# Calculemus
# (Vol. 2: Demostraciones con Lean4)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 10 de julio de 2023 (versión del 26 de noviembre de 2023)

# Índice general

# Capítulo 1

# Introducción

Este libro es una recopilación de los ejercicios de demostración con Lean4 que se han ido publicando, desde el 10 de julio de 20023, en el blog Calculemus.

La ordenación de los ejercicios es simplemente temporal según su fecha de publicación en Calculemus y el orden de los ejercicios en Calculemus responde a los que me voy encontrando en mis lecturas.

En cada ejercicio, se comienza proponiendo soluciones en lenguaje natural y, a continuación, se exponen distintas demostraciones con Lean4 ordenadas desde las más detalladas a las más automáticas. Al final de cada ejercicio hay un enlace para interactuar con sus soluciones en Lean4 Web.

Las soluciones del libro están en este repositorio de GitHub.

El libro se irá actualizando periódicamente con los nuevos ejercicios que se proponen diariamente en Calculemus.

Este libro es una continuación de

- DAO (Demostración Asistida por Ordenador) con Lean que es una introducción a la demostración con Lean3 y

- Calculemus (Vol. 1: Demostraciones con Isabelle/HOL y Lean3) que es la recopilación de la primera parte de los ejercicios del blog con demostraciones en Isabelle/HOL y Lean3.

# Capítulo 2

# Demostraciones de una propiedad de los números enteros

## 2.1. ∀ m n ∈ ℕ, Even n → Even (m * n)

```
-- ---------------------------------------------------------------------
-- Demostrar que los productos de los números naturales por números
-- pares son pares.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Si n es par, entonces (por la definición de ‘Even‘) existe un k tal que
--    n = k + k         (1)
-- Por tanto,
--    mn = m(k + k)     (por (1))
--       = mk + mk      (por la propiedad distributiva)
-- Por consiguiente, mn es par.

-- Demostraciones en Lean4
-- =======================

import Mathlib.Data.Nat.Basic
import Mathlib.Data.Nat.Parity
import Mathlib.Tactic

open Nat
```

```lean
-- 1ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  ring

-- 2ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  rw [mul_add]

-- 3ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk, mul_add]

-- 4ª demostración
-- ===============

example : ∀ m n : Nat, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩; use m * k; rw [hk, mul_add]

-- 5ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  exact ⟨m * k, by rw [hk, mul_add]⟩

-- 6ª demostración
-- ===============

example : ∀ m n : Nat, Even n → Even (m * n) :=
fun m n ⟨k, hk⟩ ↦ ⟨m * k, by rw [hk, mul_add]⟩
```

```
-- 7ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  exact mul_add m k k

-- 8ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  intros m n hn
  unfold Even at *
  cases hn with
  | intro k hk =>
    use m * k
    rw [hk, mul_add]

-- 9ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) := by
  intros m n hn
  unfold Even at *
  cases hn with
  | intro k hk =>
    use m * k
    calc m * n
        = m * (k + k)    := by exact congrArg (HMul.hMul m) hk
      _ = m * k + m * k := by exact mul_add m k k

-- 10ª demostración
-- ================

example : ∀ m n : Nat, Even n → Even (m * n) := by
  intros; simp [*, parity_simps]

-- Lemas usados
-- ============

-- #check (mul_add : ∀ a b c : ℕ, a * (b + c) = a * b + a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

# Capítulo 3

# Propiedades elementales de los números reales

## 3.1. En ℝ, (ab)c = b(ac)

```
-- ---------------------------------------------------------------------
-- Demostrar que los números reales tienen la siguiente propiedad
--    (a * b) * c = b * (a * c)
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    (ab)c = (ba)c   [por la conmutativa]
--          = b(ac)   [por la asociativa]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
import Mathlib.Data.Real.Basic

-- 1ª demostración
example
  (a b c : ℝ)
  : (a * b) * c = b * (a * c) :=
calc
  (a * b) * c = (b * a) * c := by rw [mul_comm a b]
            _ = b * (a * c) := by rw [mul_assoc b a c]
```

```
-- 2ª demostración
example (a b c : ℝ) : (a * b) * c = b * (a * c) :=
by
  rw [mul_comm a b]
  rw [mul_assoc b a c]

-- 3ª demostración
example (a b c : ℝ) : (a * b) * c = b * (a * c) :=
by ring

-- Lemas usados
-- ============

-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.2. En ℝ, (cb)a = b(ac)

```
-- ---------------------------------------------------------------------
-- Demostrar que los números reales tienen la siguiente propiedad
--    (c * b) * a = b * (a * c)
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades:
--    (c * b) * a
--    = (b * c) * a    [por la conmutativa]
--    = b * (c * a)    [por la asociativa]
--    = b * (a * c)    [por la conmutativa]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
import Mathlib.Data.Real.Basic

-- 1ª demostración
example
  (a b c : ℝ)
```

```
  : (c * b) * a = b * (a * c) :=
calc
  (c * b) * a
    = (b * c) * a := by rw [mul_comm c b]
  _ = b * (c * a) := by rw [mul_assoc]
  _ = b * (a * c) := by rw [mul_comm c a]

-- 2ª demostración
example
  (a b c : ℝ)
  : (c * b) * a = b * (a * c) :=
by
  rw [mul_comm c b]
  rw [mul_assoc]
  rw [mul_comm c a]

-- 3ª demostración
example
  (a b c : ℝ)
  : (c * b) * a = b * (a * c) :=
by ring

-- Lemas usados
-- ============

-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.3. En ℝ, a(bc) = b(ac)

```
-- ---------------------------------------------------------------------
-- Demostrar que los números reales tienen la siguiente propiedad
--    a * (b * c) = b * (a * c)
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades:
--    a(bc)
--    = (ab)c    [por la asociativa]
```

```
--     = (ba)c    [por la conmutativa]
--     = b(ac)    [por la asociativa]

-- Demostraciones en Lean4
-- =======================

import Mathlib.Tactic
import Mathlib.Data.Real.Basic

-- 1ª demostración
example
  (a b c : ℝ) : a * (b * c) = b * (a * c) :=
calc
  a * (b * c)
    = (a * b) * c := by rw [←mul_assoc]
  _ = (b * a) * c := by rw [mul_comm a b]
  _ = b * (a * c) := by rw [mul_assoc]

-- 2ª demostración
example
  (a b c : ℝ) : a * (b * c) = b * (a * c) :=
by
  rw [←mul_assoc]
  rw [mul_comm a b]
  rw [mul_assoc]

-- 3ª demostración
example
  (a b c : ℝ) : a * (b * c) = b * (a * c) :=
by ring

-- Lemas usados
-- ============

-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.4. En ℝ, si ab = cd y e = f, entonces a(be) = c(df)

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c, d, e y f son números reales tales que
--    a * b = c * d y
--    e = f,
-- entonces
--    a * (b * e) = c * (d * f)
-- ---------------------------------------------------------------------

-- Demostración en leguaje natural
-- ===============================

-- Por la siguiente cadena de igualdades
--    a(be)
--    = a(bf)     [por la segunda hipótesis]
--    = (ab)f     [por la asociativa]
--    = (cd)f     [por la primera hipótesis]
--    = c(df)     [por la asociativa]

-- Demostraciones en Lean4
-- =======================

import Mathlib.Tactic
import Mathlib.Data.Real.Basic

-- 1ª demostración
example
  (a b c d e f : ℝ)
  (h1 : a * b = c * d)
  (h2 : e = f)
  : a * (b * e) = c * (d * f) :=
calc
  a * (b * e)
    = a * (b * f) := by rw [h2]
  _ = (a * b) * f := by rw [←mul_assoc]
  _ = (c * d) * f := by rw [h1]
  _ = c * (d * f) := by rw [mul_assoc]

-- 2ª demostración
example
  (a b c d e f : ℝ)
  (h1 : a * b = c * d)
```

```
  (h2 : e = f)
  : a * (b * e) = c * (d * f) :=
by
  rw [h2]
  rw [←mul_assoc]
  rw [h1]
  rw [mul_assoc]

-- 3ª demostración
example
  (a b c d e f : ℝ)
  (h1 : a * b = c * d)
  (h2 : e = f)
  : a * (b * e) = c * (d * f) :=
by
  simp [*, ←mul_assoc]

-- Lemas usados
-- ============

-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.5. En ℝ, si bc = ef, entonces ((ab)c)d = ((ae)f)d

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c, d, e y f son números reales tales que
--    b * c = e * f
-- entonces
--    ((a * b) * c) * d = ((a * e) * f) * d
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    ((ab)c)d
--    = (a(bc))d    [por la asociativa]
--    = (a(ef))d    [por la hipótesis]
--    = ((ae)f)d    [por la asociativa]

-- Demostraciones con Lean4
```

```
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

-- 1ª demostración
example
  (a b c d e f : ℝ)
  (h : b * c = e * f)
  : ((a * b) * c) * d = ((a * e) * f) * d :=
calc
  ((a * b) * c) * d
    = (a * (b * c)) * d := by rw [mul_assoc a]
  _ = (a * (e * f)) * d := by rw [h]
  _ = ((a * e) * f) * d := by rw [←mul_assoc a]

-- 2ª demostración
example
  (a b c d e f : ℝ)
  (h : b * c = e * f)
  : ((a * b) * c) * d = ((a * e) * f) * d :=
by
  rw [mul_assoc a]
  rw [h]
  rw [←mul_assoc a]

-- 3ª demostración
example
  (a b c d e f : ℝ)
  (h : b * c = e * f)
  : ((a * b) * c) * d = ((a * e) * f) * d :=
by
  rw [mul_assoc a, h, ←mul_assoc a]

-- Lemas usados
-- ============

-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.6. En ℝ, si c = ba-d y d = ab, entonces c = 0

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c y d son números reales tales que
--    c = b * a - d
--    d = a * b
-- entonces
--    c = 0
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    c = ba - d     [por la primera hipótesis]
--      = ab - d     [por la conmutativa]
--      = ab - ab    [por la segunda hipótesis]
--      = 0

-- Demostraciones en Lean4
-- =======================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

-- 1ª demostración
example
  (a b c d : ℝ)
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
calc
  c = b * a - d     := by rw [h1]
  _ = a * b - d     := by rw [mul_comm]
  _ = a * b - a * b := by rw [h2]
  _ = 0             := by rw [sub_self]

-- 2ª demostración
example
  (a b c d : ℝ)
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
by
```

```
  rw [h1]
  rw [mul_comm]
  rw [h2]
  rw [sub_self]

-- 3ª demostración
example
  (a b c d : ℝ)
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
by
  rw [h1, mul_comm, h2, sub_self]

-- Lemas usados
-- ============

-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (sub_self : ∀ (a : ℝ), a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.7. En ℝ, (a+b)(a+b) = aa+2ab+bb

```
-- ---------------------------------------------------------------------
-- Demostrar que si a y b son números reales, entonces
--    (a + b) * (a + b) = a * a + 2 * (a * b) + b * b
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    (a + b)(a + b)
--    = (a + b)a + (a + b)b    [por la distributiva]
--    = aa + ba + (a + b)b     [por la distributiva]
--    = aa + ba + (ab + bb)    [por la distributiva]
--    = aa + ba + ab + bb      [por la asociativa]
--    = aa + (ba + ab) + bb    [por la asociativa]
--    = aa + (ab + ab) + bb    [por la conmutativa]
--    = aa + 2(ab) + bb        [por def. de doble]

-- Demostraciones con Lean4
```

```
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (a b c : ℝ)

-- 1ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
calc
  (a + b) * (a + b)
    = (a + b) * a + (a + b) * b       := by rw [mul_add]
  _ = a * a + b * a + (a + b) * b     := by rw [add_mul]
  _ = a * a + b * a + (a * b + b * b) := by rw [add_mul]
  _ = a * a + b * a + a * b + b * b   := by rw [←add_assoc]
  _ = a * a + (b * a + a * b) + b * b := by rw [add_assoc (a * a)]
  _ = a * a + (a * b + a * b) + b * b := by rw [mul_comm b a]
  _ = a * a + 2 * (a * b) + b * b     := by rw [←two_mul]

-- 2ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
calc
  (a + b) * (a + b)
    = a * a + b * a + (a * b + b * b) := by rw [mul_add, add_mul, add_mul]
  _ = a * a + (b * a + a * b) + b * b := by rw [←add_assoc, add_assoc (a * a)]
  _ = a * a + 2 * (a * b) + b * b     := by rw [mul_comm b a, ←two_mul]

-- 3ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
calc
  (a + b) * (a + b)
    = a * a + b * a + (a * b + b * b) := by ring
  _ = a * a + (b * a + a * b) + b * b := by ring
  _ = a * a + 2 * (a * b) + b * b     := by ring

-- 4ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
by ring

-- 5ª demostración
example :
```

```
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
by
  rw [mul_add]
  rw [add_mul]
  rw [add_mul]
  rw [←add_assoc]
  rw [add_assoc (a * a)]
  rw [mul_comm b a]
  rw [←two_mul]

-- 6ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
by
  rw [mul_add, add_mul, add_mul]
  rw [←add_assoc, add_assoc (a * a)]
  rw [mul_comm b a, ←two_mul]

-- 7ª demostración
example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
by linarith

-- Lemas usados
-- ============

-- #check (add_assoc : ∀ a b c : ℝ, (a + b) + c = a + (b + c))
-- #check (add_mul : ∀ a b c : ℝ, (a + b) * c = a * c + b * c)
-- #check (mul_add : ∀ a b c : ℝ, a * (b + c) = a * b + a * c)
-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (two_mul : ∀ (a : ℝ), 2 * a = a + a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.8. En ℝ, (a+b)(c+d) = ac+ad+bc+bd

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c y d son números reales, entonces
--    (a + b) * (c + d) = a * c + a * d + b * c + b * d
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Por la siguiente cadena de igualdades
--    (a + b)(c + d)
--    = a(c + d) + b(c + d)    [por la distributiva]
--    = ac + ad + b(c + d)     [por la distributiva]
--    = ac + ad + (bc + bd)    [por la distributiva]
--    = ac + ad + bc + bd      [por la asociativa]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (a b c d : ℝ)

-- 1ª demostración
example
  : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
calc
  (a + b) * (c + d)
    = a * (c + d) + b * (c + d)       := by rw [add_mul]
  _ = a * c + a * d + b * (c + d)     := by rw [mul_add]
  _ = a * c + a * d + (b * c + b * d) := by rw [mul_add]
  _ = a * c + a * d + b * c + b * d   := by rw [←add_assoc]

-- 2ª demostración
example
  : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
calc
  (a + b) * (c + d)
    = a * (c + d) + b * (c + d)       := by ring
  _ = a * c + a * d + b * (c + d)     := by ring
  _ = a * c + a * d + (b * c + b * d) := by ring
  _ = a * c + a * d + b * c + b * d   := by ring

-- 3ª demostración
example : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
by ring

-- 4ª demostración
example
  : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
by
   rw [add_mul]
```

```
  rw [mul_add]
  rw [mul_add]
  rw [← add_assoc]

-- 5ª demostración
example : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
by rw [add_mul, mul_add, mul_add, ←add_assoc]

-- Lemas usados
-- ============

-- #check (add_mul : ∀ (a b c : ℝ), (a + b) * c = a * c + b * c)
-- #check (mul_add : ∀ (a b c : ℝ), a * (b + c) = a * b + a * c)
-- #check (add_assoc : ∀ (a b c : ℝ), (a + b) + c = a + (b + c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.9. En ℝ, (a+b)(a-b) = a^2-b^2

```
-- ---------------------------------------------------------------------
-- Demostrar que si a y b son números reales, entonces
--    (a + b) * (a - b) = a^2 - b^2
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades:
--    (a + b)(a - b)
--    = a(a - b) + b(a - b)           [por la distributiva]
--    = (aa - ab) + b(a - b)          [por la distributiva]
--    = (a^2 - ab) + b(a - b)         [por def. de cuadrado]
--    = (a^2 - ab) + (ba - bb)        [por la distributiva]
--    = (a^2 - ab) + (ba - b^2)       [por def. de cuadrado]
--    = (a^2 + -(ab)) + (ba - b^2)    [por def. de resta]
--    = a^2 + (-(ab) + (ba - b^2))    [por la asociativa]
--    = a^2 + (-(ab) + (ba + -b^2))   [por def. de resta]
--    = a^2 + ((-(ab) + ba) + -b^2)   [por la asociativa]
--    = a^2 + ((-(ab) + ab) + -b^2)   [por la conmutativa]
--    = a^2 + (0 + -b^2)              [por def. de opuesto]
--    = (a^2 + 0) + -b^2              [por asociativa]
--    = a^2 + -b^2                    [por def. de cero]
--    = a^2 - b^2                     [por def. de resta]
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (a b : ℝ)

-- 1ª demostración
-- ===============

example : (a + b) * (a - b) = a^2 - b^2 :=
calc
  (a + b) * (a - b)
    = a * (a - b) + b * (a - b)           := by rw [add_mul]
  _ = (a * a - a * b) + b * (a - b)       := by rw [mul_sub]
  _ = (a^2 - a * b) + b * (a - b)         := by rw [← pow_two]
  _ = (a^2 - a * b) + (b * a - b * b)     := by rw [mul_sub]
  _ = (a^2 - a * b) + (b * a - b^2)       := by rw [← pow_two]
  _ = (a^2 + -(a * b)) + (b * a - b^2)    := by ring
  _ = a^2 + (-(a * b) + (b * a - b^2))    := by rw [add_assoc]
  _ = a^2 + (-(a * b) + (b * a + -b^2))   := by ring
  _ = a^2 + ((-(a * b) + b * a) + -b^2)   := by rw [← add_assoc
                                                   (-(a * b)) (b * a) (-b^2)]
  _ = a^2 + ((-(a * b) + a * b) + -b^2)   := by rw [mul_comm]
  _ = a^2 + (0 + -b^2)                    := by rw [neg_add_self (a * b)]
  _ = (a^2 + 0) + -b^2                    := by rw [← add_assoc]
  _ = a^2 + -b^2                          := by rw [add_zero]
  _ = a^2 - b^2                           := by linarith

-- 2ª demostración
-- ===============

example : (a + b) * (a - b) = a^2 - b^2 :=
calc
  (a + b) * (a - b)
    = a * (a - b) + b * (a - b)           := by ring
  _ = (a * a - a * b) + b * (a - b)       := by ring
  _ = (a^2 - a * b) + b * (a - b)         := by ring
  _ = (a^2 - a * b) + (b * a - b * b)     := by ring
  _ = (a^2 - a * b) + (b * a - b^2)       := by ring
  _ = (a^2 + -(a * b)) + (b * a - b^2)    := by ring
  _ = a^2 + (-(a * b) + (b * a - b^2))    := by ring
  _ = a^2 + (-(a * b) + (b * a + -b^2))   := by ring
```

```
  _ = a^2 + ((-(a * b) + b * a) + -b^2) := by ring
  _ = a^2 + ((-(a * b) + a * b) + -b^2) := by ring
  _ = a^2 + (0 + -b^2)                  := by ring
  _ = (a^2 + 0) + -b^2                  := by ring
  _ = a^2 + -b^2                        := by ring
  _ = a^2 - b^2                         := by ring

-- 3ª demostración
-- ===============

example : (a + b) * (a - b) = a^2 - b^2 :=
by ring

-- 4ª demostración
-- ===============

-- El lema anterior es
lemma aux : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
by ring

-- La demostración es
example : (a + b) * (a - b) = a^2 - b^2 :=
by
  rw [sub_eq_add_neg]
  rw [aux]
  rw [mul_neg]
  rw [add_assoc (a * a)]
  rw [mul_comm b a]
  rw [neg_add_self]
  rw [add_zero]
  rw [← pow_two]
  rw [mul_neg]
  rw [← pow_two]
  rw [← sub_eq_add_neg]

-- Lemas usados
-- ============

-- #check (add_assoc : ∀ (a b c : ℝ), (a + b) + c = a + (b + c))
-- #check (add_zero : ∀ (a : ℝ), a + 0 = a)
-- #check (add_mul : ∀ (a b c : ℝ), (a + b) * c = a * c + b * c)
-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (mul_neg : ∀ (a b : ℝ), a * -b = -(a * b))
-- #check (mul_sub : ∀ (a b c : ℝ), a * (b - c) = a * b - a * c)
-- #check (neg_add_self : ∀ (a : ℝ), -a + a = 0)
```

```
-- #check (pow_two : ∀ (a : ℝ), a ^ 2 = a * a)
-- #check (sub_eq_add_neg : ∀ (a b : ℝ), a - b = a + -b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.10. En ℝ, si c = da+b y b = ad, entonces c = 2ad

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c y d son números reales tales que
--    c = d * a + b
--    b = a * d
-- entonces
--    c = 2 * a * d
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    c = da + b     [por la primera hipótesis]
--      = da + ad    [por la segunda hipótesis]
--      = ad + ad    [por la conmutativa]
--      = 2(ad)      [por la def. de doble]
--      = 2ad        [por la asociativa]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (a b c d : ℝ)

-- 1ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
calc
  c = d * a + b     := by rw [h1]
  _ = d * a + a * d := by rw [h2]
```

```
  _ = a * d + a * d := by rw [mul_comm d a]
  _ = 2 * (a * d)   := by rw [← two_mul (a * d)]
  _ = 2 * a * d     := by rw [mul_assoc]

-- 2ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by
  rw [h2] at h1
  clear h2
  rw [mul_comm d a] at h1
  rw [← two_mul (a*d)] at h1
  rw [← mul_assoc 2 a d] at h1
  exact h1

-- 3ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by rw [h1, h2, mul_comm d a, ← two_mul (a * d), mul_assoc]

-- 4ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by
  rw [h1]
  rw [h2]
  ring

-- 5ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by
  rw [h1, h2]
  ring

-- 6ª demostración
example
```

```
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by rw [h1, h2] ; ring

-- 7ª demostración
example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by linarith

-- Lemas usados
-- ============

-- #check (mul_assoc : ∀ (a b c : ℝ), (a * b) * c = a * (b * c))
-- #check (mul_comm : ∀ (a b : ℝ), a * b = b * a)
-- #check (two_mul : ∀ (a : ℝ), 2 * a = a + a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.11. En ℝ, si a+b = c, entonces (a+b)(a+b) = ac+bc

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b y c son números reales tales que
--    a + b = c,
-- entonces
--    (a + b) * (a + b) = a * c + b * c
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    (a + b)(a + b)
--    = (a + b)c        [por la hipótesis]
--    = ac + bc         [por la distributiva]

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (a b c : ℝ)

-- 1ª demostración
example
  (h : a + b = c)
  : (a + b) * (a + b) = a * c + b * c :=
calc
  (a + b) * (a + b)
    = (a + b) * c   := by exact congrArg (HMul.hMul (a + b)) h
  _ = a * c + b * c := by rw [add_mul]

-- 2ª demostración
example
  (h : a + b = c)
  : (a + b) * (a + b) = a * c + b * c :=
by
  nth_rewrite 2 [h]
  rw [add_mul]

-- Lemas usados
-- ============

-- #check (add_mul : ∀ (a b c : ℝ), (a + b) * c = a * c + b * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 3.12. Si x e y son sumas de dos cuadrados, entonces xy también lo es

```
-- ---------------------------------------------------------------------
-- Demostrar que si x e y son sumas de dos cuadrados, entonces xy
-- también lo es.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Puesto que x e y se pueden escribir como la suma de dos cuadrados,
-- existen a, b , c y d tales que
```

```
--    x = a² + b²
--    y = c² + d²
-- Entonces,
--    xy = (ac - bd)² + (ad + bc)²
-- En efecto,
--    xy = (a² + b²)(c² + d²)
--       = a²c² + b²d² + a²d² + b²c²
--       = a²c² - 2acbd + b²d² + a²d² + 2adbc + b²c²
--       = (ac - bd)² + (ad + bc)²
-- Por tanto, xy es la suma de dos cuadrados.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
variable {α : Type _} [CommRing α]
variable {x y : α}

-- (suma_de_cuadrados x) afirma que x se puede escribir como la suma
-- de dos cuadrados.
def suma_de_cuadrados (x : α) :=
  ∃ a b, x = a^2 + b^2

-- 1ª demostración
example
  (hx : suma_de_cuadrados x)
  (hy : suma_de_cuadrados y)
  : suma_de_cuadrados (x * y) :=
by
  rcases hx with ⟨a, b, xeq : x = a^2 + b^2⟩
  -- a b : α
  -- xeq : x = a ^ 2 + b ^ 2
  rcases hy with ⟨c, d, yeq : y = c^2 + d^2⟩
  -- c d : α
  -- yeq : y = c ^ 2 + d ^ 2
  have h1: x * y = (a*c - b*d)^2 + (a*d + b*c)^2 :=
    calc x * y
         = (a^2 + b^2) * (c^2 + d^2) :=
                by rw [xeq, yeq]
       _ = a^2*c^2 + b^2*d^2 + a^2*d^2 + b^2*c^2 :=
                by ring
       _ = a^2*c^2 - 2*a*c*b*d + b^2*d^2 + a^2*d^2 + 2*a*d*b*c + b^2*c^2 :=
                by ring
       _ = (a*c - b*d)^2 + (a*d + b*c)^2 :=
                by ring
```

```
  have h2 : ∃ f, x * y = (a*c - b*d)^2 + f^2 :=
    Exists.intro (a*d + b*c) h1
  have h3 : ∃ e f, x * y = e^2 + f^2 :=
    Exists.intro (a*c - b*d) h2
  show suma_de_cuadrados (x * y)
  exact h3

-- 2ª demostración
example
  (hx : suma_de_cuadrados x)
  (hy : suma_de_cuadrados y)
  : suma_de_cuadrados (x * y) :=
by
  rcases hx with ⟨a, b, xeq : x = a^2 + b^2⟩
  -- a b : α
  -- xeq : x = a ^ 2 + b ^ 2
  rcases hy with ⟨c, d, yeq : y = c^2 + d^2⟩
  -- c d : α
  -- yeq : y = c ^ 2 + d ^ 2
  have h1: x * y = (a*c - b*d)^2 + (a*d + b*c)^2 :=
    calc x * y
         = (a^2 + b^2) * (c^2 + d^2)      := by rw [xeq, yeq]
       _ = (a*c - b*d)^2 + (a*d + b*c)^2 := by ring
  have h2 : ∃ e f, x * y = e^2 + f^2 :=
    by tauto
  show suma_de_cuadrados (x * y)
  exact h2

-- 3ª demostración
example
  (hx : suma_de_cuadrados x)
  (hy : suma_de_cuadrados y)
  : suma_de_cuadrados (x * y) :=
by
  rcases hx with ⟨a, b, xeq⟩
  -- a b : α
  -- xeq : x = a ^ 2 + b ^ 2
  rcases hy with ⟨c, d, yeq⟩
  -- c d : α
  -- yeq : y = c ^ 2 + d ^ 2
  rw [xeq, yeq]
  -- ⊢ suma_de_cuadrados ((a ^ 2 + b ^ 2) * (c ^ 2 + d ^ 2))
  use a*c - b*d, a*d + b*c
  -- ⊢ (a ^ 2 + b ^ 2) * (c ^ 2 + d ^ 2)
  --   = (a * c - b * d) ^ 2 + (a * d + b * c) ^ 2
```

```
  ring

-- 4ª demostración
example
  (hx : suma_de_cuadrados x)
  (hy : suma_de_cuadrados y)
  : suma_de_cuadrados (x * y) :=
by
  rcases hx with ⟨a, b, rfl⟩
  -- ⊢ suma_de_cuadrados ((a ^ 2 + b ^ 2) * y)
  rcases hy with ⟨c, d, rfl⟩
  -- ⊢ suma_de_cuadrados ((a ^ 2 + b ^ 2) * (c ^ 2 + d ^ 2))
  use a*c - b*d, a*d + b*c
  -- ⊢ (a ^ 2 + b ^ 2) * (c ^ 2 + d ^ 2)
  --     = (a * c - b * d) ^ 2 + (a * d + b * c) ^ 2
  ring
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 4

# Propiedades elementales de los anillos

## 4.1. Si R es un anillo y a ∈ R, entonces a + 0 = a

