Matemáticas en Lean4

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 6 de junio de 2025

Índice general

Capítulo 1

Introducción

1.1. Resumen

El objetivo de este trabajo es presentar el uso de Lean4 (y su librería matemática mathlib4) mediante ejemplos matemáticos. Está basado en el libro Mathematics in Lean de Jeremy Avigad y Patrick Massot.

Los ejercicios se han ido publicando en el blog Calculemus y su código en GitHub.

1.2. Presentación panorámica de Lean

1.2.1. Ejemplo de evaluación

```
-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el valor de 2+3.
-- ---------------------------------------------------------------------

#eval 2 + 3

-- Comentario: Al poner el cursor sobre eval se escribe 5 como resultado
-- al final de la línea.
```

1.2.2. Ejemplo de comprobación con check

```
-- ---------------------------------------------------------------------
-- Ejercicio: Calcular el tipo de la expresión 2+3.
-- ---------------------------------------------------------------------
```

```
#check 2 + 3

-- Comentario: Al colocar el cursor sobre check escribe al final de la
-- línea
--    2 + 3 : Nat
-- que indica que el valor de la expresión es un número natural.
```

1.2.3. Ejemplo de definición de funciones

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar la teoría de los números naturales.
-- ---------------------------------------------------------------------

import Mathlib.Data.Nat.Basic

-- ---------------------------------------------------------------------
-- Ejercicio. Definir la función f que le suma 3 a cada número natural.
-- ---------------------------------------------------------------------

def f (x : ℕ) :=
  x + 3

-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de f.
-- ---------------------------------------------------------------------

#check f

-- Comentario: Al colocar el cursor sobre check se obtiene
--    f (x : ℕ) : ℕ

-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el valor de f(2).
-- ---------------------------------------------------------------------

#eval f 2

-- Comentario: Al colocar el cursor sobre eval escribe su valor (5).
```

1.2.4. Ejemplo de proposiciones

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar la teoría de los números naturales.
-- ---------------------------------------------------------------------

import Mathlib.Data.Nat.Basic

-- ---------------------------------------------------------------------
-- Ejercicio. Definir la proposión ultimo_teorema_de_Fermat que
-- expresa el último teorema de Fermat.
-- ---------------------------------------------------------------------

def ultimo_teorema_de_Fermat :=
  ∀ x y z n : ℕ, n > 2 → x * y * z ≠ 0 → x^n + y^n ≠ z^n

-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de ultimo_teorema_de_Fermat
-- ---------------------------------------------------------------------

#check ultimo_teorema_de_Fermat

-- Comentario: Al colocar el cursor sobre check se obtiene
--    ultimo_teorema_de_Fermat : Prop
```

1.2.5. Ejemplo de teoremas

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar la teoría de los números naturales.
-- ---------------------------------------------------------------------

import Mathlib.Data.Nat.Basic

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar el teorema facil que afirma que 2 + 3 = 5.
-- ---------------------------------------------------------------------

theorem facil : 2 + 3 = 5 := rfl

-- Comentarios:
-- 1. Para activar la ventana de objetivos (*Lean Goal*) se escribe
--    C-c TAB
-- 2. Se desactiva volviendo a escribir C-c TAB
```

```
-- 3. La táctica rfl comprueba que 2+3 y 5 son iguales por definición.

-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de facil
-- ----------------------------------------------------------------------

#check facil

-- Comentario: Colocando el cursor sobre check se obtiene
--    facil : 2 + 3 = 5

-- ----------------------------------------------------------------------
-- Ejercicio. Enunciar el teorema dificil que afirma que se verifica
-- el último teorema de Fermat, omitiendo la demostración.
-- ----------------------------------------------------------------------

def ultimo_teorema_de_Fermat :=
  ∀ x y z n : ℕ, n > 2 → x * y * z ≠ 0 → x^n + y^n ≠ z^n

-- theorem dificil : ultimo_teorema_de_Fermat :=
-- sorry

-- Comentarios:
-- 1. La palabra sorry se usa para omitir la demostración.
-- 2. Se puede verificar la teoría pulsando
--       C-c ! l
--    Se obtiene
--       Line Col Level     Message
--    24   1 info      facil : 2 + 3 = 5 (lsp)
--    37   9 warning   declaration uses 'sorry' (lsp)

-- ----------------------------------------------------------------------
-- Ejercicio 3. Calcular el tipo de dificil.
-- ----------------------------------------------------------------------

-- #check dificil

-- Comentario: Al colocar el cursor sobre check se obtiene
--    dificil : ultimo_teorema_de_Fermat
```

1.2.6. Ejemplo de demostración

```
-- ---------------------------------------------------------------------
-- 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.Algebra.Ring.Parity
import Mathlib.Tactic

variable (m n : ℕ)

open Nat

-- 1ª demostración
-- ===============

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

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

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

-- 3ª demostración
-- ===============

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

-- 8ª demostración
-- ===============

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

-- 9ª demostración
-- ===============

example : ∀ m n : ℕ, Even n → Even (m * n) :=
by
  intros m n hn
```

```
  -- m n : ℕ
  -- hn : Even n
  -- ⊢ Even (m * n)
  unfold Even at *
  -- hn : ∃ r, n = r + r
  -- ⊢ ∃ r, m * n = r + r
  cases hn with
  | intro k hk =>
    -- k : ℕ
    -- hk : n = k + k
    use m * k
    -- ⊢ m * n = m * k + m * k
    calc m * n
       = m * (k + k)   := congrArg (HMul.hMul m) hk
     _ = m * k + m * k := mul_add m k k

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

example : ∀ m n : Nat, Even n → Even (m * n) :=
by
  intros
  -- m n : ℕ
  -- a : Even n
  -- ⊢ Even (m * n)
  simp [*, parity_simps]

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

variable (a b c : ℕ)
variable (f : ℕ → ℕ)
#check (mul_add a b c : a * (b + c) = a * b + a * c)
#check (congrArg f : a = b → f a = f b)
```

Capítulo 2

Aspectos básicos del razonamiento matemático en Lean

En este capítulo se presentan los aspectos básicos del razonamiento matemático en Lean:

- cálculos,
- aplicación de lemas y teoremas y
- razonamiento sobre estructuras genéricas.

2.1. Cálculos

2.1.1. Asociativa conmutativa de los reales

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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

variable (a b c : ℝ)

-- 1ª demostración
-- ===============

example : (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]

-- Comentarios:
-- + El entorno calc permite escribir demostraciones ecuacionales.
-- + La táctica (rw [es]) reescribe una expresión usando las ecuaciones es.
-- + Al colocar el cursor sobre el nombre de un lema se ve su enunciado.
-- + Para completar el nombre de un lema basta escribir parte de su
--   nombre y completar con S-SPC (es decir, simultáneamente las teclas
--   de mayúscula y la de espacio).

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

example : (a * b) * c = b * (a * c) :=
by ring

-- Comentario: La táctica ring demuestra ecuaciones aplicando las
-- propiedades de anillos.

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

```
#check (mul_comm a b : a * b = b * a)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
```

2.1.2. Ejercicio sobre aritmética real (1)

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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

variable (a b c : ℝ)

-- 1ª demostración
-- ===============

example : (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 : (c * b) * a = b * (a * c) :=
by
```

```
  rw [mul_comm c b]
  -- ⊢ (b * c) * a = b * (a * c)
  rw [mul_assoc]
  -- ⊢ b * (c * a) = b * (a * c)
  rw [mul_comm c a]

-- 3ª demostración
-- ===============

example : (c * b) * a = b * (a * c) :=
by ring

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

#check (mul_comm a b : a * b = b * a)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
```

2.1.3. Ejercicio sobre aritmética real (2)

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

variable (a b c : ℝ)
```

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

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

-- Comentario. Con la táctica (rw [←e]) se aplica reescritura sustituyendo
-- el término derecho de la igualdad e por el izquierdo.

-- 3ª demostración
-- ===============

example : a * (b * c) = b * (a * c) :=
by ring

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

#check (mul_comm a b : a * b = b * a)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
```

2.1.4. Ejemplo de rw con hipótesis

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si a, b, c, d, e y f son números reales
-- tales que
--    a * b = c * d
--    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.Data.Real.Basic
import Mathlib.Tactic

variable (a b c d e f : ℝ)

-- 1ª demostración
-- ===============

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

```
  -- ⊢ a * (b * f) = c * (d * f)
  rw [←mul_assoc]
  -- ⊢ (a * b) * f = c * (d * f)
  rw [h1]
  -- ⊢ (c * d) * f = c * (d * f)
  rw [mul_assoc]

-- Comentario: La táctica (rw [h2]) reescribe el objetivo con la igualdad
-- de la hipótesis h2.

-- 3ª demostración
-- ===============

example
  (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 : (a * b) * c = a * (b * c))
```

2.1.5. Ejercicio de rw con hipótesis (1)

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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

variable (a b c d e f : ℝ)

-- 1ª demostración
-- ===============

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

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

#check (mul_assoc a b c : (a * b) * c = a * (b * c))
```

2.1.6. Ejercicio de rw con hipótesis (2)

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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

variable (a b c d : ℝ)

-- 1ª demostración
-- ===============

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

```
  rw [h1]
  -- ⊢ b * a - d = 0
  rw [mul_comm]
  -- ⊢ a * b - d = 0
  rw [h2]
  -- ⊢ a * b - a * b = 0
  rw [sub_self]

-- Comentario: El último lema se puede encontrar escribiendo previamente
--    exact?

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

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

2.1.7. Declaración de variables en secciones

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar la librería básica de los números reales.
-- ---------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio. Crear una sección.
-- ---------------------------------------------------------------------

section

-- ---------------------------------------------------------------------
-- Ejercicio. Declarar que a, b y c son variables sobre los números
-- reales.
-- ---------------------------------------------------------------------

variable (a b c : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de a.
-- ---------------------------------------------------------------------
```

```
#check a

-- Comentario: Al colocar el cursor sobre check se obtiene
--    a : ℝ

-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de a + b.
-- ----------------------------------------------------------------------

#check a + b

-- Comentario: Al colocar el cursor sobre check se obtiene
--    a + b : ℝ

-- ----------------------------------------------------------------------
-- Ejercicio. Comprobar que a es un número real.
-- ----------------------------------------------------------------------

#check (a : ℝ)

-- Comentario: Al colocar el cursor sobre check se obtiene
--    a : ℝ

-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--    mul_comm a b
-- ----------------------------------------------------------------------

#check mul_comm a b

-- Comentario: Al colocar el cursor sobre check se obtiene
--    mul_comm a b : a * b = b * a

-- ----------------------------------------------------------------------
-- Ejercicio. Comprobar que el tipo de
--    mul_comm a b
-- es
--    a * b = b * a
-- ----------------------------------------------------------------------

#check (mul_comm a b : a * b = b * a)

-- Comentario: Al colocar el cursor sobre check se obtiene
--    mul_comm a b : a * b = b * a
```

```
-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--    mul_assoc c a b
-- ----------------------------------------------------------------------

#check mul_assoc c a b

-- Comentario: Al colocar el cursor sobre check se obtiene
--    mul_assoc c a b : c * a * b = c * (a * b)

-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--    mul_comm a
-- ----------------------------------------------------------------------

#check mul_comm a

-- Comentario: Al colocar el cursor sobre check se obtiene
--    mul_comm a : ∀ (b : ℝ), a * b = b * a

-- ----------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--    mul_comm
-- ----------------------------------------------------------------------

#check mul_comm

-- Comentario: Al colocar el cursor sobre check se obtiene
--    mul_comm.{u_1} {G : Type u_1} [CommMagma G] (a b : G)
--    : a * b = b * a

-- ----------------------------------------------------------------------
-- Ejercicio 12. Calcular el tipo de
--    @mul_comm
-- ----------------------------------------------------------------------

#check @mul_comm

-- Comentario: Al colocar el cursor sobre check se obtiene
--    @mul_comm : ∀ {G : Type u_1} [inst : CommMagma G] (a b : G),
--    a * b = b * a

end
```

2.1.8. Demostración con calc

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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]
  -- ⊢ (a + b) * a + (a + b) * b = a * a + 2 * (a * b) + b * b
  rw [add_mul]
  -- ⊢ a * a + b * a + (a + b) * b = a * a + 2 * (a * b) + b * b
  rw [add_mul]
  -- ⊢ a * a + b * a + (a * b + b * b) = a * a + 2 * (a * b) + b * b
  rw [←add_assoc]
  -- ⊢ a * a + b * a + a * b + b * b = a * a + 2 * (a * b) + b * b
  rw [add_assoc (a * a)]
  -- ⊢ a * a + (b * a + a * b) + b * b = a * a + 2 * (a * b) + b * b
  rw [mul_comm b a]
  -- ⊢ a * a + (a * b + a * b) + b * b = a * a + 2 * (a * b) + b * b
```

```
  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]
  -- ⊢ a * a + b * a + (a * b + b * b) = a * a + 2 * (a * b) + b * b
  rw [←add_assoc, add_assoc (a * a)]
  -- ⊢ a * a + (b * a + a * b) + b * b = a * a + 2 * (a * b) + b * b
  rw [mul_comm b a, ←two_mul]

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

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

2.1.9. Ejercicio con calc

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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]
    -- ⊢ a * (c + d) + b * (c + d) = a * c + a * d + b * c + b * d
   rw [mul_add]
   -- ⊢ a * c + a * d + b * (c + d) = a * c + a * d + b * c + b * d
```

```
    rw [mul_add]
    -- ⊢ a * c + a * d + (b * c + b * d) = a * c + a * d + b * c + b * d
    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_assoc a b c : (a + b) + c = a + (b + c))
#check (add_mul a b c   : (a + b) * c = a * c + b * c)
#check (mul_add a b c   : a * (b + c) = a * b + a * c)
```

2.1.10. Ejercicio: Suma por diferencia

```
-- ----------------------------------------------------------------------
-- Ejercicio. 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 c d : 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_cancel (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 (por reescritura usando el lema anterior)
-- ==========================================================

-- 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]
  -- ⊢ (a + b) * (a + -b) = a ^ 2 - b ^ 2
  rw [aux]
  -- ⊢ a * a + a * -b + b * a + b * -b = a ^ 2 - b ^ 2
  rw [mul_neg]
  -- ⊢ a * a + -(a * b) + b * a + b * -b = a ^ 2 - b ^ 2
  rw [add_assoc (a * a)]
  -- ⊢ a * a + (-(a * b) + b * a) + b * -b = a ^ 2 - b ^ 2
  rw [mul_comm b a]
  -- ⊢ a * a + (-(a * b) + a * b) + b * -b = a ^ 2 - b ^ 2
  rw [neg_add_cancel]
  -- ⊢ a * a + 0 + b * -b = a ^ 2 - b ^ 2
  rw [add_zero]
  -- ⊢ a * a + b * -b = a ^ 2 - b ^ 2
  rw [← pow_two]
  -- ⊢ a ^ 2 + b * -b = a ^ 2 - b ^ 2
  rw [mul_neg]
  -- ⊢ a ^ 2 + -(b * b) = a ^ 2 - b ^ 2
  rw [← pow_two]
  -- ⊢ a ^ 2 + -b ^ 2 = a ^ 2 - b ^ 2
  rw [← sub_eq_add_neg]
```

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

#check (add_assoc a b c    : (a + b) + c = a + (b + c))
#check (add_mul a b c      : (a + b) * c = a * c + b * c)
#check (add_sub a b c      : a + (b - c) = a + b - c)
#check (add_zero a         : a + 0 = a)
#check (mul_comm a b       : a * b = b * a)
#check (mul_neg a b        : a * -b = -(a * b))
#check (mul_sub a b c      : a * (b - c) = a * b - a * c)
#check (neg_add_cancel a   : -a + a = 0)
#check (sub_eq_add_neg a b : a - b = a + -b)
#check (pow_two a          : a ^ 2 = a * a)
#check (sub_sub a b c      : (a - b) - c = a - (b + c))
```

2.1.11. Reescritura en hipótesis y táctica exact

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  -- h1 : c = d * a + a * d
  clear h2
  rw [mul_comm d a] at h1
  -- h1 : c = a * d + a * d
  rw [← two_mul (a*d)] at h1
  -- h1 : c = 2 * (a * d)
  rw [← mul_assoc 2 a d] at h1
  -- h1 : c = 2 * a * d
  exact h1

-- Comentarios
-- 1. La táctica (rw [e] at h) rescribe la parte izquierda de la
--    ecuación e por la derecha en la hipótesis h.
-- 2. La táctica (exact p) tiene éxito si el tipo de p se unifica con el
--    objetivo.
-- 3. La táctica (clear h) borra la hipótesis h.

-- 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]
  -- ⊢ d * a + b = 2 * a * d
  rw [h2]
  -- ⊢ d * a + a * d = 2 * a * d
  ring

-- 5ª demostración
-- ===============

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by
  rw [h1, h2]
  -- ⊢ d * a + a * d = 2 * a * d
  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 : a * b * c = a * (b * c))
#check (mul_comm a b : a * b = b * a)
#check (two_mul a : 2 * a = a + a)
```

2.1.12. Demostraciones con ring

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sean a, b, c y números reales. Demostrar, con la táctica
-- ring, que
--    (c * b) * a = b * (a * c)
--    (a + b) * (a + b) = a * a + 2 * (a * b) + b * b
--    (a + b) * (a - b) = a^2 - b^2
-- Además, si
--    c = d * a + b
--    b = a * d
-- entonces
--    c = 2 * a * d
-- ---------------------------------------------------------------------

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

variable (a b c d : ℝ)

example : (c * b) * a = b * (a * c) :=
by ring

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

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

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
```

```
by
  rw [h1, h2]
  -- ⊢ d * a + a * d = 2 * a * d
  ring
```

- Ejemplo con nth_rewrite

```
-- ---------------------------------------------------------------
-- 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   := congrArg ((a + b) * .) h
  _ = a * c + b * c := add_mul a b c

-- 2ª demostración
example
  (h : a + b = c)
  : (a + b) * (a + b) = a * c + b * c :=
by
```

```
  nth_rewrite 2 [h]
  -- ⊢ (a + b) * c = a * c + b * c
  rw [add_mul]

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

#check (add_mul a b c : (a + b) * c = a * c + b * c)
```

2.2. Demostraciones en estructuras algebraicas

2.2.1. Demostraciones en anillos

2.2.2. Axiomas de anillos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de anillos.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs

-- ---------------------------------------------------------------------
-- Ejercicio 2. Declarar R como un tipo sobre los anillos.
-- ----------------------------------------------------------------------

variable (R : Type _) [Ring R]

-- ---------------------------------------------------------------------
-- Ejercicio 3. Comprobar que R verifica los axiomas de los anillos.
-- ----------------------------------------------------------------------

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 (zero_add a :  0 + a = a)
#check (neg_add_cancel a : -a + a = 0)
#check (mul_assoc a b c : a * b * c = a * (b * c))
#check (mul_one a : a * 1 = a)
#check (one_mul a : 1 * a = a)
```

```
#check (mul_add a b c : a * (b + c) = a * b + a * c)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
```

2.2.3. Propiedades de anillos conmutativos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de las tácticas.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

-- ---------------------------------------------------------------------
-- Ejercicio 2. Declarar R como una variable de tipo de los anillos
-- conmutativos.
-- ----------------------------------------------------------------------

variable (R : Type _) [CommRing R]

-- ---------------------------------------------------------------------
-- Ejercicio 3. Declarar a, b, c y d como variables sobre R.
-- ----------------------------------------------------------------------

variable (a b c d : R)

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--     (c * b) * a = b * (a * c)
-- ----------------------------------------------------------------------

example : (c * b) * a = b * (a * c) :=
by ring

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--     (a + b) * (a + b) = a * a + 2 * (a * b) + b * b
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 6. Demostrar que
--     (a + b) * (a - b) = a^2 - b^2
```

```
-- -----------------------------------------------------------------------

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

-- ----------------------------------------------------------------------
-- Ejercicio 7. Demostrar que si
--    c = d * a + b
--    b = a * d
-- entonces
--    c = 2 * a * d
-- -----------------------------------------------------------------------

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

2.2.4. Propiedades básicas de anillos

```
-- ----------------------------------------------------------------------
-- Ejercicio 1. Importar la teoría de anillos.
-- -----------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs

-- ----------------------------------------------------------------------
-- Ejercicio 2. Crear el espacio de nombres myRing (para evitar
-- conflictos con los nombres).
-- -----------------------------------------------------------------------

namespace myRing

-- ----------------------------------------------------------------------
-- Ejercicio 2. Declarar R como una variable implícita sobre los anillos.
-- -----------------------------------------------------------------------

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

```
-- ---------------------------------------------------------------------
-- Ejercicio 3. Declarar a como una variable sobre R.
-- ----------------------------------------------------------------------

variable (a : R)

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    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]

-- 1ª demostración
-- ===============

example : a + 0 = a :=
calc a + 0
     = 0 + a := add_comm a 0
   _ = a     := zero_add a

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

theorem add_zero : a + 0 = a :=
by rw [add_comm, zero_add]

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--    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]

-- 1ª demostración
-- ===============

example : a + -a = 0 :=
calc a + -a
     = -a + a := add_comm a (-a)
   _ = 0      := neg_add_cancel a

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

theorem add_right_neg : a + -a = 0 :=
by rw [add_comm, neg_add_cancel]

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

variable (a b : R)
#check (add_comm a b : a + b = b + a)
#check (neg_add_cancel a : -a + a = 0)
#check (zero_add a : 0 + a = a)

-- ------------------------------------------------------------------
-- Ejercicio 6. Cerrar el espacio de nombre myRing.
-- -------------------------------------------------------------------

end myRing
```

```
-- ---------------------------------------------------------------------
-- Ejercicio 7. Calcular el tipo de @myRing.add_zero.
-- ----------------------------------------------------------------------

#check @myRing.add_zero

-- Comentario: Al colocar el cursor sobre check se obtiene
--    myRing.add_zero : ∀ {R : Type u_1} [inst : Ring R] (a : R),
--                        a + 0 = a

-- ---------------------------------------------------------------------
-- Ejercicio 8. Calcular el tipo de @add_zero.
-- ----------------------------------------------------------------------

#check @add_zero

-- Comentario: Al colocar el cursor sobre check se obtiene
--    @add_zero : ∀ {M : Type u_1} [inst : AddZeroClass M] (a : M), a + 0 = a
```

2.2.5. Lema neg_add_cancel_left

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la teoría de anillos.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs

-- ---------------------------------------------------------------------
-- Ejercicio 2. Crear el espacio de nombre MyRing
-- ----------------------------------------------------------------------

namespace MyRing

-- ---------------------------------------------------------------------
-- Ejercicio 3. Declarar R como una variable sobre anillos.
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que para todo 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]

-- 1ª demostración
-- ===============

example : -a + (a + b) = b :=
calc -a + (a + b) = (-a + a) + b := by exact (add_assoc (-a) a b).symm
                _ = 0 + b        := congrArg (. + b) (neg_add_cancel a)
                _ = b            := zero_add b

-- 2ª demostración
-- ===============

example
  : -a + (a + b) = b :=
by
  rw [←add_assoc]
  -- ⊢ (-a + a) + b = b
  rw [neg_add_cancel]
  -- ⊢ 0 + b = b
  rw [zero_add]

-- 3ª demostración
-- ===============

theorem neg_add_cancel_left
  : -a + (a + b) = b :=
by
  rw [←add_assoc, neg_add_cancel, zero_add]

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

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

```
-- ------------------------------------------------------------------
-- Ejercicio 6. Cerrar el espacio de nombre MyRing.
-- -------------------------------------------------------------------

end MyRing
```

2.2.6. Ejercicio neg_add_cancel_right

```
-- ------------------------------------------------------------------
-- Ejercicio 1. Importar la teoría de anillos.
-- -------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs

-- ------------------------------------------------------------------
-- Ejercicio 2. Crear el espacio de nombre MyRing.
-- -------------------------------------------------------------------

namespace MyRing

-- ------------------------------------------------------------------
-- Ejercicio 3. Declara R una variable sobre anillos.
-- -------------------------------------------------------------------

variable {R : Type _} [Ring R]

-- ------------------------------------------------------------------
-- Ejercicio 4. Declarar a y b como variables sobre R.
-- -------------------------------------------------------------------

variable (a b : R)

-- ------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--     (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
-- ========================

-- 1ª demostración
-- ===============

theorem neg_add_cancel_right : (a + b) + -b = a :=
calc
  (a + b) + -b = a + (b + -b) := by exact add_assoc a b (-b)
             _ = a + 0        := congrArg (a + .) (add_neg_cancel b)
             _ = a            := add_zero a

-- 2ª demostración
-- ===============

example : (a + b) + -b = a :=
by
  rw [add_assoc]
  -- ⊢ a + (b + -b) = a
  rw [add_neg_cancel]
  -- ⊢ a + 0 = a
  rw [add_zero]

-- 3ª demostración
-- ===============

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

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

variable (c : R)
variable (f : R → R)
#check (add_assoc a b c : a + b + c = a + (b + c))
#check (add_zero a : a + 0 = a)
#check (add_neg_cancel a : a + -a = 0)
#check (congrArg f : a = b → f a = f b)

-- ---------------------------------------------------------------------
-- Ejercicio 4. Cerrar la teoría MyRing
-- ----------------------------------------------------------------------
```

```
end MyRing
```

2.2.7. Ejercicio: Cancelativas de la suma

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la teoría de anillos.
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 2. Crear el espacio de nombre MyRing.
-- ----------------------------------------------------------------------

namespace MyRing

-- ---------------------------------------------------------------------
-- Ejercicio 3. Declara R una variable sobre anillos.
-- ----------------------------------------------------------------------

variable {R : Type _} [Ring R]

-- ---------------------------------------------------------------------
-- Ejercicio 4. Declarar a, b y c como variables sobre R.
-- ----------------------------------------------------------------------

variable {a b c : R}

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que si
--    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
-- ========================

-- 1ª demostración
-- ===============

theorem add_left_cancel
  (h : a + b = a + c)
  : b = c :=
calc
  b = 0 + b        := by rw [zero_add]
  _ = (-a + a) + b := by rw [neg_add_cancel]
  _ = -a + (a + b) := by rw [add_assoc]
  _ = -a + (a + c) := by rw [h]
  _ = (-a + a) + c := by rw [←add_assoc]
  _ = 0 + c        := by rw [neg_add_cancel]
  _ = 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 (-a + .) h
  clear h
  rw [← add_assoc] at h1
  -- h1 : (-a + a) + b = -a + (a + c)
  rw [neg_add_cancel] at h1
  -- h1 : 0 + b = -a + (a + c)
  rw [zero_add] at h1
  -- h1 : b = -a + (a + c)
  rw [← add_assoc] at h1
  -- h1 : b = (-a + a) + c
  rw [neg_add_cancel] at h1
  -- h1 : b = 0 + c
  rw [zero_add] at h1
  -- h1 : b = c
  exact h1

-- 3ª demostración
-- ===============

lemma neg_add_cancel_left (a b : R) : -a + (a + b) = b :=
by simp

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]
  -- ⊢ -a + (a + b) = c
  rw [h]
  -- ⊢ -a + (a + c) = c
  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]

-- ---------------------------------------------------------------------
-- Ejercicio 6. Demostrar que si
--    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
-- ========================

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

theorem add_right_cancel
  (h : a + b = c + b)
  : a = c :=
calc
  a = a + 0          := by rw [add_zero]
  _ = a + (b + -b) := by rw [add_neg_cancel]
  _ = (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_neg_cancel]
  _ = c              := by rw [add_zero]

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

lemma neg_add_cancel_right (a b : R) : (a + b) + -b = a :=
by simp

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

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

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

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

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

-- ---------------------------------------------------------------------
-- Ejercicio 7. Cerrar el espacio de nombre MyRing.
-- ----------------------------------------------------------------------

end MyRing
```

2.2.8. Lema mul_zero con have

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los anillos
--    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

namespace MyRing

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 exact (mul_add a 0 0).symm
                     _ = a * 0       := congrArg (a * .) (add_zero 0)
                     _ = a * 0 + 0   := by exact (add_zero (a * 0)).symm
  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]
       -- ⊢ a * (0 + 0) = a * 0 + 0
       rw [add_zero]
       -- ⊢ a * 0 = a * 0 + 0
       rw [add_zero]
  rw [add_left_cancel h]

-- 4ª 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]

-- 5ª 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

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

variable (b c : R)
variable (f : R → E)
#check (add_zero a : a + 0 = a)
#check (add_left_cancel : a + b = a + c → b = c)
#check (congrArg f : a = b → f a = f b)
#check (mul_add a b c : a * (b + c) = a * b + a * c)

end MyRing
```

2.2.9. Ejercicio zero_mul

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los anillos,
--    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

namespace MyRing

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 exact (add_mul 0 0 a).symm
                     _ = 0 * a       := congrArg (. * a) (add_zero 0)
                     _ = 0 * a + 0   := by exact (add_zero (0 * a)).symm
  exact add_left_cancel h

-- 2ª demostración
-- ===============

example : 0 * a = 0 :=
by
  have h : 0 * a + 0 * a = 0 * a + 0 :=
    by rw [←add_mul]
       -- ⊢ (0 + 0) * a = 0 * a + 0
       rw [add_zero]
       -- ⊢ 0 * a = 0 * a + 0
       rw [add_zero]
  rw [add_left_cancel h]

-- 3ª 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]

-- 4ª 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

-- 5ª demostración
-- ===============

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

-- 6ª demostración
-- ===============

example : 0 * a = 0 :=
by simp

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

variable (b c : R)
variable (f : R → R)
#check (add_left_cancel : a + b = a + c → b = c)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
#check (add_zero a : a + 0 = a)
#check (congrArg f : a = b → f a = f b)

end MyRing
```

2.2.10. Ejercicios sobre anillos

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

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

-- -----------------------------------------------------------------------
-- Ejercicio. 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
-- =========================

-- 1ª demostración (basada en la 1º en LN)
example
  (h : a + b = 0)
  : -a = b :=
calc
  -a = -a + 0       := by exact (add_zero (-a)).symm
   _ = -a + (a + b) := congrArg (-a + .) h.symm
   _ = b            := by exact (neg_add_cancel_left a b)

-- 2ª 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]

-- 3ª 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 (-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
  -- h1 : b = -a
  exact h1.symm

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

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar 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
-- ========================
```

```
-- 1ª demostración
example : (a + b) + -b = a :=
calc
  (a + b) + -b = a + (b + -b) := by exact (add_assoc a b (-b))
             _ = a + 0        := congrArg (a + .) (add_neg_cancel b)
             _ = a            := add_zero a

-- 2ª demostración
example : (a + b) + -b = a :=
calc
  (a + b) + -b = a + (b + -b) := by rw [add_assoc]
             _ = a + 0        := by rw [add_neg_cancel]
             _ = a            := by rw [add_zero]

-- 3ª demostración
example : (a + b) + -b = a :=
by
  rw [add_assoc]
  -- ⊢ a + (b + -b) = a
  rw [add_neg_cancel]
  -- ⊢ a + 0 = a
  rw [add_zero]

-- 4ª demostración
example : (a + b) + -b = a :=
by rw [add_assoc, add_neg_cancel, add_zero]

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

-- 6ª demostración
example : (a + b) + -b = a :=
  add_neg_cancel_right _ _

-- 7ª demostración
example : (a + b) + -b = a :=
by simp

-- ---------------------------------------------------------------------
-- Ejercicio. 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
-- ========================

-- 1ª demostración (basada en la 1ª en LN)
example
  (h : a + b = 0)
  : a = -b :=
calc
  a = (a + b) + -b := by exact (add_neg_cancel_right a b).symm
  _ = 0 + -b       := congrArg (. + -b) h
  _ = -b           := zero_add (-b)

-- 2ª 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]

-- 3ª 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

-- 4ª demostración (basada en la 1ª en LN)
example
  (h : a + b = 0)
  : a = -b :=
by
  have h1 : (a + b) + -b = 0 + -b := congrArg (. + -b) 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

-- 5ª demostración
example
  (h : a + b = 0)
  : a = -b :=
add_eq_zero_iff_eq_neg.mp h

-- ---------------------------------------------------------------------
-- Ejercicio. 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
-- ========================

-- 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
  -- ⊢ 0 + 0 = 0
  rw [add_zero]

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

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

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--     -(-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
-- ========================
```

```
-- 1ª demostración
example : -(-a) = a :=
by
  have h1 : -a + a = 0 := neg_add_cancel 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
  -- ⊢ -a + a = 0
  rw [neg_add_cancel]

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

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

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

variable (c : R)
variable (f : R → R)
#check (add_assoc a b c : (a + b) + c = a + (b + c))
#check (add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b)
#check (add_neg_cancel a : a + -a = 0)
#check (add_neg_cancel_right a b : (a + b) + -b = a)
#check (add_zero a : a + 0 = a)
#check (congrArg f : a = b → f a = f b)
#check (neg_add_cancel a : -a + a = 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 (zero_add a : 0 + a = a)
```

2.2.11. Subtracción en anillos

```
-- ------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de anillos.
--    2. Crear el espacio de nombres my_ring
--    3. Declarar R como una variable sobre anillos.
--    4. Declarar a y b como variables sobre R.
-- ------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs -- 1
namespace MyRing                 -- 2
variable {R : Type _} [Ring R]   -- 3
variable (a b : R)               -- 4

-- ------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    a - b = a + -b
-- ------------------------------------------------------------------

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

-- Por la definición de la resta.

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

-- 1ª demostración
theorem sub_eq_add_neg' : a - b = a + -b :=
-- by exact?
sub_eq_add_neg a b

end MyRing
```

2.2.12. Ejercicio self_sub

```
-- ------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de anillos.
--    2. Crear el espacio de nombres my_ring
--    3. Declarar R como una variable sobre anillos.
--    4. Declarar a como variable sobre R.
```

```
-- ---------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs -- 1
namespace MyRing                 -- 2
variable {R : Type _} [Ring R]   -- 3
variable (a : R)                 -- 4

-- --------------------------------------------------------------------
-- Ejercicio. Demostrar que
--     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
-- ========================

-- 1ª demostración
-- ===============

example : a - a = 0 :=
calc
  a - a = a + -a := by exact (sub_eq_add_neg a a)
      _ = 0      := add_neg_cancel a

-- 2ª demostración
-- ===============

theorem self_sub : a - a = 0 :=
calc
  a - a = a + -a := by rw [sub_eq_add_neg a a]
      _ = 0      := by rw [add_neg_cancel]

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

variable (b : R)
#check (add_neg_cancel a : a + -a = 0)
#check (sub_eq_add_neg a b : a - b = a + -b)
```

```
end MyRing
```

2.2.13. Ejercicio two_mul

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de anillos.
--    2. Crear el espacio de nombres my_ring
--    3. Declarar R como una variable sobre anillos.
--    4. Declarar a como variable sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Ring.Defs -- 1
import Mathlib.Tactic
namespace MyRing                 -- 2
variable {R : Type _} [Ring R]   -- 3
variable (a : R)                 -- 4

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    1 + 1 = 2
-- ----------------------------------------------------------------------

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

-- Por cálculo.

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

theorem one_add_one_eq_two : 1 + 1 = (2 : R) :=
by norm_num

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    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]

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

theorem two_mul : 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]

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

variable (b c : R)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
#check (one_mul a : 1 * a = a)

end MyRing
```

2.2.14. Demostraciones en grupos

2.2.15. Axiomas de grupo (versión aditiva)

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de grupos
-- ----------------------------------------------------------------------
```

```
import Mathlib.Algebra.Group.Defs

-- ----------------------------------------------------------------------
-- Ejercicio 2. Declarar A como un tipo sobre grupos aditivos.
-- -----------------------------------------------------------------------

variable (A : Type _) [AddGroup A]

-- ----------------------------------------------------------------------
-- Ejercicio 3. Comprobar que A verifica los axiomas de los grupos
-- -----------------------------------------------------------------------

variable (a b c : A)

#check (add_assoc a b c : a + b + c = a + (b + c))
#check (zero_add a : 0 + a = a)
#check (neg_add_cancel a : -a + a = 0)
```

2.2.16. Axiomas de grupo multiplicativo

```
-- ----------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de grupos
-- -----------------------------------------------------------------------

import Mathlib.Algebra.Group.Defs

-- ----------------------------------------------------------------------
-- Ejercicio 2. Declarar G como un tipo sobre grupos.
-- -----------------------------------------------------------------------

variable {G : Type _} [Group G]

-- ----------------------------------------------------------------------
-- Ejercicio 3. Comprobar que G verifica los axiomas de los grupos
-- -----------------------------------------------------------------------

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

2.2.17. Ejercicios sobre grupos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de grupo.
--    2. Crear el espacio de nombres Grupo
--    3. Declarar G como una variable sobre anillos.
--    4. Declarar a y b como variable sobre G.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Group.Defs -- 1
variable {G : Type _} [Group G]   -- 2
namespace Grupo                   -- 3
variable (a b : G)                -- 4

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    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
-- ========================

-- 1ª demostración
example : a * a⁻¹ = 1 :=
calc
  a * a⁻¹ = 1 * (a * a⁻¹)              := by exact (one_mul (a * a⁻¹)).symm
        _ = (1 * a) * a⁻¹              := (mul_assoc 1 a a⁻¹).symm
        _ = (((a⁻¹)⁻¹ * a⁻¹)  * a) * a⁻¹ := by {congr ; exact (inv_mul_cancel a⁻¹).symm}
        _ = ((a⁻¹)⁻¹ * (a⁻¹  * a)) * a⁻¹ := congrArg (. * a⁻¹) (mul_assoc a⁻¹⁻¹ a⁻¹ a)
        _ = ((a⁻¹)⁻¹ * 1) * a⁻¹        := by {congr ; exact inv_mul_cancel a}
        _ = (a⁻¹)⁻¹ * (1 * a⁻¹)        := mul_assoc a⁻¹⁻¹ 1 a⁻¹
```

```
         _ = (a⁻¹)⁻¹ * a⁻¹                 := congrArg (a⁻¹⁻¹ * .) (one_mul a⁻¹)
         _ = 1                              := inv_mul_cancel a⁻¹

-- 2ª 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 [inv_mul_cancel]
        _ = ((a⁻¹)⁻¹ * (a⁻¹  * a)) * a⁻¹  := by rw [← mul_assoc]
        _ = ((a⁻¹)⁻¹ * 1) * a⁻¹            := by rw [inv_mul_cancel]
        _ = (a⁻¹)⁻¹ * (1 * a⁻¹)            := by rw [mul_assoc]
        _ = (a⁻¹)⁻¹ * a⁻¹                  := by rw [one_mul]
        _ = 1                               := by rw [inv_mul_cancel]

-- 3ª 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

-- 4ª demostración
theorem mul_right_inv : a * a⁻¹ = 1 :=
by simp

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    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
-- ======================

-- 1ª demostración
example : a * 1 = a :=
calc
  a * 1 = a * (a⁻¹ * a) := by {congr ; exact (inv_mul_cancel a).symm}
      _ = (a * a⁻¹) * a := (mul_assoc a a⁻¹ a).symm
      _ = 1 * a         := congrArg (. * a) (mul_right_inv a)
      _ = a             := one_mul a

-- 2ª demostración
example : a * 1 = a :=
calc
  a * 1 = a * (a⁻¹ * a) := by rw [inv_mul_cancel]
      _ = (a * a⁻¹) * a := by rw [mul_assoc]
      _ = 1 * a         := by rw [mul_right_inv]
      _ = a             := by rw [one_mul]

-- 3ª 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

-- 4ª demostración
theorem mul_one : a * 1 = a :=
by simp

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que si
--    b * a = 1
-- entonces
--    a⁻¹ = b
-- ----------------------------------------------------------------------

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

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

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

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

-- 1º demostración
example
  (h : b * a = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ =  1 * a⁻¹         := by exact (one_mul a⁻¹).symm
    _ =  (b * a) * a⁻¹ := congrArg (. * a⁻¹) h.symm
    _ =  b * (a * a⁻¹) := mul_assoc b a a⁻¹
    _ =  b * 1           := congrArg (b * .) (mul_right_inv a)
    _ =  b               := mul_one b

-- 2º demostración
example
  (h : b * a = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ =  1 * a⁻¹         := by rw [one_mul]
    _ =  (b * a) * a⁻¹ := by rw [h]
    _ =  b * (a * a⁻¹) := by rw [mul_assoc]
    _ =  b * 1           := by rw [mul_right_inv]
    _ =  b               := by rw [mul_one]

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

-- 4º demostración
lemma inv_eq_of_mul_eq_one
  (h : b * a = 1)
  : a⁻¹ = b :=
calc
  a⁻¹ =  (b * a) * a⁻¹ := by simp [h]
```

```
    _ =  b              := by simp

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--    (a * b)⁻¹ = b⁻¹ * a⁻¹
-- ----------------------------------------------------------------------

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

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

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

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

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

-- 3ª demostración
example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
```

```
  have h1 : (b⁻¹ * a⁻¹) * (a * b) = 1 :=
    mul_inv_rev_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 : (b⁻¹ * a⁻¹) * (a * b) = 1 :=
    mul_inv_rev_aux a b
  simp [h1]

-- 5ª demostración
theorem mul_inv_rev : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by
  apply inv_eq_of_mul_eq_one
  -- ⊢ (b⁻¹ * a⁻¹) * (a * b) = 1
  rw [mul_inv_rev_aux]

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

variable (c : G)
variable (f : G → G)
#check (inv_mul_cancel a : a⁻¹ * a = 1)
#check (congrArg f : a = b → f a = f b)
#check (inv_eq_of_mul_eq_one a b : b * a = 1 → a⁻¹ = b)
#check (inv_mul_cancel a : a⁻¹ * a = 1)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
#check (mul_one a : a * 1 = a)
#check (mul_right_inv a : a * a⁻¹ = 1)
#check (one_mul a : 1 * a = a)

-- ---------------------------------------------------------------------
-- Ejercicio 6.  Cerrar el espacio de nombre Grupo.
-- ----------------------------------------------------------------------

end Grupo
```

2.3. Uso de lemas y teoremas

2.3.1. Propiedades reflexiva y transitiva

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la teoría de los números reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

-- ---------------------------------------------------------------------
-- Ejercicio 2. Declarar a, b y c como variables sobre los reales.
-- ----------------------------------------------------------------------

variable (a b c : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Declarar que
-- + h  es una variable de tipo a ≤ b
-- + h’ es una variable de tipo b ≤ c
-- ----------------------------------------------------------------------

variable (h : a ≤ b) (h’ : b ≤ c)

-- ---------------------------------------------------------------------
-- Ejercicio 4. Calcular el tipo de las siguientes expresiones:
--     + le_refl
--     + le_refl a
--     + le_trans
--     + le_trans h
--     + le_trans h h’
-- ----------------------------------------------------------------------

#check (le_refl : ∀ a : ℝ, a ≤ a)
#check (le_refl a : a ≤ a)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
#check (le_trans h : b ≤ c → a ≤ c)
#check (le_trans h h’ : a ≤ c)
```

2.3.2. Las tácticas apply y exact

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de los números reales
--    2. Declarar x, y y z como variables sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable (x y z : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    x ≤ y
--    y ≤ z
-- entonces
--    x ≤ z
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : y ≤ z)
  : x ≤ z :=
by
  apply le_trans
  . -- ⊢ x ≤ ?b
    apply h1
  . -- ⊢ y ≤ z
    apply h2

-- 2ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : y ≤ z)
  : x ≤ z :=
by
  apply le_trans h1
  -- ⊢ y ≤ z
  apply h2
```

```
-- 3ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : y ≤ z)
  : x ≤ z :=
by exact le_trans h1 h2

-- 4ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : y ≤ z)
  : x ≤ z :=
le_trans h1 h2

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    x ≤ x
-- ----------------------------------------------------------------------

-- 1ª demostración
example : x ≤ x :=
by apply le_refl

-- 2ª demostración
example : x ≤ x :=
by exact le_refl x

-- 3ª demostración
example (x : ℝ) : x ≤ x :=
le_refl x

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

#check (le_refl x : x ≤ x)
#check (le_trans : x ≤ y → y ≤ z → x ≤ z)
```

2.3.3. Propiedades del orden

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de los números reales
--    2. Declarar a, b y c como variables sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones:
--    + le_refl
--    + le_trans
--    + lt_of_le_of_lt
--    + lt_of_lt_of_le
--    + lt_trans
-- ----------------------------------------------------------------------

#check (le_refl a : a ≤ a)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
#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)
```

2.3.4. Ejercicio sobre orden

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

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

```
variable (a b c d e : ℝ)

-- 1ª demostración
-- ===============

example
  (h₀ : a ≤ b)
  (h₁ : b < c)
  (h₂ : c ≤ d)
  (h₃ : d < e) :
  a < e :=
by
  apply lt_of_le_of_lt h₀
  -- ⊢ b < e
  apply lt_trans h₁
  -- ⊢ c < e
  apply lt_of_le_of_lt h₂
  -- ⊢ d < e
  exact h₃

-- 2ª demostración
-- ===============

example
  (h₀ : a ≤ b)
  (h₁ : b < c)
  (h₂ : c ≤ d)
  (h₃ : d < e) :
  a < e :=
calc
  a ≤ b := h₀
  _ < c := h₁
  _ ≤ d := h₂
  _ < e := h₃

-- 3ª demostración
-- ===============

example
  (h₀ : a ≤ b)
  (h₁ : b < c)
  (h₂ : c ≤ d)
  (h₃ : d < e) :
  a < e :=
by linarith
```

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

variable (x y z : ℝ)
#check (lt_of_le_of_lt : x ≤ y → y < z → x < z)
#check (lt_trans : x < y → y < z → x < z)
```

2.3.5. Demostraciones por aritmética lineal

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Sean a, b, c, d y e números reales. Demostrar que si
--    a ≤ b
--    b < c
--    c ≤ d
--    d < e
-- entonces
--    a < e
-- ----------------------------------------------------------------------

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

variable (a b c d e : ℝ)

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

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    2 * a ≤ 3 * b
--    1 ≤ a
--    d = 2
-- entonces
--    d + a ≤ 5 * b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

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

-- Por la siguiente cadena de desigualdades
--    d + a = 2 + a      [por la hipótesis 3 (d = 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
-- ========================

-- 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
```

2.3.6. Aritmética lineal con argumentos

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

```
import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b d : ℝ)

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

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

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

-- 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)
```

2.3.7. Lemas de desigualdades en R

```
-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de los siguientes lemas
--    exp_le_exp
--    exp_lt_exp
--    log_le_log
--    log_lt_log
--    add_le_add
--    add_lt_add_of_le_of_lt
--    add_nonneg
--    add_pos
--    add_pos_of_pos_of_nonneg
--    exp_pos
-- ----------------------------------------------------------------------

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b c d : ℝ)

#check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
#check (exp_lt_exp : exp a < exp b ↔ a < b)
#check (log_le_log : 0 < a → a ≤ b → log a ≤ log b)
#check (log_lt_log : 0 < a → a < b → log a < log b)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

```
#check (add_le_add_left : a ≤ b → ∀ c, c + a ≤ c + b)
#check (add_le_add_right : a ≤ b → ∀ c, a + c ≤ b + c)
#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_left : a < b → ∀ c, c + a < c + b)
#check (add_lt_add_right : a < b → ∀ c, a + c < b + c)
#check (add_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a + b)
#check (add_pos : 0 < a → 0 < b → 0 < a + b)
#check (add_pos_of_pos_of_nonneg : 0 < a → 0 ≤ b → 0 < a + b)
#check (exp_pos : ∀ a, 0 < exp a)
```

2.3.8. Desigualdad de exponenciales (reescritura con el bicondicional)

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de exponeciales y logaritmos.
--    2. Abrir la teoría de los reales
--    3. Declarar a y b como variables sobre los reales.
-- ----------------------------------------------------------------------

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo del lema exp_le_exp
-- ----------------------------------------------------------------------

#check @exp_le_exp a b

-- Comentario: Al colocar el cursor sobre check se obtiene
--    exp_le_exp : rexp a ≤ rexp b ↔ a ≤ b

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    a ≤ b
-- entonces
--    exp a ≤ exp b
-- ----------------------------------------------------------------------
```

```
-- 1ª demostración
example
  (h : a ≤ b)
  : exp a ≤ exp b :=
by
  rw [exp_le_exp]
  -- ⊢ a ≤ b
  exact h

-- 2ª demostración
example
  (h : a ≤ b)
  : exp a ≤ exp b :=
by rwa [exp_le_exp]

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

-- Nota: Con mpr se indica en modus ponens inverso. Por ejemplo, si
-- h: A ↔ B, entonces h.mpr es B → A y h.mp es A → B

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

#check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
```

2.3.9. Eliminación de bicondicional

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de exponeciales y logaritmos.
--    2. Abrir la teoría de los reales
--    3. Declarar a, b, c, d y e como variables sobre los reales.
-- ----------------------------------------------------------------------

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

```
-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de los siguientes lemas
--    add_lt_add_of_le_of_lt
--    exp_lt_exp
--    le_refl
-- ----------------------------------------------------------------------

#check (add_lt_add_of_lt_of_le : a < b → c ≤ d → a + c < b + d)
#check (exp_lt_exp : exp a < exp b ↔ a < b)
#check (le_refl a : a ≤ a)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    a ≤ b
--    c < d
-- entonces
--    a + exp c + e < b + exp d + e
-- ----------------------------------------------------------------------

-- 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 + e < b + e^d + e

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

-- Tenemos que demostrar que
--    (a + e^c) + e < (b + e^d) + e
-- que, por la monotonía de la suma, se reduce a las siguientes dos
-- desigualdades:
--    a + e^c < b + e^d                                          (1)
--    e ≤ e                                                      (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
-- ========================

-- 1ª demostración
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + e < b + exp d + e :=
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 + e < b + exp d + e
  exact add_lt_add_right h4 e

-- 2ª demostración
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + e < b + exp d + e :=
by
  apply add_lt_add_of_lt_of_le
  . -- ⊢ a + exp c < b + exp d
    apply add_lt_add_of_le_of_lt
    . -- ⊢ a ≤ b
      exact h1
    . -- ⊢ exp c < exp d
      apply exp_lt_exp.mpr
      -- ⊢ c < d
      exact h2
  . -- ⊢ e ≤ e
    apply le_refl

-- 3ª demostración
```

```
example
  (h1 : a ≤ b)
  (h2 : c < d)
  : a + exp c + e < b + exp d + e :=
by
  apply add_lt_add_of_lt_of_le
  . -- ⊢ a + exp c < b + exp d
    apply add_lt_add_of_le_of_lt h1
    -- ⊢ exp c < exp d
    exact exp_lt_exp.mpr h2
  . -- ⊢ e ≤ e
    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 : a < b → ∀ c, a + c < b + c)
#check (exp_lt_exp : exp a < exp b ↔ a < b)
#check (le_refl a : a ≤ a)
```

2.3.10. Ejercicio sobre desigualdades

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de exponeciales y logaritmos.
--    2. Abrir la teoría de los reales
--    3. Declarar a, b, c, d y e como variables sobre los reales.
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    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
-- ========================

-- 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
  -- ⊢ exp (a + d) ≤ exp (a + f)
  apply exp_le_exp.mpr
  -- ⊢ a + d ≤ a + f
  apply add_le_add_left
  -- ⊢ d ≤ f
  exact h

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    0 < 1
-- ----------------------------------------------------------------------

example : (0 : ℝ) < 1 :=
by norm_num

-- Nota: La táctica norm_num normaliza expresiones numéricas.

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que si
--    a ≤ b
-- entonces
--    log (1 + exp a) ≤ log (1 + exp b)
-- ----------------------------------------------------------------------

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

-- Por la monotonía del logaritmo, basta demostrar que
--    0 < 1 + exp(a)                  (1)
--    1 + exp(a) ≤ 1 + exp(b)         (2)
--
-- La (1), por la suma de positivos, se reduce a
--    0 < 1                           (1.1)
--    0 < exp(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
--    exp(a) ≤ exp(b)
-- que, por la monotonía de la exponencial, se reduce a
--    a ≤ b
-- que es la hipótesis.

