q; Q⟧ ⟹ P
  · ccontr:     (¬P ⟹ False) ⟹ P

  · allI:       ⟦∀x. P x; P x ⟹ R⟧ ⟹ R
  · allE:       (⋀x. P x) ⟹ ∀x. P x
  · exI:        P x ⟹ ∃x. P x
  · exE:        ⟦∃x. P x; ⋀x. P x ⟹ Q⟧ ⟹ Q

  · refl:       t = t
  · subst:      ⟦s = t; P s⟧ ⟹ P t
  · trans:      ⟦r = s; s = t⟧ ⟹ r = t
  · sym:        s = t ⟹ t = s
  · not_sym:    t ≠ s ⟹ s ≠ t
  · ssubst:     ⟦t = s; P s⟧ ⟹ P t
  · box_equals: ⟦a = b; a = c; b = d⟧ ⟹ a: = d
  · arg_cong:   x = y ⟹ f x = f y
  · fun_cong:   f = g ⟹ f x = g x
  · cong:       ⟦f = g; x = y⟧ ⟹ f x = g y
*}

text {*
  Se usarán las reglas notnotI y mt que demostramos a continuación.
  *}

lemma notnotI: "P ⟹ ¬¬ P"
by auto

lemma mt: "⟦F ⟶ G; ¬G⟧ ⟹ ¬F"
by auto



text {* ---------------------------------------------------------------
  Ejercicio 2. Demostrar
```

```
    ∀x. P x⟶ (∀y. Q y⟶ R x y), ∃x. P x ∧ (∃y. ¬(R x y)) ⊢ ¬(∀x. Q x)
  ------------------------------------------------------------------ *}

lemma ej_2_c:
 assumes "∀x. P x⟶ (∀y. Q y⟶ R x y)"
         "∃x. P x ∧ (∃y. ¬(R x y))"
 shows "¬(∀x. Q x)"
proof
 assume "∀x. Q x"
 obtain a where 1: "P a ∧(∃y. ¬(R a y))" using assms (2) ..
 hence "∃y. ¬(R a y)" by (rule conjunct2)
 then obtain b where "¬(R a b)" ..
 have "P a" using 1 by (rule conjunct1)
 have "P a⟶ (∀y. Q y⟶ R a y)" using assms (1) ..
 hence "∀y. Q y⟶ R a y" using `P a` by (rule mp)
 hence "Q b⟶ R a b" ..
 have "Q b" using `∀x. Q x` ..
 with `Q b⟶ R a b` have "R a b" ..
 with `¬(R a b)` show False ..
 qed

text {*
  -------------------------------------------------------------------
  Ejercicio . Definir la función
     suma :: "nat list ⇒ nat"
  tal que (suma xs) es la suma de los elementos de la lista de números
  naturales xs. Por ejemplo,
     suma [3::nat,2,4] = 9
  -------------------------------------------------------------------
*}

fun suma :: "nat list ⇒ nat" where
  "suma []     = 0"
| "suma (x#xs) = x + suma xs"

value "suma [6::nat,2,4]" -- "= 12"


text {*
  -------------------------------------------------------------------
  Ejercicio. Demostrar o refutar
     suma (xs @ ys) = suma xs + suma ys
  -------------------------------------------------------------------
*}

lemma suma_append:
  "suma (xs @ ys) = suma xs + suma ys"
proof (induct xs)
  show "suma ([] @ ys) = suma [] + suma ys" by simp
next
  fix a xs
  assume HI: "suma (xs @ ys) = suma xs + suma ys"
  show "suma ((a#xs) @ ys) = suma (a#xs) + suma ys"
  proof -
    have "suma ((a#xs) @ ys) = suma (a#(xs@ys))" by simp
    also have "… = a + suma (xs@ys)" by simp
    also have "… = a + suma xs + suma ys" using HI by simp
    also have "… = suma (a#xs) + suma ys" by simp
    finally show ?thesis .
  qed
qed


end
```