```
-- ---------------------------------------------------------------------
-- Demostrar en Lean4 que si R es un anillo, entonces
--    ∀ a : R, a + 0 = a
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    a + 0 = 0 + a    [por la conmutativa de la suma]
--          = a        [por el axioma del cero por la izquierda]

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a : R)

-- Demostraciones con Lean4
-- ========================

-- 1ª demostración
example : a + 0 = a :=
calc a + 0
     = 0 + a := by rw [add_comm]
```

```
   _ = a     := by rw [zero_add]

-- 2ª demostración
example : a + 0 = a :=
by
  rw [add_comm]
  rw [zero_add]

-- 3ª demostración
example : a + 0 = a :=
by rw [add_comm, zero_add]

-- 4ª demostración
example : a + 0 = a :=
by exact add_zero a

-- 5ª demostración
example : a + 0 = a :=
  add_zero a

-- 5ª demostración
example : a + 0 = a :=
by simp

-- Lemas usados
-- ============

variable (a b : R)

-- #check (add_comm a b : a + b = b + a)
-- #check (zero_add a :  0 + a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.2. Si R es un anillo y a ∈ R, entonces a + -a = 0

```
-- ---------------------------------------------------------------------
-- Demostrar en Lean4 que si R es un anillo, entonces
--    ∀ a : R, a + -a = 0
-- ---------------------------------------------------------------------
```

```
-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    a + -a = -a + a    [por la conmutativa de la suma]
--           = 0         [por el axioma de inverso por la izquierda]

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a : R)

-- 1ª demostración
-- ===============

example : a + -a = 0 :=
calc a + -a = -a + a := by rw [add_comm]
          _ = 0      := by rw [add_left_neg]

-- 2ª demostración
-- ===============

example : a + -a = 0 :=
by
  rw [add_comm]
  rw [add_left_neg]

-- 3ª demostración
-- ===============

example : a + -a = 0 :=
by rw [add_comm, add_left_neg]

-- 4ª demostración
-- ===============

example : a + -a = 0 :=
by exact add_neg_self a

-- 5ª demostración
-- ===============

example : a + -a = 0 :=
  add_neg_self a
```

```
-- 6ª demostración
-- ===============

example : a + -a = 0 :=
by simp

-- Lemas usados
-- ============

-- variable (a b : R)
-- #check (add_comm a b : a + b = b + a)
-- #check (add_left_neg a : -a + a = 0)
-- #check (add_neg_self a : a + -a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.3. Si R es un anillo y a, b ∈ R, entonces -a + (a + b) = b

```
-- ---------------------------------------------------------------------
-- Demostrar en Lean4 que si R es un anillo, entonces
--    ∀ a, b : R, -a + (a + b) = b
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    -a + (a + b) = (-a + a) + b [por la asociativa]
--                 = 0 + b        [por inverso por la izquierda]
--                 = b            [por cero por la izquierda]

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a b : R)

-- Demostraciones con Lean4
-- ========================

-- 1ª demostración
example : -a + (a + b) = b :=
```

```
calc -a + (a + b) = (-a + a) + b := by rw [← add_assoc]
                _ = 0 + b        := by rw [add_left_neg]
                _ = b            := by rw [zero_add]

-- 2ª demostración
example : -a + (a + b) = b :=
by
  rw [←add_assoc]
  rw [add_left_neg]
  rw [zero_add]

-- 3ª demostración
example : -a + (a + b) = b :=
by rw [←add_assoc, add_left_neg, zero_add]

-- 4ª demostración
example : -a + (a + b) = b :=
by exact neg_add_cancel_left a b

-- 5ª demostración
example : -a + (a + b) = b :=
  neg_add_cancel_left a b

-- 6ª demostración
example : -a + (a + b) = b :=
by simp

-- Lemas usados
-- ============

-- variable (c : R)
-- #check (add_assoc a b c : (a + b) + c = a + (b + c))
-- #check (add_left_neg a : -a + a = 0)
-- #check (neg_add_cancel_left a b : -a + (a + b) = b)
-- #check (zero_add a :  0 + a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.4. Si R es un anillo y a, b ∈ R, entonces (a + b) + -b = a

```
-- ---------------------------------------------------------------------
-- Demostrar en Lean4 que si R es un anillo, entonces
--    ∀ a, b : R, (a + b) + -b = a
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    (a + b) + -b = a + (b + -b)    [por la asociativa]
--                 _ = a + 0           [por suma con opuesto]
--                 _ = a               [por suma con cero]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a b : R)

-- 1ª demostración
example : (a + b) + -b = a :=
calc
  (a + b) + -b = a + (b + -b) := by rw [add_assoc]
             _ = a + 0        := by rw [add_right_neg]
             _ = a            := by rw [add_zero]

-- 2ª demostración
example : (a + b) + -b = a :=
by
  rw [add_assoc]
  rw [add_right_neg]
  rw [add_zero]

-- 3ª demostración
example : (a + b) + -b = a :=
by rw [add_assoc, add_right_neg, add_zero]

-- 4ª demostración
example : (a + b) + -b = a :=
```

```
  add_neg_cancel_right a b

-- 5ª demostración
example : (a + b) + -b = a :=
  add_neg_cancel_right _ _

-- 6ª demostración
example : (a + b) + -b = a :=
by simp

-- Lemas usados
-- ============

-- variable (c : R)
-- #check (add_assoc a b c : (a + b) + c = a + (b + c))
-- #check (add_neg_cancel_right a b : (a + b) + -b = a)
-- #check (add_right_neg a : a + -a = 0)
-- #check (add_zero a :  a + 0 = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.5. Si R es un anillo y a, b, c ∈ R tales que a+b=a+c, entonces b=c

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a, b, c ∈ R tales que
--    a + b = a + c
-- entonces
--    b = c
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- ======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    b = 0 + b            [por suma con cero]
--      = (-a + a) + b     [por suma con opuesto]
--      = -a + (a + b)     [por asociativa]
--      = -a + (a + c)     [por hipótesis]
```

```
--      = (-a + a) + c    [por asociativa]
--      = 0 + c           [por suma con opuesto]
--      = c               [por suma con cero]

-- 2ª demostración en LN
-- =====================

-- Por la siguiente cadena de implicaciones
--    a + b = a + c
--    ==> -a + (a + b) = -a + (a + c)     [sumando -a]
--    ==>  (-a + a) + b = (-a + a) + c    [por la asociativa]
--    ==>  0 + b = 0 + b                  [suma con opuesto]
--    ==>  b = c                          [suma con cero]

-- 3ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    b = -a + (a + b)
--      = -a + (a + c)   [por la hipótesis]
--      = c

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable {a b c : R}

-- 1ª demostración
example
  (h : a + b = a + c)
  : b = c :=
calc
  b = 0 + b        := by rw [zero_add]
  _ = (-a + a) + b := by rw [add_left_neg]
  _ = -a + (a + b) := by rw [add_assoc]
  _ = -a + (a + c) := by rw [h]
  _ = (-a + a) + c := by rw [←add_assoc]
  _ = 0 + c        := by rw [add_left_neg]
  _ = c            := by rw [zero_add]

-- 2ª demostración
```

```
example
  (h : a + b = a + c)
  : b = c :=
by
  have h1 : -a + (a + b) = -a + (a + c) :=
    congrArg (HAdd.hAdd (-a)) h
  clear h
  rw [← add_assoc] at h1
  rw [add_left_neg] at h1
  rw [zero_add] at h1
  rw [← add_assoc] at h1
  rw [add_left_neg] at h1
  rw [zero_add] at h1
  exact h1

-- 3ª demostración
example
  (h : a + b = a + c)
  : b = c :=
calc
  b = -a + (a + b) := by rw [neg_add_cancel_left a b]
  _ = -a + (a + c) := by rw [h]
  _ = c            := by rw [neg_add_cancel_left]

-- 4ª demostración
example
  (h : a + b = a + c)
  : b = c :=
by
  rw [← neg_add_cancel_left a b]
  rw [h]
  rw [neg_add_cancel_left]

-- 5ª demostración
example
  (h : a + b = a + c)
  : b = c :=
by
  rw [← neg_add_cancel_left a b, h, neg_add_cancel_left]

-- 6ª demostración
example
  (h : a + b = a + c)
  : b = c :=
add_left_cancel h
```

```
-- Lemas usados
-- ============

-- #check (add_assoc a b c : (a + b) + c = a + (b + c))
-- #check (add_left_cancel : a + b = a + c → b = c)
-- #check (add_left_neg a : -a + a = 0)
-- #check (neg_add_cancel_left a b : -a + (a + b) = b)
-- #check (zero_add a :  0 + a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.6. Si R es un anillo y a, b, c ∈ R tales que a+b=c+b, entonces a=c

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a, b, c ∈ R tales que
--    a + b = c + b
-- entonces
--    a = c
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    a = a + 0           [por suma con cero]
--      = a + (b + -b)    [por suma con opuesto]
--      = (a + b) + -b    [por asociativa]
--      = (c + b) + -b    [por hipótesis]
--      = c + (b + -b)    [por asociativa]
--      = c + 0           [por suma con opuesto]
--      = c               [por suma con cero]

-- 2ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    a = (a + b) + -b
--      = (c + b) + -b    [por hipótesis]
```

```
--      = c

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable {a b c : R}

-- 1ª demostración con Lean4
-- =========================

example
  (h : a + b = c + b)
  : a = c :=
calc
  a = a + 0          := by rw [add_zero]
  _ = a + (b + -b) := by rw [add_right_neg]
  _ = (a + b) + -b := by rw [add_assoc]
  _ = (c + b) + -b := by rw [h]
  _ = c + (b + -b) := by rw [← add_assoc]
  _ = c + 0          := by rw [← add_right_neg]
  _ = c              := by rw [add_zero]

-- 2ª demostración con Lean4
-- =========================

example
  (h : a + b = c + b)
  : a = c :=
calc
  a = (a + b) + -b := (add_neg_cancel_right a b).symm
  _ = (c + b) + -b := by rw [h]
  _ = c              := add_neg_cancel_right c b

-- 3ª demostración con Lean4
-- =========================

example
  (h : a + b = c + b)
  : a = c :=
by
  rw [← add_neg_cancel_right a b]
```

```
  rw [h]
  rw [add_neg_cancel_right]

-- 4ª demostración con Lean4
-- =========================

example
  (h : a + b = c + b)
  : a = c :=
by
  rw [← add_neg_cancel_right a b, h, add_neg_cancel_right]

-- 5ª demostración con Lean4
-- =========================

example
  (h : a + b = c + b)
  : a = c :=
add_right_cancel h

-- Lemas usados
-- ============

-- #check (add_assoc a b c : (a + b) + c = a + (b + c))
-- #check (add_neg_cancel_right a b : (a + b) + -b = a)
-- #check (add_right_cancel : a + b = c + b → a = c)
-- #check (add_right_neg a : a + -a = 0)
-- #check (add_zero a :  a + 0 = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.7. Si R es un anillo y a ∈ R, entonces a.0 = 0

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a ∈ R, entonces
--    a * 0 = 0
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Basta aplicar la propiedad cancelativa a
--    a.0 + a.0 = a.0 + 0
```

```
-- que se demuestra mediante la siguiente cadena de igualdades
--    a.0 + a.0 = a.(0 + 0)    [por la distributiva]
--              = a.0          [por suma con cero]
--              = a.0 + 0      [por suma con cero]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable (a : R)

-- 1ª demostración
-- ===============

example : a * 0 = 0 :=
by
  have h : a * 0 + a * 0 = a * 0 + 0 :=
    calc a * 0 + a * 0 = a * (0 + 0) := by rw [mul_add a 0 0]
                     _ = a * 0       := by rw [add_zero 0]
                     _ = a * 0 + 0   := by rw [add_zero (a * 0)]
  rw [add_left_cancel h]

-- 2ª demostración
-- ===============

example : a * 0 = 0 :=
by
  have h : a * 0 + a * 0 = a * 0 + 0 :=
    calc a * 0 + a * 0 = a * (0 + 0) := by rw [← mul_add]
                     _ = a * 0       := by rw [add_zero]
                     _ = a * 0 + 0   := by rw [add_zero]
  rw [add_left_cancel h]

-- 3ª demostración
-- ===============

example : a * 0 = 0 :=
by
  have h : a * 0 + a * 0 = a * 0 + 0 :=
    by rw [← mul_add, add_zero, add_zero]
  rw [add_left_cancel h]
```

```
-- 4ª demostración
-- ===============

example : a * 0 = 0 :=
by
  have : a * 0 + a * 0 = a * 0 + 0 :=
    calc a * 0 + a * 0 = a * (0 + 0) := by simp
                     _ = a * 0       := by simp
                     _ = a * 0 + 0   := by simp
  simp

-- 5ª demostración
-- ===============

example : a * 0 = 0 :=
  mul_zero a

-- 6ª demostración
-- ===============

example : a * 0 = 0 :=
by simp

-- Lemas usados
-- ============

-- variable (b c : R)
-- #check (add_left_cancel : a + b = a + c → b = c)
-- #check (add_zero a :  a + 0 = a)
-- #check (mul_add a b c :  a * (b + c) = a * b + a * c)
-- #check (mul_zero a : a * 0 = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.8. Si R es un anillo y a ∈ R, entonces 0.a = 0

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a ∈ R, entonces
--    0 * a = 0
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Basta aplicar la propiedad cancelativa a
--    0.a + 0.a = 0.a + 0
-- que se demuestra mediante la siguiente cadena de igualdades
--    0.a + 0.a = (0 + 0).a    [por la distributiva]
--              = 0.a          [por suma con cero]
--              = 0.a + 0      [por suma con cero]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable (a : R)

-- 1ª demostración
example : 0 * a = 0 :=
by
  have h : 0 * a + 0 * a = 0 * a + 0 :=
    calc 0 * a + 0 * a = (0 + 0) * a := by rw [add_mul]
                     _ = 0 * a       := by rw [add_zero]
                     _ = 0 * a + 0   := by rw [add_zero]
  rw [add_left_cancel h]

-- 2ª demostración
example : 0 * a = 0 :=
by
  have h : 0 * a + 0 * a = 0 * a + 0 :=
    by rw [←add_mul, add_zero, add_zero]
  rw [add_left_cancel h]

-- 3ª demostración
example : 0 * a = 0 :=
by
  have : 0 * a + 0 * a = 0 * a + 0 :=
    calc 0 * a + 0 * a = (0 + 0) * a := by simp
                     _ = 0 * a       := by simp
                     _ = 0 * a + 0   := by simp
  simp

-- 4ª demostración
example : 0 * a = 0 :=
by
```

```
  have : 0 * a + 0 * a = 0 * a + 0 := by simp
  simp

-- 5ª demostración
example : 0 * a = 0 :=
by simp

-- 6ª demostración
example : 0 * a = 0 :=
zero_mul a

-- Lemas usados
-- ============

-- variable (b c : R)
-- #check (add_mul a b c :  (a + b) * c = a * c + b * c)
-- #check (add_zero a :  a + 0 = a)
-- #check (zero_mul a : 0 * a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.9. Si R es un anillo y a, b ∈ R tales que a+b=0, entonces -a=b

```
-- ---------------------------------------------------------------------
-- Demostrar que si es un anillo y a, b ∈ R tales que
--    a + b = 0
-- entonces
--    -a = b
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    -a = -a + 0          [por suma cero]
--       = -a + (a + b)    [por hipótesis]
--       = b               [por cancelativa]
```

```
-- 2ª demostración en LN
-- ---------------------

-- Sumando -a a ambos lados de la hipótesis, se tiene
--    -a + (a + b) = -a + 0
-- El término de la izquierda se reduce a b (por la cancelativa) y el de
-- la derecha a -a (por la suma con cero). Por tanto, se tiene
--     b = -a
-- Por la simetría de la igualdad, se tiene
--     -a = b

-- Demostraciones con Lean 4
-- =========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable {a b : R}

-- 1ª demostración (basada en la 1º en LN)
example
  (h : a + b = 0)
  : -a = b :=
calc
  -a = -a + 0       := by rw [add_zero]
   _ = -a + (a + b) := by rw [h]
   _ = b            := by rw [neg_add_cancel_left]

-- 2ª demostración (basada en la 1º en LN)
example
  (h : a + b = 0)
  : -a = b :=
calc
  -a = -a + 0       := by simp
   _ = -a + (a + b) := by rw [h]
   _ = b            := by simp

-- 3ª demostración (basada en la 2º en LN)
example
  (h : a + b = 0)
  : -a = b :=
by
  have h1 : -a + (a + b) = -a + 0 := congrArg (HAdd.hAdd (-a)) h
  have h2 : -a + (a + b) = b := neg_add_cancel_left a b
```

```
  have h3 : -a + 0 = -a := add_zero (-a)
  rw [h2, h3] at h1
  exact h1.symm

-- 4ª demostración
example
  (h : a + b = 0)
  : -a = b :=
neg_eq_iff_add_eq_zero.mpr h

-- Lemas usados
-- ============

-- #check (add_zero a : a + 0 = a)
-- #check (neg_add_cancel_left a b : -a + (a + b) = b)
-- #check (neg_eq_iff_add_eq_zero : -a = b ↔ a + b = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.10. Si R es un anillo y a, b ∈ R tales que a+b=0, entonces a=-b

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a, b ∈ R tales que
--    a + b = 0
-- entonces
--    a = -b
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de igualdades
--    a = (a + b) + -b    [por la concelativa]
--      = 0 + -b          [por la hipótesis]
--      = -b              [por la suma con cero]

-- 2ª demostración en LN
-- ---------------------
```

```
-- Sumando -a a ambos lados de la hipótesis, se tiene
--    (a + b) + -b = 0 + -b
-- El término de la izquierda se reduce a a (por la cancelativa) y el de
-- la derecha a -b (por la suma con cero). Por tanto, se tiene
--     a = -b

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable {a b : R}

-- 1ª demostración (basada en la 1ª en LN)
example
  (h : a + b = 0)
  : a = -b :=
calc
  a = (a + b) + -b := by rw [add_neg_cancel_right]
  _ = 0 + -b       := by rw [h]
  _ = -b           := by rw [zero_add]

-- 2ª demostración (basada en la 1ª en LN)
example
  (h : a + b = 0)
  : a = -b :=
calc
  a = (a + b) + -b := by simp
  _ = 0 + -b       := by rw [h]
  _ = -b           := by simp

-- 3ª demostración (basada en la 1ª en LN)
example
  (h : a + b = 0)
  : a = -b :=
by
  have h1 : (a + b) + -b = 0 + -b := by rw [h]
  have h2 : (a + b) + -b = a := add_neg_cancel_right a b
  have h3 : 0 + -b = -b := zero_add (-b)
  rwa [h2, h3] at h1

-- 4ª demostración
```

```
example
  (h : a + b = 0)
  : a = -b :=
add_eq_zero_iff_eq_neg.mp h

-- Lemas usados
-- ============

-- #check (add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b)
-- #check (add_neg_cancel_right a b : (a + b) + -b = a)
-- #check (zero_add a : 0 + a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.11. Si R es un anillo, entonces -0 = 0

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo, entonces
--    -0 = 0
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la suma con cero se tiene
--    0 + 0 = 0
-- Aplicándole la propiedad
--    ∀ a b ∈ R, a + b = 0 → -a = b
-- se obtiene
--    -0 = 0

-- 2ª demostración en LN
-- =====================

-- Puesto que
--    ∀ a b ∈ R, a + b = 0 → -a = b
-- basta demostrar que
--    0 + 0 = 0
-- que es cierta por la suma con cero.
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]

-- 1ª demostración (basada en la 1ª en LN)
example : (-0 : R) = 0 :=
by
  have h1 : (0 : R) + 0 = 0 := add_zero 0
  show (-0 : R) = 0
  exact neg_eq_of_add_eq_zero_left h1

-- 2ª demostración (basada en la 2ª en LN)
example : (-0 : R) = 0 :=
by
  apply neg_eq_of_add_eq_zero_left
  rw [add_zero]

-- 3ª demostración
example : (-0 : R) = 0 :=
  neg_zero

-- 4ª demostración
example : (-0 : R) = 0 :=
by simp

-- Lemas usados
-- ============

-- variable (a b : R)
-- #check (add_zero a : a + 0 = a)
-- #check (neg_eq_of_add_eq_zero_left : a + b = 0 → -b = a)
-- #check (neg_zero : -0 = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.12. Si R es un anillo y a ∈ R, entonces -(-a) = a

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a ∈ R, entonces
--     -(-a) = a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia de las siguiente propiedades demostradas en
-- ejercicios anteriores:
--    ∀ a b ∈ R, a + b = 0 → -a = b
--    ∀ a ∈ R, -a + a = 0

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic

variable {R : Type _} [Ring R]
variable {a : R}

-- 1ª demostración
example : -(-a) = a :=
by
  have h1 : -a + a = 0 := add_left_neg a
  show -(-a) = a
  exact neg_eq_of_add_eq_zero_right h1

-- 2ª demostración
example : -(-a) = a :=
by
  apply neg_eq_of_add_eq_zero_right
  rw [add_left_neg]

-- 3ª demostración
example : -(-a) = a :=
neg_neg a

-- 4ª demostración
example : -(-a) = a :=
```

```
by simp

-- Lemas usados
-- ============

-- variable (b : R)
-- #check (add_left_neg a : -a + a = 0)
-- #check (neg_eq_of_add_eq_zero_right : a + b = 0 → -a = b)
-- #check (neg_neg a : -(-a) = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.13. Si R es un anillo y a, b ∈ R, entonces a - b = a + -b

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a, b ∈ R, entonces
--    a - b = a + -b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la definición de la resta.