-- Demostraciones con Lean4
```

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

-- 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
  . -- ⊢ 0 < 1 + exp a
    apply add_pos
    . -- ⊢ 0 < 1
      exact zero_lt_one
    . -- ⊢ 0 < exp a
      exact exp_pos a
  . -- ⊢ 1 + exp a ≤ 1 + exp b
    apply add_le_add_left
    -- ⊢ exp a ≤ exp b
    exact exp_le_exp.mpr h

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

#check (add_le_add_left : a ≤ b → ∀ c, c + a ≤ c + b)
#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 : ℝ) < 1)
```

2.3.11. Búsqueda con apply?

```
-- ---------------------------------------------------------------------
-- Ejercicio . Demostrar que, para todo númeo real a,
--    0 ≤ a^2
-- ----------------------------------------------------------------------

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

variable (a : ℝ)

-- 1ª demostración
example : 0 ≤ a^2 :=
by
  -- apply?
  exact sq_nonneg a

-- 2ª demostración
example : 0 ≤ a^2 :=
sq_nonneg a

-- Notas:
-- + Nota 1: Al colocar el cursor sobre apply? (después de descomentar
--   la línea) escribe el mensaje
--      Try this: exact sq_nonneg a
-- + Nota 2: Para usar apply? hay que importar Mathlib.Tactic.

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

#check (sq_nonneg a : 0 ≤ a^2)
```

2.3.12. Ejercicio con apply?

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

```
--    c - exp b ≤ c - exp 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
-- ============

variable (d : ℝ)
variable (h : a ≤ b)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
#check (le_refl : ∀ (a : ℝ), a ≤ a)
#check (neg_le_neg : a ≤ b → -b ≤ -a)
#check (sub_le_sub_left h c : c - b ≤ c - a)
```

2.3.13. Desigualdades con calc

```
-- ----------------------------------------------------------------------
-- Ejercicio. Sean a y b números reales. Demostrar que
--    2*a*b ≤ a^2 + b^2
-- -----------------------------------------------------------------------

-- 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
import Mathlib.Tactic

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 d : 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)
```

2.3.14. Ejercicio desigualdades absolutas

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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 cuadrado,
--   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
import Mathlib.Tactic

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
  -- ⊢ a * b ≤ (a ^ 2 + b ^ 2) / 2 ∧ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
  constructor
  . -- ⊢ a * b ≤ (a ^ 2 + b ^ 2) / 2
    have h1 : a * b * 2 ≤ a ^ 2 + b ^ 2 := aux1 a b
    show a * b ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff₀ h).mpr h1
  . -- ⊢ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
    have h2 : -(a * b) * 2 ≤ a ^ 2 + b ^ 2 := aux2 a b
    show -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff₀ 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
  -- ⊢ a * b ≤ (a ^ 2 + b ^ 2) / 2 ∧ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
  constructor
  . -- ⊢ a * b ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff₀ h).mpr (aux1 a b)
  . -- ⊢ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
    exact (le_div_iff₀ 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
  -- ⊢ a * b ≤ (a ^ 2 + b ^ 2) / 2 ∧ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
  constructor
  . -- a * b ≤ (a ^ 2 + b ^ 2) / 2
    rw [le_div_iff₀ h]
    -- ⊢ a * b * 2 ≤ a ^ 2 + b ^ 2
    apply aux1
  . -- ⊢ -(a * b) ≤ (a ^ 2 + b ^ 2) / 2
    rw [le_div_iff₀ h]
    -- ⊢ -(a * b) * 2 ≤ a ^ 2 + b ^ 2
    apply aux2

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

variable (c : ℝ)
#check (abs_le' : |a| ≤ b ↔ a ≤ b ∧ -a ≤ b)
#check (le_div_iff₀ : 0 < c → (a ≤ b / c ↔ a * c ≤ b))
#check (pow_two_nonneg a : 0 ≤ a ^ 2)
```

2.4. Más sobre orden y divisibilidad

2.4.1. Mínimos y máximos

2.4.2. Caracterización del mínimo

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sean a, b y c números reales. Calcular los tipos de
--    min_le_left a b
--    min_le_right a b
--    @le_min ℝ _ a b c
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

#check (min_le_left a b : min a b ≤ a)
#check (min_le_right a b : min a b ≤ b)
#check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
```

2.4.3. Caracterización del máximo

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sean a, b y c números reales. Calcular los tipos de
--    le_max_left a b
--    le_max_right a b
--    @max_le ℝ _ a b c
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

#check (le_max_left a b : a ≤ max a b)
#check (le_max_right a b : b ≤ max a b)
#check (max_le : a ≤ c → b ≤ c → max a b ≤ c)
```

2.4.4. Conmutatividad del mínimo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ min a b ≤ b
    apply min_le_right
  . -- ⊢ min a b ≤ a
    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
  . -- ⊢ min a b ≤ min b a
    exact aux a b
  . -- ⊢ min b a ≤ min a b
    exact aux b a

-- 2ª demostración
-- ===============

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

-- 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_le_left a b : min a b ≤ a)
#check (min_le_right a b : min a b ≤ b)
```

2.4.5. Conmutatividad del máximo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ a ≤ max b a
    apply le_max_right
  . -- ⊢ b ≤ max b a
    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
  . -- ⊢ max a b ≤ max b a
    exact aux a b
  . -- ⊢ max b a ≤ max 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)
```

2.4.6. Ejercicio: Asociatividad del mínimo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ min (min a b) c ≤ a
    exact aux1a
  . -- ⊢ min (min a b) c ≤ min b c
    apply le_min
    . -- ⊢ min (min a b) c ≤ b
      exact aux1b
    . -- ⊢ 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
  . -- ⊢ min a (min b c) ≤ min a b
    apply le_min
    . -- ⊢ min a (min b c) ≤ a
      exact aux2a
    . -- ⊢ min a (min b c) ≤ b
      exact aux2b
  . -- ⊢ 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
  . -- ⊢ min (min a b) c ≤ min a (min b c)
    exact aux1
  . -- ⊢ 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
  . -- ⊢ min (min a b) c ≤ min a (min b c)
```

```
    exact aux1
  . -- ⊢ min a (min b c) ≤ min (min a b) c
    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)
```

2.4.7. Ejercicio: Mínimo de suma

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ min a b + c ≤ a + c
    apply add_le_add_right
    -- ⊢ min a b ≤ a
    exact min_le_left a b
  . -- ⊢ min a b + c ≤ b + c
    apply add_le_add_right
    -- ⊢ min a b ≤ b
    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 simp_all only [add_neg_cancel_right]
  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
  . -- ⊢ min a b + c ≤ min (a + c) (b + c)
    exact aux1
  . -- ⊢ 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
  . -- ⊢ min a b + c ≤ min (a + c) (b + c)
    exact aux1
  . -- ⊢ min (a + c) (b + c) ≤ min a b + c
    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,  b + a ≤ c + a)
#check (add_neg_cancel_right a b : (a + b) + -b = 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)
```

2.4.8. Lema abs_add

```
-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--     abs_add
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

#check abs_add

-- Comentario: Colocando el cursor sobre check se obtiene
--    abs_add.{u_1} {G : Type u_1} [AddCommGroup G] [LinearOrder G] [IsOrderedAddMonoid
--      (a b : G) : |a + b| ≤ |a| + |b|
```

2.4.9. Ejercicio: abs_sub

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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_right (|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_right]

-- 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
  -- h1 : |a| ≤ |a - b| + |b|
  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
-- ============

variable (f : ℝ → ℝ)
#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_right a b : a + b - b = a)
#check (congrArg f : a = b → f a = f b)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_le_sub_right : a ≤ b → ∀ c, a - c ≤ b - c)
```

2.4.10. Divisibilidad

2.4.11. Propiedades de divisibilidad

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar la siguientes acciones:
--    1. Importar la teoría de mcd sobre los naturales.
--    2. Declarar x, y y z como variables sobre los naturales.
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    x | y
--    y | z
-- entonces
--    x | z
-- ----------------------------------------------------------------------

example
```

```
  (h₀ : x ∣ y)
  (h₁ : y ∣ z)
  : x ∣ z :=
dvd_trans h₀ h₁

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    x ∣ y * x * z
-- ----------------------------------------------------------------------

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

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


-- 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
  -- ⊢ x ∣ y * x
  apply dvd_mul_left
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 4. Demostrar que si x ∈ ℕ, entonces
--    x ∣ x^2
-- -----------------------------------------------------------------------

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

-- Se tiene que
--    x ∣ xx
-- y, por la definición del cuadrado,
--    x ∣ x²

-- 1ª demostración
-- ===============

example : x ∣ x^2 :=
by
  have : x ∣ x * x := dvd_mul_left x x
  show x ∣ x^2
  rwa [pow_two]

-- 2ª demostración
-- ===============

example : x ∣ x^2 :=
by
  rw [pow_two]
  -- ⊢ x ∣ x * x
  apply dvd_mul_right

-- 3ª demostración
-- ===============

example : x ∣ x^2 :=
by apply dvd_mul_left

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

variable (a b : ℕ)
#check (dvd_mul_left a b : a ∣ b * a)
#check (dvd_mul_of_dvd_left : x ∣ y → ∀ (c : ℕ), x ∣ y * c)
#check (dvd_mul_right a b : a ∣ a * b)
#check (dvd_trans : x ∣ y → y ∣ z → x ∣ z)
```

```
#check (pow_two a : a ^ 2 = a * a)
```

2.4.12. Ejercicio de divisibilidad

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ x ∣ y * (x * z) + x ^ 2
    apply dvd_add
    . -- ⊢ x ∣ y * (x * z)
      apply dvd_mul_of_dvd_right
      -- ⊢ x ∣ x * z
      apply dvd_mul_right
    . -- ⊢ x ∣ x ^ 2
      rw [pow_two]
      -- ⊢ x ∣ x * x
      apply dvd_mul_right
  . -- ⊢ x ∣ w ^ 2
    rw [pow_two]
    -- ⊢ x ∣ w * w
    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
  . -- ⊢ x ∣ y * (x * z)
    apply dvd_mul_of_dvd_right
    -- ⊢ x ∣ x * z
    apply dvd_mul_right
  . -- ⊢ x ∣ x ^ 2
```

```
    rw [pow_two]
    -- ⊢ x ∣ x * x
    apply dvd_mul_right
  . -- ⊢ x ∣ w ^ 2
    rw [pow_two]
    -- ⊢ x ∣ w * w
    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_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 (pow_two x : x ^ 2 = x * x)
```

2.4.13. Propiedades de gcd y lcm

```
-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de los siguientes lemas
--    gcd_zero_right
--    gcd_zero_left
--    lcm_zero_right
--    lcm_zero_left
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

open Nat

variable (n : ℕ)

#check (gcd_zero_right n : gcd n 0 = n)
#check (gcd_zero_left n : gcd 0 n = n)
#check (lcm_zero_right n : lcm n 0 = 0)
#check (lcm_zero_left n : lcm 0 n = 0)
```

2.4.14. Conmutatividad del gcd

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    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
  . -- ⊢ gcd m n ∣ gcd n m
    exact aux m n
  . -- ⊢ gcd n m ∣ gcd 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)
```

2.5. Demostraciones sobre estructuras algebraicas

2.5.1. Órdenes

2.5.1.1. Órdenes parciales

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de órdenes
--    2. Declarar α como un tipo sobre los órdenes parciales
--    3. x, y y z como variables sobre α.
-- ----------------------------------------------------------------------

import Mathlib.Order.Basic                  -- 1
variable {α : Type _} [PartialOrder α]      -- 2
variable (x y z : α)                        -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular los tipos de las siguientes expresiones
--    x ≤ y
--    le_refl x
--    @le_trans α _ x y z
-- ----------------------------------------------------------------------

-- #check x ≤ y
-- #check le_refl x
-- #check @le_trans α _ x y z
-- #check @le_antisymm α _ x y

-- Comentario: Al colocar el cursor sobre check se obtiene
#check (x ≤ y : Prop)
#check (le_refl x : x ≤ x)
#check (le_trans : x ≤ y → y ≤ z → x ≤ z)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)

-- Nota: Las letras griegas se escriben con \a, \b, ...
```

2.5.1.2. Orden estricto

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de órdenes
--    2. Declarar α como un tipo sobre los órdenes parciales
--    3. x, y y z como variables sobre α.
-- ----------------------------------------------------------------------

import Mathlib.Order.Basic                -- 1
variable {α : Type _} [PartialOrder α]    -- 2
variable (x y z : α)                      -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones
--    x < y
--    lt_irrefl x
--    lt_trans
--    lt_of_le_of_lt
--    lt_of_lt_of_le
--    lt_iff_le_and_ne
-- ----------------------------------------------------------------------

#check (x < y : Prop)
#check (lt_irrefl x : ¬x < x)
#check (lt_trans : x < y → y < z → x < z)
#check (lt_of_le_of_lt : x ≤ y → y < z → x < z)
#check (lt_of_lt_of_le : x < y → y ≤ z → x < z)
#check (lt_iff_le_and_ne : x < y ↔ x ≤ y ∧ x ≠ y)
```

2.5.2. Retículos

2.5.2.1. Retículos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de retículos
--    2. Declarar α como un tipo sobre los retículos.
--    3. x, y y z como variables sobre α.
-- ----------------------------------------------------------------------

import Mathlib.Order.Lattice       -- 1
variable {α : Type _} [Lattice α]  -- 2
variable (x y z : α)               -- 3
```

```
-- ----------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones
--    x ⊓ y
--    @inf_le_left α _ x y
--    @inf_le_right α _ x y
--    @le_inf α _ z x y
--
--    x ⊔ y
--    @le_sup_left α _ x y
--    @le_sup_right α _ x y
--    @sup_le α _ x y z
-- -----------------------------------------------------------------------

#check (x ⊓ y : α)
#check (inf_le_left : x ⊓ y ≤ x)
#check (inf_le_right : x ⊓ y ≤ y)
#check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
#check (x ⊔ y : α)
#check (le_sup_left : x ≤ x ⊔ y)
#check (le_sup_right : y ≤ x ⊔ y)
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)

-- Comentarios:
-- 1. Para ver cómo se escribe un símbolo, se coloca el cursor sobre el
--    símbolo y se presiona C-c C-k
-- 2. El ínfimo ⊓ se escribe con \glb de ”greatest lower bound”
-- 3. El supremo ⊔ se escribe con \lub de ”least upper bound”
-- 4. En mathlib se usa inf o sup para los nombres sobre ínfimo o supremo.
-- 5. Al colocar el cursor sobre check se obtiene
--       x ⊓ y : α
--       inf_le_left : x ⊓ y ≤ x
--       inf_le_right : x ⊓ y ≤ y
--       le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y
--       x ⊔ y : α
--       le_sup_left : x ≤ x ⊔ y
--       le_sup_right: y ≤ x ⊔ y
--       sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z
```

2.5.2.2. Conmutatividad del ínfimo

```
-- ----------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ x ⊓ y ≤ y
    apply inf_le_right
  . -- ⊢ x ⊓ y ≤ x
```

```
    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
  . -- ⊢ x ⊓ y ≤ y ⊓ x
    apply aux
  . -- ⊢ y ⊓ x ≤ x ⊓ y
    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 x y

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

#check (inf_comm x y : 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)
```

2.5.2.3. Conmutatividad del supremo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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 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
  . -- ⊢ x ≤ y ⊔ x
    apply le_sup_right
  . -- ⊢ y ≤ y ⊔ x
    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
  . -- ⊢ x ⊔ y ≤ y ⊔ x
    apply aux
  . -- ⊢ y ⊔ x ≤ x ⊔ y
    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 x y

-- 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 : x ⊔ y = y ⊔ x)
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

2.5.2.4. Asociatividad del ínfimo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  · -- ⊢ (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)
    apply le_inf
    · -- ⊢ (x ⊓ y) ⊓ z ≤ x
      apply le_trans
      . -- ⊢ (x ⊓ y) ⊓ z ≤ x ⊓ y
        apply inf_le_left
      . -- ⊢ x ⊓ y ≤ x
        apply inf_le_left
    . -- ⊢ (x ⊓ y) ⊓ z ≤ y ⊓ z
      apply le_inf
      · -- ⊢ (x ⊓ y) ⊓ z ≤ y
        apply le_trans
        . -- ⊢ (x ⊓ y) ⊓ z ≤ x ⊓ y
          apply inf_le_left
        . -- ⊢ x ⊓ y ≤ y
          apply inf_le_right
      . -- ⊢ (x ⊓ y) ⊓ z ≤ z
        apply inf_le_right
```

```
  . -- ⊢ x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z
    apply le_inf
    · -- ⊢ x ⊓ (y ⊓ z) ≤ x ⊓ y
      apply le_inf
      · -- ⊢ x ⊓ (y ⊓ z) ≤ x
        apply inf_le_left
      . -- ⊢ x ⊓ (y ⊓ z) ≤ y
        apply le_trans
        . -- ⊢ x ⊓ (y ⊓ z) ≤ y ⊓ z
          apply inf_le_right
        . -- ⊢ y ⊓ z ≤ y
          apply inf_le_left
    . -- ⊢ x ⊓ (y ⊓ z) ≤ z
      apply le_trans
      . -- ⊢ x ⊓ (y ⊓ z) ≤ y ⊓ z
        apply inf_le_right
      . -- ⊢ y ⊓ z ≤ z
        apply inf_le_right

-- 3ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  apply le_antisymm
  . -- ⊢ (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)
    apply le_inf
    . -- ⊢ (x ⊓ y) ⊓ z ≤ x
      apply inf_le_of_left_le inf_le_left
    . -- ⊢ x ⊓ y ⊓ z ≤ y ⊓ z
      apply le_inf (inf_le_of_left_le inf_le_right) inf_le_right
  . -- ⊢ x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z
    apply le_inf
    . -- ⊢ x ⊓ (y ⊓ z) ≤ x ⊓ y
      apply le_inf inf_le_left (inf_le_of_right_le inf_le_left)
    . -- ⊢ x ⊓ (y ⊓ z) ≤ z
      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 x y z

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

#check (inf_assoc x y z : (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)
```

2.5.2.5. Asociatividad del supremo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  · -- ⊢ (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 ≤ 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

-- 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
  . -- ⊢ (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

-- 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 x y z

-- 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 = x ⊔ (y ⊔ z))
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

2.5.2.6. Leyes de absorción

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de retículos.
--    2. Declarar α como un tipo sobre retículos
--    3. Declarar x e y como variabkes sobre α
-- ----------------------------------------------------------------------

import Mathlib.Order.Lattice       -- 1
variable {α : Type _} [Lattice α]  -- 2
variable (x y : α)                 -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar 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
-- ========================

-- 1ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  have h1 : x ⊓ (x ⊔ y) ≤ x := inf_le_left
  have h2 : x ≤ x ⊓ (x ⊔ y) := by
    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
-- ============

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar 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
-- ========================

-- 1ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  have h1 : x ⊔ (x ⊓ y) ≤ x := by
    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 (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)
#check (sup_inf_self : x ⊔ (x ⊓ y) = x)
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

2.5.2.7. Retículos distributivos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
--    1. Importar la teoría de retículos
--    2. Declarar α como un tipo sobre los retículos.
--    3. x, y y z como variables sobre α.
-- ----------------------------------------------------------------------

import Mathlib.Order.Lattice                -- 1
variable {α : Type _} [DistribLattice α]    -- 2
variable (x y z : α)                        -- 3
```

```
-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones
--    @inf_sup_left α _ x y z
--    @inf_sup_right α _ x y z
--    @sup_inf_left α _ x y z
--    @sup_inf_right α _ x y z
-- ----------------------------------------------------------------------

--- #check @inf_sup_left α _ x y z
--- #check @inf_sup_right α _ x y z
--- #check @sup_inf_left α _ x y z
--- #check @sup_inf_right α _ x y z

-- Comentario: Al situar el cursor sobre check se obtiene
#check (inf_sup_left x y z : x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
#check (inf_sup_right x y z : (x ⊔ y) ⊓ z = (x ⊓ z) ⊔ (y ⊓ z))
#check (sup_inf_left x y z : x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
#check (sup_inf_right x y z : (x ⊓ y) ⊔ z = (x ⊔ z) ⊓ (y ⊔ z))
```

2.5.2.8. Propiedades distributivas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de retículos.
--    2. Declarar α como un tipo sobre retículos
--    3. Declarar a, b y c como variabkes sobre α
-- ----------------------------------------------------------------------

import Mathlib.Order.Lattice       -- 1
variable {α : Type _} [Lattice α]  -- 2
variable (a b c : α)               -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    ∀ 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 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
-- ========================

-- 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]

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    ∀ 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
-- ========================

-- 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 (inf_comm a b : a ⊓ b = b ⊓ a)
#check (sup_comm a b : a ⊔ b = b ⊔ a)
```

2.5.3. Anillos ordenados

2.5.3.1. Anillos ordenados

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de los anillos ordenados.
--    2. Declarar R como un tipo sobre los anillos ordenados.
--    3. Declarar a y b como variables sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Order.Ring.Defs                              -- 1
variable {R : Type _} [Ring R] [PartialOrder R] [IsStrictOrderedRing R] -- 2
variable (a b c : R)                                                -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones
--    @add_le_add_left R _ a b
--    @mul_pos R _ a b
--    zero_ne_one
--    @mul_nonneg R _ a b
-- ----------------------------------------------------------------------
```

```
#check (add_le_add_left : a ≤ b → ∀ c, c + a ≤ c + b)
#check (mul_pos : 0 < a → 0 < b → 0 < a * b)
#check (zero_ne_one : 0 ≠ 1)
#check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
```

2.5.3.2. Ejercicio sobre anillos ordenados

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
--    1. Importar la teoría de los anillos ordenados.
--    2. Declarar R como un tipo sobre los anillos ordenados.
--    3. Declarar a, b y c como variables sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Algebra.Order.Ring.Defs                              -- 1
variable {R : Type _} [Ring R] [PartialOrder R] [IsStrictOrderedRing R] -- 2
variable (a b c : R)                                                -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar 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
-- ========================

-- 1ª demostración
example : a ≤ b → 0 ≤ b - a :=
by
  intro h
```

```
  -- h : a ≤ b
  -- ⊢ 0 ≤ b - a
  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

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    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
-- ========================

-- 1ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
by
  intro h
  -- h : 0 ≤ b - a
  -- ⊢ a ≤ b
```

```
  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

-- ------------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    a ≤ b
--    0 ≤ c
-- entonces
--    a * c ≤ b * c
-- -------------------------------------------------------------------------

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

-- Se usarán los siguientes lemas:
--    mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
--    mul_nonneg                 : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
--    sub_mul a b c              : (a - b) * c = a * c - b * c)
--    sub_nonneg                 : 0 ≤ a - b ↔ b ≤ a)
--
-- 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
-- ========================

-- 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
  rw [sub_mul] at h4
  -- h4 : 0 ≤ b * c - a * c
  exact sub_nonneg.mp h4

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  apply sub_nonneg.mp
  -- ⊢ 0 ≤ b * c - a * c
```

```
  rw [← sub_mul]
  -- ⊢ 0 ≤ (b - a) * c
  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
  -- ⊢ 0 ≤ b * c - a * c
  rw [← sub_mul]
  -- ⊢ 0 ≤ (b - a) * c
  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 (add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a)
#check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * 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_add_cancel a b : a - b + b = a)
#check (sub_le_sub_right : a ≤ b → ∀ c, a - c ≤ b - c)
#check (sub_mul a b c : (a - b) * c = a * c - b * c)
#check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
#check (sub_self a : a - a = 0)
#check (zero_add a :  0 + a = a)
```

2.5.4. Espacios métricos

2.5.4.1. Espacios métricos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Ejecuta las siguientes acciones
-- 1. Importar la teoría de espacios métricos.
-- 2. Declarar X como un tipo sobre espacios métricos.
-- 3. Declarar x, y y z como variables sobre X.
-- ----------------------------------------------------------------------

import Mathlib.Topology.MetricSpace.Basic
variable {X : Type _} [MetricSpace X]
variable (x y z : X)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de las siguientes expresiones
--    dist_self x
--    dist_comm x y
--    dist_triangle x y z
-- ----------------------------------------------------------------------

#check (dist_self x : dist x x = 0)
#check (dist_comm x y : dist x y = dist y x)
#check (dist_triangle x y z : dist x z ≤ dist x y + dist y z)
```

2.5.4.2. Ejercicio en espacios métricos

```
-- ---------------------------------------------------------------------
-- 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]
    -- ⊢ dist x x ≤ dist x y + dist y 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)
```

Capítulo 3

Lógica

Este capítulo presenta el razonamiento formal en Lean4 aplicado a conectivas lógicas y cuantificadores, exponiendo las tácticas para su introducción en las conclusiones y su eliminación de las hipótesis. Como aplicación práctica de estos conceptos, se demostrarán diversas propiedades matemáticas relacionadas con límites de sucesiones.

3.1. Implicación y cuantificación universal

3.1.1. Lema con implicaciones y cuantificador universal

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de los números reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

-- ---------------------------------------------------------------------
-- Ejercicio 2. Enunciar el lema ej: ”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
```

```
variable {x y ε : ℝ}

-- 1ª demostración
-- ===============

example :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros 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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  by_cases h : (|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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  by_cases h : (|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)
```

3.1.2. Lema con implicaciones y cuantificador universal implícitos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de los números reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

-- ---------------------------------------------------------------------
-- Ejercicio 2. Enunciar, usando variables implícitas, el lema ej: ”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

variable {x y ε : ℝ}

-- 1ª demostración
-- ===============

example :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
```

```
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  by_cases h : (|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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  by_cases h : (|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)
```

3.1.3. La táctica intros

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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

variable {x y ε : ℝ}

-- 1ª demostración
-- ===============

example :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  by_cases h : (|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 :
  0 < ε → ε ≤ 1 → |x| < ε → |y| < ε → |x * y| < ε :=
by
  intros he1 he2 hx hy
  -- he1 : 0 < ε
  -- he2 : ε ≤ 1
  -- hx : |x| < ε
  -- hy : |y| < ε
  -- ⊢ |x * y| < ε
  by_cases h : (|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)
```

3.1.4. Definiciones de cotas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Importar la librería de los números reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

-- ---------------------------------------------------------------------
-- Ejercicio 2. Definir la función
--    FnUb (ℝ → ℝ) → ℝ → Prop
-- tal que (FnUb f a) afirma que a es una cota superior de f.
-- ----------------------------------------------------------------------

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

-- ---------------------------------------------------------------------
-- Ejercicio 3. Definir la función
--    FnLb (ℝ → ℝ) → ℝ → Prop
-- tal que (FnLb f a) afirma que a es una cota inferior de f.
-- ----------------------------------------------------------------------

def FnLb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x
```

3.1.5. Suma de cotas superiores

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de los números reales.
-- 2. Definir cota superior de una función.
-- 3. Definir cota inferior de una función.
-- 4. Declarar f y g como variables de funciones de ℝ en ℝ.
-- 5. Declarar a y b como variables sobre ℝ.
-- ----------------------------------------------------------------------
```

```
import Mathlib.Data.Real.Basic                                          -- 1

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop := ∀ x, f x ≤ a                     -- 2
def FnLb (f : ℝ → ℝ) (a : ℝ) : Prop := ∀ x, a ≤ f x                     -- 3

variable (f g : ℝ → ℝ)                                                  -- 4
variable (a b : ℝ)                                                      -- 5

-- --------------------------------------------------------------------
-- Ejercicio 2. 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.

-- 1ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (fun x ↦ f x + g x) (a + b) :=
by
  have h1 : ∀ x, f x + g x ≤ a + b := by
    intro x
    -- x : ℝ
    -- ⊢ f x + g x ≤ a + b
    have h2 : f x ≤ a := hfa x
    have h3 : g x ≤ b := hgb x
```

```
    show f x + g x ≤ a + b
    exact add_le_add h2 h3
  show FnUb (fun x ↦ f x + g x) (a + b)
  exact h1

-- 2ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (fun x ↦ f x + g x) (a + b) :=
by
  have h1 : ∀ x, f x + g x ≤ a + b := by
    intro x
    -- x : ℝ
    -- ⊢ f x + g x ≤ a + b
    show f x + g x ≤ a + b
    exact add_le_add (hfa x) (hgb x)
  show FnUb (fun x ↦ f x + g x) (a + b)
  exact h1

-- 3ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (fun x ↦ f x + g x) (a + b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (fun x => f x + g x) x ≤ a + b
  dsimp
  -- ⊢ f x + g x ≤ a + b
  apply add_le_add
  . -- ⊢ f x ≤ a
    apply hfa
  . -- ⊢ g x ≤ b
    apply hgb

-- Notas.
-- + Nota 1. Con ”intro x” se despliega la definición de FnUb y se introduce
--   la variable x en el contexto.
-- + Nota 2. Con ”dsimp” se simplifica la definición del lambda. El mismo
```

```
--   efecto se consigue con ”change f x + g x ≤ a + b”

-- 4ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (fun x ↦ f x + g x) (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)
```

3.1.6. Operaciones con cotas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de los números reales.
-- 2. Definir cota superior de una función.
-- 3. Definir cota inferior de una función.
-- 4. Declarar f y g como variables de funciones de ℝ en ℝ.
-- 5. Declarar a y b como variables sobre ℝ.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic                                  -- 1

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop := ∀ x, f x ≤ a            -- 2
def FnLb (f : ℝ → ℝ) (a : ℝ) : Prop := ∀ x, a ≤ f x            -- 3

variable (f g : ℝ → ℝ)                                          -- 4
variable (a b : ℝ)                                              -- 5

-- ---------------------------------------------------------------------
-- Ejercicio 2. 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
-- ========================

-- 1ª demostración
-- ===============

example
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
by
  have h1 : ∀ x, a + b ≤ f x + g x := by
    intro x
    -- x : ℝ
    -- ⊢ a + b ≤ f x + g 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 FnLb (f + g) (a + b)
  exact h1

-- 2ª demostración
-- ===============

example
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
by
```

```
  have h1 : ∀ x, a + b ≤ f x + g x := by
    intro x
    -- x : ℝ
    -- ⊢ a + b ≤ f x + g x
    show a + b ≤ f x + g x
    exact add_le_add (hfa x) (hgb x)
  show FnLb (f + g) (a + b)
  exact h1

-- 3ª demostración
-- ===============

example
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ a + b ≤ (f + g) x
  dsimp
  -- ⊢ a + b ≤ f x + g x
  apply add_le_add
  . -- ⊢ a ≤ f x
    apply hfa
  . -- ⊢ b ≤ g x
    apply hgb

-- 4ª demostración
-- ===============

example
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
λ x ↦ add_le_add (hfa x) (hgb x)

-- ---------------------------------------------------------------------
-- Ejercicio 3. 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
-- ========================

-- 1ª demostración
-- ===============

example
  (nnf : FnLb f 0)
  (nng : FnLb g 0)
  : FnLb (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x := by
    intro x
    -- x : ℝ
    -- ⊢ 0 ≤ f x * g 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 FnLb (f * g) 0
  exact h1

-- 2ª demostración
-- ===============

example
  (nnf : FnLb f 0)
  (nng : FnLb g 0)
  : FnLb (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x := by
    intro x
```

```
    -- x : ℝ
    -- ⊢ 0 ≤ f x * g x
    show 0 ≤ f x * g x
    exact mul_nonneg (nnf x) (nng x)
  show FnLb (f * g) 0
  exact h1

-- 3ª demostración
-- ===============

example
  (nnf : FnLb f 0)
  (nng : FnLb g 0)
  : FnLb (f * g) 0 :=
by
  intro x
  -- x : ℝ
  -- ⊢ 0 ≤ (f * g) x
  dsimp
  -- ⊢ 0 ≤ f x * g x
  apply mul_nonneg
  . -- ⊢ 0 ≤ f x
    apply nnf
  . -- ⊢ 0 ≤ g x
    apply nng

-- 4ª demostración
-- ===============

example
  (nnf : FnLb f 0)
  (nng : FnLb g 0)
  : FnLb (f * g) 0 :=
λ x ↦ mul_nonneg (nnf x) (nng x)

-- ---------------------------------------------------------------------
-- Ejercicio 4. 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
-- a * b es una cota superior de f * g.
-- ----------------------------------------------------------------------

-- 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) [0 ≤ 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
-- ========================

-- 1ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  (nng : FnLb g 0)
  (nna : 0 ≤ a)
  : FnUb (f * g) (a * b) :=
by
  have h1 : ∀ x, f x * g x ≤ a * b := by
    intro x
    -- x : ℝ
    -- ⊢ f x * g x ≤ a * b
    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 FnUb (f * g) (a * b)
  exact h1

-- 2ª demostración
-- ===============

example
```

```
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  (nng : FnLb g 0)
  (nna : 0 ≤ a)
  : FnUb (f * g) (a * b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x ≤ a * b
  dsimp
  -- ⊢ f x * g x ≤ a * b
  apply mul_le_mul
  . -- ⊢ f x ≤ a
    apply hfa
  . -- ⊢ g x ≤ b
    apply hgb
  . -- ⊢ 0 ≤ g x
    apply nng
  . -- ⊢ 0 ≤ a
    apply nna

-- 3ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  (nng : FnLb g 0)
  (nna : 0 ≤ a)
  : FnUb (f * g) (a * b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x ≤ a * b
  have h1:= hfa x
  have h2:= hgb x
  have h3:= nng x
  exact mul_le_mul h1 h2 h3 nna

-- 4ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
```

```
  (nng : FnLb g 0)
  (nna : 0 ≤ a)
  : FnUb (f * g) (a * b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x ≤ a * b
  specialize hfa x
  -- hfa : f x ≤ a
  specialize hgb x
  -- hgb : g x ≤ b
  specialize nng x
  -- nng : 0 ≤ g x
  exact mul_le_mul hfa hgb nng nna

-- 5ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  (nng : FnLb g 0)
  (nna : 0 ≤ a)
  : FnUb (f * g) (a * b) :=
λ x ↦ mul_le_mul (hfa x) (hgb x) (nng x) nna

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

variable (c d : ℝ)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d)
#check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
```

3.1.7. Cota_doble

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

-- ---------------------------------------------------------------------
-- Ejercicio 1. Declarar x como variable implícita sobre los reales.
-- ----------------------------------------------------------------------
```

```
variable {x : ℝ}

-- ----------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    ∃a, x < a
-- entonces
--    ∃b, x < b * 2
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (h : ∃a, x < a)
  : ∃b, x < b * 2 :=
by
  rcases h with ⟨a, hxa⟩
  -- a : ℝ
  -- hxa : x < a
  use a / 2
  -- ⊢ x < a / 2 * 2
  calc x < a         := hxa
       _ = a / 2 * 2 := (div_mul_cancel_of_invertible a 2).symm

-- 2ª demostración
-- ===============

example
  (h : ∃a, x < a)
  : ∃b, x < b * 2 :=
by
  rcases h with ⟨a, hxa⟩
  -- a : ℝ
  -- hxa : x < a
  use a / 2
  -- ⊢ x < a / 2 * 2
  linarith

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

variable (a b : ℝ)
variable (b : ℝ) [Invertible b]
#check (div_mul_cancel_of_invertible a b : a / b * b = a)
#check (two_ne_zero : 2 ≠ 0)
```

3.1.8. Generalización a monoides

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de monoides.
-- 2. Declarar α como un tipo.
-- 3. Declarar R como un monoide ordenado cancelativo.
-- 4. Declarar a, b, c y d como variables sobre R.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable {α : Type*}
variable {R : Type*} [AddCommMonoid R] [PartialOrder R] [IsOrderedCancelAddMonoid R]

variable (a b c d : R)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de
--     @add_le_add R _ a b c d
-- ----------------------------------------------------------------------

#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Definir la función
--    FnUb (α → R) → R → Prop
-- tal que (FnUb f a) afirma que a es una cota superior de f.
-- ----------------------------------------------------------------------

def FnUb’ (f : α → R) (a : R) : Prop :=
  ∀ x, f x ≤ a

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que que la suma de una cota superior de f y
-- otra de g es una cota superior de f + g.
-- ----------------------------------------------------------------------

theorem fnUb_add
  {f g : α → R}
  {a b : R}
  (hfa : FnUb’ f a)
  (hgb : FnUb’ g b)
  : FnUb’ (fun x ↦ f x + g x) (a + b) :=
fun x ↦ add_le_add (hfa x) (hgb x)
```

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

#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

3.1.9. Función monótona

```
-- ---------------------------------------------------------------------
-- Ejercicio. Explicitar la definición de función monótona poniendo el
-- nombre en la hipótesis y su definición en la conclusión.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

example
  (f : ℝ → ℝ)
  (h : Monotone f) :
  ∀ {a b}, a ≤ b → f a ≤ f b :=
@h
```

3.1.10. Suma de funciones monótonas

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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 tenemos 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)

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 := by
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (f + g) a ≤ (f + g) b
    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 := by
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (f + g) a ≤ (f + g) b
```

```
    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 := by
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (f + g) a ≤ (f + g) b
    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
  intros a b hab
  -- a b : ℝ
  -- hab : a ≤ b
  -- ⊢ (f + g) a ≤ (f + g) b
  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)
```

3.1.11. Producto de un positivo por una función monótona

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si c es no negativo y f es monótona,
-- entonces c * f 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 := by
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (fun x => c * f x) a ≤ (fun x => c * f x) b
    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
  intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (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)
```

3.1.12. Composición de funciones monótonas

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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)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)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 := by
```

```
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (f ∘ g) a ≤ (f ∘ g) b
    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 := by
    intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (f ∘ g) a ≤ (f ∘ g) b
    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
  intros a b hab
    -- a b : ℝ
    -- hab : a ≤ b
    -- ⊢ (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)
```

3.1.13. Funciones pares e impares

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de los números reales.
-- 2. Declarar f y g como variables sobre funciones de ℝ en ℝ.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic -- 1

namespace oculto

variable (f g : ℝ → ℝ)         -- 2

-- ---------------------------------------------------------------------
-- Ejercicio 2. Definir la función
--    even (ℝ → ℝ) → Prop
-- tal que (even f) afirma que f es par.
-- ----------------------------------------------------------------------

def even (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Definir la función
--    odd (ℝ → ℝ) → Prop
-- tal que (odd f) afirma que f es impar.
-- ----------------------------------------------------------------------

def odd  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- ---------------------------------------------------------------------
-- Ejercicio 4. 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
-- ========================

-- 1ª demostración
-- ===============

example
  (ef : even f)
  (eg : even g)
  : even (f + g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f + g) x = (f + g) (-x)
  have h1 : f x = f (-x) := ef x
  have h2 : g x = g (-x) := eg 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
  (ef : even f)
  (eg : even g)
  : even (f + g) :=
by
  intro x
  -- x : ℝ
```

```
  -- ⊢ (f + g) x = (f + g) (-x)
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g x    := congrArg (. + g x) (ef x)
     _ = f (-x) + g (-x) := congrArg (f (-x) + .) (eg x)
     _ = (f + g) (-x)    := rfl

-- 3ª demostración
-- ===============

example
  (ef : even f)
  (eg : even g)
  : even (f + g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f + g) x = (f + g) (-x)
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g (-x) := by rw [ef, eg]
     _ = (f + g) (-x)    := rfl

-- ---------------------------------------------------------------------
-- Ejercicio 5. 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 par]
--            = f(-x)g(-x))
--            = (f·g)(-x)

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

-- 1ª demostración
```

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

example
  (of : odd f)
  (og : odd g)
  : even (f * g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = (f * g) (-x)
  have h1 : f x = -f (-x) := of x
  have h2 : g x = -g (-x) := og 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
  (of : odd f)
  (og : odd g)
  : even (f * g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = (f * g) (-x)
  calc (f * g) x
       = f x * g x              := rfl
     _ = (-f (-x)) * g x        := congrArg (. * g x) (of x)
     _ = (-f (-x)) * (-g (-x))  := congrArg ((-f (-x)) * .) (og x)
     _ = f (-x) * g (-x)        := neg_mul_neg (f (-x)) (g (-x))
     _ = (f * g) (-x)           := rfl

-- 3ª demostración
-- ===============

example
  (of : odd f)
  (og : odd g)
  : even (f * g) :=
by
```

```
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = (f * g) (-x)
  calc (f * g) x
       = f x * g x          := rfl
     _ = -f (-x) * -g (-x) := by rw [of, og]
     _ = f (-x) * g (-x)   := by rw [neg_mul_neg]
     _ = (f * g) (-x)      := rfl

-- 4ª demostración
-- ===============

example
  (of : odd f)
  (og : odd g)
  : even (f * g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = (f * g) (-x)
  calc (f * g) x
       = f x * g x        := rfl
     _ = f (-x) * g (-x) := by rw [of, og, neg_mul_neg]
     _ = (f * g) (-x)    := rfl

-- ---------------------------------------------------------------------
-- Ejercicio 6. 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 impar; 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
-- ========================
```

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

example
  (ef : even f)
  (og : odd g)
  : odd (f * g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = -(f * g) (-x)
  have h1 : f x = f (-x) := ef x
  have h2 : g x = -g (-x) := og 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
  (ef : even f)
  (og : odd g)
  : odd (f * g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = -(f * g) (-x)
  calc (f * g) x
       = f x * g x          := rfl
    _  = f (-x) * -g (-x)   := by rw [ef, og]
    _  = -(f (-x) * g (-x)) := by rw [mul_neg]
    _  = -(f * g) (-x)      := rfl

-- 3ª demostración
-- ===============

example
  (ef : even f)
  (og : odd g)
  : odd (f * g) :=
```

```
by
  intro x
  -- x : ℝ
  -- ⊢ (f * g) x = -(f * g) (-x)
  calc (f * g) x
       = f x * g x            := rfl
     _ = -(f (-x) * g (-x)) := by rw [ef, og, mul_neg]
     _ = -((f * g) (-x))    := rfl

-- ---------------------------------------------------------------------
-- Ejercicio 7. 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
-- ========================

-- 1ª demostración
-- ===============

example
  (ef : even f)
  (og : odd g)
  : even (f ∘ g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f ∘ g) x = (f ∘ g) (-x)
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (-g (-x))  := congrArg f (og x)
     _ = f (g (-x))   := (ef (g (-x))).symm
     _ = (f ∘ g) (-x) := rfl
```

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

example
  (ef : even f)
  (og : odd g)
  : even (f ∘ g) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f ∘ g) x = (f ∘ g) (-x)
  calc (f ∘ g) x
       = f (g x)       := rfl
     _ = f (-g (-x))   := by rw [og]
     _ = f (g (-x))    := by rw [← ef]
     _ = (f ∘ g) (-x) := rfl

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

variable (a b : ℝ)
#check (congrArg f : a = b → f a = f b)
#check (mul_neg a b : a * -b = -(a * b))
#check (neg_mul_neg a b : -a * -b = a * b)

end oculto
```

3.1.14. Propiedad reflexiva del subconjunto

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ s
  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
```

3.1.15. Propiedad transitiva del subconjunto

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar la propiedad transitiva de la inclusión de
-- conjuntos.
-- ----------------------------------------------------------------------

-- 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
  -- x : α
  -- xr : x ∈ r
  -- ⊢ x ∈ t
  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
  -- ⊢ x ∈ t
  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)
```

3.1.16. Cotas superiores de conjuntos

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Declarar α como un tipo sobre órdenes parciales.
-- 2. Declarar s como una variable sobre conjuntos de elementos de tipo α
-- 3. Declarar a y b como variables sobre α.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

variable {α : Type _} [PartialOrder α] -- 1
variable (s : Set α)                   -- 2
variable (a b : α)                     -- 3

-- ---------------------------------------------------------------------
-- Ejercicio 2. Definir la función
--    SetUb : set α → α → Prop
```

```
-- tal que (SetUb s a) afirma que a es una cota superior de s.
-- ----------------------------------------------------------------------

def SetUb (s : Set α) (a : α) :=
  ∀ {x}, x ∈ s → x ≤ a

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si a es una cota superior de s y a ≤ b,
-- entonces b es una cota superior de s.
-- ----------------------------------------------------------------------

-- 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 : SetUb s a)
  (h2 : a ≤ b)
  : SetUb s b :=
by
  intro x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ≤ b
  have h3 : x ≤ a := h1 xs
  show x ≤ b
  exact le_trans h3 h2

-- 2ª demostración
-- ===============

example
  (h1 : SetUb s a)
  (h2 : a ≤ b)
  : SetUb s b :=
by
```

```
  intro x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ≤ b
  calc x ≤ a := h1 xs
       _ ≤ b := h2

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

variable (c : α)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
```

3.1.17. Funciones inyectivas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de números reales.
-- 2. Abrir el espacio de nombre de las funciones.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic -- 1
open Function                  -- 2

variable {c : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. 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
-- ========================

-- 1ª demostración
-- ===============

example : Injective ((. + c)) :=
by
  intro x1 x2 h1
  -- x1 x2 : ℝ
  -- h1 : (fun x => x + c) x1 = (fun x => x + c) x2
  -- ⊢ x1 = x2
  exact add_right_cancel h1

-- 2ª 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)

-- ---------------------------------------------------------------------
-- 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,
--    c * x₁ = c * x₂
-- y, por L1 y puesto que c ≠ 0, se tiene que
--    x₁ = x₂.

-- Demostraciones con Lean4
```

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

-- 1ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Injective ((c * .)) :=
by
  intro x1 x2 h1
  -- x1 x2 : ℝ
  -- h1 : (fun x => c * x) x1 = (fun x => c * x) x2
  -- ⊢ 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)
#check (add_right_cancel : a + b = c + b → a = c)
```

3.1.18. Composición de funciones inyectivas

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que la composición de funciones inyectivas es
-- inyectiva.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

open Function

variable {α : Type _} {β : Type _} {γ : Type _}
```

```
variable {f : α → β} {g : β → γ}

-- 1ª demostración
-- ===============

example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro x y h1
  -- x y : α
  -- h1 : (g ∘ f) x = (g ∘ f) y
  -- ⊢ x = 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
  intros x y h
  -- x y : α
  -- h : (g ∘ f) x = (g ∘ f) y
  -- ⊢ x = y
  apply hf
  -- ⊢ f x = f y
  apply hg
  -- ⊢ g (f x) = g (f y)
  apply h

-- 3ª demostración
-- ===============

example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
fun _ _ h ↦ hf (hg h)
```

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

example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
-- by exact?
Injective.comp hg hf

-- 5ª 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))
```

3.2. El cuantificador existencial

3.2.1. Existencia de valor intermedio

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Demostrar que hay algún número real entre 2 y 3.
-- ----------------------------------------------------------------------

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

-- 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
  -- ⊢ 2 < 5 / 2 ∧ 5 / 2 < 3
  norm_num

-- 4ª demostración
-- ===============

example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
⟨5 / 2, by norm_num⟩

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

variable (w : ℝ)
variable (p : ℝ → Prop)
#check (Exists.intro w : p w → Exists p)
```

3.2.2. Definición de funciones acotadas

```
import Mathlib.Data.Real.Basic

-- ---------------------------------------------------------------------
-- Ejercicio 1. Definir la función
--    FnUb (ℝ → ℝ) → ℝ → Prop
```

```
-- tal que (FnUb f a) afirma que a es una cota superior de f.
-- ----------------------------------------------------------------------

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

-- ---------------------------------------------------------------------
-- Ejercicio 2. Definir la función
--    FnLb (ℝ → ℝ) → ℝ → Prop
-- tal que (FnLb f a) afirma que a es una cota inferior de f.
-- ----------------------------------------------------------------------

def FnLb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

-- ---------------------------------------------------------------------
-- Ejercicio 3. Definir la función
--    FnHasUb (ℝ → ℝ) → Prop
-- tal que (FnHasUb f) afirma que f tiene cota superior.
-- ----------------------------------------------------------------------

def FnHasUb (f : ℝ → ℝ) :=
  ∃ a, FnUb f a

-- ---------------------------------------------------------------------
-- Ejercicio 4. Definir la función
--    FnHasLb (ℝ → ℝ) → Prop
-- tal que (FnHasLb f) afirma que f tiene cota inferior.
-- ----------------------------------------------------------------------

def FnHasLb (f : ℝ → ℝ) :=
  ∃ a, FnLb f a
```

3.2.3. Suma de funciones acotadas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría Definicion_de_funciones_acotadas
-- 2. Declarar f y g como variables de funciones de ℝ en ℝ.
-- 3. Declarar a y b como variables sobre ℝ.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_funciones_acotadas
```

```
variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si a es una cota superior de f y b lo es
-- de g, entonces a + b lo es de f + g.
-- ----------------------------------------------------------------------

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

-- Usaremos el siguiente lema:
--    L1 : a ≤ b → c ≤ d → a + c ≤ b + d
--
-- Tenemos que demostrar que
--    (∀ x ∈ ℝ) [(f + g)(x) ≤ a + b]
-- Sea x ∈ ℝ. Puesto que a es una cota superior de f, se tiene que
--    f(x) ≤ a                                                      (1)
-- y, puesto que b es una cota superior de g, se tiene que
--    g(x) ≤ b                                                      (2)
-- Por tanto,
--    (f + g)(x) = f(x) + g(x)
--               ≤ a + b          [por L1, (1) y (2)]
-- que es lo que teníamos que demostrar.

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

-- 1ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (f + g) (a + b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f + g) x ≤ a + b
  have h1 : f x ≤ a := hfa x
  have h2 : g x ≤ b := hgb x
  calc (f + g) x = f x + g x := by rfl
               _ ≤ a + b     := add_le_add h1 h2

-- 2ª demostración
```

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

example
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (f + g) (a + b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ (f + g) x ≤ a + b
  change f x + g x ≤ a + b
  -- ⊢ f x + g x ≤ a + b
  apply add_le_add
  . -- ⊢ f x ≤ a
    apply hfa
  . -- ⊢ g x ≤ b
    apply hgb

-- 3ª demostración
-- ===============

theorem FnUb_add
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (f + g) (a + b) :=
fun x ↦ add_le_add (hfa x) (hgb x)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que la suma de dos funciones acotadas
-- superiormente también lo está.
-- ----------------------------------------------------------------------

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

-- Usaremos el siguiente lema:
--    L1: FnUb f a → FnUb g b → FnUb (f + g) (a + b)
--
-- 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 L1, a+b es una cota superior de f+g. or consiguiente,
-- f+g está acotada superiormente.

-- Demostraciones con Lean4
```

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

-- 1ª demostración
-- ===============

example
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
by
  rcases ubf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnUb f a
  rcases ubg with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnUb g b
  have h : ∀ x, (f + g) x ≤ a + b :=
    FnUb_add ha hb
  have h4 : ∃ z, ∀ x, (f + g) x ≤ z :=
    Exists.intro (a + b) h
  show FnHasUb (f + g)
  exact h4

-- 2ª demostración
-- ===============

example
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
by
  rcases ubf with ⟨a, ubfa⟩
  -- a : ℝ
  -- ubfa : FnUb f a
  rcases ubg with ⟨b, ubfb⟩
  -- b : ℝ
  -- ubfb : FnUb g b
  use a + b
  -- ⊢ FnUb (f + g) (a + b)
  apply FnUb_add ubfa ubfb

-- 4ª demostración
-- ===============

example
```

```
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
by
  rcases ubf with ⟨a, ubfa⟩
  -- a : ℝ
  -- ubfa : FnUb f a
  rcases ubg with ⟨b, ubfb⟩
  -- b : ℝ
  -- ubfb : FnUb g b
  exact ⟨a + b, FnUb_add ubfa ubfb⟩

-- 5ª demostración
-- ===============

example
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
by
  obtain ⟨a, ubfa⟩ := ubf
  -- a : ℝ
  -- ubfa : FnUb f a
  obtain ⟨b, ubfb⟩ := ubg
  -- b : ℝ
  -- ubfb : FnUb g b
  exact ⟨a + b, FnUb_add ubfa ubfb⟩

-- 6ª demostración
-- ===============

example :
  FnHasUb f → FnHasUb g → FnHasUb (f + g) :=
by
  rintro ⟨a, ubfa⟩ ⟨b, ubfb⟩
  -- a : ℝ
  -- ubfa : FnUb f a
  -- b : ℝ
  -- ubfb : FnUb g b
  -- ⊢ FnHasUb (f + g)
  exact ⟨a + b, FnUb_add ubfa ubfb⟩

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

```
example :
  FnHasUb f → FnHasUb g → FnHasUb (f + g) :=
fun ⟨a, ubfa⟩ ⟨b, ubfb⟩ ↦ ⟨a + b, FnUb_add ubfa ubfb⟩

example
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
  match ubf, ubg with
    | ⟨a, ubfa⟩, ⟨b, ubgb⟩ =>
      -- a : ℝ
      -- ubfa : FnUb f a
      -- b : ℝ
      -- ubgb : FnUb g b
      ⟨a + b, FnUb_add ubfa ubgb⟩

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

variable (c d : ℝ)
variable (w : ℝ)
variable (p : ℝ → Prop)
#check (Exists.intro w : p w → Exists p)
#check (FnUb_add : FnUb f a → FnUb g b → FnUb (f + g) (a + b))
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

3.2.4. Suma de funciones acotadas inferiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría Definicion_de_funciones_acotadas
-- 2. Declarar f y g como variables de funciones de ℝ en ℝ.
-- 3. Declarar a y b como variables sobre ℝ.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_funciones_acotadas
variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si a es una cota inferior de f y b lo es
-- de g, entonces a + b lo es de f + g.
-- ----------------------------------------------------------------------
```

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

lemma FnLb_add
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
by
  intro x
  -- x : ℝ
  -- ⊢ a + b ≤ (f + g) x
  change a + b ≤ f x + g x
  -- ⊢ a + b ≤ f x + g x
  apply add_le_add
  . -- ⊢ a ≤ f x
    apply hfa
  . -- ⊢ b ≤ g x
    apply hgb

-- 2ª demostración
-- ===============

example
  (hfa : FnLb f a)
  (hgb : FnLb g b)
  : FnLb (f + g) (a + b) :=
fun x ↦ add_le_add (hfa x) (hgb x)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que la suma de dos funciones acotadas
-- inferiormente también lo está.
-- ----------------------------------------------------------------------

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

-- Usaremos el siguiente lema:
--    FnLb_add: FnLb f a → FnLb g b → FnLb (f + g) (a + b)
--
-- 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
-- ========================

-- 1ª demostración
-- ===============

example
  (lbf : FnHasLb f)
  (lbg : FnHasLb g)
  : FnHasLb (f + g) :=
by
  rcases lbf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnLb f a
  rcases lbg with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnLb g b
  have h1 : FnLb (f + g) (a + b) := FnLb_add ha hb
  have h2 : ∃ z, ∀ x, z ≤ (f + g) x :=
    Exists.intro (a + b) h1
  show FnHasLb (f + g)
  exact h2

-- 2ª demostración
-- ===============

example
  (lbf : FnHasLb f)
  (lbg : FnHasLb g)
  : FnHasLb (f + g) :=
by
  rcases lbf with ⟨a, lbfa⟩
  -- a : ℝ
  -- lbfa : FnLb f a
  rcases lbg with ⟨b, lbgb⟩
  -- b : ℝ
  -- lbgb : FnLb g b
  use a + b
  -- ⊢ FnLb (f + g) (a + b)
  apply FnLb_add lbfa lbgb

-- 3ª demostración
-- ===============
```

```
example
  (lbf : FnHasLb f)
  (lbg : FnHasLb g)
  : FnHasLb (f + g) :=
by
  rcases lbf with ⟨a, lbfa⟩
  -- a : ℝ
  -- lbfa : FnLb f a
  rcases lbg with ⟨b, lbfb⟩
  -- b : ℝ
  -- lbfb : FnLb g b
  exact ⟨a + b, FnLb_add lbfa lbfb⟩

-- 4ª demostración
-- ===============

example :
  FnHasLb f → FnHasLb g → FnHasLb (f + g) :=
by
  rintro ⟨a, lbfa⟩ ⟨b, lbfb⟩
  -- a : ℝ
  -- lbfa : FnLb f a
  -- b : ℝ
  -- lbfb : FnLb g b
  exact ⟨a + b, FnLb_add lbfa lbfb⟩

-- 5ª demostración
-- ===============

example :
  FnHasLb f → FnHasLb g → FnHasLb (f + g) :=
fun ⟨a, lbfa⟩ ⟨b, lbfb⟩ ↦ ⟨a + b, FnLb_add lbfa lbfb⟩

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

variable (c d : ℝ)
variable (w : ℝ)
variable (p : ℝ → Prop)
#check (Exists.intro w : p w → Exists p)
#check (FnLb_add : FnLb f a → FnLb g b → FnLb (f + g) (a + b))
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

3.2.5. Producto por función acotada superiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría Definicion_de_funciones_acotadas
-- 2. Declarar f como variable de funciones de ℝ en ℝ.
-- 3. Declarar a y c como variables sobre ℝ.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_funciones_acotadas

variable {f : ℝ → ℝ}
variable {a c : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. 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
-- ========================

-- 1ª demostración
-- ===============

example
  (hfa : FnUb f a)
  (h : c ≥ 0)
  : FnUb (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 : FnUb f a)
  (h : c ≥ 0)
  : FnUb (fun x ↦ c * f x) (c * a) :=
by
  intro y
  -- y : ℝ
  -- ⊢ (fun x => c * f x) y ≤ c * a
  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 : FnUb f a)
  (h : c ≥ 0)
  : FnUb (fun x ↦ c * f x) (c * a) :=
by
  intro y
  -- y : ℝ
  -- ⊢ (fun x => c * f x) y ≤ c * a
  exact mul_le_mul_of_nonneg_left (hfa y) h

-- 4ª demostración
-- ===============

lemma FnUb_mul
  (hfa : FnUb f a)
  (h : c ≥ 0)
  : FnUb (fun x ↦ c * f x) (c * a) :=
fun y ↦ mul_le_mul_of_nonneg_left (hfa y) h

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si c ≥ 0 y f está acotada superiormente,
```

```
-- entonces c * f también lo está.
-- ----------------------------------------------------------------------

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

-- Usaremos el siguiente lema:
--    FnUb_mul : FnUb f a → c ≥ 0 → FnUb (fun x ↦ c * f x) (c * a)
--
-- Puesto que f está acotada superiormente, tiene una cota superior. Sea
-- a una de dichas cotas. Entonces, por el lema FnUb_mul, ca es una cota
-- superior de cf. Por consiguiente, cf está acotada superiormente.

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

-- 1ª demostración
-- ===============

example
  (hf : FnHasUb f)
  (hc : c ≥ 0)
  : FnHasUb (fun x ↦ c * f x) :=
by
  rcases hf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnUb f a
  have h1 : FnUb (fun x ↦ c * f x) (c * a) :=
    FnUb_mul ha hc
  have h2 : ∃ z, ∀ x, (fun x ↦ c * f x) x ≤ z :=
    Exists.intro (c * a) h1
  show FnHasUb (fun x ↦ c * f x)
  exact h2

-- 2ª demostración
-- ===============

example
  (hf : FnHasUb f)
  (hc : c ≥ 0)
  : FnHasUb (fun x ↦ c * f x) :=
by
  rcases hf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnUb f a
```

```
  use c * a
  -- ⊢ FnUb (fun x => c * f x) (c * a)
  apply FnUb_mul ha hc

-- 3ª demostración
-- ===============

example
  (hf : FnHasUb f)
  (hc : c ≥ 0)
  : FnHasUb (fun x ↦ c * f x) :=
by
  rcases hf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnUb f a
  exact ⟨c * a, FnUb_mul ha hc⟩

-- 4ª demostración
-- ===============

example
  (hc : c ≥ 0)
  : FnHasUb f → FnHasUb (fun x ↦ c * f x) :=
by
  rintro ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnUb f a
  exact ⟨c * a, FnUb_mul ha hc⟩

-- 5ª demostración
-- ===============

example
  (hc : c ≥ 0)
  : FnHasUb f → FnHasUb (fun x ↦ c * f x) :=
fun ⟨a, ha⟩ ↦ ⟨c * a, FnUb_mul ha hc⟩

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

variable (b : ℝ)
#check (FnUb_mul : FnUb f a → c ≥ 0 → FnUb (fun x ↦ c * f x) (c * a))
#check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
```

3.2.6. Sumas de cotas superiores con rcases y rintros

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría Definicion_de_funciones_acotadas
-- 2. Declarar f y g como variable de funciones de ℝ en ℝ.
-- 3. Declarar a y c como variables sobre ℝ.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_funciones_acotadas

variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si a es una cota superior de f y b es una
-- cota superior de g, entonces a + b lo es de f + g.
-- ----------------------------------------------------------------------

theorem FnUb_add
  (hfa : FnUb f a)
  (hgb : FnUb g b)
  : FnUb (f + g) (a + b) :=
fun x ↦ add_le_add (hfa x) (hgb x)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si f y g está acotadas superiormente,
-- entonces f + g también lo está.
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (ubf : FnHasUb f)
  (ubg : FnHasUb g)
  : FnHasUb (f + g) :=
by
  rcases ubf with ⟨a, ubfa⟩
  -- a : ℝ
  -- ubfa : FnUb f a
  rcases ubg with ⟨b, ubfb⟩
  -- b : ℝ
  -- ubfb : FnUb g b
  exact ⟨a + b, FnUb_add ubfa ubfb⟩
```

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

example :
  FnHasUb f →
  FnHasUb g →
  FnHasUb (f + g) :=
by
  rintro ⟨a, ubfa⟩ ⟨b, ubfb⟩
  -- a b : ℝ
  -- ubfa : FnUb f a
  -- b : ℝ
  -- ubfb : FnUb g b
  -- ⊢ FnHasUb (f + g)
  exact ⟨a + b, FnUb_add ubfa ubfb⟩

-- 3ª demostración
-- ===============

example : FnHasUb f → FnHasUb g →
  FnHasUb (f + g) :=
fun ⟨a, ubfa⟩ ⟨b, ubfb⟩ ↦ ⟨a + b, FnUb_add ubfa ubfb⟩

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

variable (c d : ℝ)
#check (FnUb_add : FnUb f a → FnUb g b → FnUb (f + g) (a + b))
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

3.2.7. Producto_de_suma_de_cuadrados

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar α como un tipo sobre los anillos conmutativos.
-- 3. Declarar x e y como variables sobre α.
-- ----------------------------------------------------------------------

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

```
-- ----------------------------------------------------------------------
-- Ejercicio 2. Definir la función
--    sum_of_squares : α → Prop
-- tal que (sum_of_squares x) afirma que x se puede escribir como la suma
-- de dos cuadrados.
-- ----------------------------------------------------------------------

def sum_of_squares (x : α) :=
  ∃ a b, x = a^2 + b^2

-- ----------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si x e y se pueden escribir como la suma
-- de dos cuadrados, entonces también se puede escribir x * y.
-- ----------------------------------------------------------------------

-- 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
-- ========================

-- 1ª demostración
-- ===============

example
  (hx : sum_of_squares x)
  (hy : sum_of_squares y)
  : sum_of_squares (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
  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 sum_of_squares (x * y)
  exact h3

-- 2ª demostración
-- ===============

example
  (hx : sum_of_squares x)
  (hy : sum_of_squares y)
  : sum_of_squares (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
  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 sum_of_squares (x * y)
  exact h2

-- 3ª demostración
```

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

example
  (hx : sum_of_squares x)
  (hy : sum_of_squares y)
  : sum_of_squares (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]
  -- ⊢ sum_of_squares ((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 : sum_of_squares x)
  (hy : sum_of_squares y)
  : sum_of_squares (x * y) :=
by
  rcases hx with ⟨a, b, rfl⟩
  -- a b : α
  -- ⊢ sum_of_squares ((a ^ 2 + b ^ 2) * y)
  rcases hy with ⟨c, d, rfl⟩
  -- c d : α
  -- ⊢ sum_of_squares ((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

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

variable (w : ℝ)
variable (p : ℝ → Prop)
#check (Exists.intro w : p w → Exists p)
```

3.2.8. Transitividad de la divisibilidad

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases divbc with ⟨e, ceq⟩
  -- e : ℕ
  -- 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⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases divbc with ⟨e, ceq⟩
  -- e : ℕ
  -- 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⟩
  -- e : ℕ
  -- ⊢ a ∣ b * e
  rcases divab with ⟨d, rfl⟩
  -- d : ℕ
  -- ⊢ 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
  rcases divab with ⟨d, beq⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases 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

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

variable (f : ℕ → ℕ)
#check (Dvd.intro c : a * c = b → a ∣ b)
#check (congrArg f : a = b → f a = f b)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
```

3.2.9. Suma divisible

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. 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
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases divab with ⟨d, beq⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases divac with ⟨e, ceq⟩
  -- e : ℕ
  -- 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
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases divab with ⟨d, beq⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases divac with ⟨e, ceq⟩
  -- e : ℕ
  -- 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
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases divab with ⟨d, beq⟩
  -- d : ℕ
  -- beq : b = a * d
  rcases divac with ⟨e, ceq⟩
  -- e : ℕ
  -- ceq : c = a * e
  show a ∣ (b + c)
  exact Dvd.intro (d + e) (by linarith)

-- 4ª demostración
-- ===============

example
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
by
  obtain ⟨d, beq⟩ := divab
  -- d : ℕ
  -- beq : b = a * d
  obtain ⟨e, ceq⟩ := divac
  -- 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
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
by
```

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

-- 6ª demostración
-- ===============

example
  (divab : a ∣ b)
  (divac : a ∣ c)
  : a ∣ (b + c) :=
dvd_add divab divac

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

variable (f : ℕ → ℕ)
#check (Dvd.intro c : a * c = b → a ∣ b)
#check (congrArg f : a = b → f a = f b)
#check (dvd_add : a ∣ b →  a ∣ c → a ∣ (b + c))
#check (mul_add a b c : a * (b + c) = a * b + a * c)
```

3.2.10. Suma constante es suprayectiva

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
import Mathlib.Tactic

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 (add_right_surjective c : Surjective (fun x ↦ x + c))
#check (sub_add_cancel a b : (a - b) + b = a)
```

3.2.11. Producto por no nula es suprayectiva

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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_cancel₀ 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_cancel₀ x h

-- 3ª demostración
```

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

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
fun x ↦ ⟨x / c, mul_div_cancel₀ 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_cancel₀ a : b ≠ 0 → (a / b) * b = a)
#check (mul_comm a b : a * b = b * a)
#check (mul_div_cancel₀ a : b ≠ 0 → b * (a / b) = a)
#check (mul_left_surjective₀ : c ≠ 0 → Surjective (fun x ↦ c * x))
```

3.2.12. Propiedad de suprayectivas

```
-- ----------------------------------------------------------------------
-- Ejercicio. 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 Lean4
-- ======================

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

```
open Function

example
  {f : ℝ → ℝ}
  (h : Surjective f)
  : ∃ x, (f x)^2 = 9 :=
by
  rcases h 3 with ⟨y, hy⟩
  -- y : ℝ
  -- hy : f y = 3
  use y
  -- ⊢ (f y) ^ 2 = 9
  rw [hy]
  -- ⊢ 3 ^ 2 = 9
  norm_num
```

3.2.13. Composición de suprayectivas

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  rcases hg z with ⟨y, hy⟩
  -- y : β
  -- hy : g y = z
  rcases 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
  rcases hg z with ⟨y, hy⟩
  -- y : β
  -- hy : g y = z
  rcases 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
  rcases hg z with ⟨y, hy⟩
  -- y : β
  -- hy : g y = z
  rcases 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) :=
Surjective.comp hg hf

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

#check (Surjective.comp : Surjective g → Surjective f → Surjective (g ∘ f))
```

3.3. La negación

3.3.1. Asimétrica implica irreflexiva

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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)
```

3.3.2. Función no acotada superiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
import Mathlib.Tactic

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop := ∀ x, f x ≤ a

def FnHasUb (f : ℝ → ℝ) := ∃ a, FnUb f a

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

theorem sinCotaSup
  (h : ∀ a, ∃ x, f x > a)
  : ¬ FnHasUb f :=
by
  intros hf
  -- hf : FnHasUb f
  -- ⊢ False
  rcases hf with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnUb f b
  rcases h b with ⟨x, hx⟩
  -- x : ℝ
  -- hx : f x > b
  have : f x ≤ b := hb x
  linarith
```

3.3.3. Función no acotada inferiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
import Mathlib.Tactic

def FnLb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

def FnHasLb (f : ℝ → ℝ) : Prop :=
  ∃ a, FnLb f a

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ FnHasLb f :=
by
  intros hf
  -- hf : FnHasLb f
  -- ⊢ False
  obtain ⟨b, hb⟩ := hf
  -- b : ℝ
  -- hb : FnLb f b
  obtain ⟨x, hx⟩ := h b
  -- x : ℝ
  -- hx : f x < b
  have : b ≤ f x := hb x
  linarith

-- 2ª demostración
-- ===============

example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ FnHasLb f :=
by
  intros hf
```

```
  -- hf : FnHasLb f
  -- ⊢ False
  rcases hf with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnLb f b
  rcases h b with ⟨x, hx⟩
  -- x : ℝ
  -- hx : f x < b
  have : b ≤ f x := hb x
  linarith
```

3.3.4. La identidad no está acotada superiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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.Logica.Funcion_no_acotada_superiormente

-- 1ª demostración
-- ===============

example : ¬ FnHasUb (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 : ¬ FnHasUb (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 : ¬ FnHasUb (fun x ↦ x) :=
by
  apply sinCotaSup
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  exact fun a ↦ ⟨a + 1, by linarith⟩
```

3.3.5. Lemas sobre órdenes y negaciones

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
-- 1. Importar la librería de los reales.
-- 2. Declarar a y b como variables sobre los reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Calcular el tipo de los siguientes lemas
--    not_le_of_gt
--    not_lt_of_ge
--    lt_of_not_ge
--    le_of_not_gt
-- ----------------------------------------------------------------------
```

```
#check (not_le_of_gt : a > b → ¬ a ≤ b)
#check (not_lt_of_ge : a ≥ b → ¬ a < b)
#check (lt_of_not_ge : ¬ a ≥ b → a < b)
#check (le_of_not_gt : ¬ a > b → a ≤ b)
```

3.3.6. Propiedades de funciones monótonas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
-- 1. Importar la librería de los reales.
-- 2. Declarar f como variable de las funciones de ℝ en ℝ.
-- 3. Declarar a y b como variables sobre los reales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic
variable (f : ℝ → ℝ)
variable (a b : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. 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
-- ========================

-- 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)

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si a, b ∈ ℝ tales que
--    a ≤ b
--    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) que contradice a la hipótesis,

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

-- 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 (lt_of_not_ge : ¬ a ≥ b → a < b)
#check (not_lt_of_ge : a ≥ b → ¬a < b)
```

3.3.7. Propiedades de funciones monótonas (2)

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones
-- 1. Importar la librería de los reales.
-- 2. Declarar f como variable de las funciones de ℝ en ℝ.
-- 3. Declarar a y b como variables sobre los reales.
-- ----------------------------------------------------------------------

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

-- ---------------------------------------------------------------------
-- Ejercicio 2. 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
-- ========================

-- 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 : ℝ) ≤ 0 := h1 h2 h3
  linarith

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

variable (a c : ℝ)
#check (le_refl a : a ≤ a)
#check (monotone_const : Monotone fun _ : ℝ ↦ c)
```

3.3.8. Condición para no positivo

```
-- ---------------------------------------------------------------------
-- Ejercicio. 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
  . -- ⊢ x / 2 > 0
    exact half_pos hx0
  . -- ⊢ 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
  . -- ⊢ x / 2 > 0
    exact half_pos h
  . -- ⊢ x / 2 < x
    exact half_lt_self h

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

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

3.3.9. Negación de cuantificadores

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Inportar la librería de tácticas.
-- 2. Declarar α como una variable de tipos.
-- 3. Declarar P una variable sobre las propiedades de α.
-- ----------------------------------------------------------------------

import Mathlib.Tactic
variable {α : Type _}
variable (P : α → Prop)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    ¬ ∃ x, P x
-- entonces
--    ∀ x, ¬ P x
-- ----------------------------------------------------------------------

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

-- Sea y un elemento cualquiera. Tenemos que demostrar ¬P(y). Para ello,
-- supongamos que P(y). Entonces, (∃x)P(x) que es una contradicción con
-- la hipótesis,

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

-- 1ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
```

```
  : ∀ x, ¬ P x :=
by
  intros y h1
  -- y : α
  -- h1 : P x
  -- ⊢ False
  apply h
  -- ⊢ ∃ x, P x
  existsi y
  -- ⊢ P y
  exact h1

-- Comentario: La táctica (existsi e) es la regla de introducción del
-- existencial; es decir, sustituye en el cuerpo del objetivo
-- existencial su variable por e

-- 2ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
by
  intros y h1
  -- y : α
  -- h1 : P x
  -- ⊢ False
  apply h
  -- ⊢ ∃ x, P x
  use y
  -- ⊢ P y

-- 3ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
by
  intros y h1
  -- y : α
  -- h1 : P x
  -- ⊢ False
  apply h
  -- ⊢ ∃ x, P x
```

```
  exact ⟨y, h1⟩

-- 4ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
by
  intros y h1
  -- y : α
  -- h1 : P x
  -- ⊢ False
  exact h ⟨y, h1⟩

-- 5ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
fun y h1 ↦ h ⟨y, h1⟩

-- 6ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
by
  push_neg at h
  -- h : ∀ (x : α), ¬P x
  exact h

-- 7ª demostración
-- ===============

example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
not_exists.mp h

-- 8ª demostración
-- ===============
```

```
example
  (h : ¬ ∃ x, P x)
  : ∀ x, ¬ P x :=
by aesop

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    ∀ x, ¬ P x
-- entonces
--    ¬ ∃ x, P x
-- ----------------------------------------------------------------------

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

-- Supongamos que (∃x)P(x). Sea y tal que P(y). Puesto que (∀x)¬P)x), se
-- tiene que ¬P(y) que es una contradicción con P(y).

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

-- 1ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by
  intro h1
  -- h1 : ∃ x, P x
  -- ⊢ False
  rcases h1 with ⟨y, hy⟩
  -- y : α
  -- hy : P y
  have h2 : ¬P y := h y
  exact h2 hy

-- 2ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by
  intro h1
```

```
  -- h1 : ∃ x, P x
  -- ⊢ False
  rcases h1 with ⟨y, hy⟩
  -- y : α
  -- hy : P y
  exact (h y) hy

-- 3ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by
  rintro ⟨y, hy⟩
  -- y : α
  -- hy : P y
  exact (h y) hy

-- 4ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
fun ⟨y, hy⟩ ↦ (h y) hy

-- 5ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
not_exists_of_forall_not h

-- 6ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by aesop

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que si
```

```
--    ¬ ∀ x, P x
-- entonces
--    ∃ x, ¬ P x
-- ----------------------------------------------------------------------

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

-- Por reducción al absurdo, supongamos que ¬(∃x)¬P(x). Para obtener una
-- contradicción, demostraremos la negación de la hipótesis; es decir,
-- que (∀x)P(x). Para ello, sea y un elemento cualquiera y tenemos que
-- demostrar P(y). De nuevo, lo haremos por reducción al absurdo: Para
-- ello, supongamos que ¬P(y). Entonces, se tiene que (∃x)¬P(x) en
-- contradicción con nuestro primer supuesto de ¬(∃x)¬P(x).

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

-- 1ª demostración
-- ===============

example
  (h : ¬ ∀ x, P x)
  : ∃ x, ¬ P x :=
by
  by_contra h1
  -- h1 : ¬∃ x, ¬P x
  -- ⊢ False
  apply h
  -- ⊢ ∀ (x : α), P x
  intro y
  -- y : α
  -- ⊢ P y
  show P y
  by_contra h2
  -- h2 : ¬P y
  -- ⊢ False
  exact h1 ⟨y, h2⟩

-- Comentarios:
-- 1. La táctica (by_contra h) es la regla de reducción al absurdo; es
--    decir, si el objetivo es p añade la hipótesis (h : p) y reduce el
--    objetivo a False.
-- 2. La táctica (exact h1 ⟨x, h2⟩) es la regla de inntroducción del
--    cuantificador existencial; es decir, si el objetivo es de la forma
```

```
--    (∃y, P y) demuestra (P x) con h2 y unifica h1 con (∃x, P x).

-- 2ª demostración
-- ===============

example
  (h : ¬ ∀ x, P x)
  : ∃ x, ¬ P x :=
not_forall.mp h

-- 3ª demostración
-- ===============

example
  (h : ¬ ∀ x, P x)
  : ∃ x, ¬ P x :=
by aesop

-- ----------------------------------------------------------------------
-- Ejercicio 6. Demostrar que si
--    ∃ x, ¬ P x
-- entonces
--    ¬ ∀ x, P x
-- -----------------------------------------------------------------------

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

-- Supongamos que (∀x)P(x) y tenemos que demostrar una
-- contradicción. Por hipótesis, (∃x)¬P(x). Sea y tal que
-- ¬P(y). Entonces, como (∀x)P(x), se tiene que P(y) que es una
-- contradicción con ¬P(y).

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

-- 1ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by
  intro h1
  -- h1 : ∀ (x : α), P x
```

```
  -- ⊢ False
  rcases h with ⟨y, hy⟩
  -- y : α
  -- hy : ¬P y
  apply hy
  -- ⊢ P y
  exact (h1 y)

-- Comentarios:
-- 1. La táctica (intro h), cuando el objetivo es una negación, es la
--    regla de introducción de la negación; es decir, si el objetivo es
--    ¬P entonces añade la hipótesis (h : P) y cambia el objetivo a
--    false.
-- 2. La táctica (cases' h with x hx), cuando la hipótesis es un
--    existencial, es la regla de eliminación del existencial; es decir,
--    si h es (∃ (y : α), P y) añade las hipótesis (x : α) y (hx : P x).

-- 2ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by
  intro h1
  -- h1 : ∀ (x : α), P x
  -- ⊢ False
  rcases h with ⟨y, hy⟩
  -- y : α
  -- hy : ¬P y
  apply hy
  -- ⊢ P y
  exact (h1 y)

-- 3ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by
  intro h1
  -- h1 : ∀ (x : α), P x
  -- ⊢ False
  rcases h with ⟨y, hy⟩
```

```
  -- y : α
  -- hy : ¬P y
  exact hy (h1 y)

-- 4ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
not_forall.mpr h

-- 5ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
not_forall_of_exists_not h

-- 5ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by aesop

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

#check (not_exists : (¬∃ x, P x) ↔ ∀ (x : α), ¬P x)
#check (not_exists_of_forall_not : (∀ x, ¬ P x) → ¬ ∃ x, P x)
#check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
#check (not_forall_of_exists_not : (∃ x, ¬P x) → ¬∀ x, P x)
```

3.3.10. Doble negación

```
-- ----------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- + Importar la librería de tácticas.
-- + Declarar P como una variable proposicional.
-- ----------------------------------------------------------------------
```

```
import Mathlib.Tactic
variable (P : Prop)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--     ¬¬P
-- entonces
--    P
-- ----------------------------------------------------------------------

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

-- Por reducción al absurdo. Supongamos ¬P. Entonces, tenemos una
-- contradicción con la hipótesis (¬¬P).

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

-- 1ª demostración
-- ===============

example
  (h : ¬¬P)
  : P :=
by
  by_contra h1
  -- h1 : ¬P
  -- ⊢ False
  exact (h h1)

-- 2ª demostración
-- ===============

example
  (h : ¬¬P)
  : P :=
by_contra (fun h1 ↦ h h1)

-- 3ª demostración
-- ===============

example
  (h : ¬¬P)
```

```
  : P :=
-- not_not.mp h
of_not_not h

-- 4ª demostración
-- ===============

example
  (h : ¬¬P)
  : P :=
by tauto

-- Comentario: La táctica tauto demuestra las tautologís
-- proposionales.

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    P
-- entonces
--    ¬¬P
-- ----------------------------------------------------------------------

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

-- Supongamos ¬P. Entonces, tenemos una contradicción con la hipótesis
-- (P).

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

-- 1ª demostración
-- ===============

example
  (h : P)
  : ¬¬P :=
by
  intro h1
  -- h1 : ¬P
  -- ⊢ False
  exact (h1 h)

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

```
example
  (h : P)
  : ¬¬P :=
fun h1 ↦ h1 h

-- 3ª demostración
-- ===============

example
  (h : P)
  : ¬¬P :=
not_not_intro h

-- 4ª demostración
-- ===============

example
  (h : P)
  : ¬ ¬ P :=
by tauto

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

#check (not_not_intro : P → ¬¬P)
#check (of_not_not : ¬¬P → P)
```

3.3.11. CN no acotada superiormente

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sea f una función de ℝ en ℝ. Demostrar que si f no tiene
-- cota superior, entonces para cada a existe un x tal que f(x) > a.
-- ----------------------------------------------------------------------

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

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

-- Usaremos los siguientes lemas
--    ¬(∃x)P(x) → (∀x)¬P(x)                                         (L1)
```

```
--    ¬a > b → a ≤ b                                                  (L2)
--
-- Sea a ∈ ℝ. Tenemos que demostrar que
--    (∃x)[f(x) > a]
-- Lo haremos por reducción al absurdo. Para ello, suponemos que
--    ¬(∃x)[f(x) > a]                                                 (1)
-- y tenemos que obtener una contradicción. Aplicando L1 a (1) se tiene
--    (∀x)[¬ f(x) > a]
-- y, aplicando L2, se tiene
--    (∀x)[f(x) ≤ a]
-- Lo que significa que a es una cota superior de f y, por tanto f está
-- acotada superiormente, en cotradicción con la hipótesis.

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

-- Por la contrarecíproca, se supone que
--    ¬(∀a)(∃x)[f(x) > a]                                             (1)
-- y tenemos que demostrar que f está acotada superiormente.
--
-- Interiorizando la negación en (1) y simplificando, se tiene que
--    (∃a)(∀x)[f x ≤ a]
-- que es lo que teníamos que demostrar.

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

import Mathlib.Data.Real.Basic

def FnUb (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a
def FnHasUb (f : ℝ → ℝ) :=
  ∃ a, FnUb f a

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ¬FnHasUb f)
  : ∀ a, ∃ x, f x > a :=
by
  intro a
  -- a : ℝ
```

```
  -- ⊢ ∃ x, f x > a
  by_contra h1
  -- h1 : ¬∃ x, f x > a
  -- ⊢ False
  have h2 : ∀ x, ¬ f x > a :=
    forall_not_of_not_exists h1
  have h3 : ∀ x, f x ≤ a := by
    intro x
    -- x : ℝ
    -- ⊢ f x ≤ a
    have h3a : ¬ f x > a := h2 x
    show f x ≤ a
    exact le_of_not_gt h3a
  have h4 : FnUb f a := h3
  have h5 : ∃ b, FnUb f b := ⟨a, h4⟩
  have h6 : FnHasUb f := h5
  show False
  exact h h6

-- 2ª demostración
-- ===============

example
  (h : ¬FnHasUb f)
  : ∀ a, ∃ x, f x > a :=
by
  intro a
  -- a : ℝ
  -- ⊢ ∃ x, f x > a
  by_contra h1
  -- h1 : ¬∃ x, f x > a
  -- ⊢ False
  apply h
  -- ⊢ FnHasUb f
  use a
  -- ⊢ FnUb f a
  intro x
  -- x : ℝ
  -- ⊢ f x ≤ a
  apply le_of_not_gt
  -- ⊢ ¬f x > a
  intro h2
  -- h2 : f x > a
  -- ⊢ False
  apply h1
```

```
  -- ⊢ ∃ x, f x > a
  use x
  -- ⊢ f x > a

-- 3ª demostración
-- ===============

example
  (h : ¬FnHasUb f)
  : ∀ a, ∃ x, f x > a :=
by
  unfold FnHasUb at h
  -- h : ¬∃ a, FnUb f a
  unfold FnUb at h
  -- h : ¬∃ a, ∀ (x : ℝ), f x ≤ a
  push_neg at h
  -- ∀ (a : ℝ), ∃ x, f x > a
  exact h

-- 4ª demostración
-- ===============

example
  (h : ¬FnHasUb f)
  : ∀ a, ∃ x, f x > a :=
by
  simp only [FnHasUb, FnUb] at h
  -- h : ¬∃ a, ∀ (x : ℝ), f x ≤ a
  push_neg at h
  -- ∀ (a : ℝ), ∃ x, f x > a
  exact h

-- 5ª demostración
-- ===============

example
  (h : ¬FnHasUb f) :
  ∀ a, ∃ x, f x > a :=
by
  contrapose h
  -- h : ¬∀ (a : ℝ), ∃ x, f x > a
  -- ⊢ ¬¬FnHasUb f
  push_neg at *
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  -- ⊢ FnHasUb f
```

```
  exact h

-- 6ª demostración
-- ===============

example
  (h : ¬FnHasUb f) :
  ∀ a, ∃ x, f x > a :=
by
  contrapose! h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  -- ⊢ FnHasUb f
  exact h

-- Comentario: La táctica (contrapose! h) aplica el contrapositivo entre
-- la hipótesis h y el objetivo; es decir, si (h : P) y el objetivo es Q
-- entonces cambia la hipótesis a (h : ¬Q) el objetivo a ¬P aplicando
-- simplificaciones en ambos.

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

variable (a b : ℝ)
variable {α : Type _}
variable (P : α → Prop)
#check (forall_not_of_not_exists : (¬∃ x, P x) → ∀ x, ¬P x)
#check (le_of_not_gt : ¬a > b → a ≤ b)
```

3.3.12. CNS de acotada superiormente (uso de push_neg y simp only)

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- + Importar la teoría Definicion_de_funciones_acotadas
-- + Declarar f como una variable de ℝ en ℝ.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_funciones_acotadas
variable (f : ℝ → ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
```

```
--    ¬ ∀ a, ∃ x, f x > a
-- entonces f está acotada superiormente.
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que f es acotada superiormente; es decir, que
--    (∃a)(∀x)[f(x) ≤ a]
-- que es exactamente la fórmula obtenida interiorizando la negación en
-- la hipótesis.

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

-- 1ª demostración
-- ===============

example
  (h : ¬∀ a, ∃ x, f x > a)
  : FnHasUb f :=
by
  unfold FnHasUb
  -- ⊢ ∃ a, FnUb f a
  unfold FnUb
  -- ⊢ ∃ a, ∀ (x : ℝ), f x ≤ a
  push_neg at h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  exact h

-- 2ª demostración
-- ===============

example
  (h : ¬∀ a, ∃ x, f x > a)
  : FnHasUb f :=
by
  unfold FnHasUb FnUb
  -- ⊢ ∃ a, ∀ (x : ℝ), f x ≤ a
  push_neg at h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  exact h

-- 3ª demostración
-- ===============
```

```
example
  (h : ¬∀ a, ∃ x, f x > a)
  : FnHasUb f :=
by
  push_neg at h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  exact h

-- Comentario. La táctica (push_neg at h) interioriza las negaciones de
-- la hipótesis h.

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si f no tiene cota superior, entonces para
-- cada a existe un x tal que f(x) > a.
-- ----------------------------------------------------------------------

example
  (h : ¬FnHasUb f)
  : ∀ a, ∃ x, f x > a :=
by
  simp only [FnHasUb, FnUb] at h
  -- h : ¬∃ a, ∀ (x : ℝ), f x ≤ a
  push_neg at h
  -- ⊢ ∀ (a : ℝ), ∃ x, f x > a
  exact h

-- Comentario: La táctica (simp only [h₁, ..., hₙ] at h) simplifica la
-- hipótesis h usando sólo los lemas h₁, ..., hₙ.
```

3.3.13. CN de no monótona

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f no es monótona, entonces existen x, y
-- tales que x ≤ y y f(y) < f(x).
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas.
--    ¬(∀x)P(x) ↔ (∃ x)¬P(x)                                         (L1)
--    ¬(p → q) ↔ p ∧ ¬q                                               (L2)
```

```
--    (∀a, b ∈ ℝ)[¬b ≤ a → a < b]                                        (L3)
--
-- Por la definición de función monótona,
--    ¬(∀x)(∀y)[x ≤ y → f(x) ≤ f(y)]
-- Aplicando L1 se tiene
--    (∃x)¬(∀y)[x ≤ y → f(x) ≤ f(y)]
-- Sea a tal que
--    ¬(∀y)[a ≤ y → f(a) ≤ f(y)]
-- Aplicando L1 se tiene
--    (∃y)¬[a ≤ y → f(a) ≤ f(y)]
-- Sea b tal que
--    ¬[a ≤ b → f(a) ≤ f(b)]
-- Aplicando L2 se tiene que
--    a ≤ b ∧ ¬(f(a) ≤ f(b))
-- Aplicando L3 se tiene que
--    a ≤ b ∧ f(b) < f(a)
-- Por tanto,
--    (∃x,y)[x ≤ y ∧ f(y) < f(x)]

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

import Mathlib.Tactic
variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ¬Monotone f)
  : ∃ x y, x ≤ y ∧ f y < f x :=
by
  have h1 : ¬∀ x y, x ≤ y → f x ≤ f y := h
  have h2 : ∃ x, ¬(∀ y, x ≤ y → f x ≤ f y) := not_forall.mp h1
  rcases h2 with ⟨a, ha⟩
  -- a : ℝ
  -- ha : ¬∀ (y : ℝ), a ≤ y → f a ≤ f y
  have h3 : ∃ y, ¬(a ≤ y → f a ≤ f y) := not_forall.mp ha
  rcases h3 with ⟨b, hb⟩
  -- b : ℝ
  -- hb : ¬(a ≤ b → f a ≤ f b)
  have h4 : a ≤ b ∧ ¬(f a ≤ f b) := _root_.not_imp.mp hb
  have h5 : a ≤ b ∧ f b < f a := ⟨h4.1, lt_of_not_le h4.2⟩
  use a, b
  -- ⊢ a ≤ b ∧ f b < f a
```

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

example
  (h : ¬Monotone f)
  : ∃ x y, x ≤ y ∧ f y < f x :=
by
  simp only [Monotone] at h
  -- h : ¬∀ {a b : ℝ}, a ≤ b → f a ≤ f b
  push_neg at h
  -- h : Exists fun {a} => Exists fun {b} => a ≤ b ∧ f b < f a
  exact h

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

variable {α : Type _}
variable (P : α → Prop)
variable (p q : Prop)
variable (a b : ℝ)
#check (_root_.not_imp : ¬(p → q) ↔ p ∧ ¬q)
#check (lt_of_not_le : ¬b ≤ a → a < b)
#check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
```

3.3.14. Principio de explosión

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si 0 < 0, entonces a > 37 para cualquier
-- número a.
-- ----------------------------------------------------------------------

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

-- Basta demostrar una cotradicción, ya que de una contradicción se
-- sigue cualquier cosa.
--
-- La hipótesis es una contradicción con la propiedad irreflexiva de la
-- relación <.

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

```
import Mathlib.Tactic

variable (a : ℕ)

-- 1ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by
  exfalso
  -- ⊢ False
  show False
  exact lt_irrefl 0 h

-- Comentario: La táctica exfalso sustituye el objetivo por false.

-- 2ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by
  exfalso
  -- ⊢ False
  apply lt_irrefl 0 h

-- 3ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
absurd h (lt_irrefl 0)

-- 4ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by
```

```
  have : ¬ 0 < 0 :=  lt_irrefl 0
  contradiction

-- Comentario: La táctica contradiction busca dos hipótesis
-- contradictorias.

-- 5ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by linarith

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

variable (p q : Prop)
#check (absurd : p → ¬p → q)
#check (lt_irrefl a : ¬a < a)
```

3.4. Conjunción y bicondicional

3.4.1. Introducción de la conjunción

```
-- ----------------------------------------------------------------------
-- Ejercicio. Sean x e y dos números tales que
--    x ≤ y
--    ¬ y ≤ x
-- entonces
--    x ≤ y ∧ x ≠ y
-- -----------------------------------------------------------------------

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

-- Como la conclusión es una conjunción, tenemos que desmostrar sus dos
-- partes. La primera parte (x ≤ y) coincide con la hipótesis. Para
-- demostrar la segunda parte (x ≠ y), supongamos que x = y; entonces
-- y ≤ x en contradicción con la segunda hipótesis.

-- Demostraciones con Lean4
```

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

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

variable {x y : ℝ}

-- 1ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by
  constructor
  . -- ⊢ x ≤ y
    exact h1
  . -- ⊢ x ≠ y
    intro h3
    -- h3 : x = y
    -- ⊢ False
    have h4 : y ≤ x := h3.symm.le
    show False
    exact h2 h4

-- Comentario: La táctica constructor, cuando el objetivo es una
-- conjunción (P ∧ Q), aplica la regla de introducción de la conjunción;
-- es decir, sustituye el objetivo por dos nuevos subobjetivos (P y Q).

-- 2ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by
  constructor
  . -- ⊢ x ≤ y
    exact h1
  . -- ⊢ x ≠ y
    intro h3
    -- h3 : x = y
    -- ⊢ False
```

```
    exact h2 (h3.symm.le)

-- 3ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬y ≤ x)
  : x ≤ y ∧ x ≠ y :=
⟨h1, fun h3 ↦ h2 (h3.symm.le)⟩

-- Comentario: La notación ⟨h0, h1⟩, cuando el objetivo es una conjunción
-- (P ∧ Q), aplica la regla de introducción de la conjunción donde h0 es
-- una prueba de P y h1 de Q.

-- 4ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by
  constructor
  . -- ⊢ x ≤ y
    exact h1
  . -- ⊢ x ≠ y
    intro h3
    -- h3 : x = y
    -- ⊢ False
    apply h2
    -- ⊢ y ≤ x
    rw [h3]

-- 5ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by
  constructor
  . -- ⊢ x ≤ y
    exact h1
```

```
  . -- ⊢ x ≠ y
    intro h3
    -- h3 : x = y
    -- ⊢ False
    exact h2 (by rw [h3])

-- 6ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬ y ≤ x)
  : x ≤ y ∧ x ≠ y :=
⟨h1, fun h ↦ h2 (by rw [h])⟩

-- 7ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬ y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by
  have h3 : x ≠ y
  . contrapose! h2
    -- ⊢ y ≤ x
    rw [h2]
  exact ⟨h1, h3⟩

-- 8ª demostración
-- ===============

example
  (h1 : x ≤ y)
  (h2 : ¬ y ≤ x)
  : x ≤ y ∧ x ≠ y :=
by aesop
```

3.4.2. Eliminación de la conjunción

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los reales, si
--    x ≤ y ∧ x ≠ y
```

```
-- entonces
--    ¬ y ≤ x
-- ----------------------------------------------------------------------

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

-- Supongamos que y ≤ x. Entonces, por la antisimetría y la primera
-- parte de la hipótesis, se tiene que x = y que contradice la segunda
-- parte de la hipótesis.

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

import Mathlib.Data.Real.Basic

variable {x y : ℝ}

-- 1ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
  : ¬ y ≤ x :=
by
  intro h1
  rcases h with ⟨h2, h3⟩
  -- h2 : x ≤ y
  -- h3 : x ≠ y
  have h4 : x = y := le_antisymm h2 h1
  show False
  exact h3 h4

-- 2ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
  : ¬ y ≤ x :=
by
  intro h1
  -- h1 : y ≤ x
  -- ⊢ False
  have h4 : x = y := le_antisymm h.1 h1
  show False
```

```
  exact h.2 h4

-- 3ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
  : ¬ y ≤ x :=
by
  intro h1
  -- h1 : y ≤ x
  -- ⊢ False
  show False
  exact h.2 (le_antisymm h.1 h1)

-- 4ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
  : ¬ y ≤ x :=
fun h1 ↦ h.2 (le_antisymm h.1 h1)

-- 5ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
  : ¬ y ≤ x :=
by
  intro h'
  -- h' : y ≤ x
  -- ⊢ False
  apply h.right
  -- ⊢ x = y
  exact le_antisymm h.left h'

-- Comentario: Si h es una conjunción (P ∧ Q), entonces h.left es P y
-- h.right es Q.

-- 6ª demostración
-- ===============

example
  (h : x ≤ y ∧ x ≠ y)
```

```
  : ¬ y ≤ x :=
by
  cases' h with h1 h2
  -- h1 : x ≤ y
  -- h2 : x ≠ y
  contrapose! h2
  -- h2 : y ≤ x
  -- ⊢ x = y
  exact le_antisymm h1 h2

-- Comentario: La táctica (cases' h with h1 h2) si la hipótesis h es una
-- conjunción (P ∧ Q), aplica la regla de eliminación de la conjunción;
-- es decir, sustituye h por las hipótesis (h1 : P) y (h2 : Q).

-- 7ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → ¬ y ≤ x :=
by
  rintro ⟨h1, h2⟩ h'
  -- h1 : x ≤ y
  -- h2 : x ≠ y
  -- h' : y ≤ x
  -- ⊢ False
  exact h2 (le_antisymm h1 h')

-- Comentario: La táctica (rintro ⟨h1, h2⟩ h')
-- + si el objetivo es de la forma (P ∧ Q → (R → S)) añade las hipótesis
--   (h1 : P), (h2 : Q), (h' : R) y sustituye el objetivo por S.
-- + si el objetivo es de la forma (P ∧ Q → ¬R) añade las hipótesis
--   (h1 : P), (h2 : Q), (h' : R) y sustituye el objetivo por false.

-- 8ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → ¬ y ≤ x :=
fun ⟨h1, h2⟩ h' ↦ h2 (le_antisymm h1 h')

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

#check (le_antisymm : x ≤ y → y ≤ x → x = y)
```

3.4.3. Uso de conjunción

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sean m y n números naturales. Demostrar que si
--    m ∣ n ∧ m ≠ n
-- entonces
--    m ∣ n ∧ ¬(n ∣ m)
-- ----------------------------------------------------------------------

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

-- La primera parte de la conclusión coincide con la primera de la
-- hipótesis. Nos queda demostrar la segunda parte; es decir, que
-- ¬(n ∣ m). Para ello, supongamos que n ∣ m. Entonces, por la propiedad
-- antisimétrica de la divisibilidad y la primera parte de la hipótesis,
-- se tiene que m = n en contradicción con la segunda parte de la
-- hipótesis.

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

import Mathlib.Data.Nat.GCD.Basic

variable {m n : ℕ}

-- 1ª demostración
-- ===============

example
  (h : m ∣ n ∧ m ≠ n)
  : m ∣ n ∧ ¬ n ∣ m :=
by
  constructor
  . -- ⊢ m ∣ n
    exact h.left
  . -- ⊢ ¬n ∣ m
    intro h1
    -- h1 : n ∣ m
    have h2 : m = n := dvd_antisymm h.left h1
    show False
    exact h.right h2

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

```
example
  (h : m ∣ n ∧ m ≠ n)
  : m ∣ n ∧ ¬ n ∣ m :=
by
  constructor
  . -- ⊢ m ∣ n
    exact h.left
  . -- ⊢ ¬n ∣ m
    intro h1
    -- h1 : n ∣ m
    exact h.right (dvd_antisymm h.left h1)

-- 3ª demostración
-- ===============

example
  (h : m ∣ n ∧ m ≠ n)
  : m ∣ n ∧ ¬ n ∣ m :=
⟨h.left, fun h1 ↦ h.right (dvd_antisymm h.left h1)⟩

-- 4ª demostración
-- ===============

example
  (h : m ∣ n ∧ m ≠ n)
  : m ∣ n ∧ ¬ n ∣ m :=
by
  rcases h with ⟨h1, h2⟩
  -- h1 : m ∣ n
  -- h2 : m ≠ n
  constructor
  . -- ⊢ m ∣ n
    exact h1
  . -- ⊢ ¬n ∣ m
    contrapose! h2
    -- h2 : n ∣ m
    -- ⊢ m = n
    apply dvd_antisymm h1 h2

-- 5ª demostración
-- ===============

example
  (h : m ∣ n ∧ m ≠ n)
```

```
  : m ∣ n ∧ ¬ n ∣ m :=
by
  obtain ⟨h1, h2⟩ := h
  constructor
  . -- ⊢ m ∣ n
    exact h1
  . -- ⊢ ¬n ∣ m
    contrapose! h2
    -- h2 : n ∣ m
    -- ⊢ m = n
    apply dvd_antisymm h1 h2

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

#check (dvd_antisymm : m ∣ n → n ∣ m → m = n)
```

3.4.4. Existenciales y conjunciones anidadas

```
import Mathlib.Data.Real.Basic
import Mathlib.Data.Nat.Prime.Defs
import Mathlib.Tactic

variable (x y : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que (∃x ∈ ℝ)[2 < x < 3]
-- ---------------------------------------------------------------------

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

-- Podemos usar el número 5/2 y comprobar que 2 < 5/2 < 3.

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

-- 1ª demostración
-- ===============

example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  use 5 / 2
```

```
  show 2 < 5 / 2 ∧ 5 / 2 < 3
  constructor
  . show 2 < 5 / 2
    norm_num
  . show 5 / 2 < 3
    norm_num

-- 2ª demostración
-- ===============

example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  use 5 / 2
  constructor
  . norm_num
  . norm_num

-- 3ª demostración
-- ===============

example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
by
  use 5 / 2
  constructor <;> norm_num

-- 4ª demostración
-- ===============

example : ∃ x : ℝ, 2 < x ∧ x < 3 :=
⟨5/2, by norm_num⟩

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si (∃z ∈ ℝ)[x < z < y], entonces x < y.
-- ---------------------------------------------------------------------

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

-- Sea z tal que verifica las siguientes relaciones:
--    x < z                                                          (1)
--    z < y                                                          (2)
-- Aplicando la propiedad transitiva a (1) y (2) se tiene que
--    x < y.

-- Demostraciones con Lean4
```

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

-- 1ª demostración
-- ===============

example : (∃ z : ℝ, x < z ∧ z < y) → x < y :=
by
  rintro ⟨z, h1 : x < z, h2 : z < y⟩
  show x < y
  exact lt_trans h1 h2

-- 2ª demostración
-- ===============

example : (∃ z : ℝ, x < z ∧ z < y) → x < y :=
by
  rintro ⟨z, h1, h2⟩
  exact lt_trans h1 h2

-- 3ª demostración
-- ===============

example : (∃ z : ℝ, x < z ∧ z < y) → x < y :=
fun ⟨_, h1, h2⟩ ↦ lt_trans h1 h2

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que existen números primos m y n tales que
-- 4 < m < n < 10.
-- ---------------------------------------------------------------------

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

-- Basta considerar los números 5 y 7, ya que son primos y
-- 4 < 5 < 7 < 10.

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

example :
  ∃ m n : ℕ, 4 < m ∧ m < n ∧ n < 10 ∧ Nat.Prime m ∧ Nat.Prime n :=
by
  use 5, 7
  -- ⊢ 4 < 5 ∧ 5 < 7 ∧ 7 < 10 ∧ Nat.Prime 5 ∧ Nat.Prime 7
  norm_num
```

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que x ≤ y ∧ x ≠ y → x ≤ y ∧ y ≰ x
-- ---------------------------------------------------------------------

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

-- Supongamos que
--    x ≤ y                                                         (1)
--    x ≠ y                                                         (2)
-- Entonces, se tiene x ≤ y (por (1)) y, para probar y ≰ x, supongamos
-- que y ≤ x. Por (1), se tiene que x = y, en contradicción con (2).

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

-- 1ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1 : x ≤ y, h2 : x ≠ y⟩
  constructor
  . show x ≤ y
    exact h1
  . show ¬ y ≤ x
    rintro h3 : y ≤ x
    -- ⊢ False
    have h4 : x = y := le_antisymm h1 h3
    show False
    exact h2 h4

-- 2ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1 : x ≤ y, h2 : x ≠ y⟩
  -- ⊢ x ≤ y ∧ ¬y ≤ x
  constructor
  . show x ≤ y
    exact h1
  . show ¬ y ≤ x
    rintro h3 : y ≤ x
```

```
    -- ⊢ False
    show False
    exact h2 (le_antisymm h1 h3)

-- 3ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1 : x ≤ y, h2 : x ≠ y⟩
  constructor
  . show x ≤ y
    exact h1
  . show ¬ y ≤ x
    exact fun h3 ↦ h2 (le_antisymm h1 h3)

-- 4ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1, h2⟩
  exact ⟨h1, fun h3 ↦ h2 (le_antisymm h1 h3)⟩

-- 5ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
  fun ⟨h1, h2⟩ ↦ ⟨h1, fun h3 ↦ h2 (le_antisymm h1 h3)⟩

-- 6ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1 : x ≤ y, h2 : x ≠ y⟩
  use h1
  exact fun h3 ↦ h2 (le_antisymm h1 h3)

-- 7ª demostración
-- ===============

example : x ≤ y ∧ x ≠ y → x ≤ y ∧ ¬ y ≤ x :=
by
  rintro ⟨h1, h2⟩
```

```
  -- h1 : x ≤ y
  -- h2 : x ≠ y
  -- ⊢ x ≤ y ∧ ¬y ≤ x
  use h1
  -- ¬y ≤ x
  contrapose! h2
  -- h2 : y ≤ x
  -- ⊢ x = y
  apply le_antisymm h1 h2

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

variable (z : ℝ)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)
#check (lt_trans : x < y → y < z → x < z)
```

3.4.5. CNS de distintos

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sean x, y números reales tales que x ≤ y. Entonces,
-- ¬y ≤ x ↔ x ≠ y.
-- ---------------------------------------------------------------------

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

-- Para demostrar la equivalencia, demostraremos cada una de las
-- implicaciones.
--
-- Para demostrar la primera, supongamos que y ≰ x y que x =
-- y. Entonces, y ≤ x que es una contradicción.
--
-- Para demostrar la segunda, supongamos que x ≠ y y que y ≤
-- x. Entonces, por la hipótesis y la antisimetría, se tiene que x = y
-- lo que es una contradicción.