-- Demostración en Lean4
-- =====================

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a b : R)

example : a - b = a + -b :=
-- by exact?
sub_eq_add_neg a b

-- Lemas usados
-- ============

-- #check (sub_eq_add_neg a b : a - b = a + -b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.14. Si R es un anillo y a ∈ R, entonces a - a = 0

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a ∈ R, entonces
--     a - a = 0
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades:
--    a - a = a + -a    [por definición de resta]
--          = 0         [por suma con opuesto]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs
variable {R : Type _} [Ring R]
variable (a : R)

-- 1ª demostración
example : a - a = 0 :=
calc
  a - a = a + -a := by rw [sub_eq_add_neg a a]
      _ = 0      := by rw [add_right_neg]

-- 2ª demostración
example : a - a = 0 :=
sub_self a

-- 3ª demostración
example : a - a = 0 :=
by simp

-- Lemas usados
-- ============

-- #check (add_right_neg a : a + -a = 0)
-- #check (sub_eq_add_neg a b : a - b = a + -b)
-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.15. En los anillos, 1 + 1 = 2

```
-- ---------------------------------------------------------------------
-- Demostrar que en los anillos,
--    1 + 1 = 2
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por cálculo.

-- Demostración con Lean4
-- ======================

import Mathlib.Algebra.Ring.Defs
import Mathlib.Tactic
variable {R : Type _} [Ring R]

-- Demostraciones con Lean4
-- ========================

-- 1ª demostración
example : 1 + 1 = (2 : R) :=
by norm_num

-- 2ª demostración
example : 1 + 1 = (2 : R) :=
one_add_one_eq_two

-- Lemas usados
-- ============

-- #check (one_add_one_eq_two : 1 + 1 = 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 4.16. Si R es un anillo y a ∈ R, entonces 2a = a+a

```
-- ---------------------------------------------------------------------
-- Demostrar que si R es un anillo y a ∈ R, entonces
--    2 * a = a + a
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    2·a = (1 + 1)·a    [por la definición de 2]
--        = 1·a + 1·a    [por la distributiva]
--        = a + a        [por producto con uno]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Ring.Defs

variable {R : Type _} [Ring R]
variable (a : R)

-- 1ª demostración
example : 2 * a = a + a :=
calc
  2 * a = (1 + 1) * a   := by rw [one_add_one_eq_two]
      _ = 1 * a + 1 * a := by rw [add_mul]
      _ = a + a         := by rw [one_mul]

-- 2ª demostración
example : 2 * a = a + a :=
by exact two_mul a

-- Lemas usados
-- ============

-- variable (b c : R)
-- #check (add_mul a b c : (a + b) * c = a * c + b * c)
-- #check (one_add_one_eq_two : (1 : R) + 1 = 2)
-- #check (one_mul a : 1 * a = a)
-- #check (two_mul a : 2 * a = a + a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

# Capítulo 5

# Propiedades elementales de los grupos

## 5.1. Si G es un grupo y a ∈ G, entonces $aa^{-1} = 1$

```
-- ---------------------------------------------------------------------
-- En Lean4, se declara que G es un grupo mediante la expresión
--    variable {G : Type _} [Group G]
--
-- Como consecuencia, se tiene los siguientes axiomas
--    mul_assoc :    ∀ a b c : G, a * b * c = a * (b * c)
--    one_mul :      ∀ a : G, 1 * a = a
--    mul_left_inv : ∀ a : G, a⁻¹ * a = 1
--
-- Demostrar que si G es un grupo y a ∈ G, entonces
--    a * a⁻¹ = 1
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de igualdades
--    a·a⁻¹ = 1·(a·a⁻¹)                 [por producto con uno]
--          = (1·a)·a⁻¹                 [por asociativa]
--          = (((a⁻¹)⁻¹·a⁻¹) ·a)·a⁻¹    [por producto con inverso]
--          = ((a⁻¹)⁻¹·(a⁻¹ ·a))·a⁻¹    [por asociativa]
--          = ((a⁻¹)⁻¹·1)·a⁻¹           [por producto con inverso]
--          = (a⁻¹)⁻¹·(1·a⁻¹)           [por asociativa]
--          = (a⁻¹)⁻¹·a⁻¹               [por producto con uno]
```

```
--          = 1                         [por producto con inverso]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Group.Defs

variable {G : Type _} [Group G]
variable (a b : G)

-- 1ª demostración
example : a * a⁻¹ = 1 :=
calc
  a * a⁻¹ = 1 * (a * a⁻¹)                := by rw [one_mul]
        _ = (1 * a) * a⁻¹                := by rw [mul_assoc]
        _ = (((a⁻¹)⁻¹ * a⁻¹)  * a) * a⁻¹ := by rw [mul_left_inv]
        _ = ((a⁻¹)⁻¹ * (a⁻¹  * a)) * a⁻¹ := by rw [← mul_assoc]
        _ = ((a⁻¹)⁻¹ * 1) * a⁻¹          := by rw [mul_left_inv]
        _ = (a⁻¹)⁻¹ * (1 * a⁻¹)          := by rw [mul_assoc]
        _ = (a⁻¹)⁻¹ * a⁻¹                := by rw [one_mul]
        _ = 1                            := by rw [mul_left_inv]

-- 2ª demostración
example : a * a⁻¹ = 1 :=
calc
  a * a⁻¹ = 1 * (a * a⁻¹)                := by simp
        _ = (1 * a) * a⁻¹                := by simp
        _ = (((a⁻¹)⁻¹ * a⁻¹)  * a) * a⁻¹ := by simp
        _ = ((a⁻¹)⁻¹ * (a⁻¹  * a)) * a⁻¹ := by simp
        _ = ((a⁻¹)⁻¹ * 1) * a⁻¹          := by simp
        _ = (a⁻¹)⁻¹ * (1 * a⁻¹)          := by simp
        _ = (a⁻¹)⁻¹ * a⁻¹                := by simp
        _ = 1                            := by simp

-- 3ª demostración
example : a * a⁻¹ = 1 :=
by simp

-- 4ª demostración
example : a * a⁻¹ = 1 :=
by exact mul_inv_self a

-- Lemas usados
-- ============
```

```
-- variable (c : G)
-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (mul_inv_self a : a * a⁻¹ = 1)
-- #check (mul_left_inv a : a⁻¹  * a = 1)
-- #check (one_mul a : 1 * a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 5.2. Si G es un grupo y a ∈ G, entonces a·1 = a

```
-- ---------------------------------------------------------------------
-- Demostrar que si G es un grupo y a ∈ G, entonces
--    a * 1 = a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se tiene por la siguiente cadena de igualdades
--    a·1 = a·(a⁻¹·a)    [por producto con inverso]
--        = (a·a⁻¹)·a    [por asociativa]
--        = 1·a          [por producto con inverso]
--        = a            [por producto con uno]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Group.Defs

variable {G : Type _} [Group G]
variable (a b : G)

-- 1ª demostración
example : a * 1 = a :=
calc
  a * 1 = a * (a⁻¹ * a) := by rw [mul_left_inv]
      _ = (a * a⁻¹) * a := by rw [mul_assoc]
      _ = 1 * a         := by rw [mul_right_inv]
      _ = a             := by rw [one_mul]

-- 2ª demostración
example : a * 1 = a :=
calc
```

```
  a * 1 = a * (a⁻¹ * a) := by simp
      _ = (a * a⁻¹) * a := by simp
      _ = 1 * a         := by simp
      _ = a             := by simp

-- 3ª demostración
example : a * 1 = a :=
by simp

-- 4ª demostración
example : a * 1 = a :=
by exact mul_one a

-- Lemas usados
-- ============

-- variable (c : G)
-- #check (mul_left_inv a : a⁻¹  * a = 1)
-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (mul_right_inv a : a * a⁻¹ = 1)
-- #check (one_mul a : 1 * a = a)
-- #check (mul_one a : a * 1 = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 5.3. Si G es un grupo y a, b ∈ G tales que ab = 1 entonces a⁻¹ = b

```
-- ---------------------------------------------------------------------
-- Demostrar que si G es un grupo y a, b ∈ G, tales que
--    a * b = 1
-- entonces
--    a⁻¹ = b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se tiene a partir de la siguiente cadena de igualdades
--    a⁻¹ = a⁻¹ * 1           [por producto por uno]
--        = a⁻¹ * (a * b)     [por hipótesis]
--        = (a⁻¹ * a) * b     [por asociativa]
```

```
--        = 1 * b             [por producto con inverso]
--        = b                 [por producto por uno]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Group.Defs

variable {G : Type _} [Group G]
variable (a b : G)

-- 1º demostración
example
  (h : a * b = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ = a⁻¹ * 1       := by rw [mul_one]
    _ = a⁻¹ * (a * b) := by rw [h]
    _ = (a⁻¹ * a) * b := by rw [mul_assoc]
    _ = 1 * b         := by rw [mul_left_inv]
    _ = b             := by rw [one_mul]

-- 2º demostración
example
  (h : a * b = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ = a⁻¹ * 1       := by simp
    _ = a⁻¹ * (a * b) := by simp [h]
    _ = (a⁻¹ * a) * b := by simp
    _ = 1 * b         := by simp
    _ = b             := by simp

-- 3º demostración
example
  (h : a * b = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ =  a⁻¹ * (a * b) := by simp [h]
    _ =  b             := by simp

-- 4º demostración
example
  (h : a * b = 1)
  : a⁻¹ = b :=
```

```
by exact inv_eq_of_mul_eq_one_right h

-- Lemas usados
-- ============

-- variable (c : G)
-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (mul_left_inv a : a⁻¹  * a = 1)
-- #check (mul_one a : a * 1 = a)
-- #check (one_mul a : 1 * a = a)
-- #check (inv_eq_of_mul_eq_one_right : a * b = 1 → a⁻¹ = b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 5.4. Si G es un grupo y a, b ∈ G, entonces $(ab)^{-1} = b^{-1}a^{-1}$

```
-- ---------------------------------------------------------------------
-- Demostrar que si G es un grupo y a, b ∈ G, entonces
--    (a * b)⁻¹ = b⁻¹ * a⁻¹
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Teniendo en cuenta la propiedad
--    ∀ a b ∈ R, ab = 1 → a⁻¹ = b,
-- basta demostrar que
--    (a·b)·(b⁻¹·a⁻¹) = 1.
-- La identidad anterior se demuestra mediante la siguiente cadena de
-- igualdades
--    (a·b)·(b⁻¹·a⁻¹) =  a·(b·(b⁻¹·a⁻¹))   [por la asociativa]
--                    =  a·((b·b⁻¹)·a⁻¹)   [por la asociativa]
--                    =  a·(1·a⁻¹)         [por producto con inverso]
--                    =  a·a⁻¹             [por producto con uno]
--                    =  1                 [por producto con inverso]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Group.Defs
```

```
variable {G : Type _} [Group G]
variable (a b : G)

lemma aux : (a * b) * (b⁻¹ * a⁻¹) = 1 :=
calc
  (a * b) * (b⁻¹ * a⁻¹)
    = a * (b * (b⁻¹ * a⁻¹)) := by rw [mul_assoc]
  _ = a * ((b * b⁻¹) * a⁻¹) := by rw [mul_assoc]
  _ = a * (1 * a⁻¹)          := by rw [mul_right_inv]
  _ = a * a⁻¹                := by rw [one_mul]
  _ = 1                      := by rw [mul_right_inv]

-- 1ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
  have h1 : (a * b) * (b⁻¹ * a⁻¹) = 1 :=
    aux a b
  show (a * b)⁻¹ = b⁻¹ * a⁻¹
  exact inv_eq_of_mul_eq_one_right h1

-- 3ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
  have h1 : (a * b) * (b⁻¹ * a⁻¹) = 1 :=
    aux a b
  show (a * b)⁻¹ = b⁻¹ * a⁻¹
  simp [h1]

-- 4ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
  have h1 : (a * b) * (b⁻¹ * a⁻¹) = 1 :=
    aux a b
  simp [h1]

-- 5ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
  apply inv_eq_of_mul_eq_one_right
  rw [aux]

-- 6ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by exact mul_inv_rev a b
```

```
-- 7ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by simp

-- Lemas usados
-- ============

-- variable (c : G)
-- #check (inv_eq_of_mul_eq_one_right : a * b = 1 → a⁻¹ = b)
-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (mul_inv_rev a b : (a * b)⁻¹ = b⁻¹ * a⁻¹)
-- #check (mul_right_inv a : a * a⁻¹ = 1)
-- #check (one_mul a : 1 * a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

# Capítulo 6

# Propiedades de orden en los números reales

## 6.1. En ℝ, si a ≤ b, b <c, c ≤ d y d <e, entonces a <e

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c, d y e son números reales tales que
--    a ≤ b,
--    b < c,
--    c ≤ d y
--    d < e,
-- entonces
--    a < e.
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de desigualdades
--    a ≤ b    [por h1]
--      < c    [por h2]
--      ≤ d    [por h3]
--      < e    [por h4]

-- 2ª demostración en LN
-- =====================
```

```
-- A partir de las hipótesis 1 (a ≤ b) y 2 (b < c) se tiene
--    a < c
-- que, junto la hipótesis 3 (c ≤ d) da
--    a < d
-- que, junto la hipótesis 4 (d < e) da
--    a < e.

-- 3ª demostración en LN
-- =====================

-- Para demostrar a < e, por la hipótesis 1 (a ≤ b) se reduce a probar
--    b < e
-- que, por la hipótesis 2 (b < c), se reduce a
--    c < e
-- que, por la hipótesis 3 (c ≤ d), se reduce a
--    d < e
-- que es cierto, por la hipótesis 4.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b c d e : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : b < c)
  (h3 : c ≤ d)
  (h4 : d < e) :
  a < e :=
calc
  a ≤ b := h1
  _ < c := h2
  _ ≤ d := h3
  _ < e := h4

-- 2ª demostración
-- ===============

example
```

```
  (h1 : a ≤ b)
  (h2 : b < c)
  (h3 : c ≤ d)
  (h4 : d < e) :
  a < e :=
by
  have h5 : a < c := lt_of_le_of_lt h1 h2
  have h6 : a < d := lt_of_lt_of_le h5 h3
  show a < e
  exact lt_trans h6 h4

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : b < c)
  (h3 : c ≤ d)
  (h4 : d < e) :
  a < e :=
by
  apply lt_of_le_of_lt h1
  apply lt_trans h2
  apply lt_of_le_of_lt h3
  exact h4

-- El desarrollo de la prueba es
--
--    a b c d e : ℝ,
--    h1 : a ≤ b,
--    h2 : b < c,
--    h3 : c ≤ d,
--    h4 : d < e
--    ⊢ a < e
-- apply lt_of_le_of_lt h1,
--    ⊢ b < e
-- apply lt_trans h2,
--    ⊢ c < e
-- apply lt_of_le_of_lt h3,
--    ⊢ d < e
-- exact h4,
--    no goals

-- 4ª demostración
-- ===============
```

```
example
  (h1 : a ≤ b)
  (h2 : b < c)
  (h3 : c ≤ d)
  (h4 : d < e) :
  a < e :=
by linarith

-- Lemas usados
-- ============

-- #check (lt_of_le_of_lt : a ≤ b → b < c → a < c)
-- #check (lt_of_lt_of_le : a < b → b ≤ c → a < c)
-- #check (lt_trans : a < b → b < c → a < c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 6.2. En ℝ, si 2a ≤ 3b, 1 ≤ a y d = 2, entonces d + a ≤ 5b

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b y c son números reales tales que
--    2 * a ≤ 3 * b
--    1 ≤ a
--    c = 2
-- entonces
--    c + a ≤ 5 * b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la siguiente cadena de desigualdades
--    c + a = 2 + a      [por la hipótesis 3 (c = 2)]
--          ≤ 2·a + a    [por la hipótesis 2 (1 ≤ a)]
--          = 3·a
--          ≤ 9/2·b      [por la hipótesis 1 (2·a ≤ 3·b)]
--          ≤ 5·b

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

-- 1ª demostración
example
  (h1 : 2 * a ≤ 3 * b)
  (h2 : 1 ≤ a)
  (h3 : c = 2)
  : c + a ≤ 5 * b :=
calc
  c + a = 2 + a     := by rw [h3]
      _ ≤ 2 * a + a := by linarith only [h2]
      _ = 3 * a     := by linarith only []
      _ ≤ 9/2 * b   := by linarith only [h1]
      _ ≤ 5 * b     := by linarith

-- 2ª demostración
example
  (h1 : 2 * a ≤ 3 * b)
  (h2 : 1 ≤ a)
  (h3 : c = 2)
  : c + a ≤ 5 * b :=
by linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 6.3. En ℝ, si 1 ≤ a y b ≤ d, entonces 2 + a + $e^b$ ≤ 3a + $e^d$

```
-- ---------------------------------------------------------------------
-- Sean a, b, y d números reales. Demostrar  que si
--    1 ≤ a
--    b ≤ d
-- entonces
--    2 + a + exp b ≤ 3 * a + exp d
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- De la primera hipótesis (1 ≤ a), multiplicando por 2, se obtiene
```

```
--    2 ≤ 2a
-- y, sumando a ambos lados, se tiene
--    2 + a ≤ 3a             (1)
-- De la hipótesis 2 (b ≤ d) y de la monotonía de la función exponencial
-- se tiene
--    e^b ≤ e^d              (2)
-- Finalmente, de (1) y (2) se tiene
--    2 + a + e^b ≤ 3a + e^d

-- Demostraciones con Lean4
-- ========================

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b d : ℝ)

-- 1ª demostración
example
  (h1 : 1 ≤ a)
  (h2 : b ≤ d)
  : 2 + a + exp b ≤ 3 * a + exp d :=
by
  have h3 : 2 + a ≤ 3 * a := calc
    2 + a = 2 * 1 + a := by linarith only []
        _ ≤ 2 * a + a := by linarith only [h1]
        _ ≤ 3 * a     := by linarith only []
  have h4 : exp b ≤ exp d := by
    linarith only [exp_le_exp.mpr h2]
  show 2 + a + exp b ≤ 3 * a + exp d
  exact add_le_add h3 h4

-- 2ª demostración
example
  (h1 : 1 ≤ a)
  (h2 : b ≤ d)
  : 2 + a + exp b ≤ 3 * a + exp d :=
calc
  2 + a + exp b
    ≤ 3 * a + exp b := by linarith only [h1]
  _ ≤ 3 * a + exp d := by linarith only [exp_le_exp.mpr h2]

-- 3ª demostración
example
```

```
  (h1 : 1 ≤ a)
  (h2 : b ≤ d)
  : 2 + a + exp b ≤ 3 * a + exp d :=
by linarith [exp_le_exp.mpr h2]

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
-- #check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

## 6.4. En ℝ, si a ≤ b y c <d, entonces a + eᶜ + f ≤ b + eᵈ + f

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b, c, d y f son números reales tales que
--    a ≤ b
--    c < d
-- entonces
--    a + eᶜ + f ≤ b + eᵈ + f
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Aplicando a la hipótesis 3 (c < d) la monotonía de la exponencial, se
-- tiene
--    e^c < e^d
-- que, junto a la hipótesis 1 (a ≤ b) y la monotonía de la suma da
--    a + e^c < b + e^d
-- y, de nuevo por la monotonía de la suma, se tiene
--    a + e^c + f < b + e^d + f

-- 2ª demostración en LN
-- =====================
```

```
-- Tenemos que demostrar que
--    (a + e^c) + f < (b + e^d) + f
-- que, por la monotonía de la suma, se reduce a las siguientes dos
-- desigualdades:
--    a + e^c < b + e^d                                                (1)
--    f ≤ f                                                            (2)
--
-- La (1), de nuevo por la monotonía de la suma, se reduce a las
-- siguientes dos:
--    a ≤ b                                                          (1.1)
--    e^c < e^d                                                      (1.2)
--
-- La (1.1) se tiene por la hipótesis 1.
--
-- La (1.2) se tiene aplicando la monotonía de la exponencial a la
-- hipótesis 2.
--
-- La (2) se tiene por la propiedad reflexiva.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Analysis.SpecialFunctions.Log.Basic
open Real
variable (a b c d f : ℝ)

-- 1ª demostración
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + f < b + exp d + f :=
by
  have h3 : exp c < exp d :=
    exp_lt_exp.mpr h2
  have h4 : a + exp c < b + exp d :=
    add_lt_add_of_le_of_lt h1 h3
  show a + exp c + f < b + exp d + f
  exact add_lt_add_right h4 f

-- 2ª demostración
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + f < b + exp d + f :=
by
```

```
  apply add_lt_add_of_lt_of_le
  { apply add_lt_add_of_le_of_lt
    { exact h1 }
    { apply exp_lt_exp.mpr
      exact h2 } }
  { apply le_refl }

-- 3ª demostración
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + f < b + exp d + f :=
by
  apply add_lt_add_of_lt_of_le
  . apply add_lt_add_of_le_of_lt h1
    apply exp_lt_exp.mpr h2
  rfl

-- Lemas usados
-- ============

-- #check (add_lt_add_of_le_of_lt : a ≤ b → c < d → a + c < b + d)
-- #check (add_lt_add_of_lt_of_le : a < b → c ≤ d → a + c < b + d)
-- #check (add_lt_add_right : b < c → ∀ (a : ℝ), b + a < c + a)
-- #check (exp_lt_exp : exp a < exp b ↔ a < b)
-- #check (le_refl a : a ≤ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.5. En ℝ, si d ≤ f, entonces c + e^(a + d) ≤ c + e^(a + f)

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, c, d y f son números reales tales que
--    d ≤ f
-- entonces
--    c + exp (a + d) ≤ c + exp (a + f)
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================
```

```
-- 1ª demostración en LN
-- =====================

-- De la hipótesis, por la monotonia de la suma, se tiene
--    a + d ≤ a + f
-- que, por la monotonía de la exponencial, da
--    exp (a + d) ≤ exp (a + f)
-- y, por la monotonía de la suma, se tiene
--    c + exp (a + d) ≤ c + exp (a + f)

-- 2ª demostración en LN
-- =====================

-- Tenemos que demostrar que
--    c + exp (a + d) ≤ c + exp (a + f)
-- Por la monotonía de la suma, se reduce a
--    exp (a + d) ≤ exp (a + f)
-- que, por la monotonía de la exponencial, se reduce a
--    a + d ≤ a + f
-- que, por la monotonía de la suma, se reduce a
--    d ≤ f
-- que es la hipótesis.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Analysis.SpecialFunctions.Log.Basic
open Real
variable (a c d f : ℝ)

-- 1ª demostración
example
  (h : d ≤ f)
  : c + exp (a + d) ≤ c + exp (a + f) :=
by
  have h1 : a + d ≤ a + f :=
    add_le_add_left h a
  have h2 : exp (a + d) ≤ exp (a + f) :=
    exp_le_exp.mpr h1
  show c + exp (a + d) ≤ c + exp (a + f)
  exact add_le_add_left h2 c

-- 2ª demostración
example
  (h : d ≤ f)
```

```
  : c + exp (a + d) ≤ c + exp (a + f) :=
by
  apply add_le_add_left
  apply exp_le_exp.mpr
  apply add_le_add_left
  exact h

-- Lemas usados
-- ============

-- variable (b : ℝ)
-- #check (add_le_add_left : b ≤ c → ∀ (a : ℝ), a + b ≤ a + c)
-- #check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.6. En ℝ, si a ≤ b, entonces log(1+e^a) ≤ log(1+e^b)

```
-- ---------------------------------------------------------------------
-- Demostrar que si a y b son números reales tales que
--    a ≤ b
-- entonces
--    log(1+e^a) ≤ log(1+e^b)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la monotonía del logaritmo, basta demostrar que
--    0 < 1 + e^a                 (1)
--    1 + e^a ≤ 1 + e^b           (2)
--
-- La (1), por la suma de positivos, se reduce a
--    0 < 1                       (1.1)
--    0 < e^a                     (1.2)
-- La (1.1) es una propiedad de los números naturales y la (1.2) de la
-- función exponencial.
--
-- La (2), por la monotonía de la suma, se reduce a
--    e^a ≤ e^b
-- que, por la monotonía de la exponencial, se reduce a
--    a ≤ b
-- que es la hipótesis.
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Analysis.SpecialFunctions.Log.Basic
open Real
variable (a b : ℝ)

-- 1ª demostración
example
  (h : a ≤ b)
  : log (1 + exp a) ≤ log (1 + exp b) :=
by
  have h1 : (0 : ℝ) < 1 :=
    zero_lt_one
  have h2 : 0 < exp a :=
    exp_pos a
  have h3 : 0 < 1 + exp a :=
    add_pos h1 h2
  have h4 : exp a ≤ exp b :=
    exp_le_exp.mpr h
  have h5 : 1 + exp a ≤ 1 + exp b :=
    add_le_add_left h4 1
  show log (1 + exp a) ≤ log (1 + exp b)
  exact log_le_log' h3 h5

-- 2ª demostracipn
example
  (h : a ≤ b)
  : log (1 + exp a) ≤ log (1 + exp b) :=
by
  apply log_le_log'
  { apply add_pos
    { exact zero_lt_one }
    { exact exp_pos a }}
  { apply add_le_add_left
    exact exp_le_exp.mpr h }

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (add_le_add_left : b ≤ c → ∀ (a : ℝ), a + b ≤ a + c)
-- #check (add_pos : 0 < a → 0 < b → 0 < a + b)
-- #check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
```

```
-- #check (exp_pos a : 0 < exp a)
-- #check (log_le_log' : 0 < a → a ≤ b → log a ≤ log b)
-- #check (zero_lt_one : 0 < 1)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.7. En ℝ, si a ≤ b, entonces c - e^b ≤ c - e^a

```
-- ---------------------------------------------------------------------
-- Sean a, b y c números reales. Demostrar que si
--    a ≤ b
-- entonces
--    c - e^b ≤ c - e^a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Aplicando la monotonía de la exponencial a la hipótesis, se tiene
--    e^a ≤ e^b
-- y, restando de c, se invierte la desigualdad
--    c - e^b ≤ c - e^a

-- Demostraciones con Lean4
-- ========================

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b c : ℝ)

-- 1ª demostración
example
  (h : a ≤ b)
  : c - exp b ≤ c - exp a :=
by
   have h1 : exp a ≤ exp b :=
     exp_le_exp.mpr h
   show c - exp b ≤ c - exp a
   exact sub_le_sub_left h1 c

-- 2ª demostración
```

```
example
  (h : a ≤ b)
  : c - exp b ≤ c - exp a :=
by
   apply sub_le_sub_left _ c
   apply exp_le_exp.mpr h

-- 3ª demostración
example
  (h : a ≤ b)
  : c - exp b ≤ c - exp a :=
sub_le_sub_left (exp_le_exp.mpr h) c

-- 4ª demostración
example
  (h : a ≤ b)
  : c - exp b ≤ c - exp a :=
by linarith [exp_le_exp.mpr h]

-- Lemas usados
-- ============

-- #check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
-- #check (sub_le_sub_left : a ≤ b → ∀ (c : ℝ), c - b ≤ c - a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.8. En $\mathbb{R}$, $2ab \leq a^2 + b^2$

```
-- ---------------------------------------------------------------------
-- Sean a y b números reales. Demostrar que
--    2ab ≤ a² + b²
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Puesto que los cuadrados son positivos, se tiene
--    (a - b)² ≥ 0
-- Desarrollando el cuadrado, se obtiene
--    a² - 2ab + b² ≥ 0
-- Sumando 2ab a ambos lados, queda
--    a² + b² ≥ 2ab
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- 1ª demostración
example : 2*a*b ≤ a^2 + b^2 :=
by
  have h1 : 0 ≤ (a - b)^2          := sq_nonneg (a - b)
  have h2 : 0 ≤ a^2 - 2*a*b + b^2 := by linarith only [h1]
  show 2*a*b ≤ a^2 + b^2
  linarith

-- 2ª demostración
example : 2*a*b ≤ a^2 + b^2 :=
by
  have h : 0 ≤ a^2 - 2*a*b + b^2
  { calc a^2 - 2*a*b + b^2
         = (a - b)^2                   := (sub_sq a b).symm
       _ ≥ 0                           := sq_nonneg (a - b) }
  calc 2*a*b
       = 2*a*b + 0                     := (add_zero (2*a*b)).symm
     _ ≤ 2*a*b + (a^2 - 2*a*b + b^2) := add_le_add (le_refl _) h
     _ = a^2 + b^2                     := by ring

-- 3ª demostración
example : 2*a*b ≤ a^2 + b^2 :=
by
  have h : 0 ≤ a^2 - 2*a*b + b^2
  { calc a^2 - 2*a*b + b^2
         = (a - b)^2         := (sub_sq a b).symm
       _ ≥ 0                 := sq_nonneg (a - b) }
  linarith only [h]

-- 4ª demostración
example : 2*a*b ≤ a^2 + b^2 :=
-- by apply?
two_mul_le_add_sq a b