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

import Mathlib.Data.Real.Basic
variable {x y : ℝ}
```

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

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . -- ⊢ ¬y ≤ x → x ≠ y
    intro h1
    -- h1 : ¬y ≤ x
    -- ⊢ x ≠ y
    intro h2
    -- h2 : x = y
    -- ⊢ False
    have h3 : y ≤ x := by rw [h2]
    show False
    exact h1 h3
  . -- ⊢ x ≠ y → ¬y ≤ x
    intro h1
    -- h1 : x ≠ y
    -- ⊢ ¬y ≤ x
    intro h2
    -- h2 : y ≤ x
    -- ⊢ False
    have h3 : x = y := le_antisymm h h2
    show False
    exact h1 h3

-- 2ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . -- ⊢ ¬y ≤ x → x ≠ y
    intro h1
    -- h1 : ¬y ≤ x
    -- ⊢ x ≠ y
    intro h2
    -- h2 : x = y
    -- ⊢ False
    show False
```

```
    exact h1 (by rw [h2])
  . -- ⊢ x ≠ y → ¬y ≤ x
    intro h1
    -- h1 : x ≠ y
    -- ⊢ ¬y ≤ x
    intro h2
    -- h2 : y ≤ x
    -- ⊢ False
    show False
    exact h1 (le_antisymm h h2)

-- 3ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . -- ⊢ ¬y ≤ x → x ≠ y
    intro h1 h2
    -- h1 : ¬y ≤ x
    -- h2 : x = y
    -- ⊢ False
    exact h1 (by rw [h2])
  . -- ⊢ x ≠ y → ¬y ≤ x
    intro h1 h2
    -- h1 : x ≠ y
    -- h2 : y ≤ x
    -- ⊢ False
    exact h1 (le_antisymm h h2)

-- 4ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . -- ⊢ ¬y ≤ x → x ≠ y
    exact fun h1 h2 ↦ h1 (by rw [h2])
  . -- ⊢ x ≠ y → ¬y ≤ x
    exact fun h1 h2 ↦ h1 (le_antisymm h h2)
```

```
-- 5ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
  ⟨fun h1 h2 ↦ h1 (by rw [h2]),
   fun h1 h2 ↦ h1 (le_antisymm h h2)⟩

-- 6ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . -- ⊢ ¬y ≤ x → x ≠ y
    intro h1
    -- h1 : ¬y ≤ x
    -- ⊢ x ≠ y
    contrapose! h1
    -- h1 : x = y
    -- ⊢ y ≤ x
    calc y = x := h1.symm
         _ ≤ x := by rfl
  . -- ⊢ x ≠ y → ¬y ≤ x
    intro h2
    -- h2 : x ≠ y
    -- ⊢ ¬y ≤ x
    contrapose! h2
    -- h2 : y ≤ x
    -- ⊢ x = y
    show x = y
    exact le_antisymm h h2

-- 7ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  · -- ⊢ ¬y ≤ x → x ≠ y
```

```
    contrapose!
    -- ⊢ x = y → y ≤ x
    rintro rfl
    -- ⊢ x ≤ x
    rfl
  . -- ⊢ x ≠ y → ¬y ≤ x
    contrapose!
    -- ⊢ y ≤ x → x = y
    exact le_antisymm h

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

#check (le_antisymm : x ≤ y → y ≤ x → x = y)
```

3.4.6. Suma nula de dos cuadrados

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de los números reales.
-- 2. Declarar x e y como variables sobre los reales.
-- ----------------------------------------------------------------------

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

variable {x y : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    x^2 + y^2 = 0
-- entonces
--   x = 0
-- ----------------------------------------------------------------------

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

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

-- Se usarán los siguientes lemas
--    (∀ x ∈ ℝ)(∀ n ∈ ℕ)[x^n = 0 → x = 0]                        (L1)
```

```
--    (∀ x, y ∈ ℝ)[x ≤ y → y ≤ x → x = y]                              (L2)
--    (∀ x, y ∈ ℝ)[0 ≤ y → x ≤ x + y]                                  (L3)
--    (∀ x ∈ ℝ)[0 ≤ x²]                                                (L4)
--
-- Por el lema L1, basta demostrar
--    x² = 0                                                           (1)
-- y, por el lema L2, basta demostrar las siguientes desigualdades
--     x² ≤ 0                                                          (2)
--     0 ≤ x²                                                          (3)
--
-- La prueba de la (2) es
--    x² ≤ x² + y²   [por L3 y L4]
--       = 0         [por la hipótesis]
--
-- La (3) se tiene por el lema L4.

-- 1ª demostración
-- ===============

example
  (h : x^2 + y^2 = 0)
  : x = 0 :=
by
  have h' : x^2 = 0 := by
    apply le_antisymm
    . -- ⊢ x ^ 2 ≤ 0
      calc x ^ 2 ≤ x^2 + y^2 := by simp [le_add_of_nonneg_right,
                                         pow_two_nonneg]
               _ = 0         := by exact h
    . -- ⊢ 0 ≤ x ^ 2
      apply pow_two_nonneg
  show x = 0
  exact pow_eq_zero h'

-- 2ª demostración
-- ===============

lemma aux
  (h : x^2 + y^2 = 0)
  : x = 0 :=
  have h' : x ^ 2 = 0 := by linarith [pow_two_nonneg x, pow_two_nonneg y]
  pow_eq_zero h'

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
```

```
--    x^2 + y^2 = 0 ↔ x = 0 ∧ y = 0
-- ----------------------------------------------------------------------

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

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

-- Para la primera implicación, supongamos que
--    x² + y² = 0                                                   (1)
-- Entonces, por el lema anterior,
--    x = 0                                                         (2)
-- Además, aplicando la conmutativa a (1), se tiene
--    y² + x² = 0
-- y, por el lema anterior,
--    y = 0                                                         (3)
-- De (2) y (3) se tiene
--    x = 0 ∧ y = 0
--
-- Para la segunda implicación, supongamos que
--    x = 0 ∧ y = 0
-- Por tanto,
--    x² + y² = 0² + 0²
--            = 0

-- 1ª demostración
-- ===============

example : x^2 + y^2 = 0 ↔ x = 0 ∧ y = 0 :=
by
  constructor
  . -- ⊢ x ^ 2 + y ^ 2 = 0 → x = 0 ∧ y = 0
    intro h
    -- h : x ^ 2 + y ^ 2 = 0
    -- ⊢ x = 0 ∧ y = 0
    constructor
    . -- ⊢ x = 0
      exact aux h
    . -- ⊢ y = 0
      rw [add_comm] at h
      -- h : x ^ 2 + y ^ 2 = 0
      exact aux h
  . -- ⊢ x = 0 ∧ y = 0 → x ^ 2 + y ^ 2 = 0
    intro h1
```

```
    -- h1 : x = 0 ∧ y = 0
    -- ⊢ x ^ 2 + y ^ 2 = 0
    rcases h1 with ⟨h2, h3⟩
    -- h2 : x = 0
    -- h3 : y = 0
    rw [h2, h3]
    -- ⊢ 0 ^ 2 + 0 ^ 2 = 0
    norm_num

-- 2ª demostración
-- ===============

example : x^2 + y^2 = 0 ↔ x = 0 ∧ y = 0 :=
by
  constructor
  . -- ⊢ x ^ 2 + y ^ 2 = 0 → x = 0 ∧ y = 0
    intro h
    -- h : x ^ 2 + y ^ 2 = 0
    -- ⊢ x = 0 ∧ y = 0
    constructor
    . -- ⊢ x = 0
      exact aux h
    . -- ⊢ y = 0
      rw [add_comm] at h
      -- h : x ^ 2 + y ^ 2 = 0
      exact aux h
  . -- ⊢ x = 0 ∧ y = 0 → x ^ 2 + y ^ 2 = 0
    rintro ⟨h1, h2⟩
    -- h1 : x = 0
    -- h2 : y = 0
    -- ⊢ x ^ 2 + y ^ 2 = 0
    rw [h1, h2]
    -- ⊢ 0 ^ 2 + 0 ^ 2 = 0
    norm_num

-- 3ª demostración
-- ===============

example : x ^ 2 + y ^ 2 = 0 ↔ x = 0 ∧ y = 0 := by
  constructor
  · -- ⊢ x ^ 2 + y ^ 2 = 0 → x = 0 ∧ y = 0
    intro h
    -- h : x ^ 2 + y ^ 2 = 0
    -- ⊢ x = 0 ∧ y = 0
    constructor
```

```
    · -- x = 0
      exact aux h
    . -- ⊢ y = 0
      rw [add_comm] at h
      -- h : y ^ 2 + x ^ 2 = 0
      exact aux h
  . -- ⊢ x = 0 ∧ y = 0 → x ^ 2 + y ^ 2 = 0
    rintro ⟨rfl, rfl⟩
    -- ⊢ 0 ^ 2 + 0 ^ 2 = 0
    norm_num

-- Comentario: La táctica constructor, si el objetivo es un bicondicional
-- (P ↔ Q), aplica la introducción del bicondicional; es decir, lo
-- sustituye por dos nuevos objetivos: P → Q y Q → P.

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

variable (n : ℕ)
#check (add_comm x y : x + y = y + x)
#check (le_add_of_nonneg_right : 0 ≤ y → x ≤ x + y)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)
#check (pow_eq_zero : x ^ n = 0 → x = 0)
#check (pow_two_nonneg x : 0 ≤ x ^ 2)
```

3.4.7. Acotación del valor absoluto

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    |x + 3| < 5
-- entonces
--    -8 < x < 2
-- ----------------------------------------------------------------------

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

-- Supongamos que
--    |x + 3| < 5
-- entonces
--    -5 < x + 3 ∧ x + 3 < 5
-- por tanto
--    -8 < x ∧ x < 2
```

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

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

variable (x y : ℝ)

-- 1ª demostración
-- ===============

example
  : |x + 3| < 5 → -8 < x ∧ x < 2 :=
by
  rw [abs_lt]
  -- ⊢ -5 < x + 3 ∧ x + 3 < 5 → -8 < x ∧ x < 2
  intro h
  -- h : -5 < x + 3 ∧ x + 3 < 5
  -- ⊢ -8 < x ∧ x < 2
  constructor
  . -- ⊢ -8 < x
    linarith
  . -- x < 2
    linarith

-- 2ª demostración
-- ===============

example
  : |x + 3| < 5 → -8 < x ∧ x < 2 :=
by
  rw [abs_lt]
  -- ⊢ -5 < x + 3 ∧ x + 3 < 5 → -8 < x ∧ x < 2
  intro h
  -- h : -5 < x + 3 ∧ x + 3 < 5
  -- ⊢ -8 < x ∧ x < 2
  constructor <;> linarith

-- Comentario: La composición (constructor <;> linarith) aplica constructor y a
-- continuación le aplica linarith a cada subojetivo.

-- 3ª demostración
-- ===============
```

```
example
  : |x + 3| < 5 → -8 < x ∧ x < 2 :=
by
  rw [abs_lt]
  -- ⊢ -5 < x + 3 ∧ x + 3 < 5 → -8 < x ∧ x < 2
  exact fun _ ↦ ⟨by linarith, by linarith⟩

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

#check (abs_lt: |x| < y ↔ -y < x ∧ x < y)
```

3.4.8. Divisor del mcd

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que 3 divide al máximo común divisor de 6 y 15.
-- ----------------------------------------------------------------------

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

-- Se usará el siguiente lema
--    (∀ k, m, n ∈ ℕ)[k | gcd m n ↔ k | m ∧ k | n]
--
-- Por el lema,
--    3 | gcd 6 15
-- se reduce a
--    3 | 6 ∧ 3 | 15
-- que se verifican fácilmente.

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

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

open Nat

-- 1ª demostración
-- ===============

example : 3 ∣ Nat.gcd 6 15 :=
by
```

```
  rw [dvd_gcd_iff]
  -- ⊢ 3 ∣ 6 ∧ 3 ∣ 15
  constructor
  . -- 3 ∣ 6
    norm_num
  . -- ⊢ 3 ∣ 15
    norm_num

-- 2ª demostración
-- ===============

example : 3 ∣ Nat.gcd 6 15 :=
by
  rw [dvd_gcd_iff]
  -- ⊢ 3 ∣ 6 ∧ 3 ∣ 15
  constructor <;> norm_num

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

variable (k m n : ℕ)
#check (dvd_gcd_iff : k ∣ Nat.gcd m n ↔ k ∣ m ∧ k ∣ n)
```

3.4.9. Funciones no monótonas

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de los números reales.
-- 2. Declarar f como una variable sobre las funciones de ℝ en ℝ.
-- ----------------------------------------------------------------------

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

variable {f : ℝ → ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que f es no monótona syss existen x e y tales
-- que x ≤ y y f(x) > f(y).
-- ----------------------------------------------------------------------

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

```
-- Por la siguiente cadena de equivalencias:
--    f es no monótona ↔ ¬(∀ x y)[x ≤ y → f(x) ≤ f(y)]
--                     ↔ (∃ x y)[x ≤ y ∧ f(x) > f(y)]

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

-- 1ª demostración
-- ===============

example :
  ¬Monotone f ↔ ∃ x y, x ≤ y ∧ f x > f y :=
calc
  ¬Monotone f
    ↔ ¬∀ x y, x ≤ y → f x ≤ f y := by rw [Monotone]
  _ ↔ ∃ x y, x ≤ y ∧ f y < f x  := by simp_all only [not_forall, not_le, exists_prop]
  _ ↔ ∃ x y, x ≤ y ∧ f x > f y  := by rfl

-- 2ª demostración
-- ===============

example :
  ¬Monotone f ↔ ∃ x y, x ≤ y ∧ f x > f y :=
calc
  ¬Monotone f
    ↔ ¬∀ x y, x ≤ y → f x ≤ f y := by rw [Monotone]
  _ ↔ ∃ x y, x ≤ y ∧ f x > f y  := by aesop

-- 3ª demostración
-- ===============

example :
  ¬Monotone f ↔ ∃ x y, x ≤ y ∧ f x > f y :=
by
  rw [Monotone]
  -- ⊢ (¬∀ {a b : ℝ}, a ≤ b → f a ≤ f b) ↔ ∃ x y, x ≤ y ∧ f x > f y
  push_neg
  -- ⊢ (Exists fun {a} => Exists fun {b} => a ≤ b ∧ f b < f a) ↔ ∃ x y, x ≤ y ∧ f x > f
  rfl

-- 4ª demostración
-- ===============

lemma not_Monotone_iff :
```

```
  ¬Monotone f ↔ ∃ x y, x ≤ y ∧ f x > f y :=
by
  rw [Monotone]
  -- ⊢ (¬∀ {a b : ℝ}, a ≤ b → f a ≤ f b) ↔ ∃ x y, x ≤ y ∧ f x > f y
  aesop

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que la función opuesta no es monótona.
-- ----------------------------------------------------------------------

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

-- Usando el lema del ejercicio anterior que afirma que una función f no
-- es monótona syss existen x e y tales que x ≤ y y f(x) > f(y), basta
-- demostrar que
--    (∃ x y)[x ≤ y ∧ -x > -y]
-- Basta elegir 2 y 3 ya que
--    2 ≤ 3 ∧ -2 > -3

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

example : ¬Monotone fun x : ℝ ↦ -x :=
by
  apply not_Monotone_iff.mpr
  -- ⊢ ∃ x y, x ≤ y ∧ -x > -y
  use 2, 3
  -- ⊢ 2 ≤ 3 ∧ -2 > -3
  norm_num

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

variable (a b : ℝ)
variable (P : ℝ → Prop)
variable (p q : Prop)
#check (exists_prop : (∃ (_ : p), q) ↔ p ∧ q)
#check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
#check (not_le : ¬a ≤ b ↔ b < a)
```

3.4.10. Caracterización de menor en órdenes parciales

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en un orden parcial
--     a < b ↔ a ≤ b ∧ a ≠ b
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ a, b)[a < b ↔ a ≤ b ∧ b ≰ a]                               (L1)
--    (∀ a, b)[a ≤ b → b ≤ a → a = b]                               (L2)
--
-- Por el lema L1, lo que tenemos que demostrar es
--    a ≤ b ∧ b ≰ a ↔ a ≤ b ∧ a ≠ b
-- Lo haremos demostrando las dos implicaciones.
--
-- (⇒) Supongamos que a ≤ b y b ≰ a. Tenemos que demostrar que
-- a ≠ b. Lo haremos por reducción al absurdo. Para ello, supongamos que
-- a = b. Entonces, b ≤ a que contradice a b ≰ a.
--
-- (⇐) Supongamos que a ≤ b y a ≠ b. Tenemos que demostrar que
-- b ≰ a. Lo haremos por reducción al absurdo. Para ello, supongamos que
-- b ≤ a. Entonces, junto con a ≤ b, se tiene que a = b que es una
-- contradicicción con a ≠ b.

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

import Mathlib.Tactic

variable {α : Type _} [PartialOrder α]
variable (a b : α)

-- 1ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  constructor
  . -- ⊢ a ≤ b ∧ ¬b ≤ a → a ≤ b ∧ a ≠ b
    rintro ⟨h1 : a ≤ b, h2 : ¬b ≤ a⟩
```

```
    -- ⊢ a ≤ b ∧ a ≠ b
    constructor
    . -- ⊢ a ≤ b
      exact h1
    . -- ⊢ a ≠ b
      rintro (h3 : a = b)
      -- ⊢ False
      have h4: b = a := h3.symm
      have h5: b ≤ a := le_of_eq h4
      show False
      exact h2 h5
  . -- ⊢ a ≤ b ∧ a ≠ b → a ≤ b ∧ ¬b ≤ a
    rintro ⟨h5 : a ≤ b , h6 : a ≠ b⟩
    -- ⊢ a ≤ b ∧ ¬b ≤ a
    constructor
    . -- ⊢ a ≤ b
      exact h5
    . -- ⊢ ¬b ≤ a
      rintro (h7 : b ≤ a)
      have h8 : a = b := le_antisymm h5 h7
      show False
      exact h6 h8

-- 2ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  constructor
  . -- ⊢ a ≤ b ∧ ¬b ≤ a → a ≤ b ∧ a ≠ b
    rintro ⟨h1 : a ≤ b, h2 : ¬b ≤ a⟩
    -- ⊢ a ≤ b ∧ a ≠ b
    constructor
    . -- ⊢ a ≤ b
      exact h1
    . -- ⊢ a ≠ b
      rintro (h3 : a = b)
      -- ⊢ False
      exact h2 (le_of_eq h3.symm)
  . -- ⊢ a ≤ b ∧ a ≠ b → a ≤ b ∧ ¬b ≤ a
    rintro ⟨h4 : a ≤ b , h5 : a ≠ b⟩
    -- ⊢ a ≤ b ∧ ¬b ≤ a
    constructor
```

```
    . -- ⊢ a ≤ b
      exact h4
    . -- ⊢ ¬b ≤ a
      rintro (h6 : b ≤ a)
      exact h5 (le_antisymm h4 h6)

-- 3ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  constructor
  . -- ⊢ a ≤ b ∧ ¬b ≤ a → a ≤ b ∧ a ≠ b
    rintro ⟨h1 : a ≤ b, h2 : ¬b ≤ a⟩
    -- ⊢ a ≤ b ∧ a ≠ b
    constructor
    . -- ⊢ a ≤ b
      exact h1
    . -- ⊢ a ≠ b
      exact fun h3 ↦ h2 (le_of_eq h3.symm)
  . -- ⊢ a ≤ b ∧ a ≠ b → a ≤ b ∧ ¬b ≤ a
    rintro ⟨h4 : a ≤ b , h5 : a ≠ b⟩
    -- ⊢ a ≤ b ∧ ¬b ≤ a
    constructor
    . -- ⊢ a ≤ b
      exact h4
    . -- ⊢ ¬b ≤ a
      exact fun h6 ↦ h5 (le_antisymm h4 h6)

-- 4ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  constructor
  . -- ⊢ a ≤ b ∧ ¬b ≤ a → a ≤ b ∧ a ≠ b
    rintro ⟨h1 : a ≤ b, h2 : ¬b ≤ a⟩
    -- ⊢ a ≤ b ∧ a ≠ b
    exact ⟨h1, fun h3 ↦ h2 (le_of_eq h3.symm)⟩
  . -- ⊢ a ≤ b ∧ a ≠ b → a ≤ b ∧ ¬b ≤ a
    rintro ⟨h4 : a ≤ b , h5 : a ≠ b⟩
```

```
    -- ⊢ a ≤ b ∧ ¬b ≤ a
    exact ⟨h4, fun h6 ↦ h5 (le_antisymm h4 h6)⟩

-- 5ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  constructor
  . -- ⊢ a ≤ b ∧ ¬b ≤ a → a ≤ b ∧ a ≠ b
    exact fun ⟨h1, h2⟩ ↦ ⟨h1, fun h3 ↦ h2 (le_of_eq h3.symm)⟩
  . -- ⊢ a ≤ b ∧ a ≠ b → a ≤ b ∧ ¬b ≤ a
    exact fun ⟨h4, h5⟩ ↦ ⟨h4, fun h6 ↦ h5 (le_antisymm h4 h6)⟩

-- 6ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a ↔ a ≤ b ∧ a ≠ b
  exact ⟨fun ⟨h1, h2⟩ ↦ ⟨h1, fun h3 ↦ h2 (le_of_eq h3.symm)⟩,
         fun ⟨h4, h5⟩ ↦ ⟨h4, fun h6 ↦ h5 (le_antisymm h4 h6)⟩⟩

-- 7ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
by
  constructor
  . -- ⊢ a < b → a ≤ b ∧ a ≠ b
    intro h
    -- h : a < b
    -- ⊢ a ≤ b ∧ a ≠ b
    constructor
    . -- ⊢ a ≤ b
      exact le_of_lt h
    . -- ⊢ a ≠ b
      exact ne_of_lt h
  . -- ⊢ a ≤ b ∧ a ≠ b → a < b
    rintro ⟨h1, h2⟩
    -- h1 : a ≤ b
    -- h2 : a ≠ b
```

```
    -- ⊢ a < b
    exact lt_of_le_of_ne h1 h2

-- 8ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
  ⟨fun h ↦ ⟨le_of_lt h, ne_of_lt h⟩,
   fun ⟨h1, h2⟩ ↦ lt_of_le_of_ne h1 h2⟩

-- 9ª demostración
-- ===============

example : a < b ↔ a ≤ b ∧ a ≠ b :=
  lt_iff_le_and_ne

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

#check (le_antisymm : a ≤ b → b ≤ a → a = b)
#check (le_of_eq : a = b → a ≤ b)
#check (le_of_eq : a = b → a ≤ b)
#check (le_of_lt : a < b → a ≤ b)
#check (lt_iff_le_and_ne : a < b ↔ a ≤ b ∧ a ≠ b)
#check (lt_iff_le_not_le : a < b ↔ a ≤ b ∧ ¬b ≤ a)
#check (lt_of_le_of_ne : a ≤ b → a ≠ b → a < b)
#check (ne_of_lt : a < b → a ≠ b)
```

3.4.11. Irreflexiva y transitiva de menor en preórdenes

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar α como una variables sobre preórdenes.
-- 3. Declarar a, b y c como variables sobre elementos de α.
-- ----------------------------------------------------------------------

import Mathlib.Tactic
variable {α : Type _} [Preorder α]
variable (a b c : α)

-- ---------------------------------------------------------------------
-- Ejercicio 1. Demostrar que que la relación menor es irreflexiva.
```

```
-- ----------------------------------------------------------------------

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

-- Se usará la siguiente propiedad de lo preórdenes
--    (∀ a, b)[a < b ↔ a ≤ b ∧ b ≰ a]
-- Con dicha propiedad, lo que tenemos que demostrar se transforma en
--    ¬(a ≤ a ∧ a ≰ a)
-- Para demostrarla, supongamos que
--    a ≤ a ∧ a ≰ a
-- lo que es una contradicción.

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

-- 1ª demostración
-- ===============

example : ¬a < a :=
by
  rw [lt_iff_le_not_le]
  -- ⊢ ¬(a ≤ a ∧ ¬a ≤ a)
  rintro ⟨h1, h2⟩
  -- h1 : a ≤ a
  -- h2 : ¬a ≤ a
  -- ⊢ False
  exact h2 h1

-- 2ª demostración
-- ===============

example : ¬a < a :=
  irrefl a

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que que la relación menor es transitiva.
-- ----------------------------------------------------------------------

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

-- Se usará la siguiente propiedad de los preórdenes
--    (∀ a, b)[a < b ↔ a ≤ b ∧ b ≰ a]
-- Con dicha propiedad, lo que tenemos que demostrar se transforma en
```

```
--    a ≤ b ∧ b ≰ a → b ≤ c ∧ c ≰ b → a ≤ c ∧ c ≰ a
-- Para demostrarla, supongamos que
--    a ≤ b                                                    (1)
--    b ≰ a                                                    (2)
--    b ≤ c                                                    (3)
--    c ≰ b                                                    (4)
-- y tenemos que demostrar las siguientes relaciones
--    a ≤ c                                                    (5)
--    c ≰ a                                                    (6)
--
-- La (5) se tiene aplicando la propiedad transitiva a (1) y (3).
--
-- Para demostrar la (6), supongamos que
--    c ≤ a                                                    (7)
-- entonces, junto a la (1), por la propieda transitiva se tiene
--    c ≤ b
-- que es una contradicción con la (4).

-- 1ª demostración
-- ===============

example : a < b → b < c → a < c :=
by
  simp only [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a → b ≤ c ∧ ¬c ≤ b → a ≤ c ∧ ¬c ≤ a
  rintro ⟨h1 : a ≤ b, _h2 : ¬b ≤ a⟩ ⟨h3 : b ≤ c, h4 : ¬c ≤ b⟩
  -- ⊢ a ≤ c ∧ ¬c ≤ a
  constructor
  . -- ⊢ a ≤ c
    exact le_trans h1 h3
  . -- ⊢ ¬c ≤ a
    contrapose! h4
    -- h4 : c ≤ a
    -- ⊢ c ≤ b
    exact le_trans h4 h1

-- 2ª demostración
-- ===============

example : a < b → b < c → a < c :=
by
  simp only [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a → b ≤ c ∧ ¬c ≤ b → a ≤ c ∧ ¬c ≤ a
  rintro ⟨h1 : a ≤ b, _h2 : ¬b ≤ a⟩ ⟨h3 : b ≤ c, h4 : ¬c ≤ b⟩
  -- ⊢ a ≤ c ∧ ¬c ≤ a
```

```
  constructor
  . -- ⊢ a ≤ c
    exact le_trans h1 h3
  . -- ⊢ ¬c ≤ a
    rintro (h5 : c ≤ a)
    -- ⊢ False
    have h6 : c ≤ b := le_trans h5 h1
    show False
    exact h4 h6

-- 3ª demostración
-- ===============

example : a < b → b < c → a < c :=
by
  simp only [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a → b ≤ c ∧ ¬c ≤ b → a ≤ c ∧ ¬c ≤ a
  rintro ⟨h1 : a ≤ b, _h2 : ¬b ≤ a⟩ ⟨h3 : b ≤ c, h4 : ¬c ≤ b⟩
  -- ⊢ a ≤ c ∧ ¬c ≤ a
  constructor
  . -- ⊢ a ≤ c
    exact le_trans h1 h3
  . -- ⊢ ¬c ≤ a
    exact fun h5 ↦ h4 (le_trans h5 h1)

-- 4ª demostración
-- ===============

example : a < b → b < c → a < c :=
by
  simp only [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a → b ≤ c ∧ ¬c ≤ b → a ≤ c ∧ ¬c ≤ a
  rintro ⟨h1 : a ≤ b, _h2 : ¬b ≤ a⟩ ⟨h3 : b ≤ c, h4 : ¬c ≤ b⟩
  -- ⊢ a ≤ c ∧ ¬c ≤ a
  exact ⟨le_trans h1 h3, fun h5 ↦ h4 (le_trans h5 h1)⟩

-- 5ª demostración
-- ===============

example : a < b → b < c → a < c :=
by
  simp only [lt_iff_le_not_le]
  -- ⊢ a ≤ b ∧ ¬b ≤ a → b ≤ c ∧ ¬c ≤ b → a ≤ c ∧ ¬c ≤ a
  exact fun ⟨h1, _h2⟩ ⟨h3, h4⟩ ↦ ⟨le_trans h1 h3,
                                  fun h5 ↦ h4 (le_trans h5 h1)⟩
```

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

example : a < b → b < c → a < c :=
  lt_trans

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

#check (irrefl a : ¬a < a)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
#check (lt_iff_le_not_le : a < b ↔ a ≤ b ∧ ¬b ≤ a)
#check (lt_trans : a < b → b < c → a < c)
```

3.5. Disyunción

3.5.1. Introducción de la disyunción (Tácticas left / right y lemas or.inl y or.inr)

```
-- ----------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de los números naturales.
-- 2. Declarar x e y como variables sobre los reales.
-- ----------------------------------------------------------------------

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

variable {x y : ℝ}

-- ----------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    y > x^2
-- entonces
--    y > 0 ∨ y < -1
-- ----------------------------------------------------------------------

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

-- Puesto que
```

```
--    (∀ x ∈ ℝ)[x² ≥ 0]
-- se tiene que
--    y > x²
--      ≥ 0
-- Por tanto, y > 0 y, al verificar la primera parte de la diyunción, se
-- verifica la disyunción.

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

-- 1ª demostración
-- ===============

example
  (h : y > x^2)
  : y > 0 ∨ y < -1 :=
by
  have h1 : y > 0 := by
    calc y > x^2 := h
         _ ≥ 0   := pow_two_nonneg x
  show y > 0 ∨ y < -1
  exact Or.inl h1

-- 2ª demostración
-- ===============

example
  (h : y > x^2)
  : y > 0 ∨ y < -1 :=
by
  left
  -- ⊢ y > 0
  calc y > x^2 := h
       _ ≥ 0   := pow_two_nonneg x

-- 3ª demostración
-- ===============

example
  (h : y > x^2)
  : y > 0 ∨ y < -1 :=
by
  left
  -- ⊢ y > 0
  linarith [pow_two_nonneg x]
```

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

example
  (h : y > x^2)
  : y > 0 ∨ y < -1 :=
by { left ; linarith [pow_two_nonneg x] }

-- Comentario: La táctica left, si el objetivo es una disjunción
-- (P ∨ Q), aplica la regla de introducción de la disyunción; es decir,
-- cambia el objetivo por P.

-- ----------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    -y > x^2 + 1
-- entonces
--    y > 0 ∨ y < -1
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ b, c ∈ ℝ)[b ≤ c → ∀ (a : ℝ),  b + a ≤ c + a)]              (L1)
--    (∀ a ∈ ℝ)[0 ≤ a^2]                                             (L2)
--    (∀ a  ∈ ℝ)[0 + a = a]                                          (L3)
--    (∀ a, b ∈ ℝ)[a < -b ↔ b < -a]                                  (L4)
--
-- Se tiene
--    -y > x^2 + 1    [por la hipótesis]
--       ≥ 0 + 1      [por L1 y L2]
--       = 1          [por L3]
-- Por tanto,
--    -y > 1
-- y, aplicando el lema L4, se tiene
--    y < -1
-- Como se verifica la segunda parte de la diyunción, se verifica la
-- disyunción.

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

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

```
example
  (h : -y > x^2 + 1)
  : y > 0 ∨ y < -1 :=
by
  have h1 : -y > 1 := by
    calc -y > x^2 + 1 := by exact h
          _ ≥ 0 + 1   := add_le_add_right (pow_two_nonneg x) 1
          _ = 1       := zero_add 1
  have h2: y < -1 := lt_neg.mp h1
  show y > 0 ∨ y < -1
  exact Or.inr h2

-- 2ª demostración
-- ===============

example
  (h : -y > x^2 + 1)
  : y > 0 ∨ y < -1 :=
by
  have h1 : -y > 1 := by linarith [pow_two_nonneg x]
  have h2: y < -1 := lt_neg.mp h1
  show y > 0 ∨ y < -1
  exact Or.inr h2

-- 3ª demostración
-- ===============

example
  (h : -y > x^2 + 1)
  : y > 0 ∨ y < -1 :=
by
  have h1: y < -1 := by linarith [pow_two_nonneg x]
  show y > 0 ∨ y < -1
  exact Or.inr h1

-- 4ª demostración
-- ===============

example
  (h : -y > x^2 + 1)
  : y > 0 ∨ y < -1 :=
by
  right
  -- ⊢ y < -1
```

```
  linarith [pow_two_nonneg x]

-- 5ª demostración
-- ===============

example
  (h : -y > x^2 + 1)
  : y > 0 ∨ y < -1 :=
by { right ; linarith [pow_two_nonneg x] }

-- Comentario: La táctica right, si el objetivo es una disjunción
-- (P ∨ Q), aplica la regla de introducción de la disyunción; es decir,
-- cambia el objetivo por Q.

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que si
--      y > 0
-- entonces
--      y > 0 ∨ y < -1
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (h : y > 0)
  : y > 0 ∨ y < -1 :=
by
  left
  -- ⊢ y > 0
  exact h

-- 2ª demostración
-- ===============

example
  (h : y > 0)
  : y > 0 ∨ y < -1 :=
Or.inl h

-- 3ª demostración
-- ===============

example
  (h : y > 0)
```

```
  : y > 0 ∨ y < -1 :=
by tauto

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que si
--    y < -1
-- entonces
--    y > 0 ∨ y < -1
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (h : y < -1)
  : y > 0 ∨ y < -1 :=
by
  right
  -- y < -1
  exact h

-- 2ª demostración
-- ===============

example
  (h : y < -1)
  : y > 0 ∨ y < -1 :=
Or.inr h

-- 3ª demostración
-- ===============

example
  (h : y < -1)
  : y > 0 ∨ y < -1 :=
by tauto

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

variable (a b : Prop)
variable (z : ℝ)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (add_le_add_right : y ≤ z → ∀ x,  y + x ≤ z + x)
```

```
#check (lt_neg : x < -y ↔ y < -x)
#check (pow_two_nonneg x : 0 ≤ x ^ 2)
#check (zero_add x :  0 + x = x)
```

3.5.2. Eliminación de la disyunción (Táctica cases)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que para todo par de números reales x e y, si
-- x < |y|, entonces x < y ó x < -y.
-- ----------------------------------------------------------------------

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

-- Se demostrará por casos según y ≥ 0.
--
-- Primer caso: Supongamos que y ≥ 0. Entonces, |y| = y. Por tanto,
-- x < y.
--
-- Segundo caso: Supongamos que y < 0. Entonces, |y| = -y. Por tanto,
-- x < -y.

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

import Mathlib.Data.Real.Basic
variable {x y : ℝ}

-- 1ª demostración
-- ===============

example : x < |y| → x < y ∨ x < -y :=
by
  intro h1
  -- h1 : x < |y|
  -- ⊢ x < y ∨ x < -y
  rcases le_or_gt 0 y with h2 | h3
  . -- h2 : 0 ≤ y
    left
    -- ⊢ x < y
    rwa [abs_of_nonneg h2] at h1
  . -- h3 : 0 > y
    right
```

```
    -- ⊢ x < -y
    rwa [abs_of_neg h3] at h1

-- 2ª demostración
-- ===============

example : x < |y| → x < y ∨ x < -y :=
lt_abs.mp

-- Comentario:
-- + La táctica (rcases h with h1 | h2), cuando h es una diyunción, aplica
--   la regla de eliminación de la disyunción; es decir, si h es (P ∨ Q)
--   abre dos casos, en el primero añade la hipótesis (h1 : P) y en el
--   segundo (h2 : Q).

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

#check (abs_of_neg : x < 0 → abs x = -x)
#check (abs_of_nonneg : 0 ≤ x → abs x = x)
#check (le_or_gt x y : x ≤ y ∨ x > y)
#check (lt_abs : x < |y| ↔ x < y ∨ x < -y)
```

3.5.3. Desigualdad triangular para valor absoluto

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la librería de los números reales.
-- 2. Declarar x, y, a y b como variables sobre los reales.
-- 3. Crear el espacio de nombres my_abs.
-- ----------------------------------------------------------------------

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

variable {x y a b : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    x ≤ |x|
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

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

-- Se usarán los siguientes lemas
--    (∀ x ∈ ℝ)[0 ≤ x → |x| = x]                                    (L1)
--    (∀ x, y ∈ ℝ)[x < y → x ≤ y]                                   (L2)
--    (∀ x ∈ ℝ)[x ≤ 0 → x ≤ -x]                                     (L3)
--    (∀ x ∈ ℝ)[x < 0 → |x| = -x]                                   (L4)
--
-- Se demostrará por casos según x ≥ 0:
--
-- Primer caso: Supongamos que x ≥ 0. Entonces,
--    x ≤ x
--      = |x|    [por L1]
--
-- Segundo caso: Supongamos que x < 0. Entonces, por el L2, se tiene
--    x ≤ 0                                                         (1)
-- Por tanto,
--    x ≤ -x     [por L3 y (1)]
--      = |x|    [por L4]

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

-- 1ª demostración
-- ===============

example : x ≤ |x| :=
by
  rcases le_or_gt 0 x with h1 | h2
  . -- h1 : 0 ≤ x
    show x ≤ |x|
    calc x ≤ x   := le_refl x
         _ = |x| := (abs_of_nonneg h1).symm
  . -- h2 : 0 > x
    have h3 : x ≤ 0 := le_of_lt h2
    show x ≤ |x|
    calc x ≤ -x  := le_neg_self_iff.mpr h3
         _ = |x| := (abs_of_neg h2).symm

-- 2ª demostración
-- ===============

example : x ≤ |x| :=
by
  rcases le_or_gt 0 x with h1 | h2
```

```
  . -- h1 : 0 ≤ x
    rw [abs_of_nonneg h1]
  . -- h2 : 0 > x
    rw [abs_of_neg h2]
    -- ⊢ x ≤ -x
    apply Left.self_le_neg
    -- ⊢ x ≤ 0
    exact le_of_lt h2

-- 3ª demostración
-- ===============

example : x ≤ |x| :=
by
  rcases (le_or_gt 0 x) with h1 | h2
  . -- h1 : 0 ≤ x
    rw [abs_of_nonneg h1]
  . -- h1 : 0 ≤ x
    rw [abs_of_neg h2]
    linarith

-- 4ª demostración
-- ===============

example : x ≤ |x| :=
  le_abs_self x

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    -x ≤ |x|
-- ----------------------------------------------------------------------

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

-- Se usarán los siguientes lemas
--    (∀ x ∈ ℝ)[0 ≤ x → -x ≤ x]                                      (L1)
--    (∀ x ∈ ℝ)[0 ≤ x → |x| = x]                                     (L2)
--    (∀ x ∈ ℝ)[x ≤ x]                                               (L3)
--    (∀ x ∈ ℝ)[x < 0) → |x| = -x]                                   (L4)
--
-- Se demostrará por casos según x ≥ 0:
--
-- Primer caso: Supongamos que x ≥ 0. Entonces,
--    -x ≤ x      [por L1]
```

```
--        = |x|     [por L2]
--
-- Segundo caso: Supongamos que x < 0. Entonces,
--     -x ≤ -x      [por L3]
--      _ = |x|     [por L4]

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

-- 1ª demostración
-- ===============

example : -x ≤ |x| :=
by
  rcases (le_or_gt 0 x) with h1 | h2
  . -- h1 : 0 ≤ x
    show -x ≤ |x|
    calc -x ≤ x   := by exact neg_le_self h1
          _ = |x| := (abs_of_nonneg h1).symm
  . -- h2 : 0 > x
    show -x ≤ |x|
    calc -x ≤ -x  := by exact le_refl (-x)
          _ = |x| := (abs_of_neg h2).symm

-- 2ª demostración
-- ===============

example : -x ≤ |x| :=
by
  rcases (le_or_gt 0 x) with h1 | h2
  . -- h1 : 0 ≤ x
    rw [abs_of_nonneg h1]
    -- ⊢ -x ≤ x
    exact neg_le_self h1
  . -- h2 : 0 > x
    rw [abs_of_neg h2]

-- 3ª demostración
-- ===============

example : -x ≤ |x| :=
by
  rcases (le_or_gt 0 x) with h1 | h2
  . -- h1 : 0 ≤ x
    rw [abs_of_nonneg h1]
```

```
    -- ⊢ -x ≤ x
    linarith
  . -- h2 : 0 > x
    rw [abs_of_neg h2]

-- 4ª demostración
-- ===============

example : -x ≤ |x| :=
  neg_le_abs x

-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    |x + y| ≤ |x| + |y|
-- ----------------------------------------------------------------------

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

-- Se usarán los siguientes lemas
--     (∀ x ∈ ℝ)[0 ≤ x → |x| = x)                          (L1)
--     (∀ a, b, c, d ∈ ℝ)[a ≤ b ∧ c ≤ d → a + c ≤ b + d    (L2)
--     (∀ x ∈ ℝ)[x ≤ |x|]                                  (L3)
--     (∀ x ∈ ℝ)[x < 0 → |x| = -x]                         (L4)
--     (∀ x, y ∈ ℝ)[-(x + y) = -x + -y]                    (L5)
--     (∀ x ∈ ℝ)[-x ≤ |x|]                                 (L6)
--
-- Se demostrará por casos según x + y ≥ 0:
--
-- Primer caso: Supongamos que x + y ≥ 0. Entonces,
--    |x + y| = x + y        [por L1]
--          _ ≤ |x| + |y|    [por L2 y L3]
--
-- Segundo caso: Supongamos que x + y < 0. Entonces,
--    |x + y| = -(x + y)     [por L4]
--            = -x + -y      [por L5]
--            ≤ |x| + |y|    [por L2 y L6]

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

-- 1ª demostración
-- ===============

example : |x + y| ≤ |x| + |y| := by
```

```
  rcases le_or_gt 0 (x + y) with h1 | h2
  · -- h1 : 0 ≤ x + y
    show |x + y| ≤ |x| + |y|
    calc |x + y| = x + y     := by exact abs_of_nonneg h1
               _ ≤ |x| + |y| := add_le_add (le_abs_self x) (le_abs_self y)
  . -- h2 : 0 > x + y
    show |x + y| ≤ |x| + |y|
    calc |x + y| = -(x + y)  := by exact abs_of_neg h2
               _ = -x + -y   := by exact neg_add x y
               _ ≤ |x| + |y| := add_le_add (neg_le_abs x) (neg_le_abs y)

-- 2ª demostración
-- ===============

example : |x + y| ≤ |x| + |y| := by
  rcases le_or_gt 0 (x + y) with h1 | h2
  · -- h1 : 0 ≤ x + y
    rw [abs_of_nonneg h1]
    -- ⊢ x + y ≤ |x| + |y|
    exact add_le_add (le_abs_self x) (le_abs_self y)
  . -- h2 : 0 > x + y
    rw [abs_of_neg h2]
    -- ⊢ -(x + y) ≤ |x| + |y|
    calc -(x + y) = -x + -y    := by exact neg_add x y
                _ ≤ |x| + |y|  := add_le_add (neg_le_abs x) (neg_le_abs y)

-- 2ª demostración
-- ===============

example : |x + y| ≤ |x| + |y| := by
  rcases le_or_gt 0 (x + y) with h1 | h2
  · -- h1 : 0 ≤ x + y
    rw [abs_of_nonneg h1]
    -- ⊢ x + y ≤ |x| + |y|
    linarith [le_abs_self x, le_abs_self y]
  . -- h2 : 0 > x + y
    rw [abs_of_neg h2]
    -- ⊢ -(x + y) ≤ |x| + |y|
    linarith [neg_le_abs x, neg_le_abs y]

-- 3ª demostración
-- ===============

example : |x + y| ≤ |x| + |y| :=
  abs_add x y
```

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

variable (c d : ℝ)
#check (Left.self_le_neg : x ≤ 0 → x ≤ -x)
#check (abs_add x y : |x + y| ≤ |x| + |y|)
#check (abs_of_neg : x < 0 → |x| = -x)
#check (abs_of_nonneg : 0 ≤ x → |x| = x)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (le_abs_self a : a ≤ |a|)
#check (le_neg_self_iff : x ≤ -x ↔ x ≤ 0)
#check (le_of_lt : x < y → x ≤ y)
#check (le_or_gt x y : x ≤ y ∨ x > y)
#check (le_refl a : a ≤ a)
#check (neg_add x y : -(x + y) = -x + -y)
#check (neg_le_abs x : -x ≤ |x|)
#check (neg_le_self : 0 ≤ x → -x ≤ x)
```

3.5.4. Cotas del valor absoluto

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de los números reales.
-- 2. Declarar x e y como variables sobre los reales.
-- 3. Iniciar el espacio de nombre my_abs.
-- ----------------------------------------------------------------------

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

variable {x y z : ℝ}

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    x < |y| ↔ x < y ∨ x < -y
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : x < |y| ↔ x < y ∨ x < -y := by
  rcases le_or_gt 0 y with h | h
```

```
  · -- h : 0 ≤ y
    rw [abs_of_nonneg h]
    -- ⊢ x < y ↔ x < y ∨ x < -y
    constructor
    · -- ⊢ x < y → x < y ∨ x < -y
      intro h'
      -- h' : x < y
      -- ⊢ x < y ∨ x < -y
      left
      -- ⊢ x < y
      exact h'
    . -- ⊢ x < y ∨ x < -y → x < y
      intro h'
      -- h' : x < y ∨ x < -y
      -- ⊢ x < y
      rcases h' with h' | h'
      · -- h' : x < y
        exact h'
      . -- h' : x < -y
        linarith
  . -- h : 0 > y
    rw [abs_of_neg h]
    -- ⊢ x < -y ↔ x < y ∨ x < -y
    constructor
    · -- ⊢ x < -y → x < y ∨ x < -y
      intro h'
      -- h' : x < -y
      -- ⊢ x < y ∨ x < -y
      right
      -- ⊢ x < -y
      exact h'
    . -- ⊢ x < y ∨ x < -y → x < -y
      intro h'
      -- h' : x < y ∨ x < -y
      -- ⊢ x < -y
      rcases h' with h' | h'
      · -- h' : x < y
        linarith
      . -- h' : x < -y
        exact h'

-- 2ª demostración
-- ===============

example : x < |y| ↔ x < y ∨ x < -y :=
```

```
by
  rw [abs_eq_max_neg]
  -- ⊢ x < max y (-y) ↔ x < y ∨ x < -y
  exact lt_max_iff

-- 3ª demostración
-- ===============

example : x < |y| ↔ x < y ∨ x < -y :=
  lt_max_iff

-- 4ª demostración
-- ===============

example : x < |y| ↔ x < y ∨ x < -y :=
  lt_abs

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    |x| < y ↔ - y < x ∧ x < y
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : |x| < y ↔ -y < x ∧ x < y := by
  rcases le_or_gt 0 x with h | h
  · -- h : 0 ≤ x
    rw [abs_of_nonneg h]
    -- ⊢ x < y ↔ -y < x ∧ x < y
    constructor
    · -- ⊢ x < y → -y < x ∧ x < y
      intro h'
      -- h' : x < y
      -- ⊢ -y < x ∧ x < y
      constructor
      · -- ⊢ -y < x
        linarith
      . -- ⊢ x < y
        exact h'
    . -- ⊢ -y < x ∧ x < y → x < y
      intro h'
      -- h' : -y < x ∧ x < y
      -- ⊢ x < y
      rcases h' with ⟨-, h2⟩
```

```
      -- h2 : x < y
      exact h2
  . -- h : 0 > x
    rw [abs_of_neg h]
    -- ⊢ -x < y ↔ -y < x ∧ x < y
    constructor
    · -- ⊢ -x < y → -y < x ∧ x < y
      intro h'
      -- h' : -x < y
      -- ⊢ -y < x ∧ x < y
      constructor
      · -- ⊢ -y < x
        linarith
      . -- ⊢ x < y
        linarith
    . -- ⊢ -y < x ∧ x < y → -x < y
      intro h'
      -- h' : -y < x ∧ x < y
      -- ⊢ -x < y
      linarith

-- 2ª demostración
-- ===============

example : |x| < y ↔ -y < x ∧ x < y :=
by
  rw [abs_eq_max_neg]
  -- ⊢ max x (-x) < y ↔ -y < x ∧ x < y
  constructor
  . -- ⊢ max x (-x) < y → -y < x ∧ x < y
    intro h1
    -- h1 : max x (-x) < y
    -- ⊢ -y < x ∧ x < y
    rw [max_lt_iff] at h1
    -- h1 : x < y ∧ -x < y
    rcases h1 with ⟨h2, h3⟩
    -- h2 : x < y
    -- h3 : -x < y
    constructor
    . -- ⊢ -y < x
      exact neg_lt.mp h3
    . -- ⊢ x < y
      exact h2
  . -- ⊢ -y < x ∧ x < y → max x (-x) < y
    intro h4
```

```
    -- h4 : -y < x ∧ x < y
    -- ⊢ max x (-x) < y
    apply max_lt_iff.mpr
    -- ⊢ x < y ∧ -x < y
    rcases h4 with ⟨h5, h6⟩
    -- h5 : -y < x
    -- h6 : x < y
    constructor
    . -- ⊢ x < y
      exact h6
    . -- ⊢ -x < y
      exact neg_lt.mp h5

-- 2ª demostración
-- ===============

example : |x| < y ↔ -y < x ∧ x < y :=
  abs_lt

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

#check (abs_eq_max_neg : |x| = max x (-x))
#check (abs_lt : |x| < y ↔ -y < x ∧ x < y)
#check (abs_of_neg : x < 0 → |x| = -x)
#check (abs_of_nonneg : 0 ≤ x → abs x = x)
#check (le_or_gt x y : x ≤ y ∨ x > y)
#check (lt_abs : x < |y| ↔ x < y ∨ x < -y)
#check (lt_max_iff : x < max y z ↔ x < y ∨ x < z)
#check (max_lt_iff : max x y < z ↔ x < z ∧ y < z)
#check (neg_lt : -x < y ↔ -y < x)
```

3.5.5. Eliminación de la disyunción con rcases

```
-- ---------------------------------------------------------------------
-- Ejercicio. Sea x un número real. Demostrar que si
--    x ≠ 0
-- entonces
--    x < 0 ∨ x > 0
-- ----------------------------------------------------------------------

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

```
-- Usando el siguiente lema
--    (∀ x y ∈ ℝ)[x < y ∨ x = y ∨ y < x]
-- se demuestra distinguiendo tres casos.
--
-- Caso 1: Supongamos que x < 0. Entonces, se verifica la disyunción ya
-- que se verifica su primera parte.
--
-- Caso 2: Supongamos que x = 0. Entonces, se tiene una contradicción
-- con la hipótesis.
--
-- Caso 3: Supongamos que x > 0. Entonces, se verifica la disyunción ya
-- que se verifica su segunda parte.

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

import Mathlib.Data.Real.Basic
variable {x : ℝ}

-- 1ª demostración
-- ===============

example
  (h : x ≠ 0)
  : x < 0 ∨ x > 0 :=
by
  rcases lt_trichotomy x 0 with hx1 | hx2 | hx3
  . -- hx1 : x < 0
    left
    -- ⊢ x < 0
    exact hx1
  . -- hx2 : x = 0
    contradiction
  . -- hx3 : 0 < x
    right
    -- ⊢ x > 0
    exact hx3

-- 2ª demostración
-- ===============

example
  (h : x ≠ 0)
  : x < 0 ∨ x > 0 :=
```

```
Ne.lt_or_lt h

-- 3ª demostración
-- ===============

example
  (h : x ≠ 0)
  : x < 0 ∨ x > 0 :=
by aesop

-- Comentarios:
-- 1. La táctica (rcases h with h1 | h2 | h3) si el objetivo es (P ∨ Q ∨ R)
--    crea tres casos añadiéndole al primero la hipótesis (h1 : P), al
--    segundo (h2 : Q) y al tercero (h3 : R).

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

variable (y : ℝ)
#check (Ne.lt_or_lt : x ≠ y → x < y ∨ y < x)
#check (lt_trichotomy x y : x < y ∨ x = y ∨ y < x)
```

3.5.6. CS de divisibilidad del producto

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si m divide a n o a k, entonces divide a
-- nk.
-- ----------------------------------------------------------------------

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

-- Se demuestra por casos.
--
-- Caso 1: Supongamos que m | n. Entonces, existe un a ∈ ℕ tal que
--    n = ma
-- Por tanto,
--    nk = (ma)k
--       = m(ak)
-- que es divisible por m.
--
-- Caso 2: Supongamos que m | k. Entonces, existe un b ∈ ℕ tal que
--    k = mb
```

```
-- Por tanto,
--    nk = n(mb)
--       = m(nb)
-- que es divisible por m.

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

import Mathlib.Tactic
variable {m n k : ℕ}

-- 1ª demostración
-- ===============

example
  (h : m ∣ n ∨ m ∣ k)
  : m ∣ n * k :=
by
  rcases h with h1 | h2
  . -- h1 : m ∣ n
    rcases h1 with ⟨a, ha⟩
    -- a : ℕ
    -- ha : n = m * a
    rw [ha]
    -- ⊢ m ∣ (m * a) * k
    rw [mul_assoc]
    -- ⊢ m ∣ m * (a * k)
    exact dvd_mul_right m (a * k)
  . -- h2 : m ∣ k
    rcases h2 with ⟨b, hb⟩
    -- b : ℕ
    -- hb : k = m * b
    rw [hb]
    -- ⊢ m ∣ n * (m * b)
    rw [mul_comm]
    -- ⊢ m ∣ (m * b) * n
    rw [mul_assoc]
    -- ⊢ m ∣ m * (b * n)
    exact dvd_mul_right m (b * n)

-- 2ª demostración
-- ===============

example
  (h : m ∣ n ∨ m ∣ k)
```

```
  : m ∣ n * k :=
by
  rcases h with h1 | h2
  . -- h1 : m ∣ n
    rcases h1 with ⟨a, ha⟩
    -- a : ℕ
    -- ha : n = m * a
    rw [ha, mul_assoc]
    -- ⊢ m ∣ m * (a * k)
    exact dvd_mul_right m (a * k)
  . -- h2 : m ∣ k
    rcases h2 with ⟨b, hb⟩
    -- b : ℕ
    -- hb : k = m * b
    rw [hb, mul_comm, mul_assoc]
    -- ⊢ m ∣ m * (b * n)
    exact dvd_mul_right m (b * n)

-- 3ª demostración
-- ===============

example
  (h : m ∣ n ∨ m ∣ k)
  : m ∣ n * k :=
by
  rcases h with ⟨a, rfl⟩ | ⟨b, rfl⟩
  . -- a : ℕ
    -- ⊢ m ∣ (m * a) * k
    rw [mul_assoc]
    -- ⊢ m ∣ m * (a * k)
    exact dvd_mul_right m (a * k)
  . -- ⊢ m ∣ n * (m * b)
    rw [mul_comm, mul_assoc]
    -- ⊢ m ∣ m * (b * n)
    exact dvd_mul_right m (b * n)

-- 4ª demostración
-- ===============

example
  (h : m ∣ n ∨ m ∣ k)
  : m ∣ n * k :=
by
  rcases h with h1 | h2
  . -- h1 : m ∣ n
```

```
    exact dvd_mul_of_dvd_left h1 k
  . -- h2 : m ∣ k
    exact dvd_mul_of_dvd_right h2 n

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

#check (dvd_mul_of_dvd_left : m ∣ n → ∀ (c : ℕ), m ∣ n * c)
#check (dvd_mul_of_dvd_right : m ∣ n → ∀ (c : ℕ), m ∣ c * n)
#check (dvd_mul_right m n : m ∣ m * n)
#check (mul_assoc m n k : m * n * k = m * (n * k))
#check (mul_comm m n : m * n = n * m)
```

3.5.7. Desigualdad con rcases

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1
-- entonces
--    z ≥ 0
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ x ∈ ℝ)[x² ≥ 0]                                          (L1)
--    (∀ x, y ∈ ℝ)[x ≥ 0 → y ≥ 0 → x + y ≥ 0]                    (L2)
--    1 ≥ 0                                                      (L3)
--
-- Sean a y b tales que
--    z = a² + b² ∨ z = a² + b² + 1
-- Entonces, por L1, se tiene que
--    a² ≥ 0                                                     (1)
--    b² ≥ 0                                                     (2)
--
-- En el primer caso, z = a² + b² y se tiene que z ≥ 0 por el lema L2
-- aplicado a (1) y (2).
--
-- En el segundo caso, z = a² + b² y se tiene que z ≥ 0 por el lema L2
-- aplicado a (1), (2) y L3.

-- Demostraciones con Lean4
```

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

import Mathlib.Data.Real.Basic
import Mathlib.Tactic
variable {z : ℝ}

-- 1ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by
  rcases h with ⟨a, b, h1⟩
  -- a b : ℝ
  -- h1 : z = a ^ 2 + b ^ 2 ∨ z = a ^ 2 + b ^ 2 + 1
  have h2 : a ^ 2 ≥ 0 := pow_two_nonneg a
  have h3 : b ^ 2 ≥ 0 := pow_two_nonneg b
  have h4 : a ^ 2 + b ^ 2 ≥ 0 := add_nonneg h2 h3
  rcases h1 with h5 | h6
  . -- h5 : z = a ^ 2 + b ^ 2
    show z ≥ 0
    calc z = a ^ 2 + b ^ 2 := h5
         _ ≥ 0             := add_nonneg h2 h3
  . -- h6 : z = a ^ 2 + b ^ 2 + 1
    show z ≥ 0
    calc z = (a ^ 2 + b ^ 2) + 1 := h6
         _ ≥ 0                   := add_nonneg h4 zero_le_one

-- 2ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by
  rcases h with ⟨a, b, h1 | h2⟩
  . -- h1 : z = a ^ 2 + b ^ 2
    have h1a : a ^ 2 ≥ 0 := pow_two_nonneg a
    have h1b : b ^ 2 ≥ 0 := pow_two_nonneg b
    show z ≥ 0
    calc z = a ^ 2 + b ^ 2 := h1
         _ ≥ 0             := add_nonneg h1a h1b
  . -- h2 : z = a ^ 2 + b ^ 2 + 1
    have h2a : a ^ 2 ≥ 0         := pow_two_nonneg a
```

```
    have h2b : b ^ 2 ≥ 0           := pow_two_nonneg b
    have h2c : a ^ 2 + b ^ 2 ≥ 0 := add_nonneg h2a h2b
    show z ≥ 0
    calc z = (a ^ 2 + b ^ 2) + 1 := h2
         _ ≥ 0                   := add_nonneg h2c zero_le_one

-- 3ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by
  rcases h with ⟨a, b, h1 | h2⟩
  . -- h1 : z = a ^ 2 + b ^ 2
    rw [h1]
    -- ⊢ a ^ 2 + b ^ 2 ≥ 0
    apply add_nonneg
    . -- ⊢ 0 ≤ a ^ 2
      apply pow_two_nonneg
    . -- ⊢ 0 ≤ b ^ 2
      apply pow_two_nonneg
  . -- h2 : z = a ^ 2 + b ^ 2 + 1
    rw [h2]
    -- ⊢ a ^ 2 + b ^ 2 + 1 ≥ 0
    apply add_nonneg
    . -- ⊢ 0 ≤ a ^ 2 + b ^ 2
      apply add_nonneg
      . -- ⊢ 0 ≤ a ^ 2
        apply pow_two_nonneg
      . -- ⊢ 0 ≤ b ^ 2
        apply pow_two_nonneg
    . -- ⊢ 0 ≤ 1
      exact zero_le_one

-- 4ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by
  rcases h with ⟨a, b, rfl | rfl⟩
  . -- ⊢ a ^ 2 + b ^ 2 ≥ 0
    apply add_nonneg
```

```
    . -- ⊢ 0 ≤ a ^ 2
      apply pow_two_nonneg
    . -- ⊢ 0 ≤ b ^ 2
      apply pow_two_nonneg
  . -- ⊢ a ^ 2 + b ^ 2 + 1 ≥ 0
    apply add_nonneg
    . -- ⊢ 0 ≤ a ^ 2 + b ^ 2
      apply add_nonneg
      . -- ⊢ 0 ≤ a ^ 2
        apply pow_two_nonneg
      . -- ⊢ 0 ≤ b ^ 2
        apply pow_two_nonneg
    . -- ⊢ 0 ≤ 1
      exact zero_le_one

-- 5ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by
  rcases h with ⟨a, b, rfl | rfl⟩
  . -- ⊢ a ^ 2 + b ^ 2 ≥ 0
    nlinarith
  . -- ⊢ a ^ 2 + b ^ 2 + 1 ≥ 0
    nlinarith

-- 6ª demostración
-- ===============

example
  (h : ∃ x y, z = x^2 + y^2 ∨ z = x^2 + y^2 + 1)
  : z ≥ 0 :=
by rcases h with ⟨a, b, rfl | rfl⟩ <;> nlinarith

-- Comentarios:
-- 1. La táctica (rcases h with ⟨a, b, h1 | h2⟩) sobre el objetivo
--    (∃ x y : ℝ, P ∨ Q) crea dos casos. Al primero le añade las
--    hipótesis (a b : ℝ) y (k1 : P). Al segundo, (a b : ℝ) y (h2 : Q).

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

variable (x y : ℝ)
```

```
#check (add_nonneg : 0 ≤ x → 0 ≤ y → 0 ≤ x + y)
#check (pow_two_nonneg x : 0 ≤ x ^ 2)
#check (zero_le_one : 0 ≤ 1)
```

3.5.8. Igualdad de cuadrados

```
-- ---------------------------------------------------------------------
-- Ejercicio 1. Realizar las siguientes acciones:
-- 1. Importar la teoría de números reales.
-- 2. Declarar x e y como variables sobre los reales.
-- ----------------------------------------------------------------------

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

variable (x y : ℝ)

-- ---------------------------------------------------------------------
-- Ejercicio 2. Demostrar que si
--    x^2 = 1
-- entonces
--    x = 1 ∨ x = -1
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ x ∈ ℝ)[x - x = 0]                                          (L1)
--    (∀ x, y ∈ ℝ)[xy = 0 → x = 0 ∨ y = 0]                          (L2)
--    (∀ x, y ∈ ℝ)[x - y = 0 ↔ x = y]                               (L3)
--    (∀ x, y ∈ ℝ)[x + y = 0 → x = -y]                              (L4)
--
-- Se tiene que
--    (x - 1)(x + 1) = x² - 1
--                   = 1 - 1      [por la hipótesis]
--                   = 0          [por L1]
-- y, por el lema L2, se tiene que
--    x - 1 = 0 ∨ x + 1 = 0
-- Acabaremos la demostración por casos.
--
-- Primer caso:
--   x - 1 = 0 ⟹ x = 1            [por L3]
```

```
--              ⟹ x = 1 ∨ x = -1
--
-- Segundo caso:
--    x + 1 = 0 ⟹ x = -1            [por L4]
--              ⟹ x = 1 ∨ x = -1

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

-- 1ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
by
  have h1 : (x - 1) * (x + 1) = 0 := by
    calc (x - 1) * (x + 1) = x^2 - 1 := by ring
                         _ = 1 - 1   := by rw [h]
                         _ = 0       := sub_self 1
  have h2 : x - 1 = 0 ∨ x + 1 = 0 := by
    apply eq_zero_or_eq_zero_of_mul_eq_zero h1
  rcases h2 with h3 | h4
  . -- h3 : x - 1 = 0
    left
    -- ⊢ x = 1
    exact sub_eq_zero.mp h3
  . -- h4 : x + 1 = 0
    right
    -- ⊢ x = -1
    exact eq_neg_of_add_eq_zero_left h4

-- 2ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
by
  have h1 : (x - 1) * (x + 1) = 0 := by nlinarith
  have h2 : x - 1 = 0 ∨ x + 1 = 0 := by aesop
  rcases h2 with h3 | h4
  . -- h3 : x - 1 = 0
    left
    -- ⊢ x = 1
```

```
    linarith
  . -- h4 : x + 1 = 0
    right
    -- ⊢ x = -1
    linarith

-- 3ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
sq_eq_one_iff.mp h

-- 3ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
by aesop

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar si
--    x^2 = y^2
-- entonces
--    x = y ∨ x = -y
-- ----------------------------------------------------------------------

-- Usaremos los siguientes lemas
--    (∀ x ∈ ℝ)[x - x = 0]                                           (L1)
--    (∀ x, y ∈ ℝ)[xy = 0 → x = 0 ∨ y = 0]                           (L2)
--    (∀ x, y ∈ ℝ)[x - y = 0 ↔ x = y]                                (L3)
--    (∀ x, y ∈ ℝ)[x + y = 0 → x = -y]                               (L4)
--
-- Se tiene que
--    (x - y)(x + y) = x² - y²
--                   = y² - y²    [por la hipótesis]
--                   = 0          [por L1]
-- y, por el lema L2, se tiene que
--    x - y = 0 ∨ x + y = 0
--
-- Acabaremos la demostración por casos.
--
-- Primer caso:
```

```
--   x - y = 0 ⟹ x = y             [por L3]
--             ⟹ x = y ∨ x = -y
--
-- Segundo caso:
--   x + y = 0 ⟹ x = -y            [por L4]
--             ⟹ x = y ∨ x = -y

-- 1ª demostración
-- ===============

example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
by
  have h1 : (x - y) * (x + y) = 0 := by
    calc (x - y) * (x + y) = x^2 - y^2 := by ring
                         _ = y^2 - y^2 := by rw [h]
                         _ = 0         := sub_self (y ^ 2)
  have h2 : x - y = 0 ∨ x + y = 0 := by
    apply eq_zero_or_eq_zero_of_mul_eq_zero h1
  rcases h2 with h3 | h4
  . -- h3 : x - y = 0
    left
    -- ⊢ x = y
    exact sub_eq_zero.mp h3
  . -- h4 : x + y = 0
    right
    -- ⊢ x = -y
    exact eq_neg_of_add_eq_zero_left h4

-- 2ª demostración
-- ===============

example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
by
  have h1 : (x - y) * (x + y) = 0 := by nlinarith
  have h2 : x - y = 0 ∨ x + y = 0 := by aesop
  rcases h2 with h3 | h4
  . -- h3 : x - y = 0
    left
    -- ⊢ x = y
    linarith
  . -- h4 : x + y = 0
```

```
    right
    -- ⊢ x = -y
    linarith

-- 2ª demostración
-- ===============

example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
sq_eq_sq_iff_eq_or_eq_neg.mp h

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

#check (eq_neg_of_add_eq_zero_left : x + y = 0 → x = -y)
#check (eq_zero_or_eq_zero_of_mul_eq_zero : x * y = 0 → x = 0 ∨ y = 0)
#check (sq_eq_one_iff : x ^ 2 = 1 ↔ x = 1 ∨ x = -1)
#check (sq_eq_sq_iff_eq_or_eq_neg : x ^ 2 = y ^ 2 ↔ x = y ∨ x = -y)
#check (sub_eq_zero : x - y = 0 ↔ x = y)
#check (sub_self x : x - x = 0)
```

3.5.9. Igualdad de cuadrados en dominios de integridad

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar las teorías:
-- + algebra.group_power de potencias en grupos
-- + tactic de tácticas
-- ----------------------------------------------------------------------

import Mathlib

-- ---------------------------------------------------------------------
-- Ejercicio. Declara R como una variable sobre dominios de integridad.
-- ----------------------------------------------------------------------

variable {R : Type _}  [CommRing R] [IsDomain R]

-- ---------------------------------------------------------------------
-- Ejercicio. Declarar x e y como variables sobre R.
-- ----------------------------------------------------------------------

variable (x y : R)
```

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar si
--    x^2 = 1
-- entonces
--    x = 1 ∨ x = -1
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ x ∈ ℝ)[x - x = 0]                                          (L1)
--    (∀ x, y ∈ ℝ)[xy = 0 → x = 0 ∨ y = 0]                          (L2)
--    (∀ x, y ∈ ℝ)[x - y = 0 ↔ x = y]                               (L3)
--    (∀ x, y ∈ ℝ)[x + y = 0 → x = -y]                              (L4)
--
-- Se tiene que
--    (x - 1)(x + 1) = x² - 1
--                   = 1 - 1      [por la hipótesis]
--                   = 0          [por L1]
-- y, por el lema L2, se tiene que
--    x - 1 = 0 ∨ x + 1 = 0
-- Acabaremos la demostración por casos.
--
-- Primer caso:
--   x - 1 = 0 ⟹ x = 1             [por L3]
--             ⟹ x = 1 ∨ x = -1
--
-- Segundo caso:
--   x + 1 = 0 ⟹ x = -1            [por L4]
--             ⟹ x = 1 ∨ x = -1

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

-- 1ª demostración
example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
by
  have h1 : (x - 1) * (x + 1) = 0 := by
    calc (x - 1) * (x + 1) = x^2 - 1 := by ring
                         _ = 1 - 1   := by rw [h]
                         _ = 0       := sub_self 1
```

```
  have h2 : x - 1 = 0 ∨ x + 1 = 0 := by
    apply eq_zero_or_eq_zero_of_mul_eq_zero h1
  rcases h2 with h3 | h4
  . -- h3 : x - 1 = 0
    left
    -- ⊢ x = 1
    exact sub_eq_zero.mp h3
  . -- h4 : x + 1 = 0
    right
    -- ⊢ x = -1
    exact eq_neg_of_add_eq_zero_left h4

-- 2ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
sq_eq_one_iff.mp h

-- 3ª demostración
-- ===============

example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
by aesop

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar si
--    x^2 = y^2
-- entonces
--    x = y ∨ x = -y
-- ----------------------------------------------------------------------

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

-- Usaremos los siguientes lemas
--    (∀ x ∈ ℝ)[x - x = 0]                                         (L1)
--    (∀ x, y ∈ ℝ)[xy = 0 → x = 0 ∨ y = 0]                         (L2)
--    (∀ x, y ∈ ℝ)[x - y = 0 ↔ x = y]                              (L3)
--    (∀ x, y ∈ ℝ)[x + y = 0 → x = -y]                             (L4)
--
-- Se tiene que
```

```
--    (x - y)(x + y) = x² - y²
--                   = y² - y²    [por la hipótesis]
--                   = 0          [por L1]
-- y, por el lema L2, se tiene que
--    x - y = 0 ∨ x + y = 0
--
-- Acabaremos la demostración por casos.
--
-- Primer caso:
--   x - y = 0 ⟹ x = y              [por L3]
--             ⟹ x = y ∨ x = -y
--
-- Segundo caso:
--   x + y = 0 ⟹ x = -y             [por L4]
--             ⟹ x = y ∨ x = -y

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

-- 1ª demostración
-- ===============

example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
by
  have h1 : (x - y) * (x + y) = 0 := by
    calc (x - y) * (x + y) = x^2 - y^2 := by ring
                         _ = y^2 - y^2 := by rw [h]
                         _ = 0         := sub_self (y ^ 2)
  have h2 : x - y = 0 ∨ x + y = 0 := by
    apply eq_zero_or_eq_zero_of_mul_eq_zero h1
  rcases h2 with h3 | h4
  . -- h3 : x - y = 0
    left
    -- ⊢ x = y
    exact sub_eq_zero.mp h3
  . -- h4 : x + y = 0
    right
    -- ⊢ x = -y
    exact eq_neg_of_add_eq_zero_left h4

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

```
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
sq_eq_sq_iff_eq_or_eq_neg.mp h

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

#check (eq_neg_of_add_eq_zero_left : x + y = 0 → x = -y)
#check (eq_zero_or_eq_zero_of_mul_eq_zero : x * y = 0 → x = 0 ∨ y = 0)
#check (sq_eq_one_iff : x ^ 2 = 1 ↔ x = 1 ∨ x = -1)
#check (sq_eq_sq_iff_eq_or_eq_neg : x ^ 2 = y ^ 2 ↔ x = y ∨ x = -y)
#check (sub_eq_zero : x - y = 0 ↔ x = y)
#check (sub_self x : x - x = 0)
```

3.5.10. Eliminación de la doble negación (Tácticas (cases em) y by_cases)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Importar la librería de tácticas
-- ----------------------------------------------------------------------

import Mathlib.Tactic
variable (P : Prop)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ¬¬P → P
-- ----------------------------------------------------------------------

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

-- Supongamos que
--    ¬¬P                                                          (1)
--
-- Por el principio del tercio excluso, se tiene
--    P ∨ ¬P
-- lo que da lugar a dos casos.
--
-- En el primer caso, se supone P que es lo que hay que demostrar.
--
```

```
-- En el primer caso, se supone ¬P que es una contradicción con (1).

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

-- 1ª demostración
-- ===============

example : ¬¬P → P :=
by
  intro h1
  -- h1 : ¬¬P
  -- ⊢ P
  have h2 : P ∨ ¬ P := em P
  rcases h2 with h3 | h4
  . -- h3 : P
    exact h3
  . -- h4 : ¬P
    exfalso
    -- ⊢ False
    exact h1 h4

-- 2ª demostración
-- ===============

example : ¬¬P → P :=
by
  intro h1
  -- h1 : ¬¬P
  -- ⊢ P
  rcases em P with h2 | h3
  . -- h2 : P
    exact h2
  . -- h3 : ¬P
    exact absurd h3 h1

-- 3ª demostración
-- ===============

example : ¬¬P → P :=
by
  intro h1
  -- h1 : ¬¬P
  -- ⊢ P
  cases em P
```

```
  . -- h2 : P
    assumption
  . -- h3 : ¬P
    contradiction

-- 4ª demostración
-- ===============

example : ¬¬P → P :=
by
  intro h
  by_cases P
  . assumption
  . contradiction

-- 4ª demostración
-- ===============

example : ¬¬P → P :=
by
  intro h1
  -- h1 : ¬¬P
  -- ⊢ P
  by_contra h
  -- h : ¬P
  -- ⊢ False
  exact h1 h

-- 5ª demostración
-- ===============

example : ¬¬P → P :=
by tauto

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

variable (Q : Prop)
#check (absurd : P → ¬P → Q)
#check (em P : P ∨ ¬P)
```

3.5.11. Implicación mediante disyunción y negación

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (P → Q) ↔ ¬P ∨ Q
-- ----------------------------------------------------------------------

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

-- Demostraremos cada una de las implicaciones.
--
-- (==>) Supongamos que P → Q. Distinguimos dos subcasos según el valor de
-- P.
--
-- Primer subcaso: suponemos P. Entonces. tenemos Q (por P → Q) y. por
-- tanto, ¬P ∨ Q.
--
-- Segundo subcaso: suponemos ¬P. Entonces. tenemos ¬P ∨ Q.
--
-- (<==) Supongamos que ¬P ∨ Q y P y tenemos que demostrar
-- Q. Distinguimos dos subcasos según ¬P ∨ Q.
--
-- Primer subcaso: Suponemos ¬P. Entonces tenemos una contradicción con
-- P.
--
-- Segundo subcaso: Suponemos Q, que es lo que tenemos que demostrar.

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

import Mathlib.Tactic
variable (P Q : Prop)

-- 1ª demostración
-- ===============

example
  : (P → Q) ↔ ¬P ∨ Q :=
by
  constructor
  . -- ⊢ (P → Q) → ¬P ∨ Q
    intro h1
    -- h1 : P → Q
    -- ⊢ ¬P ∨ Q
```

```
    by_cases h2 : P
    . -- h2 : P
      right
      -- ⊢ Q
      apply h1
      -- ⊢ P
      exact h2
    . -- h2 : ¬P
      left
      -- ⊢ ¬P
      exact h2
  . -- ⊢ ¬P ∨ Q → P → Q
    intros h3 h4
    -- h3 : ¬P ∨ Q
    -- h4 : P
    -- ⊢ Q
    rcases h3 with h3a | h3b
    . -- h : ¬P
      exact absurd h4 h3a
    . -- h : Q
      exact h3b

-- 2ª demostración
-- ===============

example
  : (P → Q) ↔ ¬P ∨ Q :=
by
  constructor
  . -- ⊢ (P → Q) → ¬P ∨ Q
    intro h1
    -- h1 : P → Q
    -- ⊢ ¬P ∨ Q
    by_cases h2: P
    . -- h2 : P
      right
      -- ⊢ Q
      exact h1 h2
    . -- h2 : ¬P
      left
      -- ⊢ ¬P
      exact h2
  . -- ⊢ ¬P ∨ Q → P → Q
    intros h3 h4
    -- h3 : ¬P ∨ Q
```

```
    -- h4 : P
    -- ⊢ Q
    cases h3
    . -- h : ¬P
      contradiction
    . -- h : Q
      assumption

-- 3ª demostración
-- ===============

example
  (P Q : Prop)
  : (P → Q) ↔ ¬P ∨ Q :=
imp_iff_not_or

-- 3ª demostración
-- ===============

example
  (P Q : Prop)
  : (P → Q) ↔ ¬P ∨ Q :=
by tauto

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

#check (absurd : P → (¬P → Q))
#check (imp_iff_not_or : (P → Q) ↔ (¬P ∨ Q))
```

3.6. Sucesiones y convergencia

3.6.1. Definicion de convergencia

```
-- ---------------------------------------------------------------------
-- Ejercicio. Definir la función
--     limite (ℕ → ℝ) → ℝ → Prop
-- tal que (limite s a) afirma que a es el límite de s.
-- ----------------------------------------------------------------------

import Mathlib.Data.Real.Basic
```

```
def limite (s : ℕ → ℝ) (a : ℝ) :=
  ∀ ε > 0, ∃ N, ∀ n ≥ N, |s n - a| < ε

#print limite

-- Comentario: Al colocar el cursor sobre print se obtiene
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--    fun s a => ∀ ε > 0, ∃ N, ∀ n ≥ N, |s n - a| < ε
```

3.6.2. Demostración por extensionalidad (La táctica ext)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (fun x y : ℝ ↦ (x + y)^2) = (fun x y : ℝ ↦ x^2 + 2*x*y + y^2)
-- ----------------------------------------------------------------------

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

-- 1ª demostración
-- ===============

example : (fun x y : ℝ ↦ (x + y)^2) = (fun x y : ℝ ↦ x^2 + 2*x*y + y^2) :=
by
  ext u v
  -- u v : ℝ
  -- ⊢ (u + v) ^ 2 = u ^ 2 + 2 * u * v + v ^ 2
  ring

-- Comentario: La táctica ext transforma las conclusiones de la forma
-- (fun x ↦ f x) = (fun x ↦ g x) en f x = g x.

-- 2ª demostración
-- ===============

example : (fun x y : ℝ ↦ (x + y)^2) = (fun x y : ℝ ↦ x^2 + 2*x*y + y^2) :=
by { ext ; ring }
```

3.6.3. Demostración por congruencia (La táctica congr)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    |a| = |a - b + b|
-- ----------------------------------------------------------------------

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

variable (a b : ℝ)

-- 1ª demostración
-- ===============

example
  : |a| = |a - b + b| :=
by
  congr
  -- a = a - b + b
  ring

-- Comentario: La táctica cong sustituye una conclusión de la forma
-- A = B por las igualdades de sus subtérminos que no no iguales por
-- definición. Por ejemplo, sustituye la conclusión (x * f y = g w * f z)
-- por las conclusiones (x = g w) y (y = z).

-- 2ª demostración
-- ===============

example
  (a b : ℝ)
  : |a| = |a - b + b| :=
by { congr ; ring }

-- 3ª demostración
-- ===============

example
  (a b : ℝ)
  : |a| = |a - b + b| :=
by ring_nf
```

3.6.4. Demostración por conversión (La táctica convert)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar, para todo a ∈ ℝ, si
--    1 < a
-- entonces
--    a < a * a
-- ----------------------------------------------------------------------

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

-- Se usarán los siguientes lemas
--    L1: 0 < 1
--    L2: (∀ a ∈ ℝ[1·a = a]
--    L3: (∀ a, b, c ∈ ℝ)[0 < a → (ba < ca ↔ b < c)]
--
-- En primer lugar, tenemos que
--    0 < a                                                       (1)
-- ya que
--    0 < 1    [por L1]
--      < a    [por la hipótesis]
-- Entonces,
--    a = 1·a   [por L2]
--      < a·a   [por L3, (1) y la hipótesis]

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

import Mathlib.Data.Real.Basic
variable {a : ℝ}

-- 1ª demostración
-- ===============

example
  (h : 1 < a)
  : a < a * a :=
by
  have h1 : 0 < a := calc
    0 < 1 := zero_lt_one
    _ < a := h
  show a < a * a
  calc a = 1 * a := (one_mul a).symm
       _ < a * a := (mul_lt_mul_right h1).mpr h
```

```
-- Comentarios: La táctica (convert e) genera nuevos subojetivos cuya
-- conclusiones son las diferencias entre el tipo de e y la conclusión.

-- 2ª demostración
-- ===============

example
  (h : 1 < a)
  : a < a * a :=
by
  convert (mul_lt_mul_right _).mpr h
  . -- ⊢ a = 1 * a
    rw [one_mul]
  . -- ⊢ 0 < a
    exact lt_trans zero_lt_one h

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

variable (a b c : ℝ)
#check (lt_trans : a < b → b < c → a < c)
#check (mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c))
#check (one_mul a : 1 * a = a)
#check (zero_lt_one : 0 < 1)
```

3.6.5. Convergencia de la función constante

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que, para todo a ∈ ℝ, la sucesión constante
--    s(n) = a
-- converge a a.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_convergencia
variable (a : ℝ)

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

-- Tenemos que demostrar que para cada ε ∈ ℝ tal que ε > 0, existe un
-- N ∈ ℕ, tal que (∀n ∈ ℕ)[n ≥ N → |s(n) - a| < ε]. Basta tomar N como
-- 0, ya que para todo n ≥ N se tiene
```

```
--    |s(n) - a| = |a - a|
--               = |0|
--               = 0
--               < ε

-- 1ª demostración
-- ===============

example : limite (fun _ : ℕ ↦ a) a :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun x => a) n - a| < ε
  use 0
  -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(fun x => a) n - a| < ε
  intros n _hn
  -- n : ℕ
  -- nge : n ≥ 0
  -- ⊢ |(fun x => a) n - a| < ε
  show |(fun _ => a) n - a| < ε
  calc |(fun _ => a) n - a| = |a - a|  := by dsimp
                          _ = |0|      := by {congr ; exact sub_self a}
                          _ = 0        := abs_zero
                          _ < ε        := hε

-- 2ª demostración
-- ===============

theorem limite_constante
  : limite (fun _ : ℕ ↦ a) a :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun x => a) n - a| < ε
  use 0
  -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(fun x => a) n - a| < ε
  intros n _hn
  -- n : ℕ
  -- nge : n ≥ 0
  -- ⊢ |(fun x => a) n - a| < ε
  dsimp
  -- ⊢ |a - a| < ε
  rw [sub_self]
```

```
  -- ⊢ |0| < ε
  rw [abs_zero]
  -- ⊢ 0 < ε
  exact hε

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

#check (abs_zero : |(0 : ℝ)| = 0)
#check (sub_self a : a - a = 0)
```

3.6.6. Convergencia de la suma

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar el límite de la suma de dos sucesiones
-- convergentes es la suma de los límites.
-- ----------------------------------------------------------------------

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

-- En la demostración usaremos los siguientes lemas
--    (∀ a ∈ ℝ)[a > 0 → a / 2 > 0]                                   (L1)
--    (∀ a, b, c ∈ ℝ)[max(a, b) ≤ c → a ≤ c]                         (L2)
--    (∀ a, b, c ∈ ℝ)[max(a, b) ≤ c → b ≤ c]                         (L3)
--    (∀ a, b ∈ ℝ)[|a + b| ≤ |a| + |b|]                              (L4)
--    (∀ a ∈ ℝ)[a / 2 + a / 2 = a]                                   (L5)
--
-- Tenemos que probar que si s es una sucesión con límite a y t otra con
-- límite , entonces el límite de s + t es a+b; es decir, que para todo
-- ε ∈ ℝ, si
--    ε > 0                                                          (1)
-- entonces
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |(s + t)(n) - (a + b)| < ε]           (2)
--
-- Por (1) y el lema L1, se tiene que
--    ε/2 > 0                                                        (3)
-- Por (3) y porque el límite de s es a, se tiene que
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |s(n) - a| < ε/2]
-- Sea N₁ ∈ ℕ tal que
--    (∀n ∈ ℕ)[n ≥ N₁ → |s(n) - a| < ε/2]                            (4)
-- Por (3) y porque el límite de t es b, se tiene que
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |t(n) - b| < ε/2]
```

```
-- Sea N₂ ∈ ℕ tal que
--    (∀n ∈ ℕ)[n ≥ N₂ → |t(n) - b| < ε/2]                         (5)
-- Sea N = max(N₁, N₂). Veamos que verifica la condición (1). Para ello,
-- sea n ∈ ℕ tal que n ≥ N. Entonces, n ≥ N₁ (por L2) y n ≥ N₂ (por
-- L3). Por tanto, por las propiedades (4) y (5) se tiene que
--    |s(n) - a| < ε/2                                            (6)
--    |t(n) - b| < ε/2                                            (7)
-- Finalmente,
--    |(s + t)(n) - (a + b)| = |(s(n) + t(n)) - (a + b)|
--                           = |(s(n) - a) + (t(n) - b)|
--                           ≤ |s(n) - a| + |t(n) - b|      [por L4]
--                           < ε / 2 + ε / 2                [por (6) y (7)
--                           = ε                            [por L5]

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

import src.Logica.Definicion_de_convergencia
import Mathlib.Tactic

variable {s t : ℕ → ℝ}
variable {a b c : ℝ}

lemma limite_suma
  (cs : limite s a)
  (ct : limite t b)
  : limite (s + t) (a + b) :=
by
  intros ε εpos
  -- ε : ℝ
  -- εpos : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(s + t) n - (a + b)| < ε
  have ε2pos : 0 < ε / 2 := half_pos εpos
  cases' cs (ε / 2) ε2pos with Ns hs
  -- Ns : ℕ
  -- hs : ∀ (n : ℕ), n ≥ Ns → |s n - a| < ε / 2
  cases' ct (ε / 2) ε2pos with Nt ht
  -- Nt : ℕ
  -- ht : ∀ (n : ℕ), n ≥ Nt → |t n - b| < ε / 2
  clear cs ct ε2pos εpos
  let N := max Ns Nt
  use N
  -- ⊢ ∀ (n : ℕ), n ≥ N → |(s + t) n - (a + b)| < ε
  intros n hn
  -- n : ℕ
```

```
  -- hn : n ≥ N
  have nNs : n ≥ Ns := le_of_max_le_left hn
  specialize hs n nNs
  -- hs : |s n - a| < ε / 2
  have nNt : n ≥ Nt := le_of_max_le_right hn
  specialize ht n nNt
  -- ht : |t n - b| < ε / 2
  clear hn nNs nNt
  calc |(s + t) n - (a + b)|
       = |s n + t n - (a + b)|     := rfl
     _ = |(s n - a) + (t n -  b)| := by { congr; ring }
     _ ≤ |s n - a| + |t n -  b|    := by apply abs_add
     _ < ε / 2 + ε / 2             := by linarith [hs, ht]
     _ = ε                         := by apply add_halves

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

#check (half_pos : a > 0 → a / 2 > 0)
#check (le_of_max_le_left : max a b ≤ c → a ≤ c)
#check (le_of_max_le_right : max a b ≤ c → b ≤ c)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
#check (add_halves a : a / 2 + a / 2 = a)
```

3.6.7. Convergencia del producto por una constante

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que que si el límite de uₙ es a, entonces el de
-- cuₙ es ca.
-- ---------------------------------------------------------------------

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

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos que demostrar que
--    (∃ N ∈ ℕ)(∀ n ≥ N)[|cuₙ - ca| < ε]                           (1)
-- Distinguiremos dos casos según sea c = 0 o no.
--
-- Primer caso: Supongamos que c = 0. Entonces, (1) se reduce a
--    (∃ N ∈ ℕ)(∀ n ≥ N)[|0·uₙ - 0·a| < ε]
-- es decir,
--    (∃ N ∈ ℕ)(∀ n ≥ N)[0 < ε]
-- que se verifica para cualquier número N, ya que ε > 0.
```

```
--
-- Segundo caso: Supongamos que c ≠ 0. Entonces, ε/|c| > 0 y, puesto que
-- el límite de uₙ es a, existe un k ∈ ℕ tal que
--    (∀ n ≥ k)[|uₙ - a| < ε/|c|]                                    (2)
-- Veamos que con k se cumple (1). En efecto, sea n ≥ k. Entonces,
--    |cuₙ - ca| = |c(uₙ - a)|
--               = |c||u n - a|
--               < |c|(ε/|c|)     [por (2)]
--               = ε

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

import src.Logica.Definicion_de_convergencia
import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable {u v : ℕ → ℝ}
variable {a : ℝ}
variable (c : ℝ)

-- 1ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun n ↦ c * (u n)) (c * a) :=
by
  by_cases hc : c = 0
  . -- hc : c = 0
    subst hc
    -- ⊢ limite (fun n => 0 * u n) (0 * a)
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => 0 * u n) n - 0 * a| < ε
    aesop
  . -- hc : ¬c = 0
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => c * u n) n - c * a| < ε
    have hc' : 0 < |c| := abs_pos.mpr hc
    have hεc : 0 < ε / |c| := div_pos hε hc'
    specialize h (ε/|c|) hεc
```

```
    -- h : ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε / |c|
    cases' h with N hN
    -- N : ℕ
    -- hN : ∀ (n : ℕ), n ≥ N → |u n - a| < ε / |c|
    use N
    -- ⊢ ∀ (n : ℕ), n ≥ N → |(fun n => c * u n) n - c * a| < ε
    intros n hn
    -- n : ℕ
    -- hn : n ≥ N
    -- ⊢ |(fun n => c * u n) n - c * a| < ε
    specialize hN n hn
    -- hN : |u n - a| < ε / |c|
    dsimp only
    calc |c * u n - c * a|
         = |c * (u n - a)| := congrArg abs (mul_sub c (u n) a).symm
       _ = |c| * |u n - a| := abs_mul c  (u n - a)
       _ < |c| * (ε / |c|) := (mul_lt_mul_left hc').mpr hN
       _ = ε               := mul_div_cancel₀ ε (ne_of_gt hc')

-- 2ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun n ↦ c * (u n)) (c * a) :=
by
  by_cases hc : c = 0
  . -- hc : c = 0
    subst hc
    -- ⊢ limite (fun n => 0 * u n) (0 * a)
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => 0 * u n) n - 0 * a| < ε
    aesop
  . -- hc : ¬c = 0
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => c * u n) n - c * a| < ε
    have hc' : 0 < |c| := abs_pos.mpr hc
    have hεc : 0 < ε / |c| := div_pos hε hc'
    specialize h (ε/|c|) hεc
    -- h : ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε / |c|
    cases' h with N hN
```

```
    -- N : ℕ
    -- hN : ∀ (n : ℕ), n ≥ N → |u n - a| < ε / |c|
    use N
    -- ⊢ ∀ (n : ℕ), n ≥ N → |(fun n => c * u n) n - c * a| < ε
    intros n hn
    -- n : ℕ
    -- hn : n ≥ N
    -- ⊢ |(fun n => c * u n) n - c * a| < ε
    specialize hN n hn
    -- hN : |u n - a| < ε / |c|
    dsimp only
    -- ⊢ |c * u n - c * a| < ε
    rw [← mul_sub]
    -- ⊢ |c * (u n - a)| < ε
    rw [abs_mul]
    -- ⊢ |c| * |u n - a| < ε
    rw [← lt_div_iff₀' hc']
    -- ⊢ |u n - a| < ε / |c|
    exact hN

-- 3ª demostración
-- ===============

theorem limite_por_constante
  (h : limite u a)
  : limite (fun n ↦ c * (u n)) (c * a) :=
by
  by_cases hc : c = 0
  . -- hc : c = 0
    subst hc
    -- ⊢ limite (fun n => 0 * u n) (0 * a)
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ n ≥ N, |(fun n => 0 * u n) n - 0 * a| < ε
    aesop
  . -- hc : ¬c = 0
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ n ≥ N, |(fun n => c * u n) n - c * a| < ε
    have hc' : 0 < |c| := by aesop
    have hεc : 0 < ε / |c| := div_pos hε hc'
    rcases h (ε/|c|) hεc with ⟨N, hN⟩
    -- N : ℕ
```

```
    -- hN : ∀ n ≥ N, |u n - a| < ε / |c|
    use N
    -- ⊢ ∀ n ≥ N, |(fun n => c * u n) n - c * a| < ε
    intros n hn
    -- n : ℕ
    -- hn : n ≥ N
    -- ⊢ |(fun n => c * u n) n - c * a| < ε
    specialize hN n hn
    -- hN : |u n - a| < ε / |c|
    dsimp only
    -- ⊢ |c * u n - c * a| < ε
    rw [← mul_sub, abs_mul, ← lt_div_iff₀' hc']
    -- ⊢ |u n - a| < ε / |c|
    exact hN

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

variable (b c : ℝ)
variable (f : ℝ → ℝ)
#check (abs_mul a b : |a * b| = |a| * |b|)
#check (abs_pos.mpr : a ≠ 0 → 0 < |a|)
#check (congrArg f : a = b → f a = f b)
#check (div_pos : 0 < a → 0 < b → 0 < a / b)
#check (lt_div_iff₀' : 0 < c → (a < b / c ↔ c * a < b))
#check (mul_div_cancel₀ a : b ≠ 0 → b * (a / b) = a)
#check (mul_lt_mul_left : 0 < a → (a * b < a * c ↔ b < c))
#check (mul_sub a b c : a * (b - c) = a * b - a * c)
```

3.6.8. Acotación de convergentes

```
import src.Logica.Definicion_de_convergencia
import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable {u : ℕ → ℝ}

-- ---------------------------------------------------------------------
-- Demostrar que si u es una sucesión convergente, entonces está
-- acotada; es decir,
--     ∃ k b. ∀n≥k. |u n| < b
-- ----------------------------------------------------------------------
```

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

-- Puesto que la sucesión uₙ es convergente, existe un a ∈ ℝ tal que
--    lim(uₙ) = a
-- Luego, existe un k ∈ ℕ tal que
--    (∀ n ∈ ℕ)[n ≥ k → |uₙ - a | < 1]                              (1)
-- Veamos que uₙ está acotada por 1 + |a|; es decir,
--    (∀ n ∈ ℕ)[n ≥ k → |uₙ| < 1 + |a|]
-- Para ello, sea n ∈ ℕ tal que
--    n ≥ k.                                                        (2)
-- Entonces,
--    |uₙ| = |uₙ - a + a|
--         ≤ |uₙ - a| + |a|
--         < 1 + |a|          [por (1) y (2)]

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

-- 1ª demostración
-- ===============

example
  (ua : limite  u a)
  : ∃ k b, ∀ n, n ≥ k → |u n| <b :=
by
  rcases ua 1 zero_lt_one with ⟨k, h⟩
  -- k : ℕ
  -- h : ∀ n ≥ k, |u n - a| < 1
  use k, 1 + |a|
  -- ⊢ ⊢ ∀ n ≥ k, |u n| ≤ 1 + |a|
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n| ≤ 1 + |a|
  specialize h n hn
  -- h : |u n - a| < 1
  calc |u n|
       = |u n - a + a|   := congrArg abs (eq_add_of_sub_eq rfl)
     _ ≤ |u n - a| + |a| := abs_add (u n - a) a
     _ < 1 + |a|         := (add_lt_add_iff_right |a|).mpr h

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

```
theorem convergentes_acotadas
  (ua : limite  u a)
  : ∃ k b, ∀ n, n ≥ k → |u n| < b :=
by
  rcases ua 1 zero_lt_one with ⟨k, h⟩
  -- k : ℕ
  -- h : ∀ (n : ℕ), n ≥ k → |u n - a| ≤ 1
  use k, 1 + |a|
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n| ≤ 1 + |a|
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n| ≤ 1 + |a|
  specialize h n hn
  -- h : |u n - a| < 1
  calc |u n|
       = |u n - a + a|   := by ring_nf
     _ ≤ |u n - a| + |a| := abs_add (u n - a) a
     _ < 1 + |a|         := by linarith

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

variable (a b c : ℝ)
variable (f : ℝ → ℝ)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
#check (add_le_add_right : b ≤ c → ∀ a,  b + a ≤ c + a)
#check (add_lt_add_iff_right a : b + a < c + a ↔ b < c)
#check (congrArg f : a = b → f a = f b)
#check (eq_add_of_sub_eq :  a - c = b → a = b + c)
#check (zero_lt_one : 0 < 1)
```

3.6.9. Producto por sucesión convergente a cero

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar si s es una sucesión convergente y el límite de
-- t es 0, entonces el límite de s * t es 0.
-- ----------------------------------------------------------------------

import src.Logica.Acotacion_de_convergentes

variable {s t : ℕ → ℝ}
variable {a : ℝ}
```

```
lemma aux_l1
  (B ε : ℝ)
  (εpos : ε > 0)
  (Bpos : 0 < B)
  (pos0 : ε / B > 0)
  (n : ℕ)
  (h0 : |s n| < B)
  (h1 : |t n - 0| < ε / B)
  : |s n| * |t n - 0| < ε :=
by
  by_cases h3 : s n = 0
  . -- h3 : s n = 0
    calc |s n| * |t n - 0|
             = |0| * |t n - 0|  := by rw [h3]
           _ = 0 * |t n - 0|    := by rw [abs_zero]
           _ = 0                := by exact zero_mul (abs (t n - 0))
           _ < ε                := by exact εpos
  . -- h3 : ¬s n = 0
    have h4 : |s n| > 0 :=
      by exact abs_pos.mpr h3
    clear h3
    have h5 : |s n| * |t n - 0| < |s n| * (ε / B) :=
      by exact mul_lt_mul_of_pos_left h1 h4
    have h6 : |s n| * (ε / B) < B * (ε / B) :=
      by exact mul_lt_mul_of_pos_right h0 pos0
    have h7 : B ≠ 0 :=
      by exact ne_of_gt Bpos
    have h8 : B * (ε / B) = ε :=
      calc B * (ε / B) = (B * B⁻¹) * ε := by ring
                     _ = 1 * ε         := by rw [Field.mul_inv_cancel B h7]
                     _ = ε             := by exact one_mul ε
    have h9 : |s n| * |t n - 0| < B * (ε / B) :=
      by exact lt_trans h5 h6
    rw [h8] at h9
    assumption


lemma aux
  (cs : limite s a)
  (ct : limite t 0)
  : limite (fun n ↦ s n * t n) 0 :=
by
  intros ε εpos
  -- ε : ℝ
```

```
  -- εpos : ε > 0
  -- ⊢ ∃ N, ∀ n ≥ N, |(fun n => s n * t n) n - 0| < ε
  dsimp
  -- ⊢ ∃ N, ∀ n ≥ N, |s n * t n - 0| < ε
  rcases convergentes_acotadas cs with ⟨N₀, B, h₀⟩
  -- N₀ : ℕ
  -- B : ℝ
  -- h₀ : ∀ n ≥ N₀, |s n| ≤ B
  have Bpos : 0 < B := lt_of_le_of_lt (abs_nonneg _) (h₀ N₀ (le_refl _))
  have pos₀ : ε / B > 0 := div_pos εpos Bpos
  rcases ct _ pos₀ with ⟨N₁, h₁⟩
  use max N₀ N₁
  intros n hn
  have hn0 : n ≥ N₀ := by
    exact le_of_max_le_left hn
  specialize h₀ n hn0
  have hn1 : n ≥ N₁ := by
    exact le_of_max_le_right hn
  specialize h₁ n hn1
  clear cs ct hn hn0 hn1 N₀ N₁
  calc
    |s n * t n - 0|
        = |s n * (t n - 0)|
          := by { congr; ring }
      _ = |s n| * |t n - 0|
          := by exact abs_mul (s n) (t n - 0)
      _ < ε
          := by exact aux_l1 B ε εpos Bpos pos₀ n h₀ h₁

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

variable (b c : ℝ)
#check (Field.mul_inv_cancel a : a ≠ 0 → a * a⁻¹ = 1)
#check (abs_mul a b : |a * b| = |a| * |b|)
#check (abs_nonneg a : 0 ≤ |a|)
#check (abs_pos : 0 < |a| ↔ a ≠ 0)
#check (abs_zero : |(0 : ℝ)| = 0)
#check (div_pos : 0 < a → 0 < b → 0 < a / b)
#check (le_of_max_le_left : max a b ≤ c → a ≤ c)
#check (le_of_max_le_right : max a b ≤ c → b ≤ c)
#check (le_refl a : a ≤ a)
#check (lt_of_le_of_lt : a ≤ b → b < c → a < c)
#check (lt_trans : a < b → b < c → a < c)
#check (mul_lt_mul_of_pos_left : b < c → 0 < a → a * b < a * c)
```

```
#check (mul_lt_mul_of_pos_right : b < c → 0 < a → b * a < c * a)
#check (ne_of_gt : b < a → a ≠ b)
#check (one_mul a : 1 * a = a)
#check (zero_mul a : 0 * a = 0)
```

3.6.10. Convergencia del producto

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar el límite del producto de dos sucesiones
-- convergentes es el producto de sus límites.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_convergencia
import src.Logica.Convergencia_de_la_funcion_constante
import src.Logica.Convergencia_de_la_suma
import src.Logica.Convergencia_del_producto_por_una_constante
import src.Logica.Acotacion_de_convergentes
import src.Logica.Producto_por_sucesion_convergente_a_cero
import Mathlib.Tactic

variable {s t : ℕ → ℝ}
variable {a b : ℝ}

theorem limite_mul
  (cs : limite s a)
  (ct : limite t b)
  : limite (fun n ↦ s n * t n) (a * b) :=
by
  have h₁ : limite (fun n ↦ s n * (t n + -b)) 0 := by
    apply aux cs
    -- ⊢ limite (fun n => t n + -b) 0
    convert limite_suma ct (limite_constante (-b))
    -- ⊢ 0 = b + -b
    ring
  convert (limite_suma h₁ (limite_por_constante b cs)) using 1
  . -- ⊢ (fun n => s n * t n) = (fun n => s n * (t n + -b)) + fun n => b * s n
    ext n
    -- ⊢ s n * t n = ((fun n => s n * (t n + -b)) + fun n => b * s n) n
    simp
    -- ⊢ s n * t n = s n * (t n + -b) + b * s n
    ring
  . -- ⊢ a * b = 0 + b * a
    ring
```