-- Lemas usados
-- ============
```

```
-- variable (c : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
-- #check (add_zero a : a + 0 = a)
-- #check (sq_nonneg a : 0 ≤ a ^ 2)
-- #check (sub_sq a b : (a - b) ^ 2 = a ^ 2 - 2 * a * b + b ^ 2)
-- #check (two_mul_le_add_sq a b : 2 * a * b ≤ a ^ 2 + b ^ 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.9. En ℝ, |ab| ≤ (a²+b²)/2]

```
-- ---------------------------------------------------------------------
-- Sean a y b números reales. Demostrar que
--    |a*b| ≤ (a^2 + b^2) / 2
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Para demostrar
--    |ab| ≤ (a² + b² / 2
-- basta demostrar estas dos desigualdades
--    ab ≤ (a² + b²) / 2                                            (1)
--    -(ab) ≤ (a² + b²) / 2                                         (2)
--
-- Para demostrar (1) basta demostrar que
--    2ab ≤ a² + b²
-- que se prueba como sigue. En primer lugar, como los cuadrados son no
-- negativos, se tiene
--   (a - b)² ≥ 0
-- Desarrollando el cuandrado,
--   a² - 2ab + b² ≥ 0
-- Sumando 2ab,
--   a² + b² ≥ 2ab
--
-- Para demostrar (2) basta demostrar que
--    -2ab ≤ a² + b²
-- que se prueba como sigue. En primer lugar, como los cuadrados son no
-- negativos, se tiene
--   (a + b)² ≥ 0
-- Desarrollando el cuandrado,
--   a² + 2ab + b² ≥ 0
-- Restando 2ab,
```

```
--   a² + b² ≥ -2ab

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lemas auxiliares
-- ================

lemma aux1 : a * b * 2 ≤ a ^ 2 + b ^ 2 := by
  have h : 0 ≤ a ^ 2 - 2 * a * b + b ^ 2
  calc
    a ^ 2 - 2 * a * b + b ^ 2
      = (a - b) ^ 2              := by ring
    _ ≥ 0                        := pow_two_nonneg (a - b)
  linarith only [h]

lemma aux2 : -(a * b) * 2 ≤ a ^ 2 + b ^ 2 := by
  have h : 0 ≤ a ^ 2 + 2 * a * b + b ^ 2
  calc
    a ^ 2 + 2 * a * b + b ^ 2
      = (a + b) ^ 2              := by ring
    _ ≥ 0                        := pow_two_nonneg (a + b)
  linarith only [h]

-- 1ª demostración
-- ===============

example : |a * b| ≤ (a ^ 2 + b ^ 2) / 2 := by
  have h : (0 : ℝ) < 2 := by norm_num
  apply abs_le'.mpr
  constructor
  { have h1 : a * b * 2 ≤ a ^ 2 + b ^ 2 := aux1 a b
    show a * b ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff h).mpr h1 }
  { have h2 : -(a * b) * 2 ≤ a ^ 2 + b ^ 2 := aux2 a b
    show -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff h).mpr h2 }

-- 2ª demostración
-- ===============
```

```
example : |a * b| ≤ (a ^ 2 + b ^ 2) / 2 := by
  have h : (0 : ℝ) < 2 := by norm_num
  apply abs_le'.mpr
  constructor
  { exact (le_div_iff h).mpr (aux1 a b) }
  { exact (le_div_iff h).mpr (aux2 a b) }

-- 3ª demostración
-- ===============

example : |a * b| ≤ (a ^ 2 + b ^ 2) / 2 := by
  have h : (0 : ℝ) < 2 := by norm_num
  apply abs_le'.mpr
  constructor
  { rw [le_div_iff h]
    apply aux1 }
  { rw [le_div_iff h]
    apply aux2 }

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (abs_le' : |a| ≤ b ↔ a ≤ b ∧ -a ≤ b)
-- #check (le_div_iff : 0 < c → (a ≤ b / c ↔ a * c ≤ b))
-- #check (pow_two_nonneg a : 0 ≤ a ^ 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.10. En ℝ, min(a,b) = min(b,a)

```
-- ---------------------------------------------------------------------
-- Sean a y b números reales. Demostrar que
--    min a b = min b a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia de la siguiente propiedad
--    min(a, b) ≤ min(b, a)                                          (1)
-- En efecto, intercambiando las variables en (1) se obtiene
--    min(b, a) ≤ min(a, b)                                          (2)
```

```
-- Finalmente de (1) y (2) se obtiene
--    min(b, a) = min(a, b)
--
-- Para demostrar (1), se observa que
--    min(a, b) ≤ b
--    min(a, b) ≤ a
-- y, por tanto,
--    min(a, b) = min(b, a)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lema auxiliar
-- =============

-- 1ª demostración del lema auxiliar
-- =================================

example : min a b ≤ min b a :=
by
  have h1 : min a b ≤ b := min_le_right a b
  have h2 : min a b ≤ a := min_le_left a b
  show min a b ≤ min b a
  exact le_min h1 h2

-- 2ª demostración del lema auxiliar
-- =================================

example : min a b ≤ min b a :=
by
  apply le_min
  { apply min_le_right }
  { apply min_le_left }

-- 3ª demostración del lema auxiliar
-- =================================

lemma aux : min a b ≤ min b a :=
by exact le_min (min_le_right a b) (min_le_left a b)

-- 1ª demostración
```

```
-- ===============

example : min a b = min b a :=
by
  apply le_antisymm
  { exact aux a b}
  { exact aux b a}

-- 2ª demostración
-- ===============

example : min a b = min b a :=
le_antisymm (aux a b) (aux b a)

-- 3ª demostración
-- ===============

example : min a b = min b a :=
min_comm a b

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (le_antisymm : a ≤ b → b ≤ a → a = b)
-- #check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
-- #check (min_comm a b : min a b = min b a)
-- #check (min_le_left a b : min a b ≤ a)
-- #check (min_le_right a b : min a b ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.11. En ℝ, max(a,b) = max(b,a)

```
-- ---------------------------------------------------------------------
-- Sean a y b números reales. Demostrar que
--    max a b = max b a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia de la siguiente propiedad
```

```
--    max(a, b) ≤ max(b, a)                                         (1)
-- En efecto, intercambiando las variables en (1) se obtiene
--    max(b, a) ≤ max(a, b)                                         (2)
-- Finalmente de (1) y (2) se obtiene
--    max(b, a) = max(a, b)
--
-- Para demostrar (1), se observa que
--    a ≤ max(b, a)
--    b ≤ max(b, a)
-- y, por tanto,
--    max(a, b) ≤ max(b, a)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lema auxiliar
-- =============

-- 1ª demostración del lema auxiliar
-- =================================

example : max a b ≤ max b a :=
by
  have h1 : a ≤ max b a := le_max_right b a
  have h2 : b ≤ max b a := le_max_left b a
  show max a b ≤ max b a
  exact max_le h1 h2

-- 2ª demostración del lema auxiliar
-- =================================

example : max a b ≤ max b a :=
by
  apply max_le
  { apply le_max_right }
  { apply le_max_left }

-- 3ª demostración del lema auxiliar
-- =================================

lemma aux : max a b ≤ max b a :=
```

```
by exact max_le (le_max_right b a) (le_max_left b a)

-- 1ª demostración
-- ===============

example : max a b = max b a :=
by
  apply le_antisymm
  { exact aux a b}
  { exact aux b a}

-- 2ª demostración
-- ===============

example : max a b = max b a :=
le_antisymm (aux a b) (aux b a)

-- 3ª demostración
-- ===============

example : max a b = max b a :=
max_comm a b

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (le_antisymm : a ≤ b → b ≤ a → a = b)
-- #check (le_max_left a b : a ≤ max a b)
-- #check (le_max_right a b : b ≤ max a b)
-- #check (max_comm a b : max a b = max b a)
-- #check (max_le : a ≤ c → b ≤ c → max a b ≤ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.12. En ℝ, min(min(a,b),c) = min(a,min(b,c))

```
-- ---------------------------------------------------------------------
-- Sean a, b y c números reales. Demostrar que
--    min (min a b) c = min a (min b c)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- =================================

-- Por la propiedad antisimétrica, la igualdad es consecuencia de las
-- siguientes desigualdades
--    min(min(a, b), c) ≤ min(a, min(b, c))                           (1)
--    min(a, min(b, c)) ≤ min(min(a, b), c)                           (2)
--
-- La (1) es consecuencia de las siguientes desigualdades
--    min(min(a, b), c) ≤ a                                          (1a)
--    min(min(a, b), c) ≤ b                                          (1b)
--    min(min(a, b), c) ≤ c                                          (1c)
-- En efecto, de (1b) y (1c) se obtiene
--    min(min(a, b), c) ≤ min(b,c)
-- que, junto con (1a) da (1).
--
-- La (2) es consecuencia de las siguientes desigualdades
--    min(a, min(b, c)) ≤ a                                          (2a)
--    min(a, min(b, c)) ≤ b                                          (2b)
--    min(a, min(b, c)) ≤ c                                          (2c)
-- En efecto, de (2a) y (2b) se obtiene
--    min(a, min(b, c)) ≤ min(a, b)
-- que, junto con (2c) da (2).
--
-- La demostración de (1a) es
--    min(min(a, b), c) ≤ min(a, b) ≤ a
-- La demostración de (1b) es
--    min(min(a, b), c) ≤ min(a, b) ≤ b
-- La demostración de (2b) es
--    min(a, min(b, c)) ≤ min(b, c) ≤ b
-- La demostración de (2c) es
--    min(a, min(b, c)) ≤ min(b, c) ≤ c
-- La (1c) y (2a) son inmediatas.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable {a b c : ℝ}

-- Lemas auxiliares
-- ================

lemma aux1a : min (min a b) c ≤ a :=
calc min (min a b) c
```

```
     ≤ min a b := by exact min_le_left (min a b) c
   _ ≤ a       := min_le_left a b

lemma aux1b : min (min a b) c ≤ b :=
calc min (min a b) c
     ≤ min a b := by exact min_le_left (min a b) c
   _ ≤ b       := min_le_right a b

lemma aux1c : min (min a b) c ≤ c :=
by exact min_le_right (min a b) c

-- 1ª demostración del lema aux1
lemma aux1 : min (min a b) c ≤ min a (min b c) :=
by
  apply le_min
  { show min (min a b) c ≤ a
    exact aux1a }
  { show min (min a b) c ≤ min b c
    apply le_min
    { show min (min a b) c ≤ b
      exact aux1b }
    { show min (min a b) c ≤ c
      exact aux1c }}

-- 2ª demostración del lema aux1
lemma aux1' : min (min a b) c ≤ min a (min b c) :=
le_min aux1a (le_min aux1b aux1c)

lemma aux2a : min a (min b c) ≤ a :=
by exact min_le_left a (min b c)

lemma aux2b : min a (min b c) ≤ b :=
calc min a (min b c)
     ≤ min b c        := by exact min_le_right a (min b c)
   _ ≤ b              := min_le_left b c

lemma aux2c : min a (min b c) ≤ c :=
calc min a (min b c)
     ≤ min b c        := by exact min_le_right a (min b c)
   _ ≤ c              := min_le_right b c

-- 1ª demostración del lema aux2
lemma aux2 : min a (min b c) ≤ min (min a b) c :=
by
  apply le_min
```

```
  { show min a (min b c) ≤ min a b
    apply le_min
    { show min a (min b c) ≤ a
      exact aux2a }
    { show min a (min b c) ≤ b
      exact aux2b }}
  { show min a (min b c) ≤ c
    exact aux2c }

-- 2ª demostración del lema aux2
lemma aux2' : min a (min b c) ≤ min (min a b) c :=
le_min (le_min aux2a aux2b) aux2c

-- 1ª demostración
-- ===============

example :
  min (min a b) c = min a (min b c) :=
by
  apply le_antisymm
  { show min (min a b) c ≤ min a (min b c)
    exact aux1 }
  { show min a (min b c) ≤ min (min a b) c
    exact aux2 }

-- 2ª demostración
-- ===============

example : min (min a b) c = min a (min b c) :=
by
  apply le_antisymm
  { exact aux1 }
  { exact aux2 }

-- 3ª demostración
-- ===============

example : min (min a b) c = min a (min b c) :=
le_antisymm aux1 aux2

-- 4ª demostración
-- ===============

example : min (min a b) c = min a (min b c) :=
min_assoc a b c
```

```
-- Lemas usados
-- ============

-- #check (le_antisymm : a ≤ b → b ≤ a → a = b)
-- #check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
-- #check (min_assoc a b c : min (min a b) c = min a (min b c))
-- #check (min_le_left a b : min a b ≤ a)
-- #check (min_le_right a b : min a b ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.13. En ℝ, min(a,b)+c = min(a+c,b+c)

```
-- ---------------------------------------------------------------------
-- Sean a, b y c números reales. Demostrar que
--    min a b + c = min (a + c) (b + c)
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Aplicando la propiedad antisimétrica a las siguientes desigualdades
--    min(a, b) + c ≤ min(a + c, b + c)                               (1)
--    min(a + c, b + c) ≤ min(a, b) + c                               (2)
--
-- Para demostrar (1) basta demostrar que se verifican las siguientes
-- desigualdades
--    min(a, b) + c ≤ a + c                                          (1a)
--    min(a, b) + c ≤ b + c                                          (1b)
-- que se tienen porque se verifican las siguientes desigualdades
--    min(a, b) ≤ a
--    min(a, b) ≤ b
--
-- Para demostrar (2) basta demostrar que se verifica
--    min(a + c, b + c) - c ≤ min(a, b)
-- que se demuestra usando (1); en efecto,
--    min(a + c, b + c) - c ≤ min(a + c - c, b + c - c)    [por (1)]
--                          = min(a, b)
```

```
-- 2ª demostración en LN
-- =====================

-- Por casos según a ≤ b.
--
-- 1º caso: Supongamos que a ≤ b. Entonces,
--    min(a, b) + c = a + c
--                  = min(a + c, b + c)
--
-- 2º caso: Supongamos que a ≰ b. Entonces,
--    min(a, b) + c = b + c
--                  = min(a + c, b + c)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable {a b c : ℝ}

-- En las demostraciones se usarán los siguientes lemas auxiliares
--    aux1 : min a b + c ≤ min (a + c) (b + c)
--    aux2 : min (a + c) (b + c) ≤ min a b + c
-- cuyas demostraciones se exponen a continuación.

-- 1ª demostración de aux1
lemma aux1 :
  min a b + c ≤ min (a + c) (b + c) :=
by
  have h1 : min a b ≤ a :=
    min_le_left a b
  have h2 : min a b + c ≤ a + c :=
    add_le_add_right h1 c
  have h3 : min a b  ≤ b :=
    min_le_right a b
  have h4 : min a b + c ≤ b + c :=
    add_le_add_right h3 c
  show min a b + c ≤ min (a + c) (b + c)
  exact le_min h2 h4

-- 2ª demostración de aux1
example :
  min a b + c ≤ min (a + c) (b + c) :=
by
  apply le_min
```

```
  { apply add_le_add_right
    exact min_le_left a b }
  { apply add_le_add_right
    exact min_le_right a b }

-- 3ª demostración de aux1
example :
  min a b + c ≤ min (a + c) (b + c) :=
le_min (add_le_add_right (min_le_left a b) c)
       (add_le_add_right (min_le_right a b) c)

-- 1ª demostración de aux2
lemma aux2 :
  min (a + c) (b + c) ≤ min a b + c :=
by
  have h1 : min (a + c) (b + c) + -c ≤ min a b
  { calc min (a + c) (b + c) + -c
         ≤ min (a + c + -c) (b + c + -c) := aux1
       _ = min a b                       := by ring_nf }
  show min (a + c) (b + c) ≤ min a b + c
  exact add_neg_le_iff_le_add.mp h1

-- 1ª demostración del ejercicio
example :
  min a b + c = min (a + c) (b + c) :=
by
  have h1 : min a b + c ≤ min (a + c) (b + c) := aux1
  have h2 : min (a + c) (b + c) ≤ min a b + c := aux2
  show min a b + c = min (a + c) (b + c)
  exact le_antisymm h1 h2

-- 2ª demostración del ejercicio
example :
  min a b + c = min (a + c) (b + c) :=
by
  apply le_antisymm
  { show min a b + c ≤ min (a + c) (b + c)
    exact aux1 }
  { show min (a + c) (b + c) ≤ min a b + c
    exact aux2 }

-- 3ª demostración del ejercicio
example :
  min a b + c = min (a + c) (b + c) :=
by
```

```
  apply le_antisymm
  { exact aux1 }
  { exact aux2 }

-- 4ª demostración del ejercicio
example :
  min a b + c = min (a + c) (b + c) :=
le_antisymm aux1 aux2

-- 5ª demostración del ejercicio
example : min a b + c = min (a + c) (b + c) :=
by
  by_cases h : a ≤ b
  { have h1 : a + c ≤ b + c := add_le_add_right h c
    calc min a b + c = a + c               := by simp [min_eq_left h]
                   _ = min (a + c) (b + c) := by simp [min_eq_left h1]}
  { have h2: b ≤ a := le_of_not_le h
    have h3 : b + c ≤ a + c := add_le_add_right h2 c
    calc min a b + c = b + c               := by simp [min_eq_right h2]
                   _ = min (a + c) (b + c) := by simp [min_eq_right h3]}

-- 6ª demostración del ejercicio
example : min a b + c = min (a + c) (b + c) :=
(min_add_add_right a b c).symm

-- Lemas usados
-- ============

-- #check (add_le_add_right : b ≤ c → ∀ (a : ℝ),  b + a ≤ c + a)
-- #check (add_neg_le_iff_le_add : a - b ≤ c ↔ a ≤ c + b)
-- #check (le_antisymm : a ≤ b → b ≤ a → a = b)
-- #check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
-- #check (min_add_add_right a b c : min (a + c) (b + c) = min a b + c)
-- #check (min_eq_left : a ≤ b → min a b = a)
-- #check (min_eq_right : b ≤ a → min a b = b)
-- #check (min_le_left a b : min a b ≤ a)
-- #check (min_le_right a b : min a b ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.14. En ℝ, |a| - |b| ≤ |a - b|

```
-- ---------------------------------------------------------------------
-- Sean a y b números reales. Demostrar que
--    |a| - |b| ≤ |a - b|
-- ----------------------------------------------------------------------

-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Por la siguiente cadena de desigualdades
--    |a| - |b| = |a - b + b| - |b|
--              ≤ (|a - b| + |b|) - |b|   [por la desigualdad triangular]
--              = |a - b|

-- 2ª demostración en LN
-- =====================

-- Por la desigualdad triangular
--    |a - b + b| ≤ |a - b| + |b|
-- simplificando en la izquierda
--    |a| ≤ |a - b| + |b|
-- y, pasando |b| a la izquierda
--    |a| - |b| ≤ |a - b|

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- 1ª demostración (basada en la 1ª en LN)
example : |a| - |b| ≤ |a - b| :=
calc |a| - |b|
     = |a - b + b| - |b| :=
          congrArg (fun x => |x| - |b|) (sub_add_cancel a b).symm
   _ ≤ (|a - b| + |b|) - |b| :=
           sub_le_sub_right (abs_add (a - b) b) (|b|)
   _ = |a - b| :=
          add_sub_cancel (|a - b|) (|b|)
```

```
-- 2ª demostración (basada en la 1ª en LN)
example : |a| - |b| ≤ |a - b| :=
calc |a| - |b|
     = |a - b + b| - |b| := by
          rw [sub_add_cancel]
   _ ≤ (|a - b| + |b|) - |b| := by
          apply sub_le_sub_right
          apply abs_add
   _ = |a - b| := by
          rw [add_sub_cancel]

-- 3ª demostración (basada en la 2ª en LN)
example : |a| - |b| ≤ |a - b| :=
by
  have h1 : |a - b + b| ≤ |a - b| + |b| := abs_add (a - b) b
  rw [sub_add_cancel] at h1
  exact abs_sub_abs_le_abs_sub a b

-- 4ª demostración
example : |a| - |b| ≤ |a - b| :=
abs_sub_abs_le_abs_sub a b

-- Lemas usados
-- ============

-- #check (abs_add a b : |a + b| ≤ |a| + |b|)
-- #check (abs_sub_abs_le_abs_sub a b : |a| - |b| ≤ |a - b|)
-- #check (add_sub_cancel a b : a + b - b = a)
-- #check (sub_add_cancel a b : a - b + b = a)
-- #check (sub_le_sub_right : a ≤ b → ∀ (c : ℝ), a - c ≤ b - c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.15. En ℝ, {0 <ε, ε ≤ 1, |x| <ε, |y| <ε} ⊢ |xy| <ε

```
-- ---------------------------------------------------------------------
-- Demostrar que para todos los números reales x, y, ε si
--    0 < ε
--    ε ≤ 1
--    |x| < ε
--    |y| < ε
```

```
-- entonces
--    |x * y| < ε
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas
--    abs_mul          : |a * b| = |a| * |b|
--    zero_mul         : 0 * a = 0
--    abs_nonneg a     : 0 ≤ |a|
--    lt_of_le_of_ne   : a ≤ b → a ≠ b → a < b
--    ne_comm          : a ≠ b ↔ b ≠ a
--    mul_lt_mul_left  : 0 < a → (a * b < a * c ↔ b < c)
--    mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c)
--    mul_le_mul_right : 0 < a → (b * a ≤ c * a ↔ b ≤ c)
--    one_mul          : 1 * a = a
--
-- Sean x y ε ∈ ℝ tales que
--    0 < ε                                                             (he1)
--    ε ≤ 1                                                             (he2)
--    |x| < ε                                                           (hx)
--    |y| < ε                                                           (hy)
-- y tenemos que demostrar que
--    |x * y| < ε
-- Lo haremos distinguiendo caso según |x| = 0.
--
-- 1º caso. Supongamos que
--    |x| = 0                                                            (1)
-- Entonces,
--    |x * y| = |x| * |y|    [por abs_mul]
--            = 0 * |y|      [por h1]
--            = 0            [por zero_mul]
--            < ε            [por he1]
--
-- 2º caso. Supongamos que
--    |x| ≠ 0                                                            (2)
-- Entonces, por lt_of_le_of_ne, abs_nonneg y ne_comm, se tiene
--    0 < x                                                              (3)
-- y, por tanto,
--    |x * y| = |x| * |y|    [por abs_mul]
--            < |x| * ε      [por mul_lt_mul_left, (3) y (hy)]
--            < ε * ε        [por mul_lt_mul_right, (he1) y (hx)]
--            ≤ 1 * ε        [por mul_le_mul_right, (he1) y (he2)]
--            = ε            [por one_mul]
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- 1ª demostración
-- ===============

example :
  ∀ {x y ε : ℝ}, 0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros x y ε he1 he2 hx hy
  by_cases h : (|x| = 0)
  . -- h : |x| = 0
    show |x * y| < ε
    calc
      |x * y|
         = |x| * |y| := abs_mul x y
      _  = 0 * |y|   := by rw [h]
      _  = 0         := zero_mul (abs y)
      _  < ε         := he1
  . -- h : ¬|x| = 0
    have h1 : 0 < |x| := by
      have h2 : 0 ≤ |x| := abs_nonneg x
      show 0 < |x|
      exact lt_of_le_of_ne h2 (ne_comm.mpr h)
    show |x * y| < ε
    calc |x * y|
         = |x| * |y| := abs_mul x y
       _ < |x| * ε   := (mul_lt_mul_left h1).mpr hy
       _ < ε * ε     := (mul_lt_mul_right he1).mpr hx
       _ ≤ 1 * ε     := (mul_le_mul_right he1).mpr he2
       _ = ε         := one_mul ε

-- 2ª demostración
-- ===============

example :
  ∀ {x y ε : ℝ}, 0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros x y ε he1 he2 hx hy
  by_cases (|x| = 0)
  . -- h : |x| = 0
    show |x * y| < ε
```

```
    calc
      |x * y| = |x| * |y| := by apply abs_mul
            _ = 0 * |y|   := by rw [h]
            _ = 0         := by apply zero_mul
            _ < ε         := by apply he1
  . -- h : ¬|x| = 0
    have h1 : 0 < |x| := by
      have h2 : 0 ≤ |x| := by apply abs_nonneg
      exact lt_of_le_of_ne h2 (ne_comm.mpr h)
    show |x * y| < ε
    calc
      |x * y| = |x| * |y| := by rw [abs_mul]
            _ < |x| * ε   := by apply (mul_lt_mul_left h1).mpr hy
            _ < ε * ε     := by apply (mul_lt_mul_right he1).mpr hx
            _ ≤ 1 * ε     := by apply (mul_le_mul_right he1).mpr he2
            _ = ε         := by rw [one_mul]

-- 3ª demostración
-- ===============

example :
  ∀ {x y ε : ℝ}, 0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros x y ε he1 he2 hx hy
  by_cases (|x| = 0)
  . -- h : |x| = 0
    show |x * y| < ε
    calc |x * y| = |x| * |y| := by simp only [abs_mul]
               _ = 0 * |y|   := by simp only [h]
               _ = 0         := by simp only [zero_mul]
               _ < ε         := by simp only [he1]
  . -- h : ¬|x| = 0
    have h1 : 0 < |x| := by
      have h2 : 0 ≤ |x| := by simp only [abs_nonneg]
      exact lt_of_le_of_ne h2 (ne_comm.mpr h)
    show |x * y| < ε
    calc
      |x * y| = |x| * |y| := by simp [abs_mul]
            _ < |x| * ε   := by simp only [mul_lt_mul_left, h1, hy]
            _ < ε * ε     := by simp only [mul_lt_mul_right, he1, hx]
            _ ≤ 1 * ε     := by simp only [mul_le_mul_right, he1, he2]
            _ = ε         := by simp only [one_mul]

-- Lemas usados
-- ============
```

```
-- variable (a b c : ℝ)
-- #check (abs_mul a b : |a * b| = |a| * |b|)
-- #check (abs_nonneg a : 0 ≤ |a|)
-- #check (lt_of_le_of_ne : a ≤ b → a ≠ b → a < b)
-- #check (mul_le_mul_right : 0 < a → (b * a ≤ c * a ↔ b ≤ c))
-- #check (mul_lt_mul_left : 0 < a → (a * b < a * c ↔ b < c))
-- #check (mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c))
-- #check (ne_comm : a ≠ b ↔ b ≠ a)
-- #check (one_mul a : 1 * a = a)
-- #check (zero_mul a : 0 * a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.16. En ℝ, a <b → ¬(b <a)

```
-- ---------------------------------------------------------------------
-- Demostrar que para todo par de numero reales a y b, si a < b entonces
-- no se tiene que b < a.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por hipótesis a < b y tenemos que demostrar que ¬(b < a). Supongamos
-- que b < a. Entonces, por la propiedad transiva a < a que es una
-- contradicción con la propiedad irreflexiva.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- 1ª demostración
example
  (h : a < b)
  : ¬ b < a :=
by
  intro h1
  -- h1 : b < a
  -- ⊢ False
```

```
  have : a < a := lt_trans h h1
  apply lt_irrefl a this

-- 2ª demostración
example
  (h : a < b)
  : ¬ b < a :=
by
  intro h1
  -- h1 : b < a
  -- ⊢ False
  exact lt_irrefl a (lt_trans h h1)

-- 3ª demostración
example
  (h : a < b)
  : ¬ b < a :=
fun h1 ↦ lt_irrefl a (lt_trans h h1)

-- 4ª demostración
example
  (h : a < b)
  : ¬ b < a :=
lt_asymm h

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (lt_asymm : a < b → ¬b < a)
-- #check (lt_irrefl a : ¬a < a)
-- #check (lt_trans : a < b → b < c → a < c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.17. Hay algún número real entre 2 y 3

```
-- ---------------------------------------------------------------------
-- Demostrar que hay algún número real entre 2 y 3.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Puesto que 2 < 5/2 < 3, basta elegir 5/2.

-- Demostracione con Lean4
-- =======================

import Mathlib.Data.Real.Basic

-- 1ª demostración
example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  have h : 2 < (5 : ℝ) / 2 ∧ (5 : ℝ) / 2 < 3 :=
    by norm_num
  show ∃ x : ℝ, 2 < x ∧ x < 3
  exact Exists.intro (5 / 2) h

-- 2ª demostración
example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  have h : 2 < (5 : ℝ) / 2 ∧ (5 : ℝ) / 2 < 3 :=
    by norm_num
  show ∃ x : ℝ, 2 < x ∧ x < 3
  exact ⟨5 / 2, h⟩

-- 3ª demostración
example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  use 5 / 2
  norm_num

-- 4ª demostración
example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
⟨5 / 2, by norm_num⟩
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 6.18. Si (∀ε >0)[x ≤ ε], entonces x ≤ 0

```
-- ---------------------------------------------------------------------
-- Sea x un número real tal que para todo número positivo ε, x ≤ ε
-- Demostrar que x ≤ 0.
-- ----------------------------------------------------------------------
```

```
-- Demostración en lenguaje natural
-- ================================

-- Basta demostrar que x ≯ 0. Para ello, supongamos que x > 0 y vamos a
-- demostrar que
--    ¬(∀ε)[ε > 0 → x ≤ ε]                                       (1)
-- que es una contradicción con la hipótesis. Interiorizando la
-- negación, (1) es equivalente a
--    (∃ε)[ε > 0 ∧ ε < x]                                        (2)
-- Para demostrar (2) se puede elegir ε = x/2 ya que, como x > 0, se
-- tiene
--    0 < x/2 < x.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (x : ℝ)

-- 1ª demostración
-- ===============

example
  (h : ∀ ε > 0, x ≤ ε)
  : x ≤ 0 :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > 0
  intro hx0
  -- hx0 : x > 0
  -- ⊢ False
  apply absurd h
  -- ⊢ ¬∀ (ε : ℝ), ε > 0 → x ≤ ε
  push_neg
  -- ⊢ ∃ ε, ε > 0 ∧ ε < x
  use x /2
  -- ⊢ x / 2 > 0 ∧ x / 2 < x
  constructor
  { show x / 2 > 0
    exact half_pos hx0 }
  { show x / 2 < x
    exact half_lt_self hx0 }

-- 2ª demostración
```

```
-- ===============

example
  (x : ℝ)
  (h : ∀ ε > 0, x ≤ ε)
  : x ≤ 0 :=
by
  contrapose! h
  -- ⊢ ∃ ε, ε > 0 ∧ ε < x
  use x / 2
  -- ⊢ x / 2 > 0 ∧ x / 2 < x
  constructor
  { show x / 2 > 0
    exact half_pos h }
  { show x / 2 < x
    exact half_lt_self h }

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- variable (p q : Prop)
-- #check (le_of_not_gt : ¬a > b → a ≤ b)
-- #check (half_lt_self : 0 < a → a / 2 < a)
-- #check (half_pos : 0 < a → 0 < a / 2)
-- #check (absurd : p → ¬p → q)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 7

# Divisibilidad

## 7.1. Si x,y,z ∈ ℕ, entonces x | yxz

```
-- ---------------------------------------------------------------------
-- Demostrar que si x, y, z ∈ ℕ, entonces
--    x | y * x * z
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la transitividad de la divisibilidad aplicada a las relaciones
--    x | yx
--    yx | yxz

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (x y z : ℕ)

-- 1ª demostración
-- ===============

example : x ∣ y * x * z :=
by
  have h1 : x ∣ y * x :=
    dvd_mul_left x y
  have h2 : (y * x) ∣ (y * x * z) :=
    dvd_mul_right (y * x) z
  show x ∣ y * x * z
```

```
  exact dvd_trans h1 h2

-- 2ª demostración
-- ===============

example : x ∣ y * x * z :=
dvd_trans (dvd_mul_left x y) (dvd_mul_right (y * x) z)

-- 3ª demostración
-- ===============

example : x ∣ y * x * z :=
by
  apply dvd_mul_of_dvd_left
  apply dvd_mul_left

-- Lemas usados
-- ============

-- #check (dvd_mul_left x y : x ∣ y * x)
-- #check (dvd_mul_right x y : x ∣ x * y)
-- #check (dvd_trans : x ∣ y → y ∣ z → x ∣ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 7.2. Si x divide a w, también divide a y(xz)+x²+w²

```
-- ------------------------------------------------------------------
-- Demostrar que si
--    x ∣ w
-- entonces
--    x ∣ y * (x * z) + x^2 + w^2
-- -------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la divisibilidad de la suma basta probar que
--    x ∣ yxz                                                    (1)
--    x ∣ x²                                                     (2)
--    x ∣ w²                                                     (3)
--
-- Para demostrar (1), por la divisibilidad del producto se tiene
```

```
--    x | xz
-- y, de nuevo por la divisibilidad del producto,
--    x | y(xz).
--
-- La propiedad (2) se tiene por la definición de cuadrado y la
-- divisibilidad del producto.
--
-- La propiedad (3) se tiene por la definición de cuadrado, la hipótesis
-- y la divisibilidad del producto.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (w x y z : ℕ)

-- 1ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  have h1 : x ∣ x * z :=
    dvd_mul_right x z
  have h2 : x ∣ y * (x * z) :=
    dvd_mul_of_dvd_right h1 y
  have h3 : x ∣ x^2 := by
    apply dvd_mul_left
  have h4 : x ∣ w * w :=
    dvd_mul_of_dvd_left h w
  have h5 : x ∣ w^2 := by
    rwa [← pow_two w] at h4
  have h6 : x ∣ y * (x * z) + x^2 :=
    dvd_add h2 h3
  show x ∣ y * (x * z) + x^2 + w^2
  exact dvd_add h6 h5

-- 2ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  apply dvd_add
  { apply dvd_add
    { apply dvd_mul_of_dvd_right
      apply dvd_mul_right }
```

```
    { rw [pow_two]
      apply dvd_mul_right }}
  { rw [pow_two]
    apply dvd_mul_of_dvd_left h }

-- 3ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  repeat' apply dvd_add
  { apply dvd_mul_of_dvd_right
    apply dvd_mul_right }
  { rw [pow_two]
    apply dvd_mul_right }
  { rw [pow_two]
    apply dvd_mul_of_dvd_left h }

-- Lemas usados
-- ============

-- #check (dvd_add : x ∣ y → x ∣ z → x ∣ y + z)
-- #check (dvd_mul_left x y : x ∣ y * x)
-- #check (dvd_mul_right x y : x ∣ x * y)
-- #check (dvd_mul_of_dvd_left : x ∣ y → ∀ (c : ℕ), x ∣ y * c)
-- #check (dvd_mul_of_dvd_right : x ∣ y → ∀ (c : ℕ), x ∣ c * y)
-- #check (pow_two x : x ^ 2 = x * x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 7.3. Transitividad de la divisibilidad

```
-- ---------------------------------------------------------------------
-- Demostrar que la relación de divisibilidad es transitiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que a | b y b | c. Entonces, existen d y e tales que
--    b = ad                                                        (1)
--    c = be                                                        (2)
-- Por tanto,
```

```
--    c = be        [por (2)]
--      = (ad)e     [por (1)]
--      = a(de)
-- Por consiguiente, a | c.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

variable {a b c : ℕ}

-- 1ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  rcases divab with ⟨d, beq : b = a * d⟩
  rcases divbc with ⟨e, ceq : c = b * e⟩
  have h1 : c = a * (d * e) :=
    calc c = b * e       := ceq
        _ = (a * d) * e := congrArg (. * e) beq
        _ = a * (d * e) := mul_assoc a d e
  show a ∣ c
  exact Dvd.intro (d * e) h1.symm

-- 2ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  rcases divab with ⟨d, beq : b = a * d⟩
  rcases divbc with ⟨e, ceq : c = b * e⟩
  use (d * e)
  -- ⊢ c = a * (d * e)
  rw [ceq, beq]
  -- ⊢ (a * d) * e = a * (d * e)
  exact mul_assoc a d e

-- 3ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
```

```
  a ∣ c :=
by
  rcases divbc with ⟨e, rfl⟩
  -- ⊢ a ∣ b * e
  rcases divab with ⟨d, rfl⟩
  -- ⊢ a ∣ a * d * e
  use (d * e)
  -- ⊢ a * d * e = a * (d * e)
  ring

-- 4ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  cases' divab with d beq
  -- d : ℕ
  -- beq : b = a * d
  cases' divbc with e ceq
  -- e : ℕ
  -- ceq : c = b * e
  rw [ceq, beq]
  -- ⊢ a ∣ a * d * e
  use (d * e)
  -- ⊢ (a * d) * e = a * (d * e)
  exact mul_assoc a d e

-- 5ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by exact dvd_trans divab divbc

-- Lemas usados
-- ============

-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (Dvd.intro c : a * c = b → a ∣ b)
-- #check (dvd_trans : a ∣ b → b ∣ c → a ∣ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 7.4. Si a divide a b y a c, entonces divide a b+c

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es un divisor de b y de c, tambien lo es de b + c.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Puesto que a divide a b y a c, existen d y e tales que
--    b = ad                                                         (1)
--    c = ae                                                         (2)
-- Por tanto,
--    b + c = ad + c      [por (1)]
--          = ad + ae     [por (2)]
--          = a(d + e)    [por la distributiva]
-- Por consiguiente, a divide a b + c.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

variable {a b c : ℕ}

-- 1ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  have h1 : b + c = a * (d + e) :=
    calc b + c
         = (a * d) + c       := congrArg (. + c) beq
       _ = (a * d) + (a * e) := congrArg ((a * d) + .) ceq
       _ = a * (d + e)       := by rw [← mul_add]
  show a ∣ (b + c)
  exact Dvd.intro (d + e) h1.symm

-- 2ª demostración
example
  (h1 : a ∣ b)
```

```
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  have h1 : b + c = a * (d + e) := by linarith
  show a ∣ (b + c)
  exact Dvd.intro (d + e) h1.symm

-- 3ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  show a ∣ (b + c)
  exact Dvd.intro (d + e) (by linarith)

-- 4ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  cases' h1 with d beq
  -- d : ℕ
  -- beq : b = a * d
  cases' h2 with e ceq
  -- e : ℕ
  -- ceq : c = a * e
  rw [ceq, beq]
  -- ⊢ a ∣ a * d + a * e
  use (d + e)
  -- ⊢ a * d + a * e = a * (d + e)
  ring

-- 5ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, rfl⟩
```

```
  -- ⊢ a ∣ a * d + c
  rcases h2 with ⟨e, rfl⟩
  -- ⊢ a ∣ a * d + a * e
  use (d + e)
  -- ⊢ a * d + a * e = a * (d + e)
  ring

-- 6ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
dvd_add h1 h2

-- Lemas usados
-- ============

-- #check (Dvd.intro c : a * c = b → a ∣ b)
-- #check (dvd_add : a ∣ b →  a ∣ c → a ∣ (b + c))
-- #check (mul_add a b c : a * (b + c) = a * b + a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 7.5. Conmutatividad del máximo común divisor

```
-- ---------------------------------------------------------------------
-- Demostrar que si m y n son números naturales, entonces
--    gcd m n = gcd n m
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ x, y ∈ ℕ)[gcd(x,y) ∣ gcd(y,x)]                            (1)
-- En efecto, sustituyendo en (1) x por m e y por n, se tiene
--    gcd(m, n) ∣ gcd(n, m)                                        (2)
-- y sustituyendo en (1) x por n e y por m, se tiene
--    gcd(n, m) ∣ gcd(m, n)                                        (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
```

```
--    gcd(m, n) = gcd(n, m)
--
-- Para demostrar (1), por la definición del máximo común divisor, basta
-- demostrar las siguientes relaciones
--    gcd(m, n) | n
--    gcd(m, n) | m
-- y ambas se tienen por la definición del máximo común divisor.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (k m n : ℕ)

open Nat

-- 1ª demostración del lema auxiliar
lemma aux : gcd m n ∣ gcd n m :=
by
  have h1 : gcd m n ∣ n :=
    gcd_dvd_right m n
  have h2 : gcd m n ∣ m :=
    gcd_dvd_left m n
  show gcd m n ∣ gcd n m
  exact dvd_gcd h1 h2

-- 2ª demostración del lema auxiliar
example : gcd m n ∣ gcd n m :=
dvd_gcd (gcd_dvd_right m n) (gcd_dvd_left m n)

-- 1ª demostración
example : gcd m n = gcd n m :=
by
  have h1 : gcd m n ∣ gcd n m := aux m n
  have h2 : gcd n m ∣ gcd m n := aux n m
  show gcd m n = gcd n m
  exact _root_.dvd_antisymm h1 h2

-- 2ª demostración
example : gcd m n = gcd n m :=
by
  apply _root_.dvd_antisymm
  { exact aux m n }
  { exact aux n m }
```

```
-- 3ª demostración
example : gcd m n = gcd n m :=
_root_.dvd_antisymm (aux m n) (aux n m)

-- 4ª demostración
example : gcd m n = gcd n m :=
-- by apply?
gcd_comm m n

-- Lemas usados
-- ============

-- #check (_root_.dvd_antisymm : m ∣ n → n ∣ m → m = n)
-- #check (dvd_gcd : k ∣ m → k ∣ n → k ∣ gcd m n)
-- #check (gcd_comm m n : gcd m n = gcd n m)
-- #check (gcd_dvd_left  m n: gcd m n ∣ m)
-- #check (gcd_dvd_right m n : gcd m n ∣ n)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 8

# Retículos

## 8.1. En los retículos, x ⊓ y = y ⊓ x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--     x ⊓ y = y ⊓ x
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ a, b)[a ⊓ b ≤ b ⊓ a]                                       (1)
-- En efecto, sustituyendo en (1) a por x y b por y, se tiene
--    x ⊓ y ≤ y ⊓ x                                                 (2)
-- y sustituyendo en (1) a por y y b por x, se tiene
--    y ⊓ x ≤ x ⊓ y                                                 (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
--    x ⊓ y = y ⊓ x
--
-- Para demostrar (1), por la definición del ínfimo, basta demostrar
-- las siguientes relaciones
--    y ⊓ x ≤ x
--    y ⊓ x ≤ y
-- y ambas se tienen por la definición del ínfimo.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
```

```
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración del lema auxiliar
lemma aux : x ⊓ y ≤ y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y :=
    inf_le_right
  have h2 : x ⊓ y ≤ x :=
    inf_le_left
  show x ⊓ y ≤ y ⊓ x
  exact le_inf h1 h2

-- 2ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
by
  apply le_inf
  { apply inf_le_right }
  { apply inf_le_left }

-- 3ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
le_inf inf_le_right inf_le_left

-- 1ª demostración
example : x ⊓ y = y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y ⊓ x :=
    aux x y
  have h2 : y ⊓ x ≤ x ⊓ y :=
    aux y x
  show x ⊓ y = y ⊓ x
  exact le_antisymm h1 h2

-- 2ª demostración
example : x ⊓ y = y ⊓ x :=
by
  apply le_antisymm
  { apply aux }
  { apply aux }

-- 3ª demostración
example : x ⊓ y = y ⊓ x :=
le_antisymm (aux x y) (aux y x)
```

```
-- 4ª demostración
example : x ⊓ y = y ⊓ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊓ y = y ⊓ x :=
-- by apply?
inf_comm

-- Lemas usados
-- ============

-- #check (inf_comm : x ⊓ y = y ⊓ x)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_le_right : x ⊓ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.2. En los retículos, x ⊔ y = y ⊔ x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊔ y = y ⊔ x
-- para todo x e y en el retículo.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ a, b)[a ⊔ b ≤ b ⊔ a]                                       (1)
-- En efecto, sustituyendo en (1) a por x y b por y, se tiene
--    x ⊔ y ≤ y ⊔ x                                                 (2)
-- y sustituyendo en (1) a por y y b por x, se tiene
--    y ⊔ x ≤ x ⊔ y                                                 (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
--    x ⊔ y = y ⊔ x
--
-- Para demostrar (1), por la definición del supremo, basta demostrar
-- las siguientes relaciones
```

```
--    x ≤ y ⊔ x
--    y ≤ y ⊔ x
-- y ambas se tienen por la definición del supremo.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración del lema auxiliar
lemma aux : x ⊔ y ≤ y ⊔ x :=
by
  have h1 : x ≤ y ⊔ x :=
    le_sup_right
  have h2 : y ≤ y ⊔ x :=
    le_sup_left
  show x ⊔ y ≤ y ⊔ x
  exact sup_le h1 h2

-- 2ª demostración del lema auxiliar
example : x ⊔ y ≤ y ⊔ x :=
by
  apply sup_le
  { apply le_sup_right }
  { apply le_sup_left }

-- 3ª demostración del lema auxiliar
example : x ⊔ y ≤ y ⊔ x :=
sup_le le_sup_right le_sup_left

-- 1ª demostración
example : x ⊔ y = y ⊔ x :=
by
  have h1 : x ⊔ y ≤ y ⊔ x :=
    aux x y
  have h2 : y ⊔ x ≤ x ⊔ y :=
    aux y x
  show x ⊔ y = y ⊔ x
  exact le_antisymm h1 h2

-- 2ª demostración
example : x ⊔ y = y ⊔ x :=
by
```

```
  apply le_antisymm
  { apply aux }
  { apply aux }

-- 3ª demostración
example : x ⊔ y = y ⊔ x :=
le_antisymm (aux x y) (aux y x)

-- 4ª demostración
example : x ⊔ y = y ⊔ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊔ y = y ⊔ x :=
-- by apply?
sup_comm

-- Lemas usados
-- ============

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (sup_comm : x ⊔ y = y ⊔ x)
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.3. En los retículos, (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--     (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm : x ≤ y → y ≤ x → x = y
--    le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y
--    inf_le_left : x ⊓ y ≤ x
--    inf_le_right : x ⊓ y ≤ y)
```

```
--
-- Por le_antisym, es suficiente demostrar las siguientes relaciones:
--    (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)                                     (1)
--    x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z                                     (2)
--
-- Para demostrar (1), por le_inf, basta probar que
--    (x ⊓ y) ⊓ z ≤ x                                              (1a)
--    (x ⊓ y) ⊓ z ≤ y ⊓ z                                          (1b)
--
-- La (1a) se demuestra por la siguiente cadena de desigualdades
--    (x ⊓ y) ⊓ z ≤ x ⊓ y   [por inf_le_left]
--                ≤ x       [por inf_le_left]
--
-- Para demostrar (1b), por le_inf, basta probar que
--    (x ⊓ y) ⊓ z ≤ y                                              (1b1)
--    (x ⊓ y) ⊓ z ≤ z                                              (1b2)
--
-- La (1b1) se demuestra por la siguiente cadena de desigualdades
--    (x ⊓ y) ⊓ z ≤ x ⊓ y   [por inf_le_left]
--                ≤ y       [por inf_le_right]
--
-- La (1b2) se tiene por inf_le_right.
--
-- Para demostrar (2), por le_inf, basta probar que
--    x ⊓ (y ⊓ z) ≤ x ⊓ y                                          (2a)
--    x ⊓ (y ⊓ z) ≤ z                                              (2b)
--
-- Para demostrar (2a), por le_inf, basta probar que
--    x ⊓ (y ⊓ z) ≤ x                                              (2a1)
--    x ⊓ (y ⊓ z) ≤ y                                              (2a2)
--
-- La (2a1) se tiene por inf_le_left.
--
-- La (2a2) se demuestra por la siguiente cadena de desigualdades
--    x ⊓ (y ⊓ z) ≤ y ⊓ z   [por inf_le_right]
--                ≤ y       [por inf_le_left]
--
-- La (2b) se demuestra por la siguiente cadena de desigualdades
--    x ⊓ (y ⊓ z) ≤ y ⊓ z   [por inf_le_right]
--                ≤ z       [por inf_le_right]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
```

```
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  have h1 : (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z) := by
  { have h1a : (x ⊓ y) ⊓ z ≤ x := calc
      (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
                _ ≤ x     := by exact inf_le_left
    have h1b : (x ⊓ y) ⊓ z ≤ y ⊓ z := by
    { have h1b1 : (x ⊓ y) ⊓ z ≤ y := calc
        (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
                  _ ≤ y     := by exact inf_le_right
      have h1b2 : (x ⊓ y) ⊓ z ≤ z :=
        inf_le_right
      show (x ⊓ y) ⊓ z ≤ y ⊓ z
      exact le_inf h1b1 h1b2 }
    show (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)
    exact le_inf h1a h1b }
  have h2 : x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z := by
  { have h2a : x ⊓ (y ⊓ z) ≤ x ⊓ y := by
    { have h2a1 : x ⊓ (y ⊓ z) ≤ x :=
        inf_le_left
      have h2a2 : x ⊓ (y ⊓ z) ≤ y := calc
        x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
                  _ ≤ y     := by exact inf_le_left
      show x ⊓ (y ⊓ z) ≤ x ⊓ y
      exact le_inf h2a1 h2a2 }
    have h2b : x ⊓ (y ⊓ z) ≤ z := by calc
      x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
                _ ≤ z     := by exact inf_le_right
    show x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z
    exact le_inf h2a h2b }
  show (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊓ y ⊓ z = x ⊓ (y ⊓ z) := by
  apply le_antisymm
  · apply le_inf
```

```
    · apply le_trans
      apply inf_le_left
      apply inf_le_left
    . apply le_inf
      · apply le_trans
        apply inf_le_left
        apply inf_le_right
      . apply inf_le_right
  . apply le_inf
    · apply le_inf
      · apply inf_le_left
      . apply le_trans
        apply inf_le_right
        apply inf_le_left
    . apply le_trans
      apply inf_le_right
      apply inf_le_right

-- 3ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  apply le_antisymm
  . apply le_inf
    . apply inf_le_of_left_le inf_le_left
    . apply le_inf (inf_le_of_left_le inf_le_right) inf_le_right
  . apply le_inf
    . apply le_inf inf_le_left (inf_le_of_right_le inf_le_left)
    . apply inf_le_of_right_le inf_le_right

-- 4ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
le_antisymm
  (le_inf
    (inf_le_of_left_le inf_le_left)
    (le_inf (inf_le_of_left_le inf_le_right) inf_le_right))
  (le_inf
    (le_inf inf_le_left (inf_le_of_right_le inf_le_left))
    (inf_le_of_right_le inf_le_right))

-- 5ª demostración
-- ===============
```

```
example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
-- by apply?
inf_assoc

-- Lemas usados
-- ============

-- #check (inf_assoc : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z))
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_le_of_left_le : x ≤ z → x ⊓ y ≤ z)
-- #check (inf_le_of_right_le : y ≤ z → x ⊓ y ≤ z)
-- #check (inf_le_right : x ⊓ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.4. En los retículos, (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm  : x ≤ y → y ≤ x → x = y
--    le_sup_left  : x ≤ x ⊔ y
--    le_sup_right : y ≤ x ⊔ y
--    sup_le       : x ≤ z → y ≤ z → x ⊔ y ≤ z
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)                                      (1)
--    x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z                                      (2)
--
-- Para demostrar (1), por sup_le, basta probar
--    x ⊔ y ≤ x ⊔ (y ⊔ z)                                           (1a)
--    z ≤ x ⊔ (y ⊔ z)                                               (1b)
--
```

```
-- Para demostrar (1a), por sup_le, basta probar
--    x ≤ x ⊔ (y ⊔ z)                                                    (1a1)
--    y ≤ x ⊔ (y ⊔ z)                                                    (1a2)
--
-- La (1a1) se tiene por le_sup_left.
--
-- La (1a2) se tiene por la siguiente cadena de desigualdades:
--    y ≤ y ⊔ z           [por le_sup_left]
--      ≤ x ⊔ (y ⊔ z)    [por le_sup_right]
--
-- La (1b) se tiene por la siguiente cadena de desigualdades
--    z ≤ y ⊔ z           [por le_sup_right]
--      ≤ x ⊔ (y ⊔ z)    [por le_sup_right]
--
-- Para demostrar (2), por sup_le, basta probar
--    x ≤ (x ⊔ y) ⊔ z                                                     (2a)
--    y ⊔ z ≤ (x ⊔ y) ⊔ z                                                 (2b)
--
-- La (2a) se demuestra por la siguiente cadena de desigualdades:
--    x ≤ x ⊔ y           [por le_sup_left]
--      ≤ (x ⊔ y) ⊔ z    [por le_sup_left]
--
-- Para demostrar (2b), por sup_le, basta probar
--    y ≤ (x ⊔ y) ⊔ z                                                    (2b1)
--    z ≤ (x ⊔ y) ⊔ z                                                    (2b2)
--
-- La (2b1) se demuestra por la siguiente cadena de desigualdades:
--    y ≤ x ⊔ y           [por le_sup_right]
--      ≤ (x ⊔ y) ⊔ z    [por le_sup_left]
--
-- La (2b2) se tiene por le_sup_right.