3.6.11. Unicidad del límite

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar la unicidad de los límites de las sucesiones
-- convergentes.
-- ----------------------------------------------------------------------

import src.Logica.Definicion_de_convergencia
import Mathlib.Tactic

theorem unicidad_limite
  {s : ℕ → ℝ}
  {a b : ℝ}
  (sa : limite s a)
  (sb : limite s b)
  : a = b :=
by
  by_contra abne
  -- abne : ¬a = b
  -- ⊢ False
  have : |a - b| > 0 := by
    apply abs_pos.mpr
    -- ⊢ a - b ≠ 0
    exact sub_ne_zero_of_ne abne
  let ε := |a - b| / 2
  have εpos : ε > 0 := by
    change |a - b| / 2 > 0
    -- ⊢ |a - b| / 2 > 0
    linarith
  rcases sa ε εpos with ⟨Na, hNa⟩
  -- Na : ℕ
  -- hNa : ∀ n ≥ Na, |s n - a| < ε
  rcases sb ε εpos with ⟨Nb, hNb⟩
  -- Nb : ℕ
  -- hNb : ∀ n ≥ Nb, |s n - b| < ε
  let N := max Na Nb
  have absa : |s N - a| < ε := by
    specialize hNa N
    -- hNa : N ≥ Na → |s N - a| < ε
    apply hNa
    -- ⊢ N ≥ Na
    exact le_max_left Na Nb
  have absb : |s N - b| < ε := by
    specialize hNb N
    -- hNb : N ≥ Nb → |s N - b| < ε
```

```
    apply hNb
    -- ⊢ N ≥ Nb
    exact le_max_right Na Nb
  have : |a - b| < |a - b| :=
    calc |a - b|
         = |(a - s N) + (s N - b)|  := by {congr; ring_nf}
       _ ≤ |a - s N| + |s N - b|    := abs_add (a - s N) (s N - b)
       _ = |s N - a| + |s N - b|    := by rw [abs_sub_comm]
       _ < ε + ε                    := by exact add_lt_add absa absb
       _ = |a - b|                  := by exact add_halves (abs (a - b))
  exact lt_irrefl _ this

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

variable (a b c d : ℝ)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
#check (abs_pos : 0 < |a| ↔ a ≠ 0)
#check (abs_sub_comm a b : |a - b| = |b - a|)
#check (add_halves a : a / 2 + a / 2 = a)
#check (add_lt_add : a < b → c < d → a + c < b + d)
#check (le_max_left a b : a ≤ max a b)
#check (le_max_right a b : b ≤ max a b)
#check (sub_ne_zero_of_ne : a ≠ b → a - b ≠ 0)
```

Capítulo 4

Conjuntos y funciones

En este capítulo se muestra el razonamiento con Lean sobre las operaciones conjuntistas y sobre las funciones.

4.1. Conjuntos

4.1.1. Monotonía de la intersección

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar si
--    s ⊆ t
-- entonces
--    s ∩ u ⊆ t ∩ u
-- ----------------------------------------------------------------------

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

-- Sea x ∈ s ∩ u. Entonces, se tiene que
--   x ∈ s                                                          (1)
--   x ∈ u                                                          (2)
-- De (1) y s ⊆ t, se tiene que
--   x ∈ t                                                          (3)
-- De (3) y (2) se tiene que
--   x ∈ t ∩ u
-- que es lo que teníamos que demostrar.

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

```
import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α : Type}
variable (s t u : Set α)

-- 1ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
by
  rw [subset_def]
  -- ⊢ ∀ (x : α), x ∈ s ∩ u → x ∈ t ∩ u
  intros x h1
  -- x : α
  -- h1 : x ∈ s ∩ u
  -- ⊢ x ∈ t ∩ u
  rcases h1 with ⟨xs, xu⟩
  -- xs : x ∈ s
  -- xu : x ∈ u
  constructor
  . -- ⊢ x ∈ t
    rw [subset_def] at h
    -- h : ∀ (x : α), x ∈ s → x ∈ t
    apply h
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ x ∈ u
    exact xu

-- 2ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
by
  rw [subset_def]
  -- ⊢ ∀ (x : α), x ∈ s ∩ u → x ∈ t ∩ u
  rintro x ⟨xs, xu⟩
  -- x : α
```

```
  -- xs : x ∈ s
  -- xu : x ∈ u
  rw [subset_def] at h
  -- h : ∀ (x : α), x ∈ s → x ∈ t
  exact ⟨h x xs, xu⟩

-- 3ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
by
  simp only [subset_def]
  -- ⊢ ∀ (x : α), x ∈ s ∩ u → x ∈ t ∩ u
  rintro x ⟨xs, xu⟩
  -- x : α
  -- xs : x ∈ s
  -- xu : x ∈ u
  rw [subset_def] at h
  -- h : ∀ (x : α), x ∈ s → x ∈ t
  exact ⟨h _ xs, xu⟩

-- 4ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
by
  intros x xsu
  -- x : α
  -- xsu : x ∈ s ∩ u
  -- ⊢ x ∈ t ∩ u
  exact ⟨h xsu.1, xsu.2⟩

-- 5ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
by
  rintro x ⟨xs, xu⟩
  -- xs : x ∈ s
```

```
  -- xu : x ∈ u
  -- ⊢ x ∈ t ∩ u
  exact ⟨h xs, xu⟩

-- 6ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
  fun _ ⟨xs, xu⟩ ↦  ⟨h xs, xu⟩

-- 7ª demostración
-- ===============

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
inter_subset_inter_left u h

-- Lema usado
-- ==========

#check (subset_def : (s ⊆ t) = ∀ x ∈ s, x ∈ t)
#check (inter_subset_inter_left u : s ⊆ t → s ∩ u ⊆ t ∩ u)
```

4.1.2. Distributiva de la intersección

```
import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α : Type}
variable (s t u : Set α)

-- -----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)
-- ------------------------------------------------------------------------

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

```
-- Sea x ∈ s ∩ (t ∪ u). Entonces se tiene que
--    x ∈ s                                                  (1)
--    x ∈ t ∪ u                                              (2)
-- La relación (2) da lugar a dos casos.
--
-- Caso 1: Supongamos que x ∈ t. Entonces, por (1), x ∈ s ∩ t y, por
-- tanto, x ∈ (s ∩ t) ∪ (s ∩ u).
--
-- Caso 2: Supongamos que x ∈ u. Entonces, por (1), x ∈ s ∩ u y, por
-- tanto, x ∈ (s ∩ t) ∪ (s ∩ u).

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

-- 1ª demostración
-- ===============

example :
  s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u) :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ s ∩ (t ∪ u)
  -- ⊢ x ∈ s ∩ t ∪ s ∩ u
  rcases hx with ⟨hxs, hxtu⟩
  -- hxs : x ∈ s
  -- hxtu : x ∈ t ∪ u
  rcases hxtu with (hxt | hxu)
  . -- hxt : x ∈ t
    left
    -- ⊢ x ∈ s ∩ t
    constructor
    . -- ⊢ x ∈ s
      exact hxs
    . -- hxt : x ∈ t
      exact hxt
  . -- hxu : x ∈ u
    right
    -- ⊢ x ∈ s ∩ u
    constructor
    . -- ⊢ x ∈ s
      exact hxs
    . -- ⊢ x ∈ u
      exact hxu
```

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

example :
  s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u) :=
by
  rintro x ⟨hxs, hxt | hxu⟩
  -- x : α
  -- hxs : x ∈ s
  -- ⊢ x ∈ s ∩ t ∪ s ∩ u
  . -- hxt : x ∈ t
    left
    -- ⊢ x ∈ s ∩ t
    exact ⟨hxs, hxt⟩
  . -- hxu : x ∈ u
    right
    -- ⊢ x ∈ s ∩ u
    exact ⟨hxs, hxu⟩

-- 3ª demostración
-- ===============

example :
  s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u) :=
by
  rintro x ⟨hxs, hxt | hxu⟩
  -- x : α
  -- hxs : x ∈ s
  -- ⊢ x ∈ s ∩ t ∪ s ∩ u
  . -- hxt : x ∈ t
    exact Or.inl ⟨hxs, hxt⟩
  . -- hxu : x ∈ u
    exact Or.inr ⟨hxs, hxu⟩

-- 4ª demostración
-- ===============

example :
  s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u) :=
by
  intro x hx
  -- x : α
  -- hx : x ∈ s ∩ (t ∪ u)
  -- ⊢ x ∈ s ∩ t ∪ s ∩ u
```

```
  aesop

-- 5ª demostración
-- ===============

example :
  s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u) :=
by rw [inter_union_distrib_left]

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)
-- ----------------------------------------------------------------------

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

-- Sea x ∈ (s ∩ t) ∪ (s ∩ u). Entonces son posibles dos casos.
--
-- 1º caso: Supongamos que x ∈ s ∩ t. Entonces, x ∈ s y x ∈ t (y, por
-- tanto, x ∈ t ∪ u). Luego, x ∈ s ∩ (t ∪ u).
--
-- 2º caso: Supongamos que x ∈ s ∩ u. Entonces, x ∈ s y x ∈ u (y, por
-- tanto, x ∈ t ∪ u). Luego, x ∈ s ∩ (t ∪ u).

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

-- 1ª demostración
-- ===============

example : (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u):=
by
  intros x hx
  -- x : α
  -- hx : x ∈ (s ∩ t) ∪ (s ∩ u)
  -- ⊢ x ∈ s ∩ (t ∪ u)
  rcases hx with (xst | xsu)
  . -- xst : x ∈ s ∩ t
    constructor
    . -- ⊢ x ∈ s
      exact xst.1
    . -- ⊢ x ∈ t ∪ u
      left
      -- ⊢ x ∈ t
```

```
      exact xst.2
  . -- xsu : x ∈ s ∩ u
    constructor
    . -- ⊢ x ∈ s
      exact xsu.1
    . -- ⊢ x ∈ t ∪ u
      right
      -- ⊢ x ∈ u
      exact xsu.2

-- 2ª demostración
-- ===============

example : (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u):=
by
  rintro x (⟨xs, xt⟩ | ⟨xs, xu⟩)
  . -- x : α
    -- xs : x ∈ s
    -- xt : x ∈ t
    -- ⊢ x ∈ s ∩ (t ∪ u)
    use xs
    -- ⊢ x ∈ t ∪ u
    left
    -- ⊢ x ∈ t
    exact xt
  . -- x : α
    -- xs : x ∈ s
    -- xu : x ∈ u
    -- ⊢ x ∈ s ∩ (t ∪ u)
    use xs
    -- ⊢ x ∈ t ∪ u
    right
    -- ⊢ x ∈ u
    exact xu

-- 3ª demostración
-- ===============

example : (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u):=
by rw [inter_union_distrib_left s t u]

-- 4ª demostración
-- ===============

example : (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u):=
```

```
by
  intros x hx
  -- x : α
  -- hx : x ∈ s ∩ t ∪ s ∩ u
  -- ⊢ x ∈ s ∩ (t ∪ u)
  aesop

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

variable (a b : Prop)
variable (u : Set α)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (inter_union_distrib_left s t u : s ∩ (t ∪ u) = (s ∩ t) ∪ (s ∩ u))
```

4.1.3. Diferencia de diferencia

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (s \ t) \ u ⊆ s \ (t ∪ u)
-- ----------------------------------------------------------------------

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

-- Sea x ∈ (s \ t) \ u. Entonces, se tiene que
--    x ∈ s                                                      (1)
--    x ∉ t                                                      (2)
--    x ∉ u                                                      (3)
-- Tenemos que demostrar que
--    x ∈ s \ (t ∪ u)
-- pero, por (1), se reduce a
--    x ∉ t ∪ u
-- que se verifica por (2) y (3).

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

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set
```

```
variable {α : Type}
variable (s t u : Set α)

-- 1ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ (s \ t) \ u
  -- ⊢ x ∈ s \ (t ∪ u)
  rcases hx with ⟨hxst, hxnu⟩
  -- hxst : x ∈ s \ t
  -- hxnu : ¬x ∈ u
  rcases hxst with ⟨hxs, hxnt⟩
  -- hxs : x ∈ s
  -- hxnt : ¬x ∈ t
  constructor
  . -- ⊢ x ∈ s
    exact hxs
  . -- ⊢ ¬x ∈ t ∪ u
    by_contra hxtu
    -- hxtu : x ∈ t ∪ u
    -- ⊢ False
    rcases hxtu with (hxt | hxu)
    . -- hxt : x ∈ t
      apply hxnt
      -- ⊢ x ∈ t
      exact hxt
    . -- hxu : x ∈ u
      apply hxnu
      -- ⊢ x ∈ u
      exact hxu

-- 2ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  rintro x ⟨⟨hxs, hxnt⟩, hxnu⟩
  -- x : α
  -- hxnu : ¬x ∈ u
  -- hxs : x ∈ s
```

```
  -- hxnt : ¬x ∈ t
  -- ⊢ x ∈ s \ (t ∪ u)
  constructor
  . -- ⊢ x ∈ s
    exact hxs
  . -- ⊢ ¬x ∈ t ∪ u
    by_contra hxtu
    -- hxtu : x ∈ t ∪ u
    -- ⊢ False
    rcases hxtu with (hxt | hxu)
    . -- hxt : x ∈ t
      exact hxnt hxt
    . -- hxu : x ∈ u
      exact hxnu hxu

-- 3ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  rintro x ⟨⟨xs, xnt⟩, xnu⟩
  -- x : α
  -- xnu : ¬x ∈ u
  -- xs : x ∈ s
  -- xnt : ¬x ∈ t
  -- ⊢ x ∈ s \ (t ∪ u)
  use xs
  -- ⊢ ¬x ∈ t ∪ u
  rintro (xt | xu)
  . -- xt : x ∈ t
    -- ⊢ False
    contradiction
  . -- xu : x ∈ u
    -- ⊢ False
    contradiction

-- 4ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  rintro x ⟨⟨xs, xnt⟩, xnu⟩
  -- x : α
  -- xnu : ¬x ∈ u
  -- xs : x ∈ s
```

```
  -- xnt : ¬x ∈ t
  -- ⊢ x ∈ s \ (t ∪ u)
  use xs
  -- ⊢ ¬x ∈ t ∪ u
  rintro (xt | xu) <;> contradiction

-- 5ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  intro x xstu
  -- x : α
  -- xstu : x ∈ (s \ t) \ u
  -- ⊢ x ∈ s \ (t ∪ u)
  simp at *
  -- ⊢ x ∈ s ∧ ¬(x ∈ t ∨ x ∈ u)
  aesop

-- 6ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by
  intro x xstu
  -- x : α
  -- xstu : x ∈ (s \ t) \ u
  -- ⊢ x ∈ s \ (t ∪ u)
  aesop

-- 7ª demostración
-- ===============

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
by rw [diff_diff]

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s \ (t ∪ u) ⊆ (s \ t) \ u
-- ----------------------------------------------------------------------

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

-- Sea x ∈ s \ (t ∪ u). Entonces,
```

```
--    x ∈ s                                                         (1)
--    x ∉ t ∪ u                                                     (2)
-- Tenemos que demostrar que x ∈ (s \ t) \ u; es decir, que se verifican
-- las relaciones
--    x ∈ s \ t                                                     (3)
--    x ∉ u                                                         (4)
-- Para demostrar (3) tenemos que demostrar las relaciones
--    x ∈ s                                                         (5)
--    x ∉ t                                                         (6)
-- La (5) se tiene por la (1). Para demostrar la (6), supongamos que
-- x ∈ t; entonces, x ∈ t ∪ u, en contracción con (2). Para demostrar la
-- (4), supongamos que x ∈ u; entonces, x ∈ t ∪ u, en contracción con
-- (2).

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

-- 1ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ s \ (t ∪ u)
  -- ⊢ x ∈ (s \ t) \ u
  constructor
  . -- ⊢ x ∈ s \ t
    constructor
    . -- ⊢ x ∈ s
      exact hx.1
    . -- ⊢ ¬x ∈ t
      intro xt
      -- xt : x ∈ t
      -- ⊢ False
      apply hx.2
      -- ⊢ x ∈ t ∪ u
      left
      -- ⊢ x ∈ t
      exact xt
  . -- ⊢ ¬x ∈ u
    intro xu
    -- xu : x ∈ u
    -- ⊢ False
    apply hx.2
```

```
    -- ⊢ x ∈ t ∪ u
    right
    -- ⊢ x ∈ u
    exact xu

-- 2ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
by
  rintro x ⟨xs, xntu⟩
  -- x : α
  -- xs : x ∈ s
  -- xntu : ¬x ∈ t ∪ u
  -- ⊢ x ∈ (s \ t) \ u
  constructor
  . -- ⊢ x ∈ s \ t
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ¬x ∈ t
      intro xt
      -- xt : x ∈ t
      -- ⊢ False
      exact xntu (Or.inl xt)
  . -- ⊢ ¬x ∈ u
    intro xu
    -- xu : x ∈ u
    -- ⊢ False
    exact xntu (Or.inr xu)

-- 2ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
  fun _ ⟨xs, xntu⟩ ↦ ⟨⟨xs, fun xt ↦ xntu (Or.inl xt)⟩,
                     fun xu ↦ xntu (Or.inr xu)⟩

-- 4ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
by
  rintro x ⟨xs, xntu⟩
  -- x : α
```

```
  -- xs : x ∈ s
  -- xntu : ¬x ∈ t ∪ u
  -- ⊢ x ∈ (s \ t) \ u
  aesop

-- 5ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
by intro ; aesop

-- 6ª demostración
-- ===============

example : s \ (t ∪ u) ⊆ (s \ t) \ u :=
by rw [diff_diff]

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

variable (a b : Prop)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (diff_diff : (s \ t) \ u = s \ (t ∪ u))
```

4.1.4. Conmutativa de la intersección

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∩ t = t ∩ s
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que
--    (∀ x)[x ∈ s ∩ t ↔ x ∈ t ∩ s]
-- Demostratemos la equivalencia por la doble implicación.
--
-- Sea x ∈ s ∩ t. Entonces, se tiene
--    x ∈ s                                                      (1)
--    x ∈ t                                                      (2)
-- Luego x ∈ t ∩ s (por (2) y (1)).
```

```
--
-- La segunda implicación se demuestra análogamente.

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

import Mathlib.Data.Set.Basic
open Set

variable {α : Type}
variable (s t : Set α)

-- 1ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ t ↔ x ∈ t ∩ s
  simp only [mem_inter_iff]
  -- ⊢ x ∈ s ∧ x ∈ t ↔ x ∈ t ∧ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∧ x ∈ t → x ∈ t ∧ x ∈ s
    intro h
    -- h : x ∈ s ∧ x ∈ t
    -- ⊢ x ∈ t ∧ x ∈ s
    constructor
    . -- ⊢ x ∈ t
      exact h.2
    . -- ⊢ x ∈ s
      exact h.1
  . -- ⊢ x ∈ t ∧ x ∈ s → x ∈ s ∧ x ∈ t
    intro h
    -- h : x ∈ t ∧ x ∈ s
    -- ⊢ x ∈ s ∧ x ∈ t
    constructor
    . -- ⊢ x ∈ s
      exact h.2
    . -- ⊢ x ∈ t
      exact h.1

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

```
example : s ∩ t = t ∩ s :=
by
  ext
  -- x : α
  -- ⊢ x ∈ s ∩ t ↔ x ∈ t ∩ s
  simp only [mem_inter_iff]
  -- ⊢ x ∈ s ∧ x ∈ t ↔ x ∈ t ∧ x ∈ s
  exact ⟨fun h ↦ ⟨h.2, h.1⟩,
         fun h ↦ ⟨h.2, h.1⟩⟩

-- 3ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
by
  ext
  -- x : α
  -- ⊢ x ∈ s ∩ t ↔ x ∈ t ∩ s
  exact ⟨fun h ↦ ⟨h.2, h.1⟩,
         fun h ↦ ⟨h.2, h.1⟩⟩

-- 4ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ t ↔ x ∈ t ∩ s
  simp only [mem_inter_iff]
  -- ⊢ x ∈ s ∧ x ∈ t ↔ x ∈ t ∧ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∧ x ∈ t → x ∈ t ∧ x ∈ s
    rintro ⟨xs, xt⟩
    -- xs : x ∈ s
    -- xt : x ∈ t
    -- ⊢ x ∈ t ∧ x ∈ s
    exact ⟨xt, xs⟩
  . -- ⊢ x ∈ t ∧ x ∈ s → x ∈ s ∧ x ∈ t
    rintro ⟨xt, xs⟩
    -- xt : x ∈ t
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∧ x ∈ t
    exact ⟨xs, xt⟩
```

```
-- 5ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ t ↔ x ∈ t ∩ s
  simp only [mem_inter_iff]
  -- ⊢ x ∈ s ∧ x ∈ t ↔ x ∈ t ∧ x ∈ s
  simp only [And.comm]

-- 6ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
ext (fun _ ↦ And.comm)

-- 7ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
by ext ; simp [And.comm]

-- 8ª demostración
-- ===============

example : s ∩ t = t ∩ s :=
inter_comm s t

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

variable (x : α)
variable (a b : Prop)
#check (And.comm : a ∧ b ↔ b ∧ a)
#check (inter_comm s t : s ∩ t = t ∩ s)
#check (mem_inter_iff x s t : x ∈ s ∩ t ↔ x ∈ s ∧ x ∈ t)
```

4.1.5. Identidades conjuntistas

```
import Mathlib.Data.Set.Basic
import Mathlib.Tactic
open Set

variable {α : Type}
variable (s t : Set α)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∩ (s ∪ t) = s
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que
--    (∀ x)[x ∈ s ∩ (s ∪ t) ↔ x ∈ s]
-- y lo haremos demostrando las dos implicaciones.
--
-- (⟹) Sea x ∈ s ∩ (s ∪ t). Entonces, x ∈ s.
--
-- (⟸) Sea x ∈ s. Entonces, x ∈ s ∪ t y, por tanto,
-- x ∈ s ∩ (s ∪ t).

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

-- 1ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ (s ∪ t) ↔ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∩ (s ∪ t) → x ∈ s
    intros h
  -- h : x ∈ s ∩ (s ∪ t)
  -- ⊢ x ∈ s
    exact h.1
  . -- ⊢ x ∈ s → x ∈ s ∩ (s ∪ t)
    intro xs
```

```
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∩ (s ∪ t)
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ x ∈ s ∪ t
      left
      -- ⊢ x ∈ s
      exact xs

-- 2ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ (s ∪ t) ↔ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∩ (s ∪ t) → x ∈ s
    intro h
    -- h : x ∈ s ∩ (s ∪ t)
    -- ⊢ x ∈ s
    exact h.1
  . -- ⊢ x ∈ s → x ∈ s ∩ (s ∪ t)
    intro xs
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∩ (s ∪ t)
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ x ∈ s ∪ t
      exact (Or.inl xs)

-- 3ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by
  ext
  -- x : α
  -- ⊢ x ∈ s ∩ (s ∪ t) ↔ x ∈ s
  exact ⟨fun h ↦ h.1,
         fun xs ↦ ⟨xs, Or.inl xs⟩⟩
```

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

example : s ∩ (s ∪ t) = s :=
by
  ext
  -- x : α
  -- ⊢ x ∈ s ∩ (s ∪ t) ↔ x ∈ s
  exact ⟨And.left,
         fun xs ↦ ⟨xs, Or.inl xs⟩⟩

-- 5ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ (s ∪ t) ↔ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∩ (s ∪ t) → x ∈ s
    rintro ⟨xs, -⟩
    -- xs : x ∈ s
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ x ∈ s → x ∈ s ∩ (s ∪ t)
    intro xs
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∩ (s ∪ t)
    use xs
    -- ⊢ x ∈ s ∪ t
    left
    -- ⊢ x ∈ s
    exact xs

-- 6ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by
  apply subset_antisymm
  . -- ⊢ s ∩ (s ∪ t) ⊆ s
    rintro x ⟨hxs, -⟩
    -- x : α
    -- hxs : x ∈ s
```

```
    -- ⊢ x ∈ s
    exact hxs
  . -- ⊢ s ⊆ s ∩ (s ∪ t)
    intros x hxs
    -- x : α
    -- hxs : x ∈ s
    -- ⊢ x ∈ s ∩ (s ∪ t)
    exact ⟨hxs, Or.inl hxs⟩

-- 7ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
inf_sup_self

-- 8ª demostración
-- ===============

example : s ∩ (s ∪ t) = s :=
by aesop

-- ------------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∪ (s ∩ t) = s
-- -------------------------------------------------------------------------

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

-- Tenemos que demostrar que
--    (∀ x)[x ∈ s ∪ (s ∩ t) ↔ x ∈ s]
-- y lo haremos demostrando las dos implicaciones.
--
-- (⟹) Sea x ∈ s ∪ (s ∩ t). Entonces, c ∈ s o x ∈ s ∩ t. En ambos casos,
-- x ∈ s.
--
-- (⟸) Sea x ∈ s. Entonces, x ∈ s ∩ t y, por tanto, x ∈ s ∪ (s ∩ t).

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

-- 1ª demostración
-- ===============

example : s ∪ (s ∩ t) = s :=
```

```
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ (s ∩ t) ↔ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∪ (s ∩ t) → x ∈ s
    intro hx
    -- hx : x ∈ s ∪ (s ∩ t)
    -- ⊢ x ∈ s
    rcases hx with (xs | xst)
    . -- xs : x ∈ s
      exact xs
    . -- xst : x ∈ s ∩ t
      exact xst.1
  . -- ⊢ x ∈ s → x ∈ s ∪ (s ∩ t)
    intro xs
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∪ (s ∩ t)
    left
    -- ⊢ x ∈ s
    exact xs

-- 2ª demostración
-- ===============

example : s ∪ (s ∩ t) = s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ s ∩ t ↔ x ∈ s
  exact ⟨fun hx ↦ Or.elim hx id And.left,
         fun xs ↦ Or.inl xs⟩

-- 3ª demostración
-- ===============

example : s ∪ (s ∩ t) = s :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ (s ∩ t) ↔ x ∈ s
  constructor
  . -- ⊢ x ∈ s ∪ (s ∩ t) → x ∈ s
    rintro (xs | ⟨xs, -⟩) <;>
    -- xs : x ∈ s
```

```
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ x ∈ s → x ∈ s ∪ (s ∩ t)
    intro xs
    -- xs : x ∈ s
    -- ⊢ x ∈ s ∪ s ∩ t
    left
    -- ⊢ x ∈ s
    exact xs

-- 4ª demostración
-- ===============

example : s ∪ (s ∩ t) = s :=
sup_inf_self

-- ------------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (s \ t) ∪ t = s ∪ t
-- ------------------------------------------------------------------------

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

-- Tenemos que demostrar que
--    (∀ x)[x ∈ (s \ t) ∪ t ↔ x ∈ s ∪ t]
-- y lo demostraremos por la siguiente cadena de equivalencias:
--    x ∈ (s \ t) ∪ t ↔ x ∈ (s \ t) ∨ (x ∈ t)
--                    ↔ (x ∈ s ∧ x ∉ t) ∨ x ∈ t
--                    ↔ (x ∈ s ∨ x ∈ t) ∧ (x ∉ t ∨ x ∈ t)
--                    ↔ x ∈ s ∨ x ∈ t
--                    ↔ x ∈ s ∪ t

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

-- 1ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ t ↔ x ∈ s ∪ t
  calc x ∈ (s \ t) ∪ t
```

```
        ↔ x ∈ s \ t ∨ x ∈ t                     := mem_union x (s \ t) t
      _ ↔ (x ∈ s ∧ x ∉ t) ∨ x ∈ t               := by simp only [mem_diff x]
      _ ↔ (x ∈ s ∨ x ∈ t) ∧ (x ∉ t ∨ x ∈ t) := and_or_right
      _ ↔ (x ∈ s ∨ x ∈ t) ∧ True                := by simp only [em' (x ∈ t)]
      _ ↔ x ∈ s ∨ x ∈ t                         := (and_true (x ∈ s ∨ x ∈ t)).to_iff
      _ ↔ x ∈ s ∪ t                             := (mem_union x s t).symm

-- 2ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ t ↔ x ∈ s ∪ t
  constructor
  . -- ⊢ x ∈ (s \ t) ∪ t → x ∈ s ∪ t
    intro hx
    -- hx : x ∈ (s \ t) ∪ t
    -- ⊢ x ∈ s ∪ t
    rcases hx with (xst | xt)
    . -- xst : x ∈ s \ t
      -- ⊢ x ∈ s ∪ t
      left
      -- ⊢ x ∈ s
      exact xst.1
    . -- xt : x ∈ t
      -- ⊢ x ∈ s ∪ t
      right
      -- ⊢ x ∈ t
      exact xt
  . -- ⊢ x ∈ s ∪ t → x ∈ (s \ t) ∪ t
    by_cases h : x ∈ t
    . -- h : x ∈ t
      intro _xst
      -- _xst : x ∈ s ∪ t
      right
      -- ⊢ x ∈ t
      exact h
    . -- ⊢ x ∈ s ∪ t → x ∈ (s \ t) ∪ t
      intro hx
      -- hx : x ∈ s ∪ t
      -- ⊢ x ∈ (s \ t) ∪ t
      rcases hx with (xs | xt)
      . -- xs : x ∈ s
```

```
        left
        -- ⊢ x ∈ s \ t
        constructor
        . -- ⊢ x ∈ s
          exact xs
        . -- ⊢ ¬x ∈ t
          exact h
      . -- xt : x ∈ t
        right
        -- ⊢ x ∈ t
        exact xt

-- 3ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ t ↔ x ∈ s ∪ t
  constructor
  . -- ⊢ x ∈ (s \ t) ∪ t → x ∈ s ∪ t
    rintro (⟨xs, -⟩ | xt)
    . -- xs : x ∈ s
      -- ⊢ x ∈ s ∪ t
      left
      -- ⊢ x ∈ s
      exact xs
    . -- xt : x ∈ t
      -- ⊢ x ∈ s ∪ t
      right
      -- ⊢ x ∈ t
      exact xt
  . -- ⊢ x ∈ s ∪ t → x ∈ (s \ t) ∪ t
    by_cases h : x ∈ t
    . -- h : x ∈ t
      intro _xst
      -- _xst : x ∈ s ∪ t
      -- ⊢ x ∈ (s \ t) ∪ t
      right
      -- ⊢ x ∈ t
      exact h
    . -- ⊢ x ∈ s ∪ t → x ∈ (s \ t) ∪ t
      rintro (xs | xt)
      . -- xs : x ∈ s
```

```
        -- ⊢ x ∈ (s \ t) ∪ t
        left
        -- ⊢ x ∈ s \ t
        exact ⟨xs, h⟩
      . -- xt : x ∈ t
        -- ⊢ x ∈ (s \ t) ∪ t
        right
        -- ⊢ x ∈ t
        exact xt

-- 4ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
diff_union_self

-- 5ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
by
  ext
  -- x : α
  -- ⊢ x ∈ s \ t ∪ t ↔ x ∈ s ∪ t
  simp

-- 6ª demostración
-- ===============

example : (s \ t) ∪ t = s ∪ t :=
by simp

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que, para todo x,
--    x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
-- Se demuestra mediante la siguiente cadena de equivalencias:
--    x ∈ (s \ t) ∪ (t \ s)
--    ↔ x ∈ (s \ t) ∨ x ∈ (t \ s)
```

```
--    ↔ (x ∈ s ∧ x ∉ t) ∨ x ∈ (t \ s)
--    ↔ (x ∈ s ∨ x ∈ (t \ s)) ∧ (x ∉ t ∨ x ∈ (t \ s))
--    ↔ (x ∈ s ∨ (x ∈ t ∧ x ∉ s)) ∧ (x ∉ t ∨ (x ∈ t ∧ x ∉ s))
--    ↔ ((x ∈ s ∨ x ∈ t) ∧ (x ∈ s ∨ x ∉ s)) ∧ ((x ∉ t ∨ x ∈ t) ∧ (x ∉ t ∨ x ∉ s))
--    ↔ ((x ∈ s ∨ x ∈ t) ∧ True) ∧ (True ∧ (x ∉ t ∨ x ∉ s))
--    ↔ (x ∈ s ∨ x ∈ t) ∧ (x ∉ t ∨ x ∉ s)
--    ↔ (x ∈ s ∪ t) ∧ (x ∉ t ∨ x ∉ s)
--    ↔ (x ∈ s ∪ t) ∧ (x ∉ s ∨ x ∉ t)
--    ↔ (x ∈ s ∪ t) ∧ ¬(x ∈ s ∧ x ∈ t)
--    ↔ (x ∈ s ∪ t) ∧ ¬(x ∈ s ∩ t)
--    ↔ x ∈ (s ∪ t) \ (s ∩ t)

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

-- 1ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  calc x ∈ (s \ t) ∪ (t \ s)
     ↔ x ∈ (s \ t) ∨ x ∈ (t \ s) :=
         by exact mem_union x (s \ t) (t \ s)
   _ ↔ (x ∈ s ∧ x ∉ t) ∨ x ∈ (t \ s) :=
         by simp only [mem_diff]
   _ ↔ (x ∈ s ∨ x ∈ (t \ s)) ∧ (x ∉ t ∨ x ∈ (t \ s)) :=
         by exact and_or_right
   _ ↔ (x ∈ s ∨ (x ∈ t ∧ x ∉ s)) ∧ (x ∉ t ∨ (x ∈ t ∧ x ∉ s)) :=
         by simp only [mem_diff]
   _ ↔ ((x ∈ s ∨ x ∈ t) ∧ (x ∈ s ∨ x ∉ s)) ∧
       ((x ∉ t ∨ x ∈ t) ∧ (x ∉ t ∨ x ∉ s)) :=
         by simp_all only [or_and_left]
   _ ↔ ((x ∈ s ∨ x ∈ t) ∧ True) ∧
       (True ∧ (x ∉ t ∨ x ∉ s)) :=
         by simp only [em (x ∈ s), em' (x ∈ t)]
   _ ↔ (x ∈ s ∨ x ∈ t) ∧ (x ∉ t ∨ x ∉ s) :=
         by simp only [and_true (x ∈ s ∨ x ∈ t),
                       true_and (¬x ∈ t ∨ ¬x ∈ s)]
   _ ↔ (x ∈ s ∪ t) ∧ (x ∉ t ∨ x ∉ s) :=
         by simp only [mem_union]
   _ ↔ (x ∈ s ∪ t) ∧ (x ∉ s ∨ x ∉ t) :=
         by simp only [or_comm]
```

```
  _ ↔ (x ∈ s ∪ t) ∧ ¬(x ∈ s ∧ x ∈ t) :=
        by simp only [not_and_or]
  _ ↔ (x ∈ s ∪ t) ∧ ¬(x ∈ s ∩ t) :=
        by simp only [mem_inter_iff]
  _ ↔ x ∈ (s ∪ t) \ (s ∩ t)     :=
        by simp only [mem_diff]

-- 2ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  constructor
  . -- ⊢ x ∈ (s \ t) ∪ (t \ s) → x ∈ (s ∪ t) \ (s ∩ t)
    rintro (⟨xs, xnt⟩ | ⟨xt, xns⟩)
    . -- xs : x ∈ s
      -- xnt : ¬x ∈ t
      -- ⊢ x ∈ (s ∪ t) \ (s ∩ t)
      constructor
      . -- ⊢ x ∈ s ∪ t
        left
        -- ⊢ x ∈ s
        exact xs
      . -- ⊢ ¬x ∈ s ∩ t
        rintro ⟨-, xt⟩
        -- xt : x ∈ t
        -- ⊢ False
        exact xnt xt
    . -- xt : x ∈ t
      -- xns : ¬x ∈ s
      -- ⊢ x ∈ (s ∪ t) \ (s ∩ t)
      constructor
      . -- ⊢ x ∈ s ∪ t
        right
        -- ⊢ x ∈ t
        exact xt
      . -- ⊢ ¬x ∈ s ∩ t
        rintro ⟨xs, -⟩
        -- xs : x ∈ s
        -- ⊢ False
        exact xns xs
  . -- ⊢ x ∈ (s ∪ t) \ (s ∩ t) → x ∈ (s \ t) ∪ (t \ s)
```

```
    rintro ⟨xs | xt, nxst⟩
    . -- xs : x ∈ s
      -- ⊢ x ∈ (s \ t) ∪ (t \ s)
      left
      -- ⊢ x ∈ s \ t
      use xs
      -- ⊢ ¬x ∈ t
      intro xt
      -- xt : x ∈ t
      -- ⊢ False
      apply nxst
      -- ⊢ x ∈ s ∩ t
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ t
        exact xt
    . -- nxst : ¬x ∈ s ∩ t
      -- xt : x ∈ t
      -- ⊢ x ∈ (s \ t) ∪ (t \ s)
      right
      -- ⊢ x ∈ t \ s
      use xt
      -- ⊢ ¬x ∈ s
      intro xs
      -- xs : x ∈ s
      -- ⊢ False
      apply nxst
      -- ⊢ x ∈ s ∩ t
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ t
        exact xt

-- 3ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  constructor
  . -- ⊢ x ∈ (s \ t) ∪ (t \ s) → x ∈ (s ∪ t) \ (s ∩ t)
```

```
    rintro (⟨xs, xnt⟩ | ⟨xt, xns⟩)
    . -- xt : x ∈ t
      -- xns : ¬x ∈ s
      -- ⊢ x ∈ (s ∪ t) \ (s ∩ t)
      aesop
    . -- xt : x ∈ t
      -- xns : ¬x ∈ s
      -- ⊢ x ∈ (s ∪ t) \ (s ∩ t)
      aesop
  . rintro ⟨xs | xt, nxst⟩
    . -- xs : x ∈ s
      -- ⊢ x ∈ (s \ t) ∪ (t \ s)
      aesop
    . -- nxst : ¬x ∈ s ∩ t
      -- xt : x ∈ t
      -- ⊢ x ∈ (s \ t) ∪ (t \ s)
      aesop

-- 4ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  constructor
  . -- ⊢ x ∈ (s \ t) ∪ (t \ s) → x ∈ (s ∪ t) \ (s ∩ t)
    rintro (⟨xs, xnt⟩ | ⟨xt, xns⟩) <;> aesop
  . -- ⊢ x ∈ (s ∪ t) \ (s ∩ t) → x ∈ (s \ t) ∪ (t \ s)
    rintro ⟨xs | xt, nxst⟩ <;> aesop

-- 5ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext
  constructor
  . aesop
  . aesop

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

```
example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  ext
  constructor <;> aesop

-- 7ª demostración
-- ===============

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by
  rw [Set.ext_iff]
  -- ⊢ ∀ (x : α), x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  intro
  -- x : α
  -- ⊢ x ∈ (s \ t) ∪ (t \ s) ↔ x ∈ (s ∪ t) \ (s ∩ t)
  rw [iff_def]
  -- ⊢ (x ∈ (s \ t) ∪ (t \ s) → x ∈ (s ∪ t) \ (s ∩ t)) ∧
  --   (x ∈ (s ∪ t) \ (s ∩ t) → x ∈ (s \ t) ∪ (t \ s))
  aesop

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

variable (x : α)
variable (a b c : Prop)
#check (And.left : a ∧ b → a)
#check (Or.elim : a ∨ b → (a → c) → (b → c) → c)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (Set.ext_iff : s = t ↔ ∀ (x : α), x ∈ s ↔ x ∈ t)
#check (and_or_right : (a ∧ b) ∨ c ↔ (a ∨ c) ∧ (b ∨ c))
#check (and_true a : (a ∧ True) = a)
#check ((and_true a).to_iff : (a ∧ True) ↔ a)
#check (diff_union_self : (s \ t) ∪ t = s ∪ t)
#check (em a : a ∨ ¬ a)
#check (em' a : ¬ a ∨ a)
#check (iff_def : (a ↔ b) ↔ (a → b) ∧ (b → a))
#check (inf_sup_self : s ∩ (s ∪ t) = s)
#check (mem_diff x : x ∈ s \ t ↔ x ∈ s ∧ ¬x ∈ t)
#check (mem_inter_iff x s t : x ∈ s ∩ t ↔ x ∈ s ∧ x ∈ t)
#check (mem_union x s t : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
#check (not_and_or : ¬(a ∧ b) ↔ ¬a ∨ ¬b)
#check (or_and_left : a ∨ (b ∧ c) ↔ (a ∨ b) ∧ (a ∨ c))
#check (or_comm : a ∨ b ↔ b ∨ a)
#check (subset_antisymm : s ⊆ t → t ⊆ s → s = t)
```

```
#check (sup_inf_self : s ∪ (s ∩ t) = s)
#check (true_and a : (True ∧ a) = a)
```

4.1.6. Unión de pares e impares

```
-- ---------------------------------------------------------------------
-- Ejercicio. Ejecutar las siguientes acciones
-- 1. Importar la librería data.set.basic data.nat.parity
-- 2. Abrir los espacios de nombres set y nat.
-- ----------------------------------------------------------------------

import Mathlib.Tactic
import Mathlib.Algebra.Ring.Parity

open Set Nat

-- ---------------------------------------------------------------------
-- Ejercicio. Definir el conjunto de los números naturales.
-- ----------------------------------------------------------------------

def Naturales : Set ℕ := {_n | True}

-- ---------------------------------------------------------------------
-- Ejercicio. Definir el conjunto de los números pares.
-- ----------------------------------------------------------------------

def Pares     : Set ℕ := {n | Even n}

-- ---------------------------------------------------------------------
-- Ejercicio. Definir el conjunto de los números impares.
-- ----------------------------------------------------------------------

def Impares   : Set ℕ := {n | ¬Even n}

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    Pares ∪ Impares = Naturales
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que
```

```
--    {n | Even n} ∪ {n | ¬Even n} = {n | True}
-- es decir,
--    n ∈ {n | Even n} ∪ {n | ¬Even n} ↔ n ∈ {n | True}
-- que se reduce a
--    ⊢ Even n ∨ ¬Even n
-- que es una tautología.

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

-- 1ª demostración
-- ===============

example : Pares ∪ Impares = Naturales :=
by
  unfold Pares Impares Naturales
  -- ⊢ {n | Even n} ∪ {n | ¬Even n} = {n | True}
  ext n
  -- n : ℕ
  -- ⊢ n ∈ {n | Even n} ∪ {n | ¬Even n} ↔ n ∈ {n | True}
  simp only [Set.mem_setOf_eq, iff_true]
  -- ⊢ n ∈ {n | Even n} ∪ {n | ¬Even n}
  exact em (Even n)

-- 2ª demostración
-- ===============

example : Pares ∪ Impares = Naturales :=
by
  unfold Pares Impares Naturales
  -- ⊢ {n | Even n} ∪ {n | ¬Even n} = {n | True}
  ext n
  -- n : ℕ
  -- ⊢ n ∈ {n | Even n} ∪ {n | ¬Even n} ↔ n ∈ {n | True}
  tauto

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

variable (x : ℕ)
variable (a : Prop)
variable (p : ℕ → Prop)
#check (Set.mem_setOf_eq : (x ∈ {y : ℕ | p y}) = p x)
#check (em a : a ∨ ¬ a)
#check (iff_true a : (a ↔ True) = a)
```

4.1.7. Pertenencia al vacío y al universal

```
-- ---------------------------------------------------------------------
-- Ejercicio. Abrir el espacion de nombres set.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

open Set

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar ningún elemento pertenece al vacío.
-- ----------------------------------------------------------------------

example (x : ℕ) : x ∉ (∅ : Set ℕ) :=
not_false

example (x : ℕ) (h : x ∈ (∅ : Set ℕ)) : false :=
False.elim h

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar tofos los elementos pertenecen al universal.
-- ----------------------------------------------------------------------

example (x : ℕ) : x ∈ (univ : Set ℕ) :=
trivial

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

variable (a : Prop)
#check (False.elim : False → a)
#check (not_false : ¬False)
#check (trivial : True)
```

4.1.8. Primos mayores que dos

```
-- ---------------------------------------------------------------------
-- Ejercicio. Los números primos, los mayores que 2 y los impares se
-- definen por
--    def Primos      : Set ℕ := {n | Nat.Prime n}
--    def MayoresQue2 : Set ℕ := {n | n > 2}
--    def Impares     : Set ℕ := {n | ¬Even n}
```

```
--
-- Demostrar que
--    Primos ∩ MayoresQue2 ⊆ Impares
-- ----------------------------------------------------------------------

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

import Mathlib.Algebra.Ring.Parity
import Mathlib.Tactic

open Nat

def Primos      : Set ℕ := {n | Nat.Prime n}
def MayoresQue2 : Set ℕ := {n | n > 2}
def Impares     : Set ℕ := {n | ¬Even n}

-- 1ª demostración
-- ===============

example : Primos ∩ MayoresQue2 ⊆ Impares :=
by
  unfold Primos MayoresQue2 Impares
  -- ⊢ {n | Nat.Prime n} ∩ {n | n > 2} ⊆ {n | ¬Even n}
  intro n
  -- n : ℕ
  -- ⊢ n ∈ {n | Nat.Prime n} ∩ {n | n > 2} → n ∈ {n | ¬Even n}
  simp
  -- ⊢ Nat.Prime n → 2 < n → Odd n
  intro hn
  -- hn : Nat.Prime n
  -- ⊢ 2 < n → Odd n
  rcases Prime.eq_two_or_odd hn with (h | h)
  . -- h : n = 2
    rw [h]
    -- ⊢ 2 < 2 → ¬Even 2
    intro h1
    -- h1 : 2 < 2
    -- ⊢ Odd 2
    exfalso
    exact absurd h1 (lt_irrefl 2)
  . -- h : n % 2 = 1
    intro
    -- a : 2 < n
    -- ⊢ Odd n
```

```
    exact odd_iff.mpr h

-- 2ª demostración
-- ===============

example : Primos ∩ MayoresQue2 ⊆ Impares :=
by
  unfold Primos MayoresQue2 Impares
  -- ⊢ {n | Nat.Prime n} ∩ {n | n > 2} ⊆ {n | ¬Even n}
  rintro n ⟨h1, h2⟩
  -- n : ℕ
  -- h1 : n ∈ {n | Nat.Prime n}
  -- h2 : n ∈ {n | n > 2}
  -- ⊢ n ∈ {n | ¬Even n}
  simp at *
  -- h1 : Nat.Prime n
  -- h2 : 2 < n
  -- ⊢ ¬Even n
  rcases Prime.eq_two_or_odd h1 with (h3 | h4)
  . -- h3 : n = 2
    rw [h3] at h2
    -- h2 : 2 < 2
    exfalso
    -- ⊢ False
    exact absurd h2 (lt_irrefl 2)
  . -- h4 : n % 2 = 1
    exact odd_iff.mpr h4

-- 3ª demostración
-- ===============

example : Primos ∩ MayoresQue2 ⊆ Impares :=
by
  unfold Primos MayoresQue2 Impares
  -- ⊢ {n | Nat.Prime n} ∩ {n | n > 2} ⊆ {n | ¬Even n}
  rintro n ⟨h1, h2⟩
  -- n : ℕ
  -- h1 : n ∈ {n | Nat.Prime n}
  -- h2 : n ∈ {n | n > 2}
  -- ⊢ n ∈ {n | ¬Even n}
  simp at *
  -- h1 : Nat.Prime n
  -- h2 : 2 < n
  -- ⊢ ¬Even n
  rcases Prime.eq_two_or_odd h1 with (h3 | h4)
```

```
  . -- h3 : n = 2
    rw [h3] at h2
    -- h2 : 2 < 2
    linarith
  . -- h4 : n % 2 = 1
    exact odd_iff.mpr h4

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

variable (p n : ℕ)
variable (a b : Prop)
#check (Prime.eq_two_or_odd : Nat.Prime p → p = 2 ∨ p % 2 = 1)
#check (absurd : a → ¬a → b)
#check (even_iff : Even n ↔ n % 2 = 0)
#check (lt_irrefl n : ¬n < n)
#check (odd_iff : Odd n ↔ n % 2 = 1)
#check (one_ne_zero : 1 ≠ 0)
```

4.1.9. Definiciones de primo

```
import Mathlib.Algebra.Prime.Defs
import Mathlib.Data.Nat.Prime.Defs
variable (p : ℕ)

-- ---------------------------------------------------------------------
-- Ejercicio. En Mathlib hay dos definiciones de primo:
-- + (Nat.Prime p) indica que el número natural p es primo.
-- + (Prime p) indica que el número p (de un monoide conmutativo con
--    cero) es primo.
--
-- Demostrar que en los números naturales ambos conceptos coinciden.
-- ---------------------------------------------------------------------

example : Nat.Prime p ↔ Prime p :=
  Nat.prime_iff

example
  (h : Prime p)
  : Nat.Prime p :=
by
  rw [Nat.prime_iff]
  -- ⊢ Prime p
```

```
  exact h

example
  (h : Prime p)
  : Nat.Prime p :=
by
  rwa [Nat.prime_iff]

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

#check (Nat.prime_iff : Nat.Prime p ↔ Prime p)
```

4.1.10. Ejemplos con cuantificadores acotados

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería data.nat.prime data.nat.parity
-- 2. Abrir el espacio de nombres nat.
-- 3. Declarar s como variable sobre conjuntos de números naturales.
-- ----------------------------------------------------------------------

import Mathlib.Data.Nat.Prime.Basic     -- 1
open Nat                                -- 2
variable (s : Set ℕ)                    -- 3

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si los elementos de s no son pares y si los
-- elementos de s son primos, entonces los elementos de s no son pares
-- pero sí son primpos.
-- ----------------------------------------------------------------------

example
  (h₀ : ∀ x ∈ s, ¬ Even x)
  (h₁ : ∀ x ∈ s, Nat.Prime x)
  : ∀ x ∈ s, ¬ Even x ∧ Nat.Prime x :=
by
  intros x xs
  -- x : ℕ
  -- xs : x ∈ s
  -- ⊢ ¬Even x ∧ Nat.Prime x
  constructor
  . -- ⊢ ¬Even x
```

```
    apply h₀ x xs
  . -- ⊢ Nat.Prime x
    apply h₁ x xs

-- Comentario: La táctica (intros x xs) si la conclusión es (∀ y ∈ s, P y)
-- y una hipótesis es (s : Set α), entonces añade las hipótesis (x : α)
-- y (xs : x ∈ s) y cambia la conclusión a (P x).