-- Demostraciones con Lean 4
-- =========================

import Mathlib.Order.Lattice

variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
```

```
  have h1 : (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z) := by
  { have h1a : x ⊔ y ≤ x ⊔ (y ⊔ z) := by
    { have h1a1 : x ≤ x ⊔ (y ⊔ z) := by exact le_sup_left
      have h1a2 : y ≤ x ⊔ (y ⊔ z) := calc
        y ≤ y ⊔ z         := by exact le_sup_left
        _ ≤ x ⊔ (y ⊔ z) := by exact le_sup_right
      show x ⊔ y ≤ x ⊔ (y ⊔ z)
      exact sup_le h1a1 h1a2 }
    have h1b : z ≤ x ⊔ (y ⊔ z) := calc
      z ≤ y ⊔ z         := by exact le_sup_right
      _ ≤ x ⊔ (y ⊔ z) := by exact le_sup_right
    show (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    exact sup_le h1a h1b }
  have h2 : x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z := by
  { have h2a : x ≤ (x ⊔ y) ⊔ z := calc
      x ≤ x ⊔ y         := by exact le_sup_left
      _ ≤ (x ⊔ y) ⊔ z := by exact le_sup_left
    have h2b : y ⊔ z ≤ (x ⊔ y) ⊔ z := by
    { have h2b1 : y ≤ (x ⊔ y) ⊔ z := calc
        y ≤ x ⊔ y         := by exact le_sup_right
        _ ≤ (x ⊔ y) ⊔ z := by exact le_sup_left
      have h2b2 : z ≤ (x ⊔ y) ⊔ z := by
        exact le_sup_right
      show  y ⊔ z ≤ (x ⊔ y) ⊔ z
      exact sup_le h2b1 h2b2 }
    show x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    exact sup_le h2a h2b }
  show (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊔ y ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  · -- (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    apply sup_le
    · -- x ⊔ y ≤ x ⊔ (y ⊔ z)
      apply sup_le
      . -- x ≤ x ⊔ (y ⊔ z)
        apply le_sup_left
      · -- y ≤ x ⊔ (y ⊔ z)
        apply le_trans
        . -- y ≤ y ⊔ z
```

```
          apply @le_sup_left _ _ y z
        . -- y ⊔ z ≤ x ⊔ (y ⊔ z)
          apply le_sup_right
    . -- z ≤ x ⊔ (y ⊔ z)
      apply le_trans
      . -- z ≤ x ⊔ (y ⊔ z)
        apply @le_sup_right _ _ y z
      . -- y ⊔ z ≤ x ⊔ (y ⊔ z)
        apply le_sup_right
  . -- x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    apply sup_le
    · -- x ≤ (x ⊔ y) ⊔ z
      apply le_trans
      . -- x ≤ x ⊔ y
        apply @le_sup_left _ _ x y
      . -- x ⊔ y ≤ (x ⊔ y) ⊔ z
        apply le_sup_left
    . -- y ⊔ z ≤ (x ⊔ y) ⊔ z
      apply sup_le
      · -- y ≤ (x ⊔ y) ⊔ z
        apply le_trans
        . -- y ≤ x ⊔ y
          apply @le_sup_right _ _ x y
        . -- x ⊔ y ≤ (x ⊔ y) ⊔ z
          apply le_sup_left
      . -- z ≤ (x ⊔ y) ⊔ z
        apply le_sup_right

-- 3ª demostración
-- ===============

example : x ⊔ y ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  · apply sup_le
    · apply sup_le
      . apply le_sup_left
      · apply le_trans
        . apply @le_sup_left _ _ y z
        . apply le_sup_right
    . apply le_trans
      . apply @le_sup_right _ _ y z
      . apply le_sup_right
  . apply sup_le
    · apply le_trans
```

```
      . apply @le_sup_left _ _ x y
      . apply le_sup_left
    . apply sup_le
      · apply le_trans
        . apply @le_sup_right _ _ x y
        . apply le_sup_left
      . apply le_sup_right

-- 4ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . -- (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    apply sup_le
    . -- x ⊔ y ≤ x ⊔ (y ⊔ z)
      apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . -- z ≤ x ⊔ (y ⊔ z)
      apply le_sup_of_le_right le_sup_right
  . -- x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    apply sup_le
    . -- x ≤ (x ⊔ y) ⊔ z
      apply le_sup_of_le_left le_sup_left
    . -- y ⊔ z ≤ (x ⊔ y) ⊔ z
      apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 5ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . apply sup_le
    . apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . apply le_sup_of_le_right le_sup_right
  . apply sup_le
    . apply le_sup_of_le_left le_sup_left
    . apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 6ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
le_antisymm
```

```
  (sup_le
    (sup_le le_sup_left (le_sup_of_le_right le_sup_left))
    (le_sup_of_le_right le_sup_right))
  (sup_le
    (le_sup_of_le_left le_sup_left)
    (sup_le (le_sup_of_le_left le_sup_right) le_sup_right))

-- 7ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
-- by apply?
sup_assoc

-- Lemas usados
-- ============

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
-- #check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)
-- #check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.5. En los retículos, x ⊓ (x ⊔ y) = x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊓ (x ⊔ y) = x
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm  : x ≤ y → y ≤ x → x = y
--    inf_le_left  : x ⊓ y ≤ x
--    le_inf       : z ≤ x → z ≤ y → z ≤ x ⊓ y
--    le_rfl       : x ≤ x
```

```
--    le_sup_left  : x ≤ x ⊔ y
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    x ⊓ (x ⊔ y) ≤ x                                               (1)
--    x ≤ x ⊓ (x ⊔ y)                                               (2)
--
-- La (1) se tiene por inf_le_left.
--
-- Para demostrar la (2), por le_inf, basta probar las relaciones:
--    x ≤ x                                                        (2a)
--    x ≤ x ⊔ y                                                    (2b)
--
-- La (2a) se tiene por le_rfl.
--
-- La (2b) se tiene por le_sup_left

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (x y : α)

-- 1ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  have h1 : x ⊓ (x ⊔ y) ≤ x := inf_le_left
  have h2 : x ≤ x ⊓ (x ⊔ y)
  { have h2a : x ≤ x := le_rfl
    have h2b : x ≤ x ⊔ y := le_sup_left
    show x ≤ x ⊓ (x ⊔ y)
    exact le_inf h2a h2b }
  show x ⊓ (x ⊔ y) = x
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  have h1 : x ⊓ (x ⊔ y) ≤ x := by simp
  have h2 : x ≤ x ⊓ (x ⊔ y) := by simp
  show x ⊓ (x ⊔ y) = x
```

```
  exact le_antisymm h1 h2

-- 3ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  apply le_antisymm
  . -- x ⊓ (x ⊔ y) ≤ x
    apply inf_le_left
  . -- x ≤ x ⊓ (x ⊔ y)
    apply le_inf
    . -- x ≤ x
      apply le_rfl
    . -- x ≤ x ⊔ y
      apply le_sup_left

-- 4ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
le_antisymm inf_le_left (le_inf le_rfl le_sup_left)

-- 5ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
-- by apply?
inf_sup_self

-- 6ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by simp

-- Lemas usados
-- ============

-- variable (z : α)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_sup_self : x ⊓ (x ⊔ y) = x)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
-- #check (le_rfl : x ≤ x)
```

```
-- #check (le_sup_left : x ≤ x ⊔ y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.6. En los retículos, x ⊔ (x ⊓ y) = x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊔ (x ⊓ y) = x
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--   le_antisymm  : x ≤ y → y ≤ x → x = y
--   inf_le_left  : x ⊓ y ≤ x
--   le_rfl       : x ≤ x
--   le_sup_left  : x ≤ x ⊔ y
--   sup_le       : x ≤ z → y ≤ z → x ⊔ y ≤ z
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    x ⊔ (x ⊓ y) ≤ x                                            (1)
--    x ≤ x ⊔ (x ⊓ y)    [que se tiene por le_sup_left]
--
-- Para demostrar (1), por sup_le, basta probar las relaciones:
--    x ≤ x              [que se tiene por le_rfl]
--    x ⊓ y ≤ x          [que se tiene por inf_le_left]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]--
variable (x y : α)

-- 1ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  have h1 : x ⊔ (x ⊓ y) ≤ x
  { have h1a : x ≤ x := le_rfl
```

```
    have h1b : x ⊓ y ≤ x := inf_le_left
    show x ⊔ (x ⊓ y) ≤ x
    exact sup_le h1a h1b }
  have h2 : x ≤ x ⊔ (x ⊓ y) := le_sup_left
  show x ⊔ (x ⊓ y) = x
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  have h1 : x ⊔ (x ⊓ y) ≤ x := by simp
  have h2 : x ≤ x ⊔ (x ⊓ y) := by simp
  show x ⊔ (x ⊓ y) = x
  exact le_antisymm h1 h2

-- 3ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  apply le_antisymm
  . -- x ⊔ (x ⊓ y) ≤ x
    apply sup_le
    . -- x ≤ x
      apply le_rfl
    . -- x ⊓ y ≤ x
      apply inf_le_left
  . -- x ≤ x ⊔ (x ⊓ y)
    apply le_sup_left

-- 4ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
-- by apply?
sup_inf_self

-- 5ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by simp
```

```
-- Lemas usados
-- ============

-- variable (z : α)
-- #check (le_rfl : x ≤ x)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (sup_inf_self : x ⊔ (x ⊓ y) = x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.7. En los retículos, una distributiva del ínfimo implica la otra

```
-- ---------------------------------------------------------------------
-- Demostrar que si α es un retículo tal que
--    ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
-- entonces
--    (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c)
-- para todos los elementos de α.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se demuestra por la siguiente cadena de igualdades
--    (a ⊔ b) ⊓ c = c ⊓ (a ⊔ b)          [por conmutatividad de ⊓]
--                = (c ⊓ a) ⊔ (c ⊓ b)    [por la hipótesis]
--                = (a ⊓ c) ⊔ (c ⊓ b)    [por conmutatividad de ⊓]
--                = (a ⊓ c) ⊔ (b ⊓ c)    [por conmutatividad de ⊓]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (a b c : α)

-- 1ª demostración
example
```

```
  (h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
  : (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
calc
  (a ⊔ b) ⊓ c = c ⊓ (a ⊔ b)       := by rw [inf_comm]
            _ = (c ⊓ a) ⊔ (c ⊓ b) := by rw [h]
            _ = (a ⊓ c) ⊔ (c ⊓ b) := by rw [@inf_comm _ _ c a]
            _ = (a ⊓ c) ⊔ (b ⊓ c) := by rw [@inf_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
  : (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
by simp [h, inf_comm]

-- Lemas usados
-- ============

-- #check (inf_comm : a ⊓ b = b ⊓ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 8.8. En los retículos, una distributiva del supremos implica la otra

```
-- ---------------------------------------------------------------------
-- Demostrar que si α es un retículo tal que
--    ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z)
-- entonces
--    (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se demuestra por la siguiente cadena de igualdades
--    (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)          [por la conmutatividad de ⊔]
--                = (c ⊔ a) ⊓ (c ⊔ b)    [por la hipótesis]
--                = (a ⊔ c) ⊓ (c ⊔ b)    [por la conmutatividad de ⊔]
--                = (a ⊔ c) ⊓ (b ⊔ c)    [por la conmutatividad de ⊔]

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (a b c : α)

-- 1ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
calc
  (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)       := by rw [sup_comm]
            _ = (c ⊔ a) ⊓ (c ⊔ b) := by rw [h]
            _ = (a ⊔ c) ⊓ (c ⊔ b) := by rw [@sup_comm _ _ c a]
            _ = (a ⊔ c) ⊓ (b ⊔ c) := by rw [@sup_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
by simp [h, sup_comm]

-- Lemas usados
-- ============

-- #check (sup_comm : a ⊔ b = b ⊔ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 9

# Anillos ordenados

## 9.1. En los anillos ordenados, a ≤ b → 0 ≤ b - a

```
-- ---------------------------------------------------------------------
-- Demostrar que en los anillos ordenados se verifica que
--    a ≤ b → 0 ≤ b - a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    sub_self         : a - a = 0
--    sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c
--
-- Supongamos que
--    a ≤ b                                                        (1)
-- La demostración se tiene por la siguiente cadena de desigualdades:
--    0 = a - a    [por sub_self]
--      ≤ b - a    [por (1) y sub_le_sub_right]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
example : a ≤ b → 0 ≤ b - a :=
by
```

```
  intro h
  calc
    0 = a - a := (sub_self a).symm
    _ ≤ b - a := sub_le_sub_right h a

-- 2ª demostración
example : a ≤ b → 0 ≤ b - a :=
sub_nonneg.mpr

-- 3ª demostración
example : a ≤ b → 0 ≤ b - a :=
by simp

-- Lemas usados
-- ============

-- #check (sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 9.2. En los anillos ordenados, 0 ≤ b - a → a ≤ b

```
-- ---------------------------------------------------------------------
-- Demostrar que en los anillos ordenados
--    0 ≤ b - a → a ≤ b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    zero_add a : 0 + a = a
--    add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a
--    sub_add_cancel a b : a - b + b = -a
-- Supongamos que
--    0 ≤ b - a                                                    (1)
-- La demostración se tiene por la siguiente cadena de desigualdades:
--    a = 0 + a          [por zero_add]
--      ≤ (b - a) + a    [por (1) y add_le_add_right]
--      = b              [por sub_add_cancel]
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
by
  intro h
  calc
    a = 0 + a       := (zero_add a).symm
    _ ≤ (b - a) + a := add_le_add_right h a
    _ = b           := sub_add_cancel b a

-- 2ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
-- by apply?
sub_nonneg.mp

-- 3ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
by simp

-- Lemas usados
-- ============

-- #check (zero_add a : 0 + a = a)
-- #check (add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a)
-- #check (sub_add_cancel a b : a - b + b = a)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 9.3. En los anillos ordenados, $\{a \leq b, 0 \leq c\} \vdash ac \leq bc$

```
-- ---------------------------------------------------------------------
-- Demostrar que, en los anillos ordenados, si
--    a ≤ b
--    0 ≤ c
-- entonces
--    a * c ≤ b * c
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    sub_nonneg                 : 0 ≤ a - b ↔ b ≤ a)
--    mul_nonneg                 : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
--    sub_mul a b c              : (a - b) * c = a * c - b * c)
--
-- Supongamos que
--    a ≤ b                                                           (1)
--    0 ≤ c
-- De (1), por sub_nonneg, se tiene
--    0 ≤ b - a
-- y con (2), por mul_nonneg, se tiene
--    0 ≤ (b - a) * c
-- que, por sub_mul, da
--    0 ≤ b * c - a * c
-- y, aplicándole sub_nonneg, se tiene
--    a * c ≤ b * c

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
```

```
  : a * c ≤ b * c :=
by
  have h3 : 0 ≤ b - a :=
    sub_nonneg.mpr h1
  have h4 : 0 ≤ b * c - a * c := calc
    0 ≤ (b - a) * c   := mul_nonneg h3 h2
    _ = b * c - a * c := sub_mul b a c
  show a * c ≤ b * c
  exact sub_nonneg.mp h4

-- 2ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  have h3 : 0 ≤ b - a := sub_nonneg.mpr h1
  have h4 : 0 ≤ (b - a) * c := mul_nonneg h3 h2
  -- h4 : 0 ≤ b * c - a * c
  rw [sub_mul] at h4
  -- a * c ≤ b * c
  exact sub_nonneg.mp h4

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  -- 0 ≤ b * c - a * c
  apply sub_nonneg.mp
  -- 0 ≤ (b - a) * c
  rw [← sub_mul]
  apply mul_nonneg
  . -- 0 ≤ b - a
    exact sub_nonneg.mpr h1
  . -- 0 ≤ c
    exact h2

-- 4ª demostración
-- ===============
```

```
example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  apply sub_nonneg.mp
  rw [← sub_mul]
  apply mul_nonneg (sub_nonneg.mpr h1) h2

-- 5ª demostración
example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
-- by apply?
mul_le_mul_of_nonneg_right h1 h2

-- Lemas usados
-- ============

-- #check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
-- #check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
-- #check (sub_mul a b c : (a - b) * c = a * c - b * c)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 10

# Espacios métricos

## 10.1. En los espacios métricos, dist(x,y) ≥ 0

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los espacios métricos
--    0 ≤ dist x y
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    dist_comm x y             : dist x y = dist y x
--    dist_self x               : dist x x = 0
--    dist_triangle x y z       : dist x z ≤ dist x y + dist y z
--    mul_two a                 : a * 2 = a + a
--    nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a
--    zero_lt_two               : 0 < 2
--
-- Por nonneg_of_mul_nonneg_left es suficiente demostrar las siguientes
-- desigualdades:
--    0 ≤ dist x y * 2                                                 (1)
--    0 < 2                                                            (2)
--
-- La (1) se demuestra por las siguiente cadena de desigualdades:
--    0 = dist x x               [por dist_self]
--      ≤ dist x y + dist y x    [por dist_triangle]
--      = dist x y + dist x y    [por dist_comm]
--      = dist x y * 2           [por mul_two]
--
-- La (2) se tiene por zero_lt_two.
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Topology.MetricSpace.Basic
variable {X : Type _} [MetricSpace X]
variable (x y : X)

-- 1ª demostración
example : 0 ≤ dist x y :=
by
  have h1 : 0 ≤ dist x y * 2 := calc
    0 = dist x x             := (dist_self x).symm
    _ ≤ dist x y + dist y x := dist_triangle x y x
    _ = dist x y + dist x y := by rw [dist_comm x y]
    _ = dist x y * 2         := (mul_two (dist x y)).symm
  show 0 ≤ dist x y
  exact nonneg_of_mul_nonneg_left h1 zero_lt_two

-- 2ª demostración
example : 0 ≤ dist x y :=
by
  apply nonneg_of_mul_nonneg_left
  . -- 0 ≤ dist x y * 2
    calc 0 = dist x x             := by simp only [dist_self]
         _ ≤ dist x y + dist y x := by simp only [dist_triangle]
         _ = dist x y + dist x y := by simp only [dist_comm]
         _ = dist x y * 2         := by simp only [mul_two]
  . -- 0 < 2
    exact zero_lt_two

-- 3ª demostración
example : 0 ≤ dist x y :=
by
  have : 0 ≤ dist x y + dist y x := by
    rw [← dist_self x]
    apply dist_triangle
  linarith [dist_comm x y]

-- 3ª demostración
example : 0 ≤ dist x y :=
-- by apply?
dist_nonneg

-- Lemas usados
```

```
-- ============

-- variable (a b : ℝ)
-- variable (z : X)
-- #check (dist_comm x y : dist x y = dist y x)
-- #check (dist_nonneg : 0 ≤ dist x y)
-- #check (dist_self x : dist x x = 0)
-- #check (dist_triangle x y z : dist x z ≤ dist x y + dist y z)
-- #check (mul_two a : a * 2 = a + a)
-- #check (nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a)
-- #check (zero_lt_two : 0 < 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 11

# Funciones reales

## 11.1. La suma de una cota superior de f y una cota superior de g es una cota superior de f+g

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de una cota superior de f y una cota superior
-- de g es una cota superior de f + g.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
-- Por la definición de cota superior, hay que demostrar que
--    (∀ x ∈ ℝ) [f(x) + g(x) ≤ a + b]                                 (1)
-- Para ello, sea x ∈ R. Puesto que es a es una cota superior de f, se
-- tiene que
--    f(x) ≤ a                                                        (2)
-- y, puesto que b es una cota superior de g, se tiene que
--    g(x) ≤ b                                                        (3)
-- De (2) y (3), por add_le_add, se tiene que
--    f(x) + g(x) ≤ a + b
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- 1ª demostración
-- ===============

example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  have h1 : ∀ x, (f + g) x  ≤ a + b := by
  { intro x
    have h2 : f x ≤ a := hfa x
    have h3 : g x ≤ b := hgb x
    show (f + g) x ≤ a + b
    exact add_le_add h2 h3 }
  show CotaSuperior (f + g) (a + b)
  exact h1

-- 2ª demostración
-- ===============

example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  have h1 : ∀ x, (f + g) x ≤ a + b := by
  { intro x
    show (f + g) x ≤ a + b
    exact add_le_add (hfa x) (hgb x) }
  show CotaSuperior (f + g) (a + b)
  exact h1

-- 3ª demostración
-- ===============

example
```

```
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  intro x
  dsimp
  apply add_le_add
  . apply hfa
  . apply hgb

-- 4ª demostración
-- ===============

theorem sumaCotaSup
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
λ x ↦ add_le_add (hfa x) (hgb x)

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.2. La suma de una cota inferior de f y una cota inferior de g es una cota inferior de f+g

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de una cota inferior de f y una  cota inferior
-- de g es una cota inferior de f + g.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
```

```
-- Por la definición de cota inferior, hay que demostrar que
--    (∀ x ∈ ℝ) [a + b ≤ f(x) + g(x)]                              (1)
-- Para ello, sea x ∈ R. Puesto que es a es una cota inferior de f, se
-- tiene que
--    a ≤ f(x)                                                      (2)
-- y, puesto que b es una cota inferior de g, se tiene que
--    b ≤ g(x)                                                      (3)
-- De (2) y (3), por add_le_add, se tiene que
--    a + b ≤ f(x) + g(x)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- 1ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  have h1 : ∀ x, a + b ≤ f x + g x
  { intro x
    have h1a : a ≤ f x := hfa x
    have h1b : b ≤ g x := hgb x
    show a + b ≤ f x + g x
    exact add_le_add h1a h1b }
  show CotaInferior (f + g) (a + b)
  exact h1

-- 2ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  have h1 : ∀ x, a + b ≤ f x + g x
```

```
  { intro x
    show a + b ≤ f x + g x
    exact add_le_add (hfa x) (hgb x) }
  show CotaInferior (f + g) (a + b)
  exact h1

-- 3ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  intro x
  dsimp
  apply add_le_add
  . apply hfa
  . apply hgb

-- 4ª demostración
theorem sumaCotaInf
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
λ x ↦ add_le_add (hfa x) (hgb x)

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.3. El producto de funciones no negativas es no negativo

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de dos funciones no negativas es no
-- negativa.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- ================================

-- Se usará el siguiente lema
--    mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b
--
-- Hay que demostrar que
--    (∀ x ∈ ℝ) [0 ≤ f(x) * g(x)]                                        (1)
-- Para ello, sea x ∈ R. Puesto que f es no negatica, se tiene que
--    0 ≤ f(x)                                                           (2)
-- y, puesto que g es no negativa, se tiene que
--    0 ≤ g(x)                                                           (3)
-- De (2) y (3), por mul_nonneg, se tiene que
--    0 ≤ f(x) * g(x)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable (f g : ℝ → ℝ)

-- 1ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x
  { intro x
    have h2: 0 ≤ f x := nnf x
    have h3: 0 ≤ g x := nng x
    show 0 ≤ f x * g x
    exact mul_nonneg h2 h3 }
  show CotaInferior (f * g) 0
  exact h1

-- 2ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
```

```
  : CotaInferior (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x
  { intro x
    show 0 ≤ f x * g x
    exact mul_nonneg (nnf x) (nng x) }
  show CotaInferior (f * g) 0
  exact h1

-- 3ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
by
  intro x
  dsimp
  apply mul_nonneg
  . apply nnf
  . apply nng

-- 4ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
λ x ↦ mul_nonneg (nnf x) (nng x)

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.4. Si a es una cota superior no negativa de f y b es es una cota superior de la función no negativa g, entonces ab es una cota superior de fg

```
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de f, b es una cota superior
-- de g, a es no negativa y g es no negativa, entonces ab es una cota
-- superior de fg.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d
--
-- Hay que demostrar que
--    (∀ x ∈ ℝ) [f x * g x ≤ a * b]                                   (1)
-- Para ello, sea x ∈ R. Puesto que a es una cota superior de f, se tiene que
--    f(x) ≤ a                                                        (2)
-- puesto que b es una cota superior de g, se tiene que
--    g(x) ≤ b                                                        (3)
-- puesto que g es no negativa, se tiene que
--    0 ≤ g(x)                                                        (4)
-- y, puesto que a es no negativa, se tiene que
--    0 ≤ a                                                           (5)
-- De (2), (3), (4) y (5), por mul_le_mul, se tiene que
--    f x * g x ≤ a * b
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a
```

```
-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable (f g : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  have h1 : ∀ x, f x * g x ≤ a * b
  { intro x
    have h2 : f x ≤ a := hfa x
    have h3 : g x ≤ b := hgb x
    have h4 : 0 ≤ g x := nng x
    show f x * g x ≤ a * b
    exact mul_le_mul h2 h3 h4 nna }
  show CotaSuperior (f * g) (a * b)
  exact h1

-- 2ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  dsimp
  apply mul_le_mul
  . apply hfa
  . apply hgb
  . apply nng
  . apply nna

-- 3ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
```

```
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  have h1:= hfa x
  have h2:= hgb x
  have h3:= nng x
  exact mul_le_mul h1 h2 h3 nna

-- 4ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  specialize hfa x
  specialize hgb x
  specialize nng x
  exact mul_le_mul hfa hgb nng nna

-- 5ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
λ x ↦ mul_le_mul (hfa x) (hgb x) (nng x) nna

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.5. La suma de dos funciones acotadas superiormente también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones acotadas superiormente también
-- lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Del ejercicio ”La suma de una cota superior de f y una cota superior
-- de g es una cota superior de f+g” (que se encuentra en
-- https://bit.ly/3QauluK ) usaremos la definición de cota superior
-- (CotaSuperior) y el lema sumaCotaSup.
--
-- Puesto que f está acotada superiormente, tiene una cota superior. Sea
-- a una de dichas cotas. Análogamentte, puesto que g está acotada
-- superiormente, tiene una cota superior. Sea b una de dichas
-- cotas. Por el lema sumaCotaSup, a+b es una cota superior de f+g. or
-- consiguiente, f+g está acotada superiormente.

-- Demostraciones con Lean4
-- ========================

import src.Suma_de_cotas_superiores

variable {f g : ℝ → ℝ}

-- (acotadaSup f) afirma que f tiene cota superior.
def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

-- 1ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  cases’ hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  cases’ hg with b hb
  -- b : ℝ
```

```
  -- hb : CotaSuperior g b
  have h1 : CotaSuperior (f + g) (a + b) :=
    sumaCotaSup ha hb
  have h2 : ∃ z, CotaSuperior (f+g) z :=
    Exists.intro (a + b) h1
  show acotadaSup (f + g)
  exact h2

-- 2ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : FnUb f a
  cases' hg with b hb
  -- b : ℝ
  -- hb : FnUb g b
  use a + b
  apply sumaCotaSup ha hb

-- 4ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  rcases hf with ⟨a, ha⟩
  rcases hg with ⟨b, hb⟩
  exact ⟨a + b, sumaCotaSup ha hb⟩

-- 5ª demostración
example :
  acotadaSup f → acotadaSup g → acotadaSup (f + g) :=
by
  rintro ⟨a, ha⟩ ⟨b, hb⟩
  exact ⟨a + b, sumaCotaSup ha hb⟩

-- 6ª demostración
example :
  acotadaSup f → acotadaSup g → acotadaSup (f + g) :=
fun ⟨a, ha⟩ ⟨b, hb⟩ ↦ ⟨a + b, sumaCotaSup ha hb⟩
```

```
-- Lemas usados
-- ============

-- #check (sumaCotaSup : CotaSuperior f a → CotaSuperior g b → CotaSuperior (f + g) (a +
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.6. La suma de dos funciones acotadas inferiormente también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones acotadas inferiormente también
-- lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Del ejercicio "La suma de una cota inferior de f y una cota inferior
-- de g es una cota inferior de f+g" usaremos la definición de cota
-- inferior (CotaInferior) y el lema sumaCotaInf.
--
-- Puesto que f está acotada inferiormente, tiene una cota inferior. Sea
-- a una de dichas cotas. Análogamentte, puesto que g está acotada
-- inferiormente, tiene una cota inferior. Sea b una de dichas
-- cotas. Por el lema FnLb_add, a+b es una cota inferior de f+g. Por
-- consiguiente, f+g está acotada inferiormente.

-- Demostraciones con Lean4
-- ========================

import src.Suma_de_cotas_inferiores
variable {f g : ℝ → ℝ}

-- (acotadaInf f) afirma que f tiene cota inferior.
def acotadaInf (f : ℝ → ℝ) :=
  ∃ a, CotaInferior f a

-- 1ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
```

```
  : acotadaInf (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaInferior f a
  cases' hg with b hb
  -- b : ℝ
  -- hb : CotaInferior g b
  have h1 : CotaInferior (f + g) (a + b) := sumaCotaInf ha hb
  have h2 : ∃ z, CotaInferior (f + g) z :=
    Exists.intro (a + b) h1
  show acotadaInf (f + g)
  exact h2

-- 2ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
  : acotadaInf (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : FnLb f a
  cases' hg with b hgb
  -- b : ℝ
  -- hgb : FnLb g b
  use a + b
  -- ⊢ FnLb (f + g) (a + b)
  apply sumaCotaInf ha hgb

-- 3ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
  : acotadaInf (f + g) :=
by
  rcases hf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnLb f a
  rcases hg with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnLb g b
  exact ⟨a + b, sumaCotaInf ha hb⟩

-- 4ª demostración
```

```
example :
  acotadaInf f → acotadaInf g → acotadaInf (f + g) :=
by
  rintro ⟨a, ha⟩ ⟨b, hb⟩
  -- a : ℝ
  -- ha : FnLb f a
  -- b : ℝ
  -- hb : FnLb g b
  exact ⟨a + b, sumaCotaInf ha hb⟩

-- 5ª demostración
example :
  acotadaInf f → acotadaInf g → acotadaInf (f + g) :=
fun ⟨a, ha⟩ ⟨b, hb⟩ ↦ ⟨a + b, sumaCotaInf ha hb⟩

-- Lemas usados
-- ============

-- #check (sumaCotaInf : FnLb f a → FnLb g b → FnLb (f + g) (a + b))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.7. Si a es una cota superior de f y c ≥ 0, entonces ca es una cota superior de cf

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de f y c ≥ 0,
-- entonces c * a es una cota superior de c * f.
-- ----------------------------------------------------------------------

-- Demostración en lenguaj natural
-- ===============================

-- Se usará el lema
--    {b ≤ c, 0 ≤ a} ⊢ ab ≤ ac                                        (L1)
--
-- Tenemos que demostrar que
--    (∀ y ∈ ℝ) cf(y) ≤ ca.
-- Sea y ∈ R. Puesto que a es una cota de f, se tiene que
--    f(y) ≤ a
-- que, junto con c ≥ 0, por el lema L1 nos da
--    cf(y) ≤ ca
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

variable {f : ℝ → ℝ}
variable {c : ℝ}

-- Demostraciones con Lean4
-- ========================

-- 1ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  -- y : ℝ
  -- ⊢ (fun x => c * f x) y ≤ c * a
  have ha : f y ≤ a := hfa y
  calc (fun x => c * f x) y
       = c * f y := by rfl
     _ ≤ c * a   := mul_le_mul_of_nonneg_left ha h

-- 2ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  calc (fun x => c * f x) y
       = c * f y := by rfl
     _ ≤ c * a   := mul_le_mul_of_nonneg_left (hfa y) h

-- 3ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
```

```
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  show (fun x => c * f x) y ≤ c * a
  exact mul_le_mul_of_nonneg_left (hfa y) h

-- 4ª demostración
lemma CotaSuperior_mul
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
fun y ↦ mul_le_mul_of_nonneg_left (hfa y) h

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.8. Si c ≥ 0 y f está acotada superiormente, entonces c·f también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que si c ≥ 0 y f está acotada superiormente, entonces c * f
-- también lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos el siguiente lema:
--    CotaSuperior_mul : CotaSuperior f a → c ≥ 0 → CotaSuperior (fun x ↦ c * f x) (c *
--
-- Puesto que f está acotada superiormente, tiene una cota superior. Sea
-- a una de dichas cotas. Entonces, por el lema CotaSuperior_mul, ca es una cota
-- superior de cf. Por consiguiente, cf está acotada superiormente.

-- Demostraciones con Lean4
-- ========================
```

```
import src.Cota_superior_de_producto_por_escalar

variable {f : ℝ → ℝ}
variable {c : ℝ}

-- (acotadaSup f) afirma que f tiene cota superior.
def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

-- 1ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  have h1 : CotaSuperior (fun x ↦ c * f x) (c * a) :=
    CotaSuperior_mul ha hc
  have h2 : ∃ z, ∀ x, (fun x ↦ c * f x) x ≤ z :=
    Exists.intro (c * a) h1
  show acotadaSup (fun x ↦ c * f x)
  exact h2

-- 2ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  use c * a
  -- ⊢ CotaSuperior (fun x => c * f x) (c * a)
  apply CotaSuperior_mul ha hc

-- 3ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  rcases hf with ⟨a, ha⟩
```

```
  -- a : ℝ
  -- ha : CotaSuperior f a
  exact ⟨c * a, CotaSuperior_mul ha hc⟩

-- 4ª demostración
example
  (hc : c ≥ 0)
  : acotadaSup f → acotadaSup (fun x ↦ c * f x) :=
by
  rintro ⟨a, ha⟩
  -- a : ℝ
  -- ha : CotaSuperior f a
  exact ⟨c * a, CotaSuperior_mul ha hc⟩

-- 5ª demostración
example
  (hc : c ≥ 0)
  : acotadaSup f → acotadaSup (fun x ↦ c * f x) :=
fun ⟨a, ha⟩ ↦ ⟨c * a, CotaSuperior_mul ha hc⟩

-- Lemas usados
-- ============

-- #check (CotaSuperior_mul : CotaSuperior f a → c ≥ 0 → CotaSuperior (fun x ↦ c * f x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.9. Si para cada a existe un x tal que f(x) >a, entonces f no tiene cota superior

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función de ℝ en ℝ tal que para cada a,
-- existe un x tal que f x > a, entonces f no tiene cota superior.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f tiene cota superior. Sea b una de dichas cotas
-- superiores. Por la hipótesis, existe un x tal que f(x) > b. Además,
-- como b es una cota superior de f, f(x) ≤ b que contradice la
-- desigualdad anterior.
```

```
-- Demostraciones con Lean4
-- =======================

import Mathlib.Data.Real.Basic

def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

def acotadaSup (f : ℝ → ℝ) : Prop :=
  ∃ a, CotaSuperior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
example
  (h : ∀ a, ∃ x, f x > a)
  : ¬ acotadaSup f :=
by
  intros hf
  -- hf : acotadaSup f
  -- ⊢ False
  cases' hf with b hb
  -- b : ℝ
  -- hb : CotaSuperior f b
  cases' h b with x hx
  -- x : ℝ
  -- hx : f x > b
  have : f x ≤ b := hb x
  linarith

-- 2ª demostración
theorem sinCotaSup
  (h : ∀ a, ∃ x, f x > a)
  : ¬ acotadaSup f :=
by
  intros hf
  -- hf : acotadaSup f
  -- ⊢ False
  rcases hf with ⟨b, hb : CotaSuperior f b⟩
  rcases h b with ⟨x, hx : f x > b⟩
  have : f x ≤ b := hb x
  linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.10. Si para cada a existe un x tal que f(x) <a, entonces f no tiene cota inferior

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función de ℝ en ℝ tal que para cada a,
-- existe un x tal que f x < a, entonces f no tiene cota inferior.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f tiene cota inferior. Sea b una de dichas cotas
-- inferiores. Por la hipótesis, existe un x tal que f(x) < b. Además,
-- como b es una cota inferior de f, b ≤ f(x) que contradice la
-- desigualdad anterior.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

def acotadaInf (f : ℝ → ℝ) : Prop :=
  ∃ a, CotaInferior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ acotadaInf f :=
by
  intros hf
  -- hf : acotadaInf f
  -- ⊢ False
  cases' hf with b hb
  -- b : ℝ
  -- hb : CotaInferior f b
  cases' h b with x hx
  -- x : ℝ
  -- hx : f x < b
  have : b ≤ f x := hb x
```

```
  linarith

-- 2ª demostración
example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ acotadaInf f :=
by
  intros hf
  -- hf : acotadaInf f
  -- ⊢ False
  rcases hf with ⟨b, hb : CotaInferior f b⟩
  rcases h b with ⟨x, hx : f x < b⟩
  have : b ≤ f x := hb x
  linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.11. La función identidad no está acotada superiormente

```
-- ---------------------------------------------------------------------
-- Demostrar que la función identidad no está acotada superiormente.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usamos el lema de ejercicio anterior (que afirma que si para cada a,
-- existe un x tal que f x > a, entonces f no tiene cota superior) basta
-- demostrar que
--    (∀a ∈ ℝ)(∃x ∈ ℝ) [x > a]
-- Sea a ∈ ℝ. Entonces a + 1 > a y, por tanto, (∃x ∈ ℝ) [x > a].

-- Demostraciones con Lean4
-- ========================

import src.Funcion_no_acotada_superiormente

-- 1ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
```

```
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  intro a
  -- a : ℝ
  -- ⊢ ∃ x, x > a
  use a + 1
  -- ⊢ a + 1 > a
  linarith

-- 2ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  intro a
  -- a : ℝ
  -- ⊢ ∃ x, x > a
  exact ⟨a + 1, by linarith⟩

-- 3ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  exact fun a ↦ ⟨a + 1, by linarith⟩
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# 11.12. Suma de funciones monótonas

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones monótonas es monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema:
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
-- Supongamos que f y g son monótonas y teneno que demostrar que f+g
-- también lo es; que
--    ∀ a b, a ≤ b → (f + g)(a) ≤ (f + g)(b)
-- Sean a, b ∈ ℝ tales que
```

```
--    a ≤ b                                                            (1)
-- Entonces, por ser f y g monótonas se tiene
--    f(a) ≤ f(b)                                                      (2)
--    g(a) ≤ g(b)                                                      (3)
-- Entonces,
--    (f + g)(a) = f(a) + g(a)
--               ≤ f(b) + g(b)    [por add_le_add, (2) y (3)]
--               = (f + g)(b)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- 1ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    have h2 : f a ≤ f b := mf hab
    have h3 : g a ≤ g b := mg hab
    calc (f + g) a
         = f a + g a := rfl
       _ ≤ f b + g b := add_le_add h2 h3
       _ = (f + g) b := rfl }
  show Monotone (f + g)
  exact h1

-- 2ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    calc (f + g) a
         = f a + g a := rfl
       _ ≤ f b + g b := add_le_add (mf hab) (mg hab)
       _ = (f + g) b := rfl }
```

```
  show Monotone (f + g)
  exact h1

-- 3ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    show (f + g) a ≤ (f + g) b
    exact add_le_add (mf hab) (mg hab) }
  show Monotone (f + g)
  exact h1

-- 4ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  apply add_le_add
  . -- f a ≤ f b
    apply mf hab
  . --  g a ≤ g b
    apply mg hab

-- 5ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
λ _ _ hab ↦ add_le_add (mf hab) (mg hab)

-- Lemas usados
-- ============

-- variable (a b c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.13. Si c es no negativo y f es monótona, entonces cf es monótona.

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es no negativo y f es monótona, entonces cf es
-- monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el Lema
--    mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c
--
-- Tenemos que demostrar que
--    (∀ a, b ∈ ℝ) [a ≤ b → (cf)(a) ≤ (cf)(b)]
-- Sean a, b ∈ ℝ tales que a ≤ b. Puesto que f es monótona, se tiene
--    f(a) ≤ f(b).
-- y, junto con la hipótesis de que c es no negativo, usando el lema
-- mul_le_mul_of_nonneg_left, se tiene que
--    cf(a) ≤ cf(b)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f : ℝ → ℝ)
variable {c : ℝ}

-- 1ª demostración
example
  (mf : Monotone f)
  (nnc : 0 ≤ c)
  : Monotone (fun x ↦ c * f x) :=
by
  have h1 : ∀ a b, a ≤ b → (fun x ↦ c * f x) a ≤ (fun x ↦ c * f x) b
  { intros a b hab
    have h2 : f a ≤ f b := mf hab
    show (fun x ↦ c * f x) a ≤ (fun x ↦ c * f x) b
    exact mul_le_mul_of_nonneg_left h2 nnc }
  show Monotone (fun x ↦ c * f x)
  exact h1
```

```
-- 2ª demostración
example
  (mf : Monotone f)
  (nnc : 0 ≤ c)
  : Monotone (fun x ↦ c * f x) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  -- (fun x => c * f x) a ≤ (fun x => c * f x) b
  apply mul_le_mul_of_nonneg_left
  . -- f a ≤ f b
    apply mf hab
  . -- 0 ≤ c
    apply nnc

-- 3ª demostración
example (mf : Monotone f) (nnc : 0 ≤ c) :
  Monotone (fun x ↦ c * f x) :=
λ _ _ hab ↦ mul_le_mul_of_nonneg_left (mf hab) nnc

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.14. La composición de dos funciones monótonas es monótona

```
-- ---------------------------------------------------------------------
-- Demostrar que la composición de dos funciones monótonas es monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sean f y g dos funciones monótonas de ℝ en ℝ. Tenemos que demostrar
-- que f ∘ g es monótona; es decir, que
--    (∀ a, b ∈ ℝ) [a ≤ b → (f ∘ g)(a) ≤ (f ∘ g)(b)]
```

```
-- Sean a, b ∈ ℝ tales que a ≤ b. Por ser g monótona, se tiene
--    g(a) ≤ g(b)
-- y, por ser f monótona, se tiene
--    f(g(a)) ≤ f(g(b))
-- Finalmente, por la definición de composición,
--    (f ∘ g)(a) ≤ (f ∘ g)(b)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- 1ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  have h1 : ∀ a b, a ≤ b → (f ∘ g) a ≤ (f ∘ g) b
  { intros a b hab
    have h1 : g a ≤ g b := mg hab
    show (f ∘ g) a ≤ (f ∘ g) b
    exact mf h1 }
  show Monotone (f ∘ g)
  exact h1

-- 2ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  have h1 : ∀ a b, a ≤ b → (f ∘ g) a ≤ (f ∘ g) b
  { intros a b hab
    show (f ∘ g) a ≤ (f ∘ g) b
    exact mf (mg hab) }
  show Monotone (f ∘ g)
  exact h1

-- 3ª demostración
example
  (mf : Monotone f)
```

```
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  -- (f ∘ g) a ≤ (f ∘ g) b
  apply mf
  -- g a ≤ g b
  apply mg
  -- a ≤ b
  apply hab

-- 4ª demostración
example (mf : Monotone f) (mg : Monotone g) :
  Monotone (f ∘ g) :=
λ _ _ hab ↦ mf (mg hab)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.15. Si f es monótona y f(a) <f(b), entonces a <b

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es monótona y f(a) < f(b), entonces a < b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos los lemas
--    a ≱ b → a < b                                                  (L1)
--    a ≥ b → a ≮ b                                                  (L2)
--
-- Usando el lema L1, basta demostrar que a ≱ b. Lo haremos por
-- reducción al absurdo. Para ello, supongamos que a ≥ b. Como f es
-- monótona, se tiene que f(a) ≥ f(b) y, aplicando el lema L2,
-- f(a) ≮ f(b), que contradice a la hipótesis.

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic
variable (f : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  have h4 : f a ≥ f b := h1 h3
  have h5 : ¬ f a < f b := not_lt_of_ge h4
  exact h5 h2

-- 2ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  have h5 : ¬ f a < f b := not_lt_of_ge (h1 h3)
  exact h5 h2

-- 3ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
```

```
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  exact (not_lt_of_ge (h1 h3)) h2

-- 4ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  exact fun h3 ↦ (not_lt_of_ge (h1 h3)) h2

-- 5ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
lt_of_not_ge (fun h3 ↦ (not_lt_of_ge (h1 h3)) h2)

-- Lemas usados
-- ============

-- #check (lt_of_not_ge : ¬ a ≥ b → a < b)
-- #check (not_lt_of_ge : a ≥ b → ¬ a < b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.16. Si a, b ∈ ℝ tales que a ≤ b y f(b) <f(a), entonces f no es monótona

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b ∈ ℝ tales que (a ≤ b) y (f b < f a), entonces f
-- no es monótona.
```

```
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos el lema
--    a ≥ b → a ≮ b                                              (L1)
--
-- Lo demostraremos por reducción al absurdo. Para ello, supongamos que
-- f es monótona. Entonces, como a ≤ b, se tiene f(a) ≤ f(b) y, por el
-- lema L1, f b ≮ f a, en contradicción con la hipótesis.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (f : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
  -- ⊢ False
  have h4 : f a ≤ f b := h3 h1
  have h5 : ¬(f b < f a) := not_lt_of_ge h4
  exact h5 h2

-- 2ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
  -- ⊢ False
```

```
  have h5 : ¬(f b < f a) := not_lt_of_ge (h3 h1)
  exact h5 h2

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
  -- ⊢ False
  exact (not_lt_of_ge (h3 h1)) h2

-- 4ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
fun h3 ↦ (not_lt_of_ge (h3 h1)) h2

-- Lemas usados
-- ============

-- #check (not_lt_of_ge : a ≥ b → ¬a < b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.17. No para toda f : ℝ → ℝ monótona, (∀a, b)[f(a) ≤ f(b) → a ≤ b]

```
-- ---------------------------------------------------------------------
-- Demostrar que no para toda f : ℝ → ℝ monótona,
--    (∀ a b)[f(a) ≤ f(b) → a ≤ b]
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Supongamos que
--    (∀f)[f es monótona → (∀a, b)[f(a) ≤ f(b) → a ≤ b]]             (1)
-- Sea f : ℝ → ℝ la función constante igual a cero (es decir,
--    (∀x ∈ ℝ)[f(x) = 0]
-- Entonces, f es monótona y f(1) ≤ f(0) (ya que
-- f(1) = 0 ≤ 0 = f(0)). Luego, por (1), 1 ≤ 0 que es una
-- contradicción.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- 1ª demostración
-- ===============

example :
  ¬∀ {f : ℝ → ℝ}, Monotone f → ∀ {a b}, f a ≤ f b → a ≤ b :=
by
  intro h1
  -- h1 : ∀ {f : ℝ → ℝ}, Monotone f → ∀ {a b : ℝ}, f a ≤ f b → a ≤ b
  -- ⊢ False
  let f := fun _ : ℝ ↦ (0 : ℝ)
  have h2 : Monotone f := monotone_const
  have h3 : f 1 ≤ f 0 := le_refl 0
  have h4 : 1 ≤ 0 := h1 h2 h3
  linarith

-- Lemas usados
-- ============

-- variable (a c : ℝ)
-- #check (le_refl a : a ≤ a)
-- #check (monotone_const : Monotone fun _ : ℝ ↦ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.18. La suma de dos funciones pares es par

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones pares es par.
-- ----------------------------------------------------------------------
```

```
-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f y g son funciones pares. Tenemos que demostrar que
-- f+g es par; es decir, que
--    (∀ x ∈ ℝ) (f + g)(x) = (f + g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f + g) x = f x + g x
--              = f (-x) + g x    [porque f es par]
--              = f (-x) + g (-x) [porque g es par]
--              = (f + g) (-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- 1ª demostración
-- ===============

example
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
  intro x
  have h1 : f x = f (-x) := h1 x
  have h2 : g x = g (-x) := h2 x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g x    := congrArg (. + g x) h1
     _ = f (-x) + g (-x) := congrArg (f (-x) + .) h2
     _ = (f + g) (-x)    := rfl

-- 2ª demostración
-- ===============

example
```

```
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
  intro x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g x    := congrArg (. + g x) (h1 x)
     _ = f (-x) + g (-x) := congrArg (f (-x) + .) (h2 x)
     _ = (f + g) (-x)    := rfl

-- 3ª demostración
-- ===============

example
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
  intro x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g (-x) := by rw [h1, h2]
     _ = (f + g) (-x)    := rfl
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.19. El producto de dos funciones impares es par

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de dos funciones impares es par.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f y g son funciones impares. Tenemos que demostrar que
-- f·g es par; es decir, que
--    (∀ x ∈ ℝ) (f·g)(x) = (f·g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f·g) x = f(x)g(x)
```

```
--              = (-f(-x))g(x)      [porque f es impar]
--              = (-f(-x)(-g(-x))   [porque g es impar]
--              = f(-x)g(-x))
--              = (f·g)(-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  have h1 : f x = -f (-x) := h1 x
  have h2 : g x = -g (-x) := h2 x
  calc (f * g) x
       = f x * g x             := rfl
     _ = (-f (-x)) * g x       := congrArg (. * g x) h1
     _ = (-f (-x)) * (-g (-x)) := congrArg ((-f (-x)) * .) h2
     _ = f (-x) * g (-x)       := neg_mul_neg (f (-x)) (g (-x))
     _ = (f * g) (-x)          := rfl

-- 2ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x             := rfl
```

```
     _ = (-f (-x)) * g x        := congrArg (. * g x) (h1 x)
     _ = (-f (-x)) * (-g (-x)) := congrArg ((-f (-x)) * .) (h2 x)
     _ = f (-x) * g (-x)        := neg_mul_neg (f (-x)) (g (-x))
     _ = (f * g) (-x)           := rfl

-- 3ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x         := rfl
     _ = -f (-x) * -g (-x) := by rw [h1, h2]
     _ = f (-x) * g (-x)   := by rw [neg_mul_neg]
     _ = (f * g) (-x)      := rfl

-- 4ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x       := rfl
     _ = f (-x) * g (-x) := by rw [h1, h2, neg_mul_neg]
     _ = (f * g) (-x)    := rfl

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (neg_mul_neg a b : -a * -b = a * b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.20. El producto de una función par por una impar es impar

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de una función par por una impar es impar.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f es una función par y g lo es impar. Tenemos que
-- demostrar que f·g es imppar; es decir, que
--    (∀ x ∈ ℝ) (f·g)(x) = -(f·g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f·g) x = f(x)g(x)
--            = f(-x)g(x)       [porque f es par]
--            = f(-x)(-g(-x))   [porque g es impar]
--            = -f(-x)g(-x))
--            = -(f·g)(-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  have h1 : f x = f (-x) := h1 x
  have h2 : g x = -g (-x) := h2 x
```

```
  calc (f * g) x
       = f x * g x            := rfl
     _ = (f (-x)) * g x       := congrArg (. * g x) h1
     _ = (f (-x)) * (-g (-x)) := congrArg (f (-x) * .) h2
     _ = -(f (-x) * g (-x))   := mul_neg (f (-x)) (g (-x))
     _ = -(f * g) (-x)        := rfl

-- 2ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x          := rfl
    _  = f (-x) * -g (-x)   := by rw [h1, h2]
    _  = -(f (-x) * g (-x)) := by rw [mul_neg]
    _  = -(f * g) (-x)      := rfl

-- 3ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x          := rfl
     _ = -(f (-x) * g (-x)) := by rw [h1, h2, mul_neg]
     _ = -((f * g) (-x))    := rfl

-- Lemas usados
-- ===========

-- variable (a b : ℝ)
-- #check (mul_neg a b : a * -b = -(a * b))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.21. Si f es par y g es impar, entonces (f ∘ g) es par

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es par y g es impar, entonces f ∘ g es par.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f es una función par y g lo es impar. Tenemos que
-- demostrar que (f ∘ g) es par; es decir, que
--    (∀ x ∈ ℝ) (f ∘ g)(x) = (f ∘ g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f ∘ g)(x) = f(g(x))
--               = f(-g(-x))    [porque g es impar]
--               = f(g(-x))     [porque f es par]
--               = (f ∘ g)(-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (-g (-x))  := congr_arg f (h2 x)
```

```
    _  = f (g (-x))   := (h1 (g (-x))).symm
    _  = (f ∘ g) (-x) := rfl

-- 2ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (-g (-x))  := by rw [h2]
     _ = f (g (-x))   := by rw [← h1]
     _ = (f ∘ g) (-x) := rfl

-- 3ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (g (-x))   := by rw [h2, ← h1]
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 11.22. Para cualquier conjunto s, s ⊆ s

```
-- ---------------------------------------------------------------------
-- Demostrar que para cualquier conjunto s, s ⊆ s.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x) [x ∈ s → x ∈ s]
-- Sea x tal que
--    x ∈ s                                                         (1)
-- Entonces, por (1), se tiene que
```

```
--    x ∈ s
-- que es lo que teníamos que demostrar.