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si s tiene algún elemento primo impar,
-- entonces tiene algún elemento primo.
-- ----------------------------------------------------------------------

example
  (h : ∃ x ∈ s, ¬ Even x ∧ Nat.Prime x)
  : ∃ x ∈ s, Nat.Prime x :=
by
  rcases h with ⟨x, xs, -, Nat.Prime_x⟩
  -- x : ℕ
  -- xs : x ∈ s
  -- Nat.Prime_x : Nat.Prime x
  -- ⊢ ∃ x, x ∈ s ∧ Nat.Prime x
  use x, xs

-- Comentarios:
-- 1. La táctica (cases h with ⟨x, xs, h1, h2⟩) si la
--    hipótesis es (∃ y ∈ s, P y ∧ Q y) y una hipótesis es (s : Set α),
--    entonces quita h y añade las hipótesis (x : s), (h1 : P x) y
--    (h2 : Q x).
-- 2. La táctica (use x, xs) resuelve la conclusión
--    (∃ x ∈ s, P x) si xs es una prueba de (x ∈ s) y h lo es de (P x).
```

4.1.11. Ejercicios con cuantificadores acotados

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería data.nat.prime data.nat.parity
-- 2. Abrir el espacio de nombres nat
-- 3. Declarar s y t como variables sobre conjuntos de números naturales.
-- 4. Declarar el hecho (ssubt : s ⊆ t)
-- 5, Añadir ssubt como hipótesis de la teoría.
-- ----------------------------------------------------------------------
```

```
import Mathlib.Data.Nat.Prime.Basic

open Nat                              -- 2

variable (s t : Set ℕ)                -- 3
variable (ssubt : s ⊆ t)              -- 4

include ssubt                         -- 5

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    ∀ x ∈ t, ¬ Even x
--    ∀ x ∈ t, Nat.Prime x
-- entonces
--    ∀ x ∈ s, ¬ Even x ∧ Nat.Prime x
-- ----------------------------------------------------------------------

example
  (h₀ : ∀ x ∈ t, ¬Even x)
  (h₁ : ∀ x ∈ t, Nat.Prime x)
  : ∀ x ∈ s, ¬Even x ∧ Nat.Prime x :=
by
  intro x xs
  -- x : ℕ
  -- xs : x ∈ s
  -- ⊢ ¬Even x ∧ Nat.Prime x
  constructor
  · -- ⊢ ¬Even x
    apply h₀ x (ssubt xs)
  . -- ⊢ Nat.Prime x
    apply h₁ x (ssubt xs)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    ∃ x ∈ s, ¬ Even x ∧ Nat.Prime x
-- entonces
--    ∃ x ∈ t, Nat.Prime x
-- ----------------------------------------------------------------------

example (h : ∃ x ∈ s, ¬Even x ∧ Nat.Prime x) : ∃ x ∈ t, Nat.Prime x :=
by
  rcases h with ⟨x, xs, -, px⟩
  -- x : ℕ
  -- xs : x ∈ s
  -- px : Nat.Prime x
```

```
  -- ⊢ ∃ x ∈ t, Nat.Prime x
  use x, ssubt xs
```

4.1.12. Ejemplos de uniones e intersecciones generales

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones
-- 1. Importar las librerías Tactic, Set.Basic y Set.Lattice
-- 2. Abrir el espacio de nombres Set
-- 3. Declarar u y v como variables de universos.
-- 4. Declarar α como una variable de tipos en u.
-- 5. Declarar I como una variable de tipos en v.
-- 6. Declarar A y B como variables sobre funciones de I en α.
-- 7. Declarar s como variable sobre conjuntos de elementos de α.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Basic    -- 1
import Mathlib.Data.Set.Lattice
import Mathlib.Tactic

open Set                         -- 2

universe u v                     -- 3
variable (α : Type u)            -- 4
variable (I : Type v)            -- 5
variable (A B : I → Set α)       -- 6
variable (s : Set α)             -- 7

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que para cada x, se verifica que
--    x ∈ s ∩ ⋃ (i : ℕ), A i ↔ x ∈ ⋃ (i : ℕ), A i ∩ s
-- Lo demostramos mediante la siguiente cadena de equivalencias
--    x ∈ s ∩ ⋃ (i : ℕ), A i ↔ x ∈ s ∧ x ∈ ⋃ (i : ℕ), A i
--                            ↔ x ∈ s ∧ (∃ i : ℕ, x ∈ A i)
--                            ↔ ∃ i : ℕ, x ∈ s ∧ x ∈ A i
--                            ↔ ∃ i : ℕ, x ∈ A i ∧ x ∈ s
```

```
--                            ↔ ∃ i : ℕ, x ∈ A i ∩ s
--                            ↔ x ∈ ⋃ (i : ℕ), A i ∩ s

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

-- 1ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ ⋃ (i : I), A i ↔ x ∈ ⋃ (i : I), A i ∩ s
  calc x ∈ s ∩ ⋃ (i : I), A i
     ↔ x ∈ s ∧ x ∈ ⋃ (i : I), A i :=
         by simp only [mem_inter_iff]
   _ ↔ x ∈ s ∧ (∃ i : I, x ∈ A i) :=
         by simp only [mem_iUnion]
   _ ↔ ∃ i : I, x ∈ s ∧ x ∈ A i :=
         by simp only [exists_and_left]
   _ ↔ ∃ i : I, x ∈ A i ∧ x ∈ s :=
         by simp only [and_comm]
   _ ↔ ∃ i : I, x ∈ A i ∩ s :=
         by simp only [mem_inter_iff]
   _ ↔ x ∈ ⋃ (i : I), A i ∩ s :=
         by simp only [mem_iUnion]

-- 2ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ ⋃ (i : I), A i ↔ x ∈ ⋃ (i : I), A i ∩ s
  constructor
  . -- ⊢ x ∈ s ∩ ⋃ (i : I), A i → x ∈ ⋃ (i : I), A i ∩ s
    intro h
    -- h : x ∈ s ∩ ⋃ (i : I), A i
    -- ⊢ x ∈ ⋃ (i : I), A i ∩ s
    rw [mem_iUnion]
    -- ⊢ ∃ i, x ∈ A i ∩ s
    rcases h with ⟨xs, xUAi⟩
    -- xs : x ∈ s
```

```
    -- xUAi : x ∈ ⋃ (i : I), A i
    rw [mem_iUnion] at xUAi
    -- xUAi : ∃ i, x ∈ A i
    rcases xUAi with ⟨i, xAi⟩
    -- i : I
    -- xAi : x ∈ A i
    use i
    -- ⊢ x ∈ A i ∩ s
    constructor
    . -- ⊢ x ∈ A i
      exact xAi
    . -- ⊢ x ∈ s
      exact xs
  . -- ⊢ x ∈ ⋃ (i : I), A i ∩ s → x ∈ s ∩ ⋃ (i : I), A i
    intro h
    -- h : x ∈ ⋃ (i : I), A i ∩ s
    -- ⊢ x ∈ s ∩ ⋃ (i : I), A i
    rw [mem_iUnion] at h
    -- h : ∃ i, x ∈ A i ∩ s
    rcases h with ⟨i, hi⟩
    -- i : I
    -- hi : x ∈ A i ∩ s
    rcases hi with ⟨xAi, xs⟩
    -- xAi : x ∈ A i
    -- xs : x ∈ s
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ x ∈ ⋃ (i : I), A i
      rw [mem_iUnion]
      -- ⊢ ∃ i, x ∈ A i
      use i
      -- ⊢ x ∈ A i

-- 3ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ ⋃ (i : I), A i ↔ x ∈ ⋃ (i : I), A i ∩ s
  simp
  -- ⊢ (x ∈ s ∧ ∃ i, x ∈ A i) ↔ (∃ i, x ∈ A i) ∧ x ∈ s
  constructor
```

```
  . -- ⊢ (x ∈ s ∧ ∃ i, x ∈ A i) → (∃ i, x ∈ A i) ∧ x ∈ s
    rintro ⟨xs, ⟨i, xAi⟩⟩
    -- xs : x ∈ s
    -- i : I
    -- xAi : x ∈ A i
    -- ⊢ (∃ i, x ∈ A i) ∧ x ∈ s
    exact ⟨⟨i, xAi⟩, xs⟩
  . -- ⊢ (∃ i, x ∈ A i) ∧ x ∈ s → x ∈ s ∧ ∃ i, x ∈ A i
    rintro ⟨⟨i, xAi⟩, xs⟩
    -- xs : x ∈ s
    -- i : I
    -- xAi : x ∈ A i
    -- ⊢ x ∈ s ∧ ∃ i, x ∈ A i
    exact ⟨xs, ⟨i, xAi⟩⟩

-- 3ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∩ ⋃ (i : I), A i ↔ x ∈ ⋃ (i : I), A i ∩ s
  aesop

-- 4ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by ext; aesop

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

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que para x se verifica
--    x ∈ ⋂ i, (A i ∩ B i) ↔ x ∈ (⋂ i, A i) ∩ (⋂ i, B i)
-- Lo demostramos mediante la siguiente cadena de equivalencias
```

```
--    x ∈ ⋂ i, (A i ∩ B i) ↔ (∀ i)[x ∈ A i ∩ B i]
--                         ↔ (∀ i)[x ∈ A i ∧ x ∈ B i]
--                         ↔ (∀ i)[x ∈ A i] ∧ (∀ i)[x ∈ B i]
--                         ↔ x ∈ (⋂ i, A i) ∧ x ∈ (⋂ i, B i)
--                         ↔ x ∈ (⋂ i, A i) ∩ (⋂ i, B i)

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

-- 1ª demostración
-- ===============

example : (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ ⋂ (i : ℕ), A i ∩ B i ↔ x ∈ (⋂ (i : ℕ), A i) ∩ ⋂ (i : ℕ), B i
  calc x ∈ ⋂ i, A i ∩ B i
     ↔ ∀ i, x ∈ A i ∩ B i :=
         by exact mem_iInter
   _ ↔ ∀ i, x ∈ A i ∧ x ∈ B i :=
         by simp only [mem_inter_iff]
   _ ↔ (∀ i, x ∈ A i) ∧ (∀ i, x ∈ B i) :=
         by exact forall_and
   _ ↔ x ∈ (⋂ i, A i) ∧ x ∈ (⋂ i, B i) :=
         by simp only [mem_iInter]
   _ ↔ x ∈ (⋂ i, A i) ∩ ⋂ i, B i :=
         by simp only [mem_inter_iff]

-- 2ª demostración
-- ===============

example : (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ ⋂ (i : ℕ), A i ∩ B i ↔ x ∈ (⋂ (i : ℕ), A i) ∩ ⋂ (i : ℕ), B i
  simp only [mem_inter_iff, mem_iInter]
  -- ⊢ (∀ (i : ℕ), x ∈ A i ∧ x ∈ B i) ↔ (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
  constructor
  . -- ⊢ (∀ (i : ℕ), x ∈ A i ∧ x ∈ B i) → (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
    intro h
    -- h : ∀ (i : ℕ), x ∈ A i ∧ x ∈ B i
    -- ⊢ (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
    constructor
```

```
    . -- ⊢ ∀ (i : ℕ), x ∈ A i
      intro i
      -- i : ℕ
      -- ⊢ x ∈ A i
      exact (h i).1
    . -- ⊢ ∀ (i : ℕ), x ∈ B i
      intro i
      -- i : ℕ
      -- ⊢ x ∈ B i
      exact (h i).2
  . -- ⊢ ((∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i) → ∀ (i : ℕ), x ∈ A i ∧ x ∈ B i
    intros h i
    -- h : (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
    -- i : ℕ
    -- ⊢ x ∈ A i ∧ x ∈ B i
    rcases h with ⟨h1, h2⟩
    -- h1 : ∀ (i : ℕ), x ∈ A i
    -- h2 : ∀ (i : ℕ), x ∈ B i
    constructor
    . -- ⊢ x ∈ A i
      exact h1 i
    . -- ⊢ x ∈ B i
      exact h2 i

-- 3ª demostración
-- ===============

example : (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ ⋂ (i : ℕ), A i ∩ B i ↔ x ∈ (⋂ (i : ℕ), A i) ∩ ⋂ (i : ℕ), B i
  simp only [mem_inter_iff, mem_iInter]
  -- ⊢ (∀ (i : ℕ), x ∈ A i ∧ x ∈ B i) ↔ (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
  exact ⟨fun h ↦ ⟨fun i ↦ (h i).1, fun i ↦ (h i).2⟩,
         fun ⟨h1, h2⟩ i ↦ ⟨h1 i, h2 i⟩⟩

-- 4ª demostración
-- ===============

example : (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
by
  ext
  -- x : α
  -- ⊢ x ∈ ⋂ (i : ℕ), A i ∩ B i ↔ x ∈ (⋂ (i : ℕ), A i) ∩ ⋂ (i : ℕ), B i
```

```
  simp only [mem_inter_iff, mem_iInter]
  -- ⊢ (∀ (i : ℕ), x ∈ A i ∧ x ∈ B i) ↔ (∀ (i : ℕ), x ∈ A i) ∧ ∀ (i : ℕ), x ∈ B i
  aesop

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

variable (x : α)
variable (a b : Set α)
variable (ι : Sort v)
variable (s : ι → Set α)
variable (p q : α → Prop)
variable (P Q : Prop)
#check (and_comm : P ∧ Q ↔ Q ∧ P)
#check (exists_and_left : (∃ (x : α), Q ∧ p x) ↔ Q ∧ ∃ (x : α), p x)
#check (forall_and : (∀ (x : α), p x ∧ q x) ↔ (∀ (x : α), p x) ∧ ∀ (x : α), q x)
#check (mem_iInter : x ∈ ⋂ (i : ι), s i ↔ ∀ (i : ι), x ∈ s i)
#check (mem_iUnion : x ∈ ⋃ i, A i ↔ ∃ i, x ∈ A i)
#check (mem_inter_iff x a b : x ∈ a ∩ b ↔ x ∈ a ∧ x ∈ b)
```

4.1.13. Ejercicios de uniones e intersecciones generales

```
-- ------------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s)
-- -------------------------------------------------------------------------

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

-- Tenemos que demostrar que para todo x,
--    x ∈ s ∪ ⋂ i, A i ↔ x ∈ ⋂ i, A i ∪ s
-- Lo haremos mediante la siguiente cadena de equivalencias
--    x ∈ s ∪ ⋂ i, A i ↔ x ∈ s ∨ x ∈ ⋂ i, A i
--                     ↔ x ∈ s ∨ ∀ i, x ∈ A i
--                     ↔ ∀ i, x ∈ s ∨ x ∈ A i
--                     ↔ ∀ i, x ∈ A i ∨ x ∈ s
--                     ↔ ∀ i, x ∈ A i ∪ s
--                     ↔ x ∈ ⋂ i, A i ∪ s

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

```
import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α : Type}
variable (s : Set α)
variable (A : ℕ → Set α)

-- 1ª demostración
-- ===============

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ ⋂ (i : ℕ), A i ↔ x ∈ ⋂ (i : ℕ), A i ∪ s
  calc x ∈ s ∪ ⋂ i, A i
     ↔ x ∈ s ∨ x ∈ ⋂ i, A i :=
          by simp only [mem_union]
   _ ↔ x ∈ s ∨ ∀ i, x ∈ A i :=
          by simp only [mem_iInter]
   _ ↔ ∀ i, x ∈ s ∨ x ∈ A i :=
          by simp only [forall_or_left]
   _ ↔ ∀ i, x ∈ A i ∨ x ∈ s  :=
          by simp only [or_comm]
   _ ↔ ∀ i, x ∈ A i ∪ s  :=
          by simp only [mem_union]
   _ ↔ x ∈ ⋂ i, A i ∪ s :=
          by simp only [mem_iInter]

-- 2ª demostración
-- ===============

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ ⋂ (i : ℕ), A i ↔ x ∈ ⋂ (i : ℕ), A i ∪ s
  simp only [mem_union, mem_iInter]
  -- ⊢ (x ∈ s ∨ ∀ (i : ℕ), x ∈ A i) ↔ ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
  constructor
  . -- ⊢ (x ∈ s ∨ ∀ (i : ℕ), x ∈ A i) → ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
    intros h i
    -- h : x ∈ s ∨ ∀ (i : ℕ), x ∈ A i
```

```
    -- i : ℕ
    -- ⊢ x ∈ A i ∨ x ∈ s
    rcases h with (xs | xAi)
    . -- xs : x ∈ s
      right
      -- ⊢ x ∈ s
      exact xs
    . -- xAi : ∀ (i : ℕ), x ∈ A i
      left
      -- ⊢ x ∈ A i
      exact xAi i
  . -- ⊢ (∀ (i : ℕ), x ∈ A i ∨ x ∈ s) → x ∈ s ∨ ∀ (i : ℕ), x ∈ A i
    intro h
    -- h : ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
    -- ⊢ x ∈ s ∨ ∀ (i : ℕ), x ∈ A i
    by_cases cxs : x ∈ s
    . -- cxs : x ∈ s
      left
      -- ⊢ x ∈ s
      exact cxs
    . -- cns : ¬x ∈ s
      right
      -- ⊢ ∀ (i : ℕ), x ∈ A i
      intro i
      -- i : ℕ
      -- ⊢ x ∈ A i
      rcases h i with (xAi | xs)
      . -- ⊢ x ∈ A i
        exact xAi
      . -- xs : x ∈ s
        exact absurd xs cxs

-- 3ª demostración
-- ===============

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ s ∪ ⋂ (i : ℕ), A i ↔ x ∈ ⋂ (i : ℕ), A i ∪ s
  simp only [mem_union, mem_iInter]
  -- ⊢ (x ∈ s ∨ ∀ (i : ℕ), x ∈ A i) ↔ ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
  constructor
  . -- ⊢ (x ∈ s ∨ ∀ (i : ℕ), x ∈ A i) → ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
    rintro (xs | xI) i
```

```
    . -- xs : x ∈ s
      -- i : ℕ
      -- ⊢ x ∈ A i ∨ x ∈ s
      right
      -- ⊢ x ∈ s
      exact xs
    . -- xI : ∀ (i : ℕ), x ∈ A i
      -- i : ℕ
      -- ⊢ x ∈ A i ∨ x ∈ s
      left
      -- ⊢ x ∈ A i
      exact xI i
  . -- ⊢ (∀ (i : ℕ), x ∈ A i ∨ x ∈ s) → x ∈ s ∨ ∀ (i : ℕ), x ∈ A i
    intro h
    -- h : ∀ (i : ℕ), x ∈ A i ∨ x ∈ s
    -- ⊢ x ∈ s ∨ ∀ (i : ℕ), x ∈ A i
    by_cases cxs : x ∈ s
    . -- cxs : x ∈ s
      left
      -- ⊢ x ∈ s
      exact cxs
    . -- cxs : ¬x ∈ s
      right
      -- ⊢ ∀ (i : ℕ), x ∈ A i
      intro i
      -- i : ℕ
      -- ⊢ x ∈ A i
      cases h i
      . -- h : x ∈ A i
        assumption
      . -- h : x ∈ s
        contradiction

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

variable (x : α)
variable (s t : Set α)
variable (a b q : Prop)
variable (p : ℕ → Prop)
#check (absurd : a → ¬a → b)
#check (forall_or_left : (∀ x, q ∨ p x) ↔ q ∨ ∀ x, p x)
#check (mem_iInter : x ∈ ⋂ i, A i ↔ ∀ i, x ∈ A i)
#check (mem_union x s t : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
#check (or_comm : a ∨ b ↔ b ∨ a)
```

4.1.14. Ejemplos de uniones e intersecciones generales (2)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la teoría data.set.lattice
-- 2. Importar la teoría data.nat.prime
-- 3. Abrir los espacios de nombre set y nat.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Lattice
import Mathlib.Data.Nat.Prime.Basic
import Mathlib.Data.Nat.Prime.Infinite

open Set Nat

-- ---------------------------------------------------------------------
-- Ejercicio. Definir el conjunto de los números primos.
-- ----------------------------------------------------------------------

def primes : Set ℕ := {x | Nat.Prime x}

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (⋃ p ∈ primes, {x | p^2 ∣ x}) = {x | ∃ p ∈ primes, p^2 ∣ x}
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : (⋃ p ∈ primes, { x | p ^ 2 ∣ x }) =
          { x | ∃ p ∈ primes, p ^ 2 ∣ x } :=
by
  ext
  -- ⊢ x ∈ ⋃ p ∈ primes, {x | p ^ 2 ∣ x} ↔ x ∈ {x | ∃ p ∈ primes, p ^ 2 ∣ x}
  rw [mem_iUnion₂]
  -- ⊢ (∃ i, ∃ (_ : i ∈ primes), x ∈ {x | i ^ 2 ∣ x}) ↔ x ∈ {x | ∃ p ∈ primes, p ^ 2 ∣ x
  simp

-- 2ª demostración
-- ===============

example : (⋃ p ∈ primes, { x | p ^ 2 ∣ x }) = { x | ∃ p ∈ primes, p ^ 2 ∣ x } :=
by
```

```
  ext
  -- ⊢ x ∈ ⋃ p ∈ primes, {x | p ^ 2 ∣ x} ↔ x ∈ {x | ∃ p ∈ primes, p ^ 2 ∣ x}
  simp

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (⋂ p ∈ primes, {x | ¬ p ∣ x}) ⊆ {x | x = 1}
-- ----------------------------------------------------------------------

example : (⋂ p ∈ primes, { x | ¬p ∣ x }) ⊆ { x | x = 1 } :=
by
  intro x
  -- x : ℕ
  -- ⊢ x ∈ ⋂ p ∈ primes, {x | ¬p ∣ x} → x ∈ {x | x = 1}
  contrapose!
  -- ⊢ x ∉ {x | x = 1} → x ∉ ⋂ p ∈ primes, {x | ¬p ∣ x}
  simp
  -- ⊢ ¬x = 1 → ∃ x_1 ∈ primes, x_1 ∣ x
  apply Nat.exists_prime_and_dvd

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (⋃ p ∈ primes, {x | x ≤ p}) = univ
-- ----------------------------------------------------------------------

example : (⋃ p ∈ primes, {x | x ≤ p}) = univ :=
by
  apply eq_univ_of_forall
  -- ⊢ ∀ (x : ℕ), x ∈ ⋃ p ∈ primes, {x | x ≤ p}
  intro x
  -- x : ℕ
  -- ⊢ x ∈ ⋃ p ∈ primes, {x | x ≤ p}
  simp
  -- ⊢ ∃ i ∈ primes, x ≤ i
  dsimp [primes]
  -- ⊢ ∃ i, Nat.Prime i ∧ x ≤ i
  obtain ⟨p, pge, primep⟩ := exists_infinite_primes x
  -- p : ℕ
  -- pge : x ≤ p
  -- primep : Nat.Prime p
  use p

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

```
variable (α : Type u)
variable (I : Type v)
variable (J : Type w)
variable (A : I → J → Set α)
variable (x : α)
variable (n : ℕ)
variable (s : Set α)
#check (Nat.exists_prime_and_dvd : n ≠ 1 → ∃ p, Nat.Prime p ∧ p ∣ n)
#check (eq_univ_of_forall : (∀ x, x ∈ s) → s = univ)
#check (exists_infinite_primes n : ∃ p, n ≤ p ∧ Nat.Prime p)
#check (mem_iUnion₂ : x ∈ ⋃ i, ⋃ j, A i j ↔ ∃ i j, x ∈ A i j)
```

4.1.15. Ejemplos de uniones e intersecciones generales (3)

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería data.set.lattice
-- 2. Abrir el espacio de nombres set.
-- 3. Declarar α una variable de tipos.
-- 4. Declarar s una vabiable sobre conjuntos de conjuntos de elementos
--    de α.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Lattice -- 1
open Set                        -- 2
variable {α : Type*}            -- 3
variable (s : Set (Set α))      -- 4

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ⋃₀ s = ⋃ t ∈ s, t
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : ⋃₀ s = ⋃ t ∈ s, t :=
by
  ext x
  -- ⊢ x ∈ ⋃₀ s ↔ x ∈ ⋃ t ∈ s, t
  rw [mem_iUnion₂]
```

```
  -- ⊢ x ∈ ⋃₀ s ↔ ∃ i, ∃ (_ : i ∈ s), x ∈ i
  simp

-- 2ª demostración
-- ===============

example : ⋃₀ s = ⋃ t ∈ s, t :=
sUnion_eq_biUnion

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ⋂₀ s = ⋂ t ∈ s, t
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : ⋂₀ s = ⋂ t ∈ s, t := by
  ext x
  rw [mem_iInter₂]
  rfl

-- 2ª demostración
-- ===============

example : ⋂₀ s = ⋂ t ∈ s, t :=
sInter_eq_biInter

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

variable (α : Type u)
variable (x : α)
variable (A : I → J → Set α)
variable (s : Set (Set α))
#check (mem_iInter₂ : x ∈ ⋂ i, ⋂ j, A i j ↔ ∀ i j, x ∈ A i j)
#check (mem_iUnion₂ : x ∈ ⋃ i, ⋃ j, A i j ↔ ∃ i j, x ∈ A i j)
#check (sInter_eq_biInter : ⋂₀ s = ⋂ i ∈ s, i)
#check (sUnion_eq_biUnion : ⋃₀ s = ⋃ i ∈ s, i)
```

4.2. Funciones

4.2.1. Preimagen de la intersección

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que, para todo x,
--    x ∈ f⁻¹[u ∩ v] ↔ x ∈ f⁻¹[u] ∩ f⁻¹[v]
-- Lo haremos mediante la siguiente cadena de equivalencias
--    x ∈ f⁻¹[u ∩ v] ↔ f x ∈ u ∩ v
--                   ↔ f x ∈ u ∧ f x ∈ v
--                   ↔ x ∈ f⁻¹[u] ∧ x ∈ f⁻¹[v]
--                   ↔ x ∈ f⁻¹[u] ∩ f⁻¹[v]

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

import Mathlib.Data.Set.Function

variable {α β : Type _}
variable (f : α → β)
variable (u v : Set β)

open Set

-- 1ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∩ v) ↔ x ∈ f ⁻¹' u ∩ f ⁻¹' v
  calc x ∈ f ⁻¹' (u ∩ v)
     ↔ f x ∈ u ∩ v :=
         by simp only [mem_preimage]
   _ ↔ f x ∈ u ∧ f x ∈ v :=
         by simp only [mem_inter_iff]
```

```
    _ ↔ x ∈ f ⁻¹' u ∧ x ∈ f ⁻¹' v :=
          by simp only [mem_preimage]
    _ ↔ x ∈ f ⁻¹' u ∩ f ⁻¹' v :=
          by simp only [mem_inter_iff]

-- 2ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∩ v) ↔ x ∈ f ⁻¹' u ∩ f ⁻¹' v
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∩ v) → x ∈ f ⁻¹' u ∩ f ⁻¹' v
    intro h
    -- h : x ∈ f ⁻¹' (u ∩ v)
    -- ⊢ x ∈ f ⁻¹' u ∩ f ⁻¹' v
    constructor
    . -- ⊢ x ∈ f ⁻¹' u
      apply mem_preimage.mpr
      -- ⊢ f x ∈ u
      rw [mem_preimage] at h
      -- h : f x ∈ u ∩ v
      exact mem_of_mem_inter_left h
    . -- ⊢ x ∈ f ⁻¹' v
      apply mem_preimage.mpr
      -- ⊢ f x ∈ v
      rw [mem_preimage] at h
      -- h : f x ∈ u ∩ v
      exact mem_of_mem_inter_right h
  . -- ⊢ x ∈ f ⁻¹' u ∩ f ⁻¹' v → x ∈ f ⁻¹' (u ∩ v)
    intro h
    -- h : x ∈ f ⁻¹' u ∩ f ⁻¹' v
    -- ⊢ x ∈ f ⁻¹' (u ∩ v)
    apply mem_preimage.mpr
    -- ⊢ f x ∈ u ∩ v
    constructor
    . -- ⊢ f x ∈ u
      apply mem_preimage.mp
      -- ⊢ x ∈ f ⁻¹' u
      exact mem_of_mem_inter_left h
    . -- ⊢ f x ∈ v
      apply mem_preimage.mp
      -- ⊢ x ∈ f ⁻¹' v
```

```
      exact mem_of_mem_inter_right h

-- 3ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∩ v) ↔ x ∈ f ⁻¹' u ∩ f ⁻¹' v
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∩ v) → x ∈ f ⁻¹' u ∩ f ⁻¹' v
    intro h
    -- h : x ∈ f ⁻¹' (u ∩ v)
    -- ⊢ x ∈ f ⁻¹' u ∩ f ⁻¹' v
    constructor
    . -- ⊢ x ∈ f ⁻¹' u
      simp at *
      -- h : f x ∈ u ∧ f x ∈ v
      -- ⊢ f x ∈ u
      exact h.1
    . -- ⊢ x ∈ f ⁻¹' v
      simp at *
      -- h : f x ∈ u ∧ f x ∈ v
      -- ⊢ f x ∈ v
      exact h.2
  . -- ⊢ x ∈ f ⁻¹' u ∩ f ⁻¹' v → x ∈ f ⁻¹' (u ∩ v)
    intro h
    -- h : x ∈ f ⁻¹' u ∩ f ⁻¹' v
    -- ⊢ x ∈ f ⁻¹' (u ∩ v)
    simp at *
    -- h : f x ∈ u ∧ f x ∈ v
    -- ⊢ f x ∈ u ∧ f x ∈ v
    exact h

-- 4ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
by aesop

-- 5ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
```

```
preimage_inter

-- 6ª demostración
-- ===============

example : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v :=
rfl

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

variable (x : α)
variable (s t : Set α)
#check (mem_inter_iff x s t : x ∈ s ∩ t ↔ x ∈ s ∧ x ∈ t)
#check (mem_of_mem_inter_left : x ∈ s ∩ t → x ∈ s)
#check (mem_of_mem_inter_right : x ∈ s ∩ t → x ∈ t)
#check (mem_preimage : x ∈ f ⁻¹' u ↔ f x ∈ u)
#check (preimage_inter : f ⁻¹' (u ∩ v) = f ⁻¹' u ∩ f ⁻¹' v)
```

4.2.2. Imagen de la unión

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f '' (s ∪ t) = (f '' s) ∪ (f '' t)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar, para todo y, que
--    y ∈ f[s ∪ t] ↔ y ∈ f[s] ∪ f[t]
-- Lo haremos mediante la siguiente cadena de equivalencias
--    y ∈ f[s ∪ t] ↔ (∃x)(x ∈ s ∪ t ∧ f x = y)
--                 ↔ (∃x)((x ∈ s ∨ x ∈ t) ∧ f x = y)
--                 ↔ (∃x)((x ∈ s ∧ f x = y) ∨ (x ∈ t ∧ f x = y))
--                 ↔ (∃x)(x ∈ s ∧ f x = y) ∨ (∃x)(x ∈ t ∧ f x = y)
--                 ↔ y ∈ f[s] ∨ y ∈ f[t]
--                 ↔ y ∈ f[s] ∪ f[t]

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

import Mathlib.Data.Set.Function
```

```
variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)

open Set

-- 1ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  calc y ∈ f '' (s ∪ t)
     ↔ ∃ x, x ∈ s ∪ t ∧ f x = y :=
         by simp only [mem_image]
   _ ↔ ∃ x, (x ∈ s ∨ x ∈ t) ∧ f x = y :=
         by simp only [mem_union]
   _ ↔ ∃ x, (x ∈ s ∧ f x = y) ∨ (x ∈ t ∧ f x = y) :=
         by simp only [or_and_right]
   _ ↔ (∃ x, x ∈ s ∧ f x = y) ∨ (∃ x, x ∈ t ∧ f x = y) :=
         by simp only [exists_or]
   _ ↔ y ∈ f '' s ∨ y ∈ f '' t :=
         by simp only [mem_image]
   _ ↔ y ∈ f '' s ∪ f '' t :=
         by simp only [mem_union]

-- 2ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  constructor
  . -- ⊢ y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t
    intro h
    -- h : y ∈ f '' (s ∪ t)
    -- ⊢ y ∈ f '' s ∪ f '' t
    rw [mem_image] at h
    -- h : ∃ x, x ∈ s ∪ t ∧ f x = y
    rcases h with ⟨x, hx⟩
```

```
    -- x : α
    -- hx : x ∈ s ∪ t ∧ f x = y
    rcases hx with ⟨xst, fxy⟩
    -- xst : x ∈ s ∪ t
    -- fxy : f x = y
    rw [←fxy]
    -- ⊢ f x ∈ f '' s ∪ f '' t
    rw [mem_union] at xst
    -- xst : x ∈ s ∨ x ∈ t
    rcases xst with (xs | xt)
    . -- xs : x ∈ s
      apply mem_union_left
      -- ⊢ f x ∈ f '' s
      apply mem_image_of_mem
      -- ⊢ x ∈ s
      exact xs
    . -- xt : x ∈ t
      apply mem_union_right
      -- ⊢ f x ∈ f '' t
      apply mem_image_of_mem
      -- ⊢ x ∈ t
      exact xt
  . -- ⊢ y ∈ f '' s ∪ f '' t → y ∈ f '' (s ∪ t)
    intro h
    -- h : y ∈ f '' s ∪ f '' t
    -- ⊢ y ∈ f '' (s ∪ t)
    rw [mem_union] at h
    -- h : y ∈ f '' s ∨ y ∈ f '' t
    rcases h with (yfs | yft)
    . -- yfs : y ∈ f '' s
      rw [mem_image]
      -- ⊢ ∃ x, x ∈ s ∪ t ∧ f x = y
      rw [mem_image] at yfs
      -- yfs : ∃ x, x ∈ s ∧ f x = y
      rcases yfs with ⟨x, hx⟩
      -- x : α
      -- hx : x ∈ s ∧ f x = y
      rcases hx with ⟨xs, fxy⟩
      -- xs : x ∈ s
      -- fxy : f x = y
      use x
      -- ⊢ x ∈ s ∪ t ∧ f x = y
      constructor
      . -- ⊢ x ∈ s ∪ t
        apply mem_union_left
```

```
        -- ⊢ x ∈ s
        exact xs
      . -- ⊢ f x = y
        exact fxy
    . -- yft : y ∈ f '' t
      rw [mem_image]
      -- ⊢ ∃ x, x ∈ s ∪ t ∧ f x = y
      rw [mem_image] at yft
      -- yft : ∃ x, x ∈ t ∧ f x = y
      rcases yft with ⟨x, hx⟩
      -- x : α
      -- hx : x ∈ t ∧ f x = y
      rcases hx with ⟨xt, fxy⟩
      -- xt : x ∈ t
      -- fxy : f x = y
      use x
      -- ⊢ x ∈ s ∪ t ∧ f x = y
      constructor
      . -- ⊢ x ∈ s ∪ t
        apply mem_union_right
        -- ⊢ x ∈ t
        exact xt
      . -- ⊢ f x = y
        exact fxy

-- 3ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  constructor
  . -- ⊢ y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t
    rintro ⟨x, xst, rfl⟩
    -- x : α
    -- xst : x ∈ s ∪ t
    -- ⊢ f x ∈ f '' s ∪ f '' t
    rcases xst with (xs | xt)
    . -- xs : x ∈ s
      left
      -- ⊢ f x ∈ f '' s
      exact mem_image_of_mem f xs
    . -- xt : x ∈ t
```

```
      right
      -- ⊢ f x ∈ f '' t
      exact mem_image_of_mem f xt
  . -- ⊢ y ∈ f '' s ∪ f '' t → y ∈ f '' (s ∪ t)
    rintro (yfs | yft)
    . -- yfs : y ∈ f '' s
      rcases yfs with ⟨x, xs, rfl⟩
      -- x : α
      -- xs : x ∈ s
      -- ⊢ f x ∈ f '' (s ∪ t)
      apply mem_image_of_mem
      -- ⊢ x ∈ s ∪ t
      left
      -- ⊢ x ∈ s
      exact xs
    . -- yft : y ∈ f '' t
      rcases yft with ⟨x, xt, rfl⟩
      -- x : α
      -- xs : x ∈ s
      -- ⊢ f x ∈ f '' (s ∪ t)
      apply mem_image_of_mem
      -- ⊢ x ∈ s ∪ t
      right
      -- ⊢ x ∈ t
      exact xt

-- 4ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  constructor
  . -- ⊢ y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t
    rintro ⟨x, xst, rfl⟩
    -- x : α
    -- xst : x ∈ s ∪ t
    -- ⊢ f x ∈ f '' s ∪ f '' t
    rcases xst with (xs | xt)
    . -- xs : x ∈ s
      left
      -- ⊢ f x ∈ f '' s
      use x, xs
```

```
    . -- xt : x ∈ t
      right
      -- ⊢ f x ∈ f '' t
      use x, xt
  . rintro (yfs | yft)
    . -- yfs : y ∈ f '' s
      rcases yfs with ⟨x, xs, rfl⟩
      -- x : α
      -- xs : x ∈ s
      -- ⊢ f x ∈ f '' (s ∪ t)
      use x, Or.inl xs
    . -- yft : y ∈ f '' t
      rcases yft with ⟨x, xt, rfl⟩
      -- x : α
      -- xt : x ∈ t
      -- ⊢ f x ∈ f '' (s ∪ t)
      use x, Or.inr xt

-- 5ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  constructor
  . -- ⊢ y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t
    rintro ⟨x, xs | xt, rfl⟩
    . -- x : α
      -- xs : x ∈ s
      -- ⊢ f x ∈ f '' s ∪ f '' t
      left
      -- ⊢ f x ∈ f '' s
      use x, xs
    . -- x : α
      -- xt : x ∈ t
      -- ⊢ f x ∈ f '' s ∪ f '' t
      right
      -- ⊢ f x ∈ f '' t
      use x, xt
  . -- ⊢ y ∈ f '' s ∪ f '' t → y ∈ f '' (s ∪ t)
    rintro (⟨x, xs, rfl⟩ | ⟨x, xt, rfl⟩)
    . -- x : α
      -- xs : x ∈ s
```

```
      -- ⊢ f x ∈ f '' (s ∪ t)
      use x, Or.inl xs
    . -- x : α
      -- xt : x ∈ t
      -- ⊢ f x ∈ f '' (s ∪ t)
      use x, Or.inr xt

-- 6ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  constructor
  . -- ⊢ y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t
    aesop
  . -- ⊢ y ∈ f '' s ∪ f '' t → y ∈ f '' (s ∪ t)
    aesop

-- 7ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  constructor <;> aesop

-- 8ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' (s ∪ t) ↔ y ∈ f '' s ∪ f '' t
  rw [iff_def]
  -- ⊢ (y ∈ f '' (s ∪ t) → y ∈ f '' s ∪ f '' t) ∧ (y ∈ f '' s ∪ f '' t → y ∈ f '' (s ∪ t
  aesop

-- 9ª demostración
-- ===============

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
```

```
image_union f s t

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

variable (x : α)
variable (y : β)
variable (a b c : Prop)
variable (p q : α → Prop)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (exists_or : (∃ x, p x ∨ q x) ↔ (∃ x, p x) ∨ ∃ x, q x)
#check (iff_def : (a ↔ b) ↔ (a → b) ∧ (b → a))
#check (image_union f s t : f '' (s ∪ t) = f '' s ∪ f '' t)
#check (mem_image f s y : (y ∈ f '' s ↔ ∃ (x : α), x ∈ s ∧ f x = y))
#check (mem_image_of_mem  f : x ∈ s → f x ∈ f '' s)
#check (mem_union x s t : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
#check (mem_union_left t : x ∈ s → x ∈ s ∪ t)
#check (mem_union_right s : x ∈ t → x ∈ s ∪ t)
#check (or_and_right : (a ∨ b) ∧ c ↔ a ∧ c ∨ b ∧ c)
```

4.2.3. Preimagen de imagen

```
-- ---------------------------------------------------------------------
-- Demostrar que si s es un subconjunto del dominio de la función f,
-- entonces s está contenido en la imagen inversa de la imagen de s por
-- f; es decir,
--    s ⊆ f⁻¹[f[s]]
-- ----------------------------------------------------------------------

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

-- Se demuestra mediante la siguiente cadena de implicaciones
--    x ∈ s ⟹ f(x) ∈ f[s]
--          ⟹ x ∈ f⁻¹[f[s]]

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

import Mathlib.Data.Set.Function

open Set
```

```
variable {α β : Type _}
variable (f : α → β)
variable (s : Set α)

-- 1ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
by
  intros x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ f ⁻¹' (f '' s)
  have h1 : f x ∈ f '' s := mem_image_of_mem f xs
  show x ∈ f ⁻¹' (f '' s)
  exact mem_preimage.mp h1

-- 2ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
by
  intros x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ f ⁻¹' (f '' s)
  apply mem_preimage.mpr
  -- ⊢ f x ∈ f '' s
  apply mem_image_of_mem
  -- ⊢ x ∈ s
  exact xs

-- 3ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
by
  intros x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ f ⁻¹' (f '' s)
  apply mem_image_of_mem
  -- ⊢ x ∈ s
  exact xs
```

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

example : s ⊆ f ⁻¹' (f '' s) :=
fun _ ↦ mem_image_of_mem f

-- 5ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
by
  intros x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ f ⁻¹' (f '' s)
  show f x ∈ f '' s
  use x, xs

-- 6ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
by
  intros x xs
  -- x : α
  -- xs : x ∈ s
  -- ⊢ x ∈ f ⁻¹' (f '' s)
  use x, xs

-- 7ª demostración
-- ===============

example : s ⊆ f ⁻¹' (f '' s) :=
subset_preimage_image f s

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

-- variable (x : α)
-- variable (t : Set β)
-- #check (mem_preimage : x ∈ f ⁻¹' t ↔ f x ∈ t)
-- #check (mem_image_of_mem f : x ∈ s → f x ∈ f '' s)
-- #check (subset_preimage_image f s : s ⊆ f ⁻¹' (f '' s))
```

4.2.4. Inclusión de la imagen

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f[s] ⊆ u ↔ s ⊆ f⁻¹[u]
-- ----------------------------------------------------------------------

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

-- Los demostraremos probando las dos implicaciones.
--
-- (⟹) Supongamos que
--    f[s] ⊆ u                                                        (1)
-- y tenemos que demostrar que
--    s ⊆ f⁻¹[u]
-- Se prueba mediante las siguientes implicaciones
--    x ∈ s ⟹ f(x) ∈ f[s]
--          ⟹ f(x) ∈ u       [por (1)]
--          ⟹ x ∈ f⁻¹[u]
--
-- (⟸) Supongamos que
--    s ⊆ f⁻¹[u]                                                      (2)
-- y tenemos que demostrar que
--    f[s] ⊆ u
-- Para ello, sea y ∈ f[s]. Entonces, existe un
--    x ∈ s                                                           (3)
-- tal que
--    y = f(x)                                                        (4)
-- Entonces,
--    x ∈ f⁻¹[u]    [por (2) y (3)]
--    ⟹ f(x) ∈ u
--    ⟹ y ∈ u     [por (4)]

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

import Mathlib.Data.Set.Function

open Set

variable {α β : Type _}
variable (f : α → β)
variable (s : Set α)
variable (u : Set β)
```

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

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
calc f '' s ⊆ u
   ↔ ∀ y, y ∈ f '' s → y ∈ u :=
       by simp only [subset_def]
 _ ↔ ∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u :=
       by simp only [mem_image]
 _ ↔ ∀ x, x ∈ s → f x ∈ u := by
       constructor
       . -- (∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u) → (∀ x, x ∈ s → f x ∈ u)
         intro h x xs
         -- h : ∀ (y : β), (∃ x, x ∈ s ∧ f x = y) → y ∈ u
         -- x : α
         -- xs : x ∈ s
         -- ⊢ f x ∈ u
         exact h (f x) (by use x, xs)
       . -- (∀ x, x ∈ s → f x ∈ u) → (∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u)
         intro h y hy
         -- h : ∀ (x : α), x ∈ s → f x ∈ u
         -- y : β
         -- hy : ∃ x, x ∈ s ∧ f x = y
         -- ⊢ y ∈ u
         obtain ⟨x, hx⟩ := hy
         -- x : α
         -- hx : x ∈ s ∧ f x = y
         have h1 : y = f x := hx.2.symm
         have h2 : f x ∈ u := h x hx.1
         show y ∈ u
         exact mem_of_eq_of_mem h1 h2
 _ ↔ ∀ x, x ∈ s → x ∈ f ⁻¹' u :=
       by simp only [mem_preimage]
 _ ↔ s ⊆ f ⁻¹' u :=
       by simp only [subset_def]

-- 2ª demostración
-- ===============

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
calc f '' s ⊆ u
   ↔ ∀ y, y ∈ f '' s → y ∈ u :=
       by simp only [subset_def]
 _ ↔ ∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u :=
```

```
        by simp only [mem_image]
  _ ↔ ∀ x, x ∈ s → f x ∈ u := by
        constructor
        . -- (∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u) → (∀ x, x ∈ s → f x ∈ u)
          intro h x xs
          -- h : ∀ (y : β), (∃ x, x ∈ s ∧ f x = y) → y ∈ u
          -- x : α
          -- xs : x ∈ s
          -- ⊢ f x ∈ u
          apply h (f x)
          -- ⊢ ∃ x_1, x_1 ∈ s ∧ f x_1 = f x
          use x, xs
        . -- (∀ x, x ∈ s → f x ∈ u) → (∀ y, (∃ x, x ∈ s ∧ f x = y) → y ∈ u)
          intro h y hy
          -- h : ∀ (x : α), x ∈ s → f x ∈ u
          -- y : β
          -- hy : ∃ x, x ∈ s ∧ f x = y
          -- ⊢ y ∈ u
          obtain ⟨x, hx⟩ := hy
          -- x : α
          -- hx : x ∈ s ∧ f x = y
          rw [←hx.2]
          -- ⊢ f x ∈ u
          apply h x
          -- ⊢ x ∈ s
          exact hx.1
  _ ↔ ∀ x, x ∈ s → x ∈ f ⁻¹' u :=
        by simp only [mem_preimage]
  _ ↔ s ⊆ f ⁻¹' u :=
        by simp only [subset_def]

-- 3ª demostración
-- ===============

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
by
  constructor
  . -- ⊢ f '' s ⊆ u → s ⊆ f ⁻¹' u
    intros h x xs
    -- h : f '' s ⊆ u
    -- x : α
    -- xs : x ∈ s
    -- ⊢ x ∈ f ⁻¹' u
    apply mem_preimage.mpr
    -- ⊢ f x ∈ u
```

```
    apply h
    -- ⊢ f x ∈ f '' s
    apply mem_image_of_mem
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ s ⊆ f ⁻¹' u → f '' s ⊆ u
    intros h y hy
    -- h : s ⊆ f ⁻¹' u
    -- y : β
    -- hy : y ∈ f '' s
    -- ⊢ y ∈ u
    rcases hy with ⟨x, xs, fxy⟩
    -- x : α
    -- xs : x ∈ s
    -- fxy : f x = y
    rw [←fxy]
    -- ⊢ f x ∈ u
    exact h xs

-- 4ª demostración
-- ===============

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
by
  constructor
  . -- ⊢ f '' s ⊆ u → s ⊆ f ⁻¹' u
    intros h x xs
    -- h : f '' s ⊆ u
    -- x : α
    -- xs : x ∈ s
    -- ⊢ x ∈ f ⁻¹' u
    apply h
    -- ⊢ f x ∈ f '' s
    apply mem_image_of_mem
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ s ⊆ f ⁻¹' u → f '' s ⊆ u
    rintro h y ⟨x, xs, rfl⟩
    -- h : s ⊆ f ⁻¹' u
    -- x : α
    -- xs : x ∈ s
    -- ⊢ f x ∈ u
    exact h xs

-- 5ª demostración
```

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

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
image_subset_iff

-- 4ª demostración
-- ===============

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
by simp

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

variable (x y : α)
variable (z : β)
variable (t : Set α)
#check (image_subset_iff : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u)
#check (mem_image f s z : (z ∈ f '' s ↔ ∃ x, x ∈ s ∧ f x = z))
#check (mem_image_of_mem f : x ∈ s → f x ∈ f '' s)
#check (mem_of_eq_of_mem : x = y → y ∈ s → x ∈ s)
#check (mem_preimage : x ∈ f ⁻¹' u ↔ f x ∈ u)
#check (subset_def : (s ⊆ t) = ∀ x ∈ s, x ∈ t)
```

4.2.5. Ejercicios de imágenes y preimágenes

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es inyectiva, entonces
--    f ⁻¹' (f '' s) ⊆ s
-- ----------------------------------------------------------------------

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

-- Sea x tal que
--    x ∈ f⁻¹[f[s]]
-- Entonces,
--    f(x) ∈ f[s]
-- y, por tanto, existe un
--    y ∈ s                                                         (1)
-- tal que
--    f(y) = f(x)                                                   (2)
-- De (2), puesto que f es inyectiva, se tiene que
```

```
--    y = x                                                              (3)
-- Finalmente, de (3) y (1), se tiene que
--    x ∈ s
-- que es lo que teníamos que demostrar.

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

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set Function

variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)
variable (u v : Set β)

-- 1ª demostración
-- ===============

example
  (h : Injective f)
  : f ⁻¹' (f '' s) ⊆ s :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' (f '' s)
  -- ⊢ x ∈ s
  have h1 : f x ∈ f '' s := mem_preimage.mp hx
  have h2 : ∃ y, y ∈ s ∧ f y = f x := (mem_image f s (f x)).mp h1
  obtain ⟨y, hy : y ∈ s ∧ f y = f x⟩ := h2
  obtain ⟨ys : y ∈ s, fyx : f y = f x⟩ := hy
  have h3 : y = x := h fyx
  show x ∈ s
  exact h3 ▸ ys

-- 2ª demostración
-- ===============

example
  (h : Injective f)
  : f ⁻¹' (f '' s) ⊆ s :=
by
  intros x hx
```

```
  -- x : α
  -- hx : x ∈ f ⁻¹' (f '' s)
  -- ⊢ x ∈ s
  rw [mem_preimage] at hx
  -- hx : f x ∈ f '' s
  rw [mem_image] at hx
  -- hx : ∃ x_1, x_1 ∈ s ∧ f x_1 = f x
  rcases hx with ⟨y, hy⟩
  -- y : α
  -- hy : y ∈ s ∧ f y = f x
  rcases hy with ⟨ys, fyx⟩
  -- ys : y ∈ s
  -- fyx : f y = f x
  unfold Injective at h
  -- h : ∀ {a₁ a₂ : α}, f a₁ = f a₂ → a₁ = a₂
  have h1 : y = x := h fyx
  rw [←h1]
  -- ⊢ y ∈ s
  exact ys

-- 3ª demostración
-- ===============

example
  (h : Injective f)
  : f ⁻¹' (f '' s) ⊆ s :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' (f '' s)
  -- ⊢ x ∈ s
  rw [mem_preimage] at hx
  -- hx : f x ∈ f '' s
  rcases hx with ⟨y, ys, fyx⟩
  -- y : α
  -- ys : y ∈ s
  -- fyx : f y = f x
  rw [←h fyx]
  -- ⊢ y ∈ s
  exact ys

-- 4ª demostración
-- ===============

example
```

```
  (h : Injective f)
  : f ⁻¹' (f '' s) ⊆ s :=
by
  rintro x ⟨y, ys, hy⟩
  -- x y : α
  -- ys : y ∈ s
  -- hy : f y = f x
  -- ⊢ x ∈ s
  rw [←h hy]
  -- ⊢ y ∈ s
  exact ys

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f '' (f⁻¹' u) ⊆ u
-- ----------------------------------------------------------------------

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

-- Sea y ∈ f[f⁻¹[u]]. Entonces existe un x tal que
--    x ∈ f⁻¹[u]                                                    (1)
--    f(x) = y                                                      (2)
-- Por (1),
--    f(x) ∈ u
-- y, por (2),
--    y ∈ u

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

-- 1ª demostración
-- ===============

example : f '' (f⁻¹' u) ⊆ u :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' (f ⁻¹' u)
  -- ⊢ y ∈ u
  obtain ⟨x : α, h1 : x ∈ f ⁻¹' u ∧ f x = y⟩ := h
  obtain ⟨hx : x ∈ f ⁻¹' u, fxy : f x = y⟩ := h1
  have h2 : f x ∈ u := mem_preimage.mp hx
  show y ∈ u
  exact fxy ▸ h2
```

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

example : f '' (f⁻¹' u) ⊆ u :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' (f ⁻¹' u)
  -- ⊢ y ∈ u
  rcases h with ⟨x, h2⟩
  -- x : α
  -- h2 : x ∈ f ⁻¹' u ∧ f x = y
  rcases h2 with ⟨hx, fxy⟩
  -- hx : x ∈ f ⁻¹' u
  -- fxy : f x = y
  rw [←fxy]
  -- ⊢ f x ∈ u
  exact hx

-- 3ª demostración
-- ===============

example : f '' (f⁻¹' u) ⊆ u :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' (f ⁻¹' u)
  -- ⊢ y ∈ u
  rcases h with ⟨x, hx, fxy⟩
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- fxy : f x = y
  rw [←fxy]
  -- ⊢ f x ∈ u
  exact hx

-- 4ª demostración
-- ===============

example : f '' (f⁻¹' u) ⊆ u :=
by
  rintro y ⟨x, hx, fxy⟩
  -- y : β
  -- x : α
```

```
  -- hx : x ∈ f ⁻¹' u
  -- fxy : f x = y
  -- ⊢ y ∈ u
  rw [←fxy]
  -- ⊢ f x ∈ u
  exact hx

-- 5ª demostración
-- ===============

example : f '' (f⁻¹' u) ⊆ u :=
by
  rintro y ⟨x, hx, rfl⟩
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- ⊢ f x ∈ u
  exact hx

-- 6ª demostración
-- ===============

example : f '' (f⁻¹' u) ⊆ u :=
image_preimage_subset f u

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es suprayectiva, entonces
--    u ⊆ f '' (f⁻¹' u)
-- ----------------------------------------------------------------------

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

-- Sea y ∈ u. Por ser f suprayectiva, exite un x tal que
--    f(x) = y                                                      (1)
-- Por tanto, x ∈ f⁻¹[u] y
--    f(x) ∈ f[f⁻¹[u]]                                              (2)
-- Finalmente, por (1) y (2),
--    y ∈ f[f⁻¹[u]]

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

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

```
example
  (h : Surjective f)
  : u ⊆ f '' (f⁻¹' u) :=
by
  intros y yu
  -- y : β
  -- yu : y ∈ u
  -- ⊢ y ∈ f '' (f ⁻¹' u)
  rcases h y with ⟨x, fxy⟩
  -- x : α
  -- fxy : f x = y
  use x
  -- ⊢ x ∈ f ⁻¹' u ∧ f x = y
  constructor
  . -- ⊢ x ∈ f ⁻¹' u
    apply mem_preimage.mpr
    -- ⊢ f x ∈ u
    rw [fxy]
    -- ⊢ y ∈ u
    exact yu
  . -- ⊢ f x = y
    exact fxy

-- 2ª demostración
-- ===============

example
  (h : Surjective f)
  : u ⊆ f '' (f⁻¹' u) :=
by
  intros y yu
  -- y : β
  -- yu : y ∈ u
  -- ⊢ y ∈ f '' (f ⁻¹' u)
  rcases h y with ⟨x, fxy⟩
  -- x : α
  -- fxy : f x = y
  -- ⊢ y ∈ f '' (f ⁻¹' u)
  use x
  -- ⊢ x ∈ f ⁻¹' u ∧ f x = y
  constructor
  . show f x ∈ u
    rw [fxy]
    -- ⊢ y ∈ u
    exact yu
```

```
  · show f x = y
    exact fxy

-- 3ª demostración
-- ===============

example
  (h : Surjective f)
  : u ⊆ f '' (f⁻¹' u) :=
by
  intros y yu
  -- y : β
  -- yu : y ∈ u
  -- ⊢ y ∈ f '' (f ⁻¹' u)
  rcases h y with ⟨x, fxy⟩
  -- x : α
  -- fxy : f x = y
  aesop

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    s ⊆ t
-- entonces
--    f '' s ⊆ f '' t
-- ----------------------------------------------------------------------

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

-- Sea y ∈ f[s]. Entonces, existe un x tal que
--    x ∈ s                                                          (1)
--    f(x) = y                                                       (2)
-- Por (1) y la hipótesis,
--    x ∈ t                                                          (3)
-- Por (3),
--    f(x) ∈ f[t]                                                    (4)
-- y, por (2) y (4),
--    y ∈ f[t]

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

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

```
example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s
  -- ⊢ y ∈ f '' t
  rw [mem_image] at hy
  -- hy : ∃ x, x ∈ s ∧ f x = y
  rcases hy with ⟨x, hx⟩
  -- x : α
  -- hx : x ∈ s ∧ f x = y
  rcases hx with ⟨xs, fxy⟩
  -- xs : x ∈ s
  -- fxy : f x = y
  use x
  -- ⊢ x ∈ t ∧ f x = y
  constructor
  . -- ⊢ x ∈ t
    exact h xs
  . -- ⊢ f x = y
    exact fxy

-- 2ª demostración
-- ===============

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s
  -- ⊢ y ∈ f '' t
  rcases hy with ⟨x, xs, fxy⟩
  -- x : α
  -- xs : x ∈ s
  -- fxy : f x = y
  use x
  -- ⊢ x ∈ t ∧ f x = y
  exact ⟨h xs, fxy⟩

-- 3ª demostración
-- ===============
```

```
example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
image_subset f h

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    u ⊆ v
-- entonces
--    f ⁻¹' u ⊆ f ⁻¹' v
-- ----------------------------------------------------------------------

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

-- Por la siguiente cadena de implicaciones:
--    x ∈ f⁻¹[u] ⟹ f(x) ∈ u
--               ⟹ f(x) ∈ v      [porque u ⊆ v]
--               ⟹ x ∈ f⁻¹[v]

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

-- 1ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- ⊢ x ∈ f ⁻¹' v
  have h1 : f x ∈ u := mem_preimage.mp hx
  have h2 : f x ∈ v := h h1
  show x ∈ f ⁻¹' v
  exact mem_preimage.mpr h2

-- 2ª demostración
-- ===============

example
  (h : u ⊆ v)
```

```
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- ⊢ x ∈ f ⁻¹' v
  apply mem_preimage.mpr
  -- ⊢ f x ∈ v
  apply h
  -- ⊢ f x ∈ u
  apply mem_preimage.mp
  -- ⊢ x ∈ f ⁻¹' u
  exact hx

-- 3ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- ⊢ x ∈ f ⁻¹' v
  apply h
  -- ⊢ f x ∈ u
  exact hx

-- 4ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ f ⁻¹' u
  -- ⊢ x ∈ f ⁻¹' v
  exact h hx

-- 5ª demostración
-- ===============
```

```
example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
fun _ hx ↦ h hx

-- 6ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by intro x; apply h

-- 7ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
preimage_mono h

-- 8ª demostración
-- ===============

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by tauto

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f ⁻¹' (u ∪ v) = (f ⁻¹' u) ∪ (f ⁻¹' v)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que, para todo x,
--    x ∈ f⁻¹[u ∪ v] ↔ x ∈ f⁻¹[u] ∪ f⁻¹[v]
-- Lo haremos demostrando las dos implicaciones.
--
-- (⟹) Supongamos que x ∈ f⁻¹[u ∪ v]. Entonces, f(x) ∈ u ∪ v.
-- Distinguimos dos casos:
--
-- Caso 1: Supongamos que f(x) ∈ u. Entonces, x ∈ f⁻¹[u] y, por tanto,
```

```
-- x ∈ f⁻¹[u] ∪ f⁻¹[v].
--
-- Caso 2: Supongamos que f(x) ∈ v. Entonces, x ∈ f⁻¹[v] y, por tanto,
-- x ∈ f⁻¹[u] ∪ f⁻¹[v].
--
-- (⟸) Supongamos que x ∈ f⁻¹[u] ∪ f⁻¹[v]. Distinguimos dos casos.
--
-- Caso 1: Supongamos que x ∈ f⁻¹[u]. Entonces, f(x) ∈ u y, por tanto,
-- f(x) ∈ u ∪ v. Luego, x ∈ f⁻¹[u ∪ v].
--
-- Caso 2: Supongamos que x ∈ f⁻¹[v]. Entonces, f(x) ∈ v y, por tanto,
-- f(x) ∈ u ∪ v. Luego, x ∈ f⁻¹[u ∪ v].

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


-- 1ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∪ v) ↔ x ∈ f ⁻¹' u ∪ f ⁻¹' v
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∪ v) → x ∈ f ⁻¹' u ∪ f ⁻¹' v
    intro h
    -- h : x ∈ f ⁻¹' (u ∪ v)
    -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v
    rw [mem_preimage] at h
    -- h : f x ∈ u ∪ v
    rcases h with fxu | fxv
    . -- fxu : f x ∈ u
      left
      -- ⊢ x ∈ f ⁻¹' u
      apply mem_preimage.mpr
      -- ⊢ f x ∈ u
      exact fxu
    . -- fxv : f x ∈ v
      right
      -- ⊢ x ∈ f ⁻¹' v
      apply mem_preimage.mpr
      -- ⊢ f x ∈ v
      exact fxv
```

```
  . -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v → x ∈ f ⁻¹' (u ∪ v)
    intro h
    -- h : x ∈ f ⁻¹' u ∪ f ⁻¹' v
    -- ⊢ x ∈ f ⁻¹' (u ∪ v)
    rw [mem_preimage]
    -- ⊢ f x ∈ u ∪ v
    rcases h with xfu | xfv
    . -- xfu : x ∈ f ⁻¹' u
      rw [mem_preimage] at xfu
      -- xfu : f x ∈ u
      left
      -- ⊢ f x ∈ u
      exact xfu
    . -- xfv : x ∈ f ⁻¹' v
      rw [mem_preimage] at xfv
      -- xfv : f x ∈ v
      right
      -- ⊢ f x ∈ v
      exact xfv

-- 2ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∪ v) ↔ x ∈ f ⁻¹' u ∪ f ⁻¹' v
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∪ v) → x ∈ f ⁻¹' u ∪ f ⁻¹' v
    intros h
    -- h : x ∈ f ⁻¹' (u ∪ v)
    -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v
    rcases h with fxu | fxv
    . -- fxu : f x ∈ u
      left
      -- ⊢ x ∈ f ⁻¹' u
      exact fxu
    . -- fxv : f x ∈ v
      right
      -- ⊢ x ∈ f ⁻¹' v
      exact fxv
  . -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v → x ∈ f ⁻¹' (u ∪ v)
    intro h
    -- h : x ∈ f ⁻¹' u ∪ f ⁻¹' v
```

```
    -- ⊢ x ∈ f ⁻¹' (u ∪ v)
    rcases h with xfu | xfv
    . -- xfu : x ∈ f ⁻¹' u
      left
      -- ⊢ f x ∈ u
      exact xfu
    . -- xfv : x ∈ f ⁻¹' v
      right
      -- ⊢ f x ∈ v
      exact xfv

-- 3ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∪ v) ↔ x ∈ f ⁻¹' u ∪ f ⁻¹' v
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∪ v) → x ∈ f ⁻¹' u ∪ f ⁻¹' v
    rintro (fxu | fxv)
    . -- fxu : f x ∈ u
      -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v
      exact Or.inl fxu
    . -- fxv : f x ∈ v
      -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v
      exact Or.inr fxv
  . -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v → x ∈ f ⁻¹' (u ∪ v)
    rintro (xfu | xfv)
    . -- xfu : x ∈ f ⁻¹' u
      -- ⊢ x ∈ f ⁻¹' (u ∪ v)
      exact Or.inl xfu
    . -- xfv : x ∈ f ⁻¹' v
      -- ⊢ x ∈ f ⁻¹' (u ∪ v)
      exact Or.inr xfv

-- 4ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∪ v) ↔ x ∈ f ⁻¹' u ∪ f ⁻¹' v
```

```
  constructor
  . -- ⊢ x ∈ f ⁻¹' (u ∪ v) → x ∈ f ⁻¹' u ∪ f ⁻¹' v
    aesop
  . -- ⊢ x ∈ f ⁻¹' u ∪ f ⁻¹' v → x ∈ f ⁻¹' (u ∪ v)
    aesop

-- 5ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (u ∪ v) ↔ x ∈ f ⁻¹' u ∪ f ⁻¹' v
  aesop

-- 6ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by ext ; aesop

-- 7ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by ext ; rfl

-- 8ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
rfl

-- 9ª demostración
-- ===============

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
preimage_union

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

example : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v :=
by simp
```

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f '' (s ∩ t) ⊆ (f '' s) ∩ (f '' t)
-- ----------------------------------------------------------------------

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

-- Sea tal que
--    y ∈ f[s ∩ t]
-- Por tanto, existe un x tal que
--   x ∈ s ∩ t                                                     (1)
--   f(x) = y                                                      (2)
-- Por (1), se tiene que
--   x ∈ s                                                         (3)
--   x ∈ t                                                         (4)
-- Por (2) y (3), se tiene
--   y ∈ f[s]                                                      (5)
-- Por (2) y (4), se tiene
--   y ∈ f[t]                                                      (6)
-- Por (5) y (6), se tiene
--   y ∈ f[s] ∩ f[t]

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

-- 1ª demostración
-- ===============

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' (s ∩ t)
  -- ⊢ y ∈ f '' s ∩ f '' t
  rcases hy with ⟨x, hx⟩
  -- x : α
  -- hx : x ∈ s ∩ t ∧ f x = y
  rcases hx with ⟨xst, fxy⟩
  -- xst : x ∈ s ∩ t
  -- fxy : f x = y
  constructor
  . -- ⊢ y ∈ f '' s
    use x
```

```
    -- ⊢ x ∈ s ∧ f x = y
    constructor
    . -- ⊢ x ∈ s
      exact xst.1
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' t
    use x
    -- ⊢ x ∈ t ∧ f x = y
    constructor
    . -- ⊢ x ∈ t
      exact xst.2
    . -- ⊢ f x = y
      exact fxy

-- 2ª demostración
-- ===============

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' (s ∩ t)
  -- ⊢ y ∈ f '' s ∩ f '' t
  rcases hy with ⟨x, ⟨xs, xt⟩, fxy⟩
  -- x : α
  -- fxy : f x = y
  -- xs : x ∈ s
  -- xt : x ∈ t
  constructor
  . -- ⊢ y ∈ f '' s
    use x
  . -- ⊢ y ∈ f '' t
    use x

-- 3ª demostración
-- ===============

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
image_inter_subset f s t

-- 4ª demostración
-- ===============

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
```

```
by intro ; aesop

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es inyectiva, entonces
--    (f '' s) ∩ (f '' t) ⊆ f '' (s ∩ t)
-- ----------------------------------------------------------------------

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

-- Sea y ∈ f[s] ∩ f[t]. Entonces, existen x₁ y x₂ tales que
--    x₁ ∈ s                                                     (1)
--    f(x₁) = y                                                  (2)
--    x₂ ∈ t                                                     (3)
--    f(x₂) = y                                                  (4)
-- De (2) y (4) se tiene que
--    f(x₁) = f(x₂)
-- y, por ser f inyectiva, se tiene que
--    x₁ = x₂
-- y, por (1), se tiene que
--    x₂ ∈ t
-- y, por (3), se tiene que
--    x₂ ∈ s ∩ t
-- Por tanto,
--    f(x₂) ∈ f[s ∩ t]
-- y, por (4),
--    y ∈ f[s ∩ t]

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

-- 1ª demostración
-- ===============

example
  (h : Injective f)
  : f '' s ∩ f '' t ⊆ f '' (s ∩ t) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s ∩ f '' t
  -- ⊢ y ∈ f '' (s ∩ t)
  rcases hy with ⟨hy1, hy2⟩
  -- hy1 : y ∈ f '' s
  -- hy2 : y ∈ f '' t
```

```
  rcases hy1 with ⟨x1, hx1⟩
  -- x1 : α
  -- hx1 : x1 ∈ s ∧ f x1 = y
  rcases hx1 with ⟨x1s, fx1y⟩
  -- x1s : x1 ∈ s
  -- fx1y : f x1 = y
  rcases hy2 with ⟨x2, hx2⟩
  -- x2 : α
  -- hx2 : x2 ∈ t ∧ f x2 = y
  rcases hx2 with ⟨x2t, fx2y⟩
  -- x2t : x2 ∈ t
  -- fx2y : f x2 = y
  have h1 : f x1 = f x2 := Eq.trans fx1y fx2y.symm
  have h2 : x1 = x2 := h (congrArg f (h h1))
  have h3 : x2 ∈ s := by rwa [h2] at x1s
  have h4 : x2 ∈ s ∩ t := by exact ⟨h3, x2t⟩
  have h5 : f x2 ∈ f '' (s ∩ t) := mem_image_of_mem f h4
  show y ∈ f '' (s ∩ t)
  rwa [fx2y] at h5

-- 2ª demostración
-- ===============

example
  (h : Injective f)
  : f '' s ∩ f '' t ⊆ f '' (s ∩ t) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s ∩ f '' t
  -- ⊢ y ∈ f '' (s ∩ t)
  rcases hy  with ⟨hy1, hy2⟩
  -- hy1 : y ∈ f '' s
  -- hy2 : y ∈ f '' t
  rcases hy1 with ⟨x1, hx1⟩
  -- x1 : α
  -- hx1 : x1 ∈ s ∧ f x1 = y
  rcases hx1 with ⟨x1s, fx1y⟩
  -- x1s : x1 ∈ s
  -- fx1y : f x1 = y
  rcases hy2 with ⟨x2, hx2⟩
  -- x2 : α
  -- hx2 : x2 ∈ t ∧ f x2 = y
  rcases hx2 with ⟨x2t, fx2y⟩
  -- x2t : x2 ∈ t
```

```
  -- fx2y : f x2 = y
  use x1
  -- ⊢ x1 ∈ s ∩ t ∧ f x1 = y
  constructor
  . -- ⊢ x1 ∈ s ∩ t
    constructor
    . -- ⊢ x1 ∈ s
      exact x1s
    . -- ⊢ x1 ∈ t
      convert x2t
      -- ⊢ x1 = x2
      apply h
      -- ⊢ f x1 = f x2
      rw [← fx2y] at fx1y
      -- fx1y : f x1 = f x2
      exact fx1y
  . -- ⊢ f x1 = y
    exact fx1y

-- 3ª demostración
-- ===============

example
  (h : Injective f)
  : f '' s ∩ f '' t ⊆ f '' (s ∩ t) :=
by
  rintro y ⟨⟨x1, x1s, fx1y⟩, ⟨x2, x2t, fx2y⟩⟩
  -- y : β
  -- x1 : α
  -- x1s : x1 ∈ s
  -- fx1y : f x1 = y
  -- x2 : α
  -- x2t : x2 ∈ t
  -- fx2y : f x2 = y
  -- ⊢ y ∈ f '' (s ∩ t)
  use x1
  -- ⊢ x1 ∈ s ∩ t ∧ f x1 = y
  constructor
  . -- ⊢ x1 ∈ s ∩ t
    constructor
    . -- ⊢ x1 ∈ s
      exact x1s
    . -- ⊢ x1 ∈ t
      convert x2t
      -- ⊢ x1 = x2
```

```
      apply h
      -- ⊢ f x1 = f x2
      rw [← fx2y] at fx1y
      -- fx1y : f x1 = f x2
      exact fx1y
  . -- ⊢ f x1 = y
    exact fx1y

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (f '' s) \ (f '' t) ⊆ f '' (s \ t)
-- ----------------------------------------------------------------------

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

-- Sea y ∈ f[s] \ f[t]. Entonces,
--    y ∈ f[s]                                                      (1)
--    y ∉ f[t]                                                      (2)
-- Por (1), existe un x tal que
--    x ∈ s                                                         (3)
--    f(x) = y                                                      (4)
-- Por tanto, para demostrar que y ∈ f[s \ t], basta probar que
-- x ∉ t. Para ello, supongamos que x ∈ t. Entonces, por (4),
-- y ∈ f[t] en contradicción con (2).

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

-- 1ª demostración
-- ===============

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s \ f '' t
  -- ⊢ y ∈ f '' (s \ t)
  rcases hy with ⟨yfs, ynft⟩
  -- yfs : y ∈ f '' s
  -- ynft : ¬y ∈ f '' t
  rcases yfs with ⟨x, hx⟩
  -- x : α
  -- hx : x ∈ s ∧ f x = y
  rcases hx with ⟨xs, fxy⟩
```

```
  -- xs : x ∈ s
  -- fxy : f x = y
  have h1 : x ∉ t := by
    intro xt
    -- xt : x ∈ t
    -- ⊢ False
    have h2 : f x ∈ f '' t := mem_image_of_mem f xt
    have h3 : y ∈ f '' t := by rwa [fxy] at h2
    show False
    exact ynft h3
  have h4 : x ∈ s \ t := mem_diff_of_mem xs h1
  have h5 : f x ∈ f '' (s \ t) := mem_image_of_mem f h4
  show y ∈ f '' (s \ t)
  rwa [fxy] at h5

-- 2ª demostración
-- ===============

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ f '' s \ f '' t
  -- ⊢ y ∈ f '' (s \ t)
  rcases hy with ⟨yfs, ynft⟩
  -- yfs : y ∈ f '' s
  -- ynft : ¬y ∈ f '' t
  rcases yfs with ⟨x, hx⟩
  -- x : α
  -- hx : x ∈ s ∧ f x = y
  rcases hx with ⟨xs, fxy⟩
  -- xs : x ∈ s
  -- fxy : f x = y
  use x
  -- ⊢ x ∈ s \ t ∧ f x = y
  constructor
  . -- ⊢ x ∈ s \ t
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ ¬x ∈ t
      intro xt
      -- xt : x ∈ t
      -- ⊢ False
      apply ynft
```

```
      -- ⊢ y ∈ f '' t
      rw [←fxy]
      -- ⊢ f x ∈ f '' t
      apply mem_image_of_mem
      -- ⊢ x ∈ t
      exact xt
  . -- ⊢ f x = y
    exact fxy

-- 3ª demostración
-- ===============

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
by
  rintro y ⟨⟨x, xs, fxy⟩, ynft⟩
  -- y : β
  -- ynft : ¬y ∈ f '' t
  -- x : α
  -- xs : x ∈ s
  -- fxy : f x = y
  -- ⊢ y ∈ f '' (s \ t)
  use x
  -- ⊢ x ∈ s \ t ∧ f x = y
  aesop

-- 4ª demostración
-- ===============

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
fun y ⟨⟨x, xs, fxy⟩, ynft⟩ ↦ ⟨x, by aesop⟩

-- 5ª demostración
-- ===============

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
subset_image_diff f s t
```

```
-- Ejercicio. Demostrar que
--    (f ⁻¹' u) \ (f ⁻¹' v) ⊆ f ⁻¹' (u \ v)
-- ----------------------------------------------------------------------

example : (f ⁻¹' u) \ (f ⁻¹' v) ⊆ f ⁻¹' (u \ v) :=
by
```

```
  rintro x ⟨hxu,hxv⟩
  -- x : α
  -- hxu : x ∈ f ⁻¹' u
  -- hxv : x ∉ f ⁻¹' v
  -- ⊢ x ∈ f ⁻¹' (u \ v)
  simp
  -- ⊢ f x ∈ u ∧ f x ∉ v
  constructor
  . -- ⊢ f x ∈ u
    exact hxu
  . -- ⊢ f x ∉ v
    intro h
    -- h : f x ∈ v
    -- ⊢ False
    apply hxv
    -- ⊢ x ∈ f ⁻¹' v
    exact h

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (f '' s) ∩ v = f '' (s ∩ (f ⁻¹' v))
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que, para toda y,
--    y ∈ f[s] ∩ v ↔ y ∈ f[s ∩ f⁻¹[v]]
-- Lo haremos probando las dos implicaciones.
--
-- (⟹) Supongamos que y ∈ f[s] ∩ v. Entonces, se tiene que
--    y ∈ f[s]                                                      (1)
--    y ∈ v                                                         (2)
-- Por (1), existe un x tal que
--    x ∈ s                                                         (3)
--    f(x) = y                                                      (4)
-- Por (2) y (4),
--    f(x) ∈ v
-- y, por tanto,
--    x ∈ f⁻¹[v]
-- que, junto con (3), da
--    x ∈ s ∩ f⁻¹[v]
-- y, por tanto,
--    f(x) ∈ f[s ∩ f⁻¹[v]]
-- que, junto con (4), da
```

```
--    y ∈ f[s ∩ f⁻¹[v]]
--
-- (⟸) Supongamos que y ∈ f[s ∩ f⁻¹[v]]. Entonces, existe un x tal que
--    x ∈ s ∩ f⁻¹[v]                                                (5)
--    f(x) = y                                                      (6)
-- Por (1), se tiene que
--    x ∈ s                                                         (7)
--    x ∈ f⁻¹[v]                                                    (8)
-- Por (7) se tiene que
--    f(x) ∈ f[s]
-- y, junto con (6), se tiene que
--    y ∈ f[s]                                                      (9)
-- Por (8), se tiene que
--    f(x) ∈ v
-- y, junto con (6), se tiene que
--    y ∈ v                                                        (10)
-- Por (9) y (19), se tiene que
--    y ∈ f[s] ∩ v

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

-- 1ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ v ↔ y ∈ f '' (s ∩ f ⁻¹' v)
  have h1 : y ∈ f '' s ∩ v → y ∈ f '' (s ∩ f ⁻¹' v) := by
    intro hy
    -- hy : y ∈ f '' s ∩ v
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v)
    have h1a : y ∈ f '' s := hy.1
    obtain ⟨x : α, hx: x ∈ s ∧ f x = y⟩ := h1a
    have h1b : x ∈ s := hx.1
    have h1c : f x = y := hx.2
    have h1d : y ∈ v := hy.2
    have h1e : f x ∈ v := by rwa [←h1c] at h1d
    have h1f : x ∈ s ∩ f ⁻¹' v := mem_inter h1b h1e
    have h1g : f x ∈ f '' (s ∩ f ⁻¹' v) := mem_image_of_mem f h1f
    show y ∈ f '' (s ∩ f ⁻¹' v)
    rwa [h1c] at h1g
  have h2 : y ∈ f '' (s ∩ f ⁻¹' v) → y ∈ f '' s ∩ v :=  by
```

```
    intro hy
    -- hy : y ∈ f '' (s ∩ f ⁻¹' v)
    -- ⊢ y ∈ f '' s ∩ v
    obtain ⟨x : α, hx : x ∈ s ∩ f ⁻¹' v ∧ f x = y⟩ := hy
    have h2a : x ∈ s := hx.1.1
    have h2b : f x ∈ f '' s := mem_image_of_mem f h2a
    have h2c : y ∈ f '' s := by rwa [hx.2] at h2b
    have h2d : x ∈ f ⁻¹' v := hx.1.2
    have h2e : f x ∈ v := mem_preimage.mp h2d
    have h2f : y ∈ v := by rwa [hx.2] at h2e
    show y ∈ f '' s ∩ v
    exact mem_inter h2c h2f
  show y ∈ f '' s ∩ v ↔ y ∈ f '' (s ∩ f ⁻¹' v)
  exact ⟨h1, h2⟩

-- 2ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ v ↔ y ∈ f '' (s ∩ f ⁻¹' v)
  constructor
  . -- ⊢ y ∈ f '' s ∩ v → y ∈ f '' (s ∩ f ⁻¹' v)
    intro hy
    -- hy : y ∈ f '' s ∩ v
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v)
    cases' hy with hyfs yv
    -- hyfs : y ∈ f '' s
    -- yv : y ∈ v
    cases' hyfs with x hx
    -- x : α
    -- hx : x ∈ s ∧ f x = y
    cases' hx with xs fxy
    -- xs : x ∈ s
    -- fxy : f x = y
    use x
    -- ⊢ x ∈ s ∩ f ⁻¹' v ∧ f x = y
    constructor
    . -- ⊢ x ∈ s ∩ f ⁻¹' v
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ f ⁻¹' v
```

```
        rw [mem_preimage]
        -- ⊢ f x ∈ v
        rw [fxy]
        -- ⊢ y ∈ v
        exact yv
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v) → y ∈ f '' s ∩ v
    intro hy
    -- hy : y ∈ f '' (s ∩ f ⁻¹' v)
    -- ⊢ y ∈ f '' s ∩ v
    cases' hy with x hx
    -- x : α
    -- hx : x ∈ s ∩ f ⁻¹' v ∧ f x = y
    constructor
    . -- ⊢ y ∈ f '' s
      use x
      -- ⊢ x ∈ s ∧ f x = y
      constructor
      . -- ⊢ x ∈ s
        exact hx.1.1
      . -- ⊢ f x = y
        exact hx.2
    . -- ⊢ y ∈ v
      cases' hx with hx1 fxy
      -- hx1 : x ∈ s ∩ f ⁻¹' v
      -- fxy : f x = y
      rw [←fxy]
      -- ⊢ f x ∈ v
      rw [←mem_preimage]
      -- ⊢ x ∈ f ⁻¹' v
      exact hx1.2

-- 3ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ v ↔ y ∈ f '' (s ∩ f ⁻¹' v)
  constructor
  . -- ⊢ y ∈ f '' s ∩ v → y ∈ f '' (s ∩ f ⁻¹' v)
    rintro ⟨⟨x, xs, fxy⟩, yv⟩
    -- yv : y ∈ v
```

```
    -- x : α
    -- xs : x ∈ s
    -- fxy : f x = y
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v)
    use x
    -- ⊢ x ∈ s ∩ f ⁻¹' v ∧ f x = y
    constructor
    . -- ⊢ x ∈ s ∩ f ⁻¹' v
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ f ⁻¹' v
        rw [mem_preimage]
        -- ⊢ f x ∈ v
        rw [fxy]
        -- ⊢ y ∈ v
        exact yv
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v) → y ∈ f '' s ∩ v
    rintro ⟨x, ⟨xs, xv⟩, fxy⟩
    -- x : α
    -- fxy : f x = y
    -- xs : x ∈ s
    -- xv : x ∈ f ⁻¹' v
    -- ⊢ y ∈ f '' s ∩ v
    constructor
    . -- ⊢ y ∈ f '' s
      use x, xs
    . -- ⊢ y ∈ v
      rw [←fxy]
      -- ⊢ f x ∈ v
      rw [←mem_preimage]
      -- ⊢ x ∈ f ⁻¹' v
      exact xv

-- 4ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ v ↔ y ∈ f '' (s ∩ f ⁻¹' v)
  constructor
```

```
  . -- ⊢ y ∈ f '' s ∩ v → y ∈ f '' (s ∩ f ⁻¹' v)
    rintro ⟨⟨x, xs, fxy⟩, yv⟩
    -- yv : y ∈ v
    -- x : α
    -- xs : x ∈ s
    -- fxy : f x = y
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v)
    aesop
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' v) → y ∈ f '' s ∩ v
    rintro ⟨x, ⟨xs, xv⟩, fxy⟩
    -- x : α
    -- fxy : f x = y
    -- xs : x ∈ s
    -- xv : x ∈ f ⁻¹' v
    -- ⊢ y ∈ f '' s ∩ v
    aesop

-- 5ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by ext ; constructor <;> aesop

-- 6ª demostración
-- ===============

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
(image_inter_preimage f s v).symm

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f '' (s ∩ f ⁻¹' u) ⊆ (f '' s) ∪ u
-- ----------------------------------------------------------------------

example : f '' (s ∩ f ⁻¹' u) ⊆ (f '' s) ∪ u :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' (s ∩ f ⁻¹' u)
  -- ⊢ y ∈ f '' s ∪ u
  rcases h with ⟨x,⟨xs,xu⟩,fxy⟩
  -- x : α
  -- fxy : f x = y
  -- xs : x ∈ s
  -- xu : x ∈ f ⁻¹' u
```

```
  right
  -- ⊢ y ∈ u
  rw [←fxy]
  -- ⊢ f x ∈ u
  exact xu

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∩ f ⁻¹' u ⊆ f ⁻¹' ((f '' s) ∩ u) :=
-- ----------------------------------------------------------------------

example : s ∩ f ⁻¹' u ⊆ f ⁻¹' ((f '' s) ∩ u) :=
by
  rintro x ⟨xs,xu⟩
  -- x : α
  -- xs : x ∈ s
  -- xu : x ∈ f ⁻¹' u
  -- ⊢ x ∈ f ⁻¹' (f '' s ∩ u)
  simp at xu
  -- xu : f x ∈ u
  constructor
  . -- ⊢ f x ∈ f '' s
    exact mem_image_of_mem f xs
  . -- ⊢ f x ∈ u
    exact xu

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    s ∪ f ⁻¹' v ⊆ f ⁻¹' ((f '' s) ∪ v)
-- ----------------------------------------------------------------------

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

-- Sea x ∈ s ∪ f⁻¹[v]. Entonces, se puede dar dos casos.
--
-- Caso 1: Supongamos que x ∈ s. Entonces, se tiene
--    f(x) ∈ f[s]
--    f(x) ∈ f[s] ∪ v
--    x ∈ f⁻¹[f[s] ∪ v]
--
-- Caso 2: Supongamos que x ∈ f⁻¹[v]. Entonces, se tiene
--    f(x) ∈ v
--    f(x) ∈ f[s] ∪ v
--    x ∈ f⁻¹[f[s] ∪ v]
```

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

-- 1ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ s ∪ f ⁻¹' v
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  rcases hx with xs | xv
  . -- xs : x ∈ s
    have h1 : f x ∈ f '' s := mem_image_of_mem f xs
    have h2 : f x ∈ f '' s ∪ v := mem_union_left v h1
    show x ∈ f ⁻¹' (f '' s ∪ v)
    exact mem_preimage.mpr h2
  . -- xv : x ∈ f ⁻¹' v
    have h3 : f x ∈ v := mem_preimage.mp xv
    have h4 : f x ∈ f '' s ∪ v := mem_union_right (f '' s) h3
    show x ∈ f ⁻¹' (f '' s ∪ v)
    exact mem_preimage.mpr h4

-- 2ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ s ∪ f ⁻¹' v
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  rw [mem_preimage]
  -- ⊢ f x ∈ f '' s ∪ v
  rcases hx with xs | xv
  . -- xs : x ∈ s
    apply mem_union_left
    -- ⊢ f x ∈ f '' s
    apply mem_image_of_mem
    -- ⊢ x ∈ s
    exact xs
  . -- xv : x ∈ f ⁻¹' v
    apply mem_union_right
```

```
    -- ⊢ f x ∈ v
    rw [←mem_preimage]
    -- ⊢ x ∈ f ⁻¹' v
    exact xv

-- 3ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  intros x hx
  -- x : α
  -- hx : x ∈ s ∪ f ⁻¹' v
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  rcases hx with xs | xv
  . -- xs : x ∈ s
    rw [mem_preimage]
    -- ⊢ f x ∈ f '' s ∪ v
    apply mem_union_left
    -- ⊢ f x ∈ f '' s
    apply mem_image_of_mem
    -- ⊢ x ∈ s
    exact xs
  . -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
    rw [mem_preimage]
    -- ⊢ f x ∈ f '' s ∪ v
    apply mem_union_right
    -- ⊢ f x ∈ v
    exact xv

-- 4ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  rintro x (xs | xv)
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  . -- xs : x ∈ s
    left
    -- ⊢ f x ∈ f '' s
    exact mem_image_of_mem f xs
  . -- xv : x ∈ f ⁻¹' v
    right
    -- ⊢ f x ∈ v
```

```
    exact xv

-- 5ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  rintro x (xs | xv)
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  . -- xs : x ∈ s
    exact Or.inl (mem_image_of_mem f xs)
  . -- xv : x ∈ f ⁻¹' v
    exact Or.inr xv

-- 5ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
by
  intros x h
  -- x : α
  -- h : x ∈ s ∪ f ⁻¹' v
  -- ⊢ x ∈ f ⁻¹' (f '' s ∪ v)
  exact Or.elim h (fun xs ↦ Or.inl (mem_image_of_mem f xs)) Or.inr

-- 6ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
fun _ h ↦ Or.elim h (fun xs ↦ Or.inl (mem_image_of_mem f xs)) Or.inr

-- 7ª demostración
-- ===============

example : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v) :=
union_preimage_subset s v f

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

variable (x : α)
variable (z : β)
variable (t : Set α)
variable (a b c : Prop)
```

```
#check (Eq.trans : a = b → b = c → a = c)
#check (Or.elim : a ∨ b → (a → c) → (b → c) → c)
#check (Or.inl : a → a ∨ b)
#check (Or.inr : b → a ∨ b)
#check (image_inter_preimage f s v : f '' (s ∩ f ⁻¹' v) = f '' s ∩ v)
#check (image_inter_subset f s t : f '' (s ∩ t) ⊆ f '' s ∩ f '' t)
#check (image_preimage_subset f u : f '' (f⁻¹' u) ⊆ u)
#check (image_subset f : s ⊆ t → f '' s ⊆ f '' t)
#check (mem_diff_of_mem : x ∈ s → x ∉ t → x ∈ s \ t)
#check (mem_image f s z : (z ∈ f '' s ↔ ∃ x, x ∈ s ∧ f x = z))
#check (mem_image_of_mem f : x  ∈ s → f x ∈ f '' s)
#check (mem_inter : x ∈ s → x ∈ t → x ∈ s ∩ t)
#check (mem_preimage : x ∈ f ⁻¹' v ↔ f x ∈ v)
#check (mem_union_left t : x ∈ s → x ∈ s ∪ t)
#check (mem_union_right s : x ∈ t → x ∈ s ∪ t)
#check (preimage_mono : u ⊆ v → f ⁻¹' u ⊆ f ⁻¹' v)
#check (preimage_union : f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v)
#check (subset_image_diff f s t : f '' s \ f '' t ⊆ f '' (s \ t))
#check (union_preimage_subset s v f : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v))
```

4.2.6. Ejercicios de imágenes y uniones

```
import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set Function

variable {α β I : Type _}
variable (f : α → β)
variable (A : I → Set α)
variable (B : I → Set β)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f[⋃ᵢAᵢ] = ⋃ᵢf[Aᵢ]
-- ---------------------------------------------------------------------

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

-- Tenemos que demostrar que, para todo y,
--    y ∈ f[⋃ᵢAᵢ] ↔ y ∈ ⋃ᵢf[Aᵢ]
-- Lo haremos demostrando las dos implicaciones.
```

```
--
-- (⟹) Supongamos que y ∈ f[⋃ᵢAᵢ]. Entonces, existe un x tal que
--    x ∈ ⋃ᵢAᵢ                                                  (1)
--    f(x) = y                                                  (2)
-- Por (1), existe un i tal que
--    i ∈ ℕ                                                     (3)
--    x ∈ Aᵢ                                                    (4)
-- Por (4),
--    f(x) ∈ f[Aᵢ]
-- Por (3),
--    f(x) ∈ ⋃ᵢf[Aᵢ]
-- y, por (2),
--    y ∈ ⋃ᵢf[Aᵢ]
--
-- (⟸) Supongamos que y ∈ ⋃ᵢf[Aᵢ]. Entonces, existe un i tal que
--    i ∈ ℕ                                                     (5)
--    y ∈ f[Aᵢ]                                                 (6)
-- Por (6), existe un x tal que
--    x ∈ Aᵢ                                                    (7)
--    f(x) = y                                                  (8)
-- Por (5) y (7),
--    x ∈ ⋃ᵢAᵢ
-- Luego,
--    f(x) ∈ f[⋃ᵢAᵢ]
-- y, por (8),
--    y ∈ f[⋃ᵢAᵢ]

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

-- 1ª demostración
-- ===============

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i ↔ y ∈ ⋃ (i : ℕ), f '' A i
  constructor
  . -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i → y ∈ ⋃ (i : ℕ), f '' A i
    intro hy
    -- hy : y ∈ f '' ⋃ (i : ℕ), A i
    -- ⊢ y ∈ ⋃ (i : ℕ), f '' A i
    have h1 : ∃ x, x ∈ ⋃ i, A i ∧ f x = y :=
      (mem_image f (⋃ i, A i) y).mp hy
```

```
    obtain ⟨x, hx : x ∈ ⋃ i, A i ∧ f x = y⟩ := h1
    have xUA : x ∈ ⋃ i, A i := hx.1
    have fxy : f x = y := hx.2
    have xUA : ∃ i, x ∈ A i := mem_iUnion.mp xUA
    obtain ⟨i, xAi : x ∈ A i⟩ := xUA
    have h2 : f x ∈ f '' A i := mem_image_of_mem f xAi
    have h3 : f x ∈ ⋃ i, f '' A i := mem_iUnion_of_mem i h2
    show y ∈ ⋃ i, f '' A i
    rwa [fxy] at h3
  . -- ⊢ y ∈ ⋃ (i : ℕ), f '' A i → y ∈ f '' ⋃ (i : ℕ), A i
    intro hy
    -- hy : y ∈ ⋃ (i : ℕ), f '' A i
    -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i
    have h4 : ∃ i, y ∈ f '' A i := mem_iUnion.mp hy
    obtain ⟨i, h5 : y ∈ f '' A i⟩ := h4
    have h6 : ∃ x, x ∈ A i ∧ f x = y :=
      (mem_image f (A i) y).mp h5
    obtain ⟨x, h7 : x ∈ A i ∧ f x = y⟩ := h6
    have h8 : x ∈ A i := h7.1
    have h9 : x ∈ ⋃ i, A i := mem_iUnion_of_mem i h8
    have h10 : f x ∈ f '' (⋃ i, A i) := mem_image_of_mem f h9
    show y ∈ f '' (⋃ i, A i)
    rwa [h7.2] at h10

-- 2ª demostración
-- ===============

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i ↔ y ∈ ⋃ (i : ℕ), f '' A i
  constructor
  . -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i → y ∈ ⋃ (i : ℕ), f '' A i
    intro hy
    -- hy : y ∈ f '' ⋃ (i : ℕ), A i
    -- ⊢ y ∈ ⋃ (i : ℕ), f '' A i
    rw [mem_image] at hy
    -- hy : ∃ x, x ∈ ⋃ (i : ℕ), A i ∧ f x = y
    cases' hy with x hx
    -- x : α
    -- hx : x ∈ ⋃ (i : ℕ), A i ∧ f x = y
    cases' hx with xUA fxy
    -- xUA : x ∈ ⋃ (i : ℕ), A i
    -- fxy : f x = y
```

```
    rw [mem_iUnion] at xUA
    -- xUA : ∃ i, x ∈ A i
    cases' xUA with i xAi
    -- i : ℕ
    -- xAi : x ∈ A i
    rw [mem_iUnion]
    -- ⊢ ∃ i, y ∈ f '' A i
    use i
    -- ⊢ y ∈ f '' A i
    rw [←fxy]
    -- ⊢ f x ∈ f '' A i
    apply mem_image_of_mem
    -- ⊢ x ∈ A i
    exact xAi
  . -- ⊢ y ∈ ⋃ (i : ℕ), f '' A i → y ∈ f '' ⋃ (i : ℕ), A i
    intro hy
    -- hy : y ∈ ⋃ (i : ℕ), f '' A i
    -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i
    rw [mem_iUnion] at hy
    -- hy : ∃ i, y ∈ f '' A i
    cases' hy with i yAi
    -- i : ℕ
    -- yAi : y ∈ f '' A i
    cases' yAi with x hx
    -- x : α
    -- hx : x ∈ A i ∧ f x = y
    cases' hx with xAi fxy
    -- xAi : x ∈ A i
    -- fxy : f x = y
    rw [←fxy]
    -- ⊢ f x ∈ f '' ⋃ (i : ℕ), A i
    apply mem_image_of_mem
    -- ⊢ x ∈ ⋃ (i : ℕ), A i
    rw [mem_iUnion]
    -- ⊢ ∃ i, x ∈ A i
    use i

-- 3ª demostración
-- ===============

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' ⋃ (i : ℕ), A i ↔ y ∈ ⋃ (i : ℕ), f '' A i
```

```
  simp
  -- ⊢ (∃ x, (∃ i, x ∈ A i) ∧ f x = y) ↔ ∃ i x, x ∈ A i ∧ f x = y
  constructor
  . -- ⊢ (∃ x, (∃ i, x ∈ A i) ∧ f x = y) → ∃ i x, x ∈ A i ∧ f x = y
    rintro ⟨x, ⟨i, xAi⟩, fxy⟩
    -- x : α
    -- fxy : f x = y
    -- i : ℕ
    -- xAi : x ∈ A i
    -- ⊢ ∃ i x, x ∈ A i ∧ f x = y
    use i, x, xAi
  . -- ⊢ (∃ i x, x ∈ A i ∧ f x = y) → ∃ x, (∃ i, x ∈ A i) ∧ f x = y
    rintro ⟨i, x, xAi, fxy⟩
    -- i : ℕ
    -- x : α
    -- xAi : x ∈ A i
    -- fxy : f x = y
    -- ⊢ ∃ x, (∃ i, x ∈ A i) ∧ f x = y
    exact ⟨x, ⟨i, xAi⟩, fxy⟩

-- 4ª demostración
-- ===============

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
image_iUnion

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i
-- ----------------------------------------------------------------------

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

-- Sea y tal que
--    y ∈ f[⋂ᵢ Aᵢ]                                                   (1)
-- Tenemos que demostrar que y ∈ ⋂ᵢ f[Aᵢ]. Para ello, sea i ∈ I, tenemos
-- que demostrar que y ∈ f[Aᵢ].
--
-- Por (1), existe un x tal que
--    x ∈ ⋂ᵢ Aᵢ                                                      (2)
--    f(x) = y                                                       (3)
-- Por (2),
--    x ∈ Aᵢ
-- y, por tanto,
```

```
--    f(x) ∈ f[Aᵢ]
-- que, junto con (3), da que
--    y ∈ f[Aᵢ]

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

-- 1ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' ⋂ (i : I), A i
  -- ⊢ y ∈ ⋂ (i : I), f '' A i
  have h1 : ∃ x, x ∈ ⋂ i, A i ∧ f x = y :=
    (mem_image f (⋂ i, A i) y).mp h
  obtain ⟨x, hx : x ∈ ⋂ i, A i ∧ f x = y⟩ := h1
  have h2 : x ∈ ⋂ i, A i := hx.1
  have h3 : f x = y := hx.2
  have h4 : ∀ i, y ∈ f '' A i := by
    intro i
    have h4a : x ∈ A i := mem_iInter.mp h2 i
    have h4b : f x ∈ f '' A i := mem_image_of_mem f h4a
    show y ∈ f '' A i
    rwa [h3] at h4b
  show y ∈ ⋂ i, f '' A i
  exact mem_iInter.mpr h4

-- 2ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' ⋂ (i : I), A i
  -- ⊢ y ∈ ⋂ (i : I), f '' A i
  apply mem_iInter_of_mem
  -- ⊢ ∀ (i : I), y ∈ f '' A i
  intro i
  -- i : I
  -- ⊢ y ∈ f '' A i
  cases' h with x hx
```

```
  -- x : α
  -- hx : x ∈ ⋂ (i : I), A i ∧ f x = y
  cases' hx with xIA fxy
  -- xIA : x ∈ ⋂ (i : I), A i
  -- fxy : f x = y
  rw [←fxy]
  -- ⊢ f x ∈ f '' A i
  apply mem_image_of_mem
  -- ⊢ x ∈ A i
  exact mem_iInter.mp xIA i

-- 3ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
by
  intros y h
  -- y : β
  -- h : y ∈ f '' ⋂ (i : I), A i
  -- ⊢ y ∈ ⋂ (i : I), f '' A i
  apply mem_iInter_of_mem
  -- ⊢ ∀ (i : I), y ∈ f '' A i
  intro i
  -- i : I
  -- ⊢ y ∈ f '' A i
  rcases h with ⟨x, xIA, rfl⟩
  -- x : α
  -- xIA : x ∈ ⋂ (i : I), A i
  -- ⊢ f x ∈ f '' A i
  exact mem_image_of_mem f (mem_iInter.mp xIA i)

-- 4ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
by
  intro y
  -- y : β
  -- ⊢ y ∈ f '' ⋂ (i : I), A i → y ∈ ⋂ (i : I), f '' A i
  simp
  -- ⊢ ∀ (x : α), (∀ (i : I), x ∈ A i) → f x = y → ∀ (i : I), ∃ x, x ∈ A i ∧ f x = y
  intros x xIA fxy i
  -- x : α
  -- xIA : ∀ (i : I), x ∈ A i
  -- fxy : f x = y
```

```
  -- i : I
  -- ⊢ ∃ x, x ∈ A i ∧ f x = y
  use x, xIA i

-- 5ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
image_iInter_subset A f

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es inyectiva e I no vacío, entonces
--    (⋂ i, f '' A i) ⊆ f '' (⋂ i, A i)
-- ----------------------------------------------------------------------

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

-- Sea y ∈ ⋂ᵢf[Aᵢ]. Entonces,
--    (∀i ∈ I)y ∈ f[Aᵢ]                                             (1)
--    y ∈ f[Aᵢ]
-- Por tanto, existe un x ∈ Aᵢ tal que
--    f(x) = y                                                      (2)
--
-- Veamos que x ∈ ⋂ᵢAᵢ. Para ello, sea j ∈ I. Por (1),
--    y ∈ f[Aⱼ]
-- Luego, existe un z tal que
--    z ∈ Aⱼ                                                        (3)
--    f(z) = y
-- Por (2),
--    f(x) = f(z)
-- y, por ser f inyectiva,
--    x = z
-- y, Por (3),
--    x ∈ Aⱼ
--
-- Puesto que x ∈ ⋂ᵢAᵢ se tiene que f(x) ∈ f[⋂ᵢAᵢ] y, por (2),
-- y ∈ f[⋂ᵢAᵢ].

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

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

```
example
  (i : I)
  (injf : Injective f)
  : (⋂ i, f '' A i) ⊆ f '' (⋂ i, A i) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ ⋂ (i : I), f '' A i
  -- ⊢ y ∈ f '' ⋂ (i : I), A i
  have h1 : ∀ (i : I), y ∈ f '' A i := mem_iInter.mp hy
  have h2 : y ∈ f '' A i := h1 i
  obtain ⟨x : α, h3 : x ∈ A i ∧ f x = y⟩ := h2
  have h4 : f x = y := h3.2
  have h5 : ∀ i : I, x ∈ A i := by
    intro j
    have h5a : y ∈ f '' A j := h1 j
    obtain ⟨z : α, h5b : z ∈ A j ∧ f z = y⟩ := h5a
    have h5c : z ∈ A j := h5b.1
    have h5d : f z = y := h5b.2
    have h5e : f z = f x := by rwa [←h4] at h5d
    have h5f : z = x := injf h5e
    show x ∈ A j
    rwa [h5f] at h5c
  have h6 : x ∈ ⋂ i, A i := mem_iInter.mpr h5
  have h7 : f x ∈ f '' (⋂ i, A i) := mem_image_of_mem f h6
  show y ∈ f '' (⋂ i, A i)
  rwa [h4] at h7

-- 2ª demostración
-- ===============

example
  (i : I)
  (injf : Injective f)
  : (⋂ i, f '' A i) ⊆ f '' (⋂ i, A i) :=
by
  intros y hy
  -- y : β
  -- hy : y ∈ ⋂ (i : I), f '' A i
  -- ⊢ y ∈ f '' ⋂ (i : I), A i
  rw [mem_iInter] at hy
  -- hy : ∀ (i : I), y ∈ f '' A i
  rcases hy i with ⟨x, -, fxy⟩
  -- x : α
  -- fxy : f x = y
```

```
  use x
  -- ⊢ x ∈ ⋂ (i : I), A i ∧ f x = y
  constructor
  . -- ⊢ x ∈ ⋂ (i : I), A i
    apply mem_iInter_of_mem
    -- ⊢ ∀ (i : I), x ∈ A i
    intro j
    -- j : I
    -- ⊢ x ∈ A j
    rcases hy j with ⟨z, zAj, fzy⟩
    -- z : α
    -- zAj : z ∈ A j
    -- fzy : f z = y
    convert zAj
    -- ⊢ x = z
    apply injf
    -- ⊢ f x = f z
    rw [fxy]
    -- ⊢ y = f z
    rw [←fzy]
  . -- ⊢ f x = y
    exact fxy

-- 3ª demostración
-- ===============

example
  (i : I)
  (injf : Injective f)
  : (⋂ i, f '' A i) ⊆ f '' (⋂ i, A i) :=
by
  intro y
  -- y : β
  -- ⊢ y ∈ ⋂ (i : I), f '' A i → y ∈ f '' ⋂ (i : I), A i
  simp
  -- ⊢ (∀ (i : I), ∃ x, x ∈ A i ∧ f x = y) → ∃ x, (∀ (i : I), x ∈ A i) ∧ f x = y
  intro h
  -- h : ∀ (i : I), ∃ x, x ∈ A i ∧ f x = y
  -- ⊢ ∃ x, (∀ (i : I), x ∈ A i) ∧ f x = y
  rcases h i with ⟨x, -, fxy⟩
  -- x : α
  -- fxy : f x = y
  use x
  -- ⊢ (∀ (i : I), x ∈ A i) ∧ f x = y
  constructor
```

```
  . -- ⊢ ∀ (i : I), x ∈ A i
    intro j
    -- j : I
    -- ⊢ x ∈ A j
    rcases h j with ⟨z, zAi, fzy⟩
    -- z : α
    -- zAi : z ∈ A j
    -- fzy : f z = y
    have : f x = f z := by rw [fxy, fzy]
    -- this : f x = f z
    have : x = z := injf this
    -- this : x = z
    rw [this]
    -- ⊢ z ∈ A j
    exact zAi
  . -- ⊢ f x = y
    exact fxy

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i)
-- ----------------------------------------------------------------------

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

-- Tenemos que demostrar que, para todo x,
--    x ∈ f⁻¹[⋃ᵢ Bᵢ] ↔ x ∈ ⋃ᵢ f⁻¹[Bᵢ]
-- y lo haremos demostrando las dos implicaciones.
--
-- (⟹) Supongamos que x ∈ f⁻¹[⋃ᵢ Bᵢ]. Entonces, por la definición de la
-- imagen inversa,
--    f(x) ∈ ⋃ᵢ Bᵢ
-- y, por la definición de la unión, existe un i tal que
--    f(x) ∈ Bᵢ
-- y, por la definición de la imagen inversa,
--    x ∈ f⁻¹[Bᵢ]
-- y, por la definición de la unión,
--    x ∈ ⋃ᵢ f⁻¹[Bᵢ]
--
-- (⟸) Supongamos que x ∈ ⋃ᵢ f⁻¹[Bᵢ]. Entonces, por la definición de la
-- unión, existe un i tal que
--    x ∈ f⁻¹[Bᵢ]
-- y, por la definición de la imagen inversa,
--    f(x) ∈ Bᵢ
```

```
-- y, por la definición de la unión,
--    f(x) ∈ ⋃ᵢ Bᵢ
-- y, por la definición de la imagen inversa,
--    x ∈ f⁻¹[