-- Demostraciones con Lean 4
-- ==========================

import Mathlib.Tactic

variable {α : Type _}
variable (s : Set α)

-- 1ª demostración
example : s ⊆ s :=
by
  intro x xs
  exact xs

-- 2ª demostración
example : s ⊆ s :=
  fun (x : α) (xs : x ∈ s) ↦ xs

-- 3ª demostración
example : s ⊆ s :=
  fun _ xs ↦ xs

-- 4ª demostración
example : s ⊆ s :=
  -- by exact?
  rfl.subset

-- 5ª demostración
example : s ⊆ s :=
by rfl
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Capítulo 12

# Teoría de conjuntos

## 12.1. Si r ⊆ s y s ⊆ t, entonces r ⊆ t

```
-- ----------------------------------------------------------------------
-- Demostrar que si r ⊆ s y s ⊆ t, entonces r ⊆ t.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural (LN)
-- =====================================

-- 1ª demostración en LN
-- ---------------------

-- Tenemos que demostrar que
--    (∀ x) [x ∈ r → x ∈ t]
-- Sea x tal que
--    x ∈ r.
-- Puesto que r ⊆ s, se tiene que
--    x ∈ s
-- y, puesto que s ⊆ t, se tiene que
--    x ∈ t
-- que es lo que teníamos que demostrar.

-- 2ª demostración en LN
-- ---------------------

-- Tenemos que demostrar que
--    (∀ x) [x ∈ r → x ∈ t]
-- Sea x tal que
--    x ∈ r
-- Tenemos que demostrar que
```

```
--    x ∈ t
-- que, puesto que s ⊆ t, se reduce a
--    x ∈ s
-- que, puesto que r ⊆ s, se redece a
--    x ∈ r
-- que es lo que hemos supuesto.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Set

variable {α : Type _}
variable (r s t : Set α)

-- 1ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by
  intros x xr
  -- xr : x ∈ r
  have xs : x ∈ s := rs xr
  show x ∈ t
  exact st xs

-- 2ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by
  intros x xr
  -- x : α
  -- xr : x ∈ r
  apply st
  -- ⊢ x ∈ s
  apply rs
  -- ⊢ x ∈ r
  exact xr

-- 3ª demostración
```

```
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
fun _ xr ↦ st (rs xr)

-- 4ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
-- by exact?
Subset.trans rs st

-- 5ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by tauto

-- Lemas usados
-- ============

-- #check (Subset.trans : r ⊆ s → s ⊆ t → r ⊆ t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.2. Si a es una cota superior de s y a ≤ b, entonces b es una cota superior de s

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de s y a ≤ b, entonces b es
-- una cota superior de s.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

variable {α : Type _} [PartialOrder α]
variable (s : Set α)
variable (a b : α)
```

```
-- (CotaSupConj s a) afirma que a es una cota superior del conjunto s.
def CotaSupConj (s : Set α) (a : α) :=
  ∀ {x}, x ∈ s → x ≤ a

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x) [x ∈ s → x ≤ b]
-- Sea x tal que x ∈ s. Entonces,
--    x ≤ a   [porque a es una cota superior de s]
--      ≤ b
-- Por tanto, x ≤ b.

-- 1ª demostración
example
  (h1 : CotaSupConj s a)
  (h2 : a ≤ b)
  : CotaSupConj s b :=
by
  intro x (xs : x ∈ s)
  have h3 : x ≤ a := h1 xs
  show x ≤ b
  exact le_trans h3 h2

-- 2ª demostración
example
  (h1 : CotaSupConj s a)
  (h2 : a ≤ b)
  : CotaSupConj s b :=
by
  intro x (xs : x ∈ s)
  calc x ≤ a := h1 xs
       _ ≤ b := h2
-
-- Lemas usados
-- ============

-- variable (c : α)
-- #check (le_trans : a ≤ b → b ≤ c → a ≤ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.3. La función (x ↦ x + c) es inyectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que, para todo c la función
--    f(x) = x + c
-- es inyectiva
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el lema
--    (∀ a, b, c) [a + b = c + b → a = c]                           (L1)
-- Hay que demostrar que
--    (∀ x₁ x₂) [f(x₁) = f(x₂) → x₁ = x₂]
-- Sean x₁, x₂ tales que f(x₁) = f(x₂). Entonces,
--    x₁ + c = x₂ + c
-- y, por L1, x₁ = x₂.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function

variable {c : ℝ}

-- 1ª demostración
example : Injective ((. + c)) :=
by
  intro (x1 : ℝ) (x2 : ℝ) (h1 : x1 + c = x2 + c)
  show x1 = x2
  exact add_right_cancel h1

-- 2ª demostración
example : Injective ((. + c)) :=
by
  intro x1 x2 h1
  show x1 = x2
  exact add_right_cancel h1

-- 3ª demostración
example : Injective ((. + c)) :=
  fun _ _ h ↦ add_right_cancel h
```

```
-- Lemas usados
-- ============

-- variable {a b : ℝ}
-- #check (add_right_cancel : a + b = c + b → a = c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.4. Si c ≠ 0, entonces la función (x ↦ cx) es inyectiva

```
-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que para todo c distinto de cero la función
--    f(x) = c * x
-- es inyectiva
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el lema
--    (∀ a, b, c) [a ≠ 0 → (a * b = a * c ↔ b = c))]             (L1)
-- Hay que demostrar que
--    (∀ x₁, x₂) [f(x₁) = f(x₂) → x₁ = x₂]
-- Sean x₁, x₂ tales que f(x₁) = f(x₂). Entonces,
--    cx₁ = cx₂
-- y, por L1 y puesto que c ≠ 0, se tiene que
--    x₁ = x₂.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function
variable {c : ℝ}

-- 1ª demostración
example
  (h : c ≠ 0)
  : Injective ((c * .)) :=
by
  intro (x1 : ℝ) (x2 : ℝ) (h1 : c * x1 = c * x2)
```

```
  show x1 = x2
  exact (mul_right_inj' h).mp h1

-- 2ª demostración
example
  (h : c ≠ 0)
  : Injective ((c * .)) :=
fun _ _ h1 ↦ mul_left_cancel₀ h h1

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_right_inj' : a ≠ 0 → (a * b = a * c ↔ b = c))
-- #check (mul_left_cancel₀ : a ≠ 0 → a * b = a * c → b = c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.5. La composición de funciones inyectivas es inyectiva

```
-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Tenemos que demostrar que
--    (∀ x, y) [(g ∘ f)(x) = (g ∘ f)(y) → x = y]
-- Sean x, y tales que
--    (g ∘ f)(x) = (g ∘ f)(y)
-- Entonces, por la definición de la composición,
--    g(f(x)) = g(f(y))
-- y, ser g inyectiva,
--    f(x) = f(y)
-- y, ser f inyectiva,
--    x = y

-- 2ª demostración en LN
-- =====================

-- Tenemos que demostrar que
```

```
--    (∀ x, y) [(g ∘ f)(x) = (g ∘ f)(y) → x = y]
-- Sean x, y tales que
--    (g ∘ f)(x) = (g ∘ f)(y)                                  (1)
-- y tenemos que demostrar que
--    x = y                                                    (2)
-- El objetivo (2), usando que f es inyectiva, se reduce a
--    f(x) = f(y)
-- que, usando que g es inyectiva, se reduce a
--    g(f(x)) = g(f(y))
-- que, por la definición de la composición, coincide con (1).

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Function

variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}

-- 1ª demostración (basada en la 1ª en LN)
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro (x : α) (y : α) (h1: (g ∘ f) x = (g ∘ f) y)
  have h2: g (f x) = g (f y) := h1
  have h3: f x = f y := hg h2
  show x = y
  exact hf h3

-- 2ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro (x : α) (y : α) (h1: (g ∘ f) x = (g ∘ f) y)
  have h2: f x = f y := hg h1
  show x = y
  exact hf h2

-- 3ª demostración
```

```
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro x y h
  exact hf (hg h)

-- 4ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
fun _ _ h ↦ hf (hg h)

-- 5ª demostración (basada en la 2ª en LN)
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intros x y h
  -- x y : α
  -- h : (g ∘ f) x = (g ∘ f) y
  apply hf
  -- ⊢ f x = f y
  apply hg
  -- ⊢ g (f x) = g (f y)
  apply h

-- 6ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
-- by exact?
Injective.comp hg hf

-- 7ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by tauto
```

```
-- Lemas usados
-- ============

-- #check (Injective.comp : Injective g → Injective f → Injective (g ∘ f))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.6. La función (x ↦ x + c) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que para todo número real c, la función
--    f(x) = x + c
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x ∈ ℝ)(∃ y ∈ ℝ)[y+c = x]
-- Sea x ∈ ℝ. Entonces, y = x-c ∈ ℝ y
--    y + c = (x - c) + c
--          = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable {c : ℝ}

open Function

-- 1ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  dsimp
  -- ⊢ (x - c) + c = x
```

```
  exact sub_add_cancel x c

-- 2ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  change (x - c) + c = x
  -- ⊢ (x - c) + c = x
  exact sub_add_cancel x c

-- 3ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  exact sub_add_cancel x c

-- 4ª demostración
example : Surjective (fun x ↦ x + c) :=
fun x ↦ ⟨x - c, sub_add_cancel x c⟩

-- 5ª demostración
example : Surjective (fun x ↦ x + c) :=
fun x ↦ ⟨x - c, by ring⟩

-- 6ª demostración
example : Surjective (fun x ↦ x + c) :=
add_right_surjective c

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (sub_add_cancel a b : (a - b) + b = a)
-- #check (add_right_surjective c : Surjective (fun x ↦ x + c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.7. Si c ≠ 0, entonces la función (x ↦ cx) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es un número real no nulo, entonces la función
--    f(x) = c * x
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Hay que demostrar que
--    (∀ x ∈ ℝ)(∃ y ∈ ℝ)[cy = x]
-- Sea x ∈ ℝ. Entonces, y = x/c ∈ R y
--    cy = c(x/c)
--       = y

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable {c : ℝ}
open Function

-- 1ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x) a = x
  use (x / c)
  -- ⊢ (fun x => c * x) (x / c) = x
  dsimp
  -- ⊢ c * (x / c) = x
  rw [mul_comm]
  -- ⊢ (x / c) * c = x
  exact div_mul_cancel x h

-- 2ª demostración
example
  (h : c ≠ 0)
```

```
  : Surjective (fun x ↦ c * x) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x) a = x
  use (x / c)
  -- ⊢ (fun x => c * x) (x / c) = x
  exact mul_div_cancel' x h

-- 3ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
fun x ↦ ⟨x / c, mul_div_cancel' x h⟩

-- 4ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
mul_left_surjective₀ h

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (div_mul_cancel a : b ≠ 0 → (a / b) * b = a)
-- #check (mul_comm a b : a * b = b * a)
-- #check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
-- #check (mul_left_surjective₀ : c ≠ 0 → Surjective (fun x ↦ c * x))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.8. Si c ≠ 0, entonces la función (x ↦ cx + d) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es un número real no nulo, entonces la función
--    f(x) = c * x + d
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- ==================================

-- Hay que demostrar que
--    (∀x ∈ ℝ)(∃y ∈ ℝ)[cy+d = x]
-- Sea x ∈ ℝ. Entonces, y = (x-d)/c ∈ R y
--    cy = c((x-d)/c)+d
--       = (x-d)+d
--       = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable {c d : ℝ}
open Function

-- 1ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  dsimp
  -- ⊢ c * ((x - d) / c) + d = x
  show c * ((x - d) / c) + d = x
  calc c * ((x - d) / c) + d
         = (x - d) + d := congrArg (. + d) (mul_div_cancel' (x - d) h)
       _ = x           := sub_add_cancel x d

-- 2ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
```

```
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  dsimp
  -- ⊢ c * ((x - d) / c) + d = x
  simp [mul_div_cancel', h]

-- 3ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  simp [mul_div_cancel', h]

-- 4ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
fun x ↦ ⟨(x - d) / c, by simp [mul_div_cancel', h]⟩

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
-- #check (sub_add_cancel a b : a - b + b = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.9. Si f: ℝ → ℝ es suprayectiva, entonces ∃x ∈ ℝ tal que f(x)² = 9

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función suprayectiva de ℝ en ℝ,
-- entonces existe un x tal que (f x)^2 = 9.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Al ser f suprayectiva, existe un y tal que f(y) = 3. Por tanto,
-- f(y)² = 9.

-- Demostración con Lean9
-- ======================

import Mathlib.Data.Real.Basic

open Function

example
  {f : ℝ → ℝ}
  (h : Surjective f)
  : ∃ x, (f x)^2 = 9 :=
by
  cases' h 3 with y hy
  -- y : ℝ
  -- hy : f y = 3
  use y
  -- ⊢ f y ^ 2 = 9
  rw [hy]
  -- ⊢ 3 ^ 2 = 9
  norm_num
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

## 12.10. La composición de funciones suprayectivas es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que la composición de funciones suprayectivas es
-- suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- ==================================

-- Supongamos que f : A → B y g : B → C son suprayectivas. Tenemos que
-- demostrar que
--     (∀z ∈ C)(∃x ∈ A)[g(f(x)) = z]
-- Sea z ∈ C. Por ser g suprayectiva, existe un y ∈ B tal que
--     g(y) = z                                                   (1)
-- Por ser f suprayectiva, existe un x ∈ A tal que
--     f(x) = y                                                   (2)
-- Por tanto,
--     g(f(x)) = g(y)   [por (2)]
--             = z      [por (1)]

-- Demostraciones con lean4
-- ========================

import Mathlib.Tactic
open Function
variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}

-- 1ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  use x
  -- ⊢ (g ∘ f) x = z
  dsimp
  -- ⊢ g (f x) = z
  rw [hx]
  -- ⊢ g y = z
  exact hy

-- 2ª demostración
```

```
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  use x
  -- ⊢ (g ∘ f) x = z
  dsimp
  -- ⊢ g (f x) = z
  rw [hx, hy]

-- 3ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  exact ⟨x, by dsimp ; rw [hx, hy]⟩

-- 4ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
```

```
  -- ⊢ ∃ a, (g ∘ f) a = z
  rcases hg z with ⟨y, hy : g y = z⟩
  rcases hf y with ⟨x, hx : f x = y⟩
  exact ⟨x, by dsimp ; rw [hx, hy]⟩

-- 5ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
Surjective.comp hg hf

-- Lemas usados
-- ============

-- #check (Surjective.comp : Surjective g → Surjective f → Surjective (g ∘ f))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

# Bibliografía


[1] J. A. Alonso. Lean para matemáticos [1], 2021.

[2] J. A. Alonso. Matemáticas en Lean [2], 2021.

[3] J. A. Alonso. DAO (Demostración Asistida por Ordenador) con Lean [3], 2021.

[4] J. A. Alonso. Calculemus (Vol. 1: Demostraciones con Isabelle/HOL y Lean3) [4] , 2021.

[5] J. Avigad, L. de Moura, and S. Kong. Theorem Proving in Lean4 [5], 2021.

[6] J. Avigad, G. Ebner, and S. Ullrich. The Lean4 Manual [6], 2021.

[7] J. Avigad, M. J. H. Heule, and W. Nawrocki. Logic and mechanized reasoning [7] , 2023.

[8] J. Avigad, R. Y. Lewis, and F. van Doorn. Logic and proof [8] , 2021.

[9] J. Avigad and P. Massot. Mathematics in Lean [9] , 2023.

[10] A. Baanen, A. Bentkamp, J. Blanchette, J. Hölzl, and J. Limperg. The Hitchhiker's Guide to Logical Verification [10], 2020.


[1] https://github.com/jaalonso/Lean_para_matematicos

[2] https://github.com/jaalonso/Matematicas_en_Lean

[3] https://raw.githubusercontent.com/jaalonso/DAO_con_Lean/master/DAO_con_Lean.pdf

[4] https://raw.githubusercontent.com/jaalonso/Calculemus/master/Calculemus.pdf

[5] https://leanprover.github.io/theorem_proving_in_lean4/

[6] https://leanprover.github.io/lean4/doc/

[7] https://avigad.github.io/lamr/logic_and_mechanized_reasoning.pdf

[8] https://leanprover.github.io/logic_and_proof/logic_and_proof.pdf

[9] https://leanprover-community.github.io/mathematics_in_lean/

[10] https://raw.githubusercontent.com/blanchette/logical_verification_2020/master/hitchhikers_guide.pdf

[11] M. Ballard. Transition to advanced mathematics (Thinking and communicating like a mathematician) [11].

[12] K. Buzzard. Sets and logic (in Lean) [12].

[13] K. Buzzard. Functions and relations (in Lean) [13].

[14] K. Buzzard. Course on formalising mathematics [14], 2021.

[15] K. Buzzard. Course on formalising mathematics [15], 2023.

[16] K. Buzzard and M. Pedramfar. The Natural Number Game, version 1.3.3 [16].

[17] D. T. Christiansen. Functional programming in Lean [17], 2023.

[18] M. Dvořák. Lean 4 Cheatsheet [18].

[19] S. Hazratpour. Introduction to proofs [19], 2022.

[20] S. Hazratpour. Introduction to proofs with Lean proof assistant [20], 2022.

[21] R. Lewis. Formal proof and verification, 2022 [21], 2022.

[22] R. Lewis. Discrete structures and probability [22], 2023.

[23] C. Löh. Exploring formalisation (A primer in human-readable mathematics in Lean 3 with examples from simplicial topology) [23], 2022.

[24] H. Macbeth. The mechanics of proof [24], 2023.

[25] P. Massot. Introduction aux mathématiques formalisées [25].

[11] https://300.f22.matthewrobertballard.com/
[12] https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C1.html
[13] https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C2.html
[14] https://github.com/ImperialCollegeLondon/formalising-mathematics
[15] https://github.com/ImperialCollegeLondon/formalising-mathematics-2023
[16] https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/
[17] https://leanprover.github.io/functional_programming_in_lean/
[18] https://raw.githubusercontent.com/madvorak/lean4-cheatsheet/main/lean-tactics.pdf
[19] https://introproofs.github.io/s22/
[20] https://sinhp.github.io/teaching/2022-introduction-to-proofs-with-Lean
[21] https://github.com/BrownCS1951x/fpv2022
[22] https://github.com/brown-cs22/CS22-Lean-2023
[23] https://loeh.app.uni-regensburg.de/mapa/main.pdf
[24] https://hrmacbeth.github.io/math2001/index.html
[25] https://www.imo.universite-paris-saclay.fr/~pmassot/enseignement/math114/

[26] F. L. Roux. Code Lean contenant les preuves d'un cours standard sur les espaces métriques [26], 2020.

[27] W. Schulze. Learning LeanProver [27].

[28] Varios. LFTCM 2020: Lean for the Curious Mathematician 2020 [28].

[29] D. J. Velleman. How to prove it with Lean [29].

[26] https://github.com/FredericLeRoux/LEAN_ESPACES_METRIQUES
[27] https://youtube.com/playlist?list=PLYwF9EIrl42RFQgbmcR_LSCWRIx2WKbXs
[28] https://leanprover-community.github.io/lftcm2020/schedule.html
[29] https://djvelleman.github.io/HTPIwL/

# Lemas usados

```
import Mathlib.Algebra.Group.Basic
import Mathlib.Algebra.Order.Ring.Defs        -- 1
import Mathlib.Algebra.Ring.Defs
import Mathlib.Analysis.SpecialFunctions.Log.Basic
import Mathlib.Data.Real.Basic
import Mathlib.Order.Lattice
import Mathlib.Topology.MetricSpace.Basic

-- Números naturales
-- =================

section naturales
variable (x y z k m n : ℕ)
#check (_root_.dvd_antisymm : m ∣ n → n ∣ m → m = n)
#check (dvd_add : x ∣ y → x ∣ z → x ∣ y + z)
#check (dvd_gcd : k ∣ m → k ∣ n → k ∣ gcd m n)
#check (dvd_mul_left x y : x ∣ y * x)
#check (dvd_mul_of_dvd_left : x ∣ y → ∀ (c : ℕ), x ∣ y * c)
#check (dvd_mul_of_dvd_right : x ∣ y → ∀ (c : ℕ), x ∣ c * y)
#check (dvd_mul_right x y : x ∣ x * y)
#check (dvd_trans : x ∣ y → y ∣ z → x ∣ z)
#check (Dvd.intro k : m * k = n → m ∣ n)
#check (gcd_comm m n : gcd m n = gcd n m)
#check (gcd_dvd_left  m n: gcd m n ∣ m)
#check (gcd_dvd_right m n : gcd m n ∣ n)
end naturales

-- Números reales
-- ==============

section reales
open Real
variable (a b c d : ℝ)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
```

```
#check (abs_le' : |a| ≤ b ↔ a ≤ b ∧ -a ≤ b)
#check (abs_mul a b : |a * b| = |a| * |b|)
#check (abs_nonneg a : 0 ≤ |a|)
#check (abs_sub_abs_le_abs_sub a b : |a| - |b| ≤ |a - b|)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (add_le_add_left : b ≤ c → ∀ (a : ℝ), a + b ≤ a + c)
#check (add_le_add_right : b ≤ c → ∀ (a : ℝ),  b + a ≤ c + a)
#check (add_lt_add_of_le_of_lt : a ≤ b → c < d → a + c < b + d)
#check (add_lt_add_of_lt_of_le : a < b → c ≤ d → a + c < b + d)
#check (add_lt_add_right : b < c → ∀ (a : ℝ), b + a < c + a)
#check (add_neg_le_iff_le_add : a - b ≤ c ↔ a ≤ c + b)
#check (add_pos : 0 < a → 0 < b → 0 < a + b)
#check (add_sub_cancel a b : a + b - b = a)
#check (div_mul_cancel a : b ≠ 0 → (a / b) * b = a)
#check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
#check (exp_lt_exp : exp a < exp b ↔ a < b)
#check (exp_pos a : 0 < exp a)
#check (half_lt_self : 0 < a → a / 2 < a)
#check (half_pos : 0 < a → 0 < a / 2)
#check (le_antisymm : a ≤ b → b ≤ a → a = b)
#check (le_div_iff : 0 < c → (a ≤ b / c ↔ a * c ≤ b))
#check (le_max_left a b : a ≤ max a b)
#check (le_max_right a b : b ≤ max a b)
#check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
#check (le_of_not_gt : ¬a > b → a ≤ b)
#check (le_refl a : a ≤ a)
#check (log_le_log' : 0 < a → a ≤ b → log a ≤ log b)
#check (lt_asymm : a < b → ¬b < a)
#check (lt_irrefl a : ¬a < a)
#check (lt_of_lt_of_le : a < b → b ≤ c → a < c)
#check (lt_of_le_of_lt : a ≤ b → b < c → a < c)
#check (lt_of_le_of_ne : a ≤ b → a ≠ b → a < b)
#check (lt_of_not_ge : ¬ a ≥ b → a < b)
#check (lt_trans : a < b → b < c → a < c)
#check (max_comm a b : max a b = max b a)
#check (max_le : a ≤ c → b ≤ c → max a b ≤ c)
#check (min_add_add_right a b c : min (a + c) (b + c) = min a b + c)
#check (min_assoc a b c : min (min a b) c = min a (min b c))
#check (min_comm a b : min a b = min b a)
#check (min_eq_left : a ≤ b → min a b = a)
#check (min_eq_right : b ≤ a → min a b = b)
#check (min_le_left a b : min a b ≤ a)
#check (min_le_right a b : min a b ≤ b)
#check (mul_comm a b : a * b = b * a)
#check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
```

```
#check (mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d)
#check (mul_le_mul_right : 0 < a → (b * a ≤ c * a ↔ b ≤ c))
#check (mul_left_cancel₀ : a ≠ 0 → a * b = a * c → b = c)
#check (mul_lt_mul_left : 0 < a → (a * b < a * c ↔ b < c))
#check (mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c))
#check (mul_neg a b : a * -b = -(a * b))
#check (mul_right_inj' :  a ≠ 0 → (a * b = a * c ↔ b = c))
#check (mul_sub a b c : a * (b - c) = a * b - a * c)
#check (mul_two a : a * 2 = a + a)
#check (ne_comm : a ≠ b ↔ b ≠ a)
#check (neg_add_self a : -a + a = 0)
#check (neg_mul_neg a b : -a * -b = a * b)
#check (nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a)
#check (not_lt_of_ge : a ≥ b → ¬ a < b)
#check (pow_two a : a ^ 2 = a * a)
#check (pow_two_nonneg a : 0 ≤ a ^ 2)
#check (sq_nonneg a : 0 ≤ a ^ 2)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_le_sub_left : a ≤ b → ∀ (c : ℝ), c - b ≤ c - a)
#check (sub_le_sub_right : a ≤ b → ∀ (c : ℝ), a - c ≤ b - c)
#check (sub_sq a b : (a - b) ^ 2 = a ^ 2 - 2 * a * b + b ^ 2)
#check (two_mul a : 2 * a = a + a)
#check (two_mul_le_add_sq a b : 2 * a * b ≤ a ^ 2 + b ^ 2)
#check (zero_lt_one : 0 < 1)
#check (zero_lt_two : 0 < 2)
end reales

-- Anillos
-- =======

section anillos
variable {R : Type _} [Ring R]
variable (a b c : R)
#check (add_assoc a b c : (a + b) + c = a + (b + c))
#check (add_comm a b : a + b = b + a)
#check (add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b)
#check (add_left_cancel : a + b = a + c → b = c)
#check (add_left_neg a : -a + a = 0)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
#check (add_neg_cancel_right a b : (a + b) + -b = a)
#check (add_neg_self a : a + -a = 0)
#check (add_right_cancel : a + b = c + b → a = c)
#check (add_right_neg a : a + -a = 0)
#check (add_zero a : a + 0 = a)
#check (mul_add a b c : a * (b + c) = a * b + a * c)
```

```
#check (mul_zero a : a * 0 = 0)
#check (neg_add_cancel_left a b : -a + (a + b) = b)
#check (neg_eq_iff_add_eq_zero : -a = b ↔ a + b = 0)
#check (neg_eq_of_add_eq_zero_left : a + b = 0 → -b = a)
#check (neg_eq_of_add_eq_zero_right : a + b = 0 → -a = b)
#check (neg_neg a : -(-a) = a)
#check (neg_zero : -0 = 0)
#check (one_add_one_eq_two : (1 : R) + 1 = 2)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_eq_add_neg a b : a - b = a + -b)
#check (sub_mul a b c : (a - b) * c = a * c - b * c)
#check (sub_self a : a - a = 0)
#check (two_mul a : 2 * a = a + a)
#check (zero_add a : 0 + a = a)
#check (zero_mul a : 0 * a = 0)
end anillos

-- Grupos
-- ======

section grupos
variable {G : Type _} [Group G]
variable (a b c : G)
#check (inv_eq_of_mul_eq_one_right : a * b = 1 → a⁻¹ = b)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
#check (mul_inv_self a : a * a⁻¹ = 1)
#check (mul_inv_rev a b : (a * b)⁻¹ = b⁻¹ * a⁻¹)
#check (mul_left_inv a : a⁻¹  * a = 1)
#check (mul_one a : a * 1 = a)
#check (mul_right_inv a : a * a⁻¹ = 1)
#check (one_mul a : 1 * a = a)
end grupos

-- Retículos
-- =========

section reticulos
variable {α : Type _} [Lattice α]
variable (x y z : α)
#check (inf_assoc : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z))
#check (inf_comm : x ⊓ y = y ⊓ x)
#check (inf_le_left : x ⊓ y ≤ x)
#check (inf_le_of_left_le : x ≤ z → x ⊓ y ≤ z)
#check (inf_le_of_right_le : y ≤ z → x ⊓ y ≤ z)
#check (inf_le_right : x ⊓ y ≤ y)
```

```
#check (inf_sup_self : x ⊓ (x ⊔ y) = x)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)
#check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
#check (le_rfl : x ≤ x)
#check (le_sup_left : x ≤ x ⊔ y)
#check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
#check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
#check (le_sup_right : y ≤ x ⊔ y)
#check (le_trans : x ≤ y → y ≤ z → x ≤ z)
#check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
#check (sup_comm : x ⊔ y = y ⊔ x)
#check (sup_inf_self : x ⊔ (x ⊓ y) = x)
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
end reticulos

-- AnillosOrdenados
-- ================

section AnillosOrdenados
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)
#check (add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a)
#check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
#check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
#check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
#check (sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c)
#check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
end AnillosOrdenados

-- Espacios métricos
-- =================

section EspacioMetrico
variable {X : Type _} [MetricSpace X]
variable (x y z : X)
#check (dist_comm x y : dist x y = dist y x)
#check (dist_nonneg : 0 ≤ dist x y)
#check (dist_self x : dist x x = 0)
#check (dist_triangle x y z : dist x z ≤ dist x y + dist y z)
end EspacioMetrico

-- Conjuntos
-- =========

section Conjuntos
```

```
open Set
variable {α : Type _}
variable (r s t : Set α)
#check (Subset.trans : r ⊆ s → s ⊆ t → r ⊆ t)
end Conjuntos

-- Órdenes parciales
-- =================

section OrdenParcial
variable {α : Type _} [PartialOrder α]
variable (a b c : α)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
#check (monotone_const : Monotone fun _ : ℝ ↦ c)
end OrdenParcial

-- Funciones
-- =========

section Funciones
open Function
variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}
variable (c : ℝ)
#check (Injective.comp : Injective g → Injective f → Injective (g ∘ f))
#check (Surjective.comp : Surjective g → Surjective f → Surjective (g ∘ f))
#check (add_right_surjective c : Surjective (fun x ↦ x + c))
#check (mul_left_surjective₀ : c ≠ 0 → Surjective (fun x ↦ c * x))
end Funciones

-- Lógica
-- ======

section Logica
variable (p q : Prop)
variable {α : Type _}
variable (P : α → Prop)
#check (absurd : p → ¬p → q)
#check (forall_not_of_not_exists : (¬∃ x, P x) → ∀ x, ¬P x)
#check (not_exists : (¬∃ x, P x) ↔ ∀ (x : α), ¬P x)
#check (not_exists_of_forall_not : (∀ x, P x → q) → (∃ x, P x) → q)
#check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
#check (not_forall_of_exists_not : (∃ x, ¬P x) → ¬∀ x, P x)
#check (not_not_intro : p → ¬¬p)
#check (of_not_not : ¬¬p → p)
```

```
end Logica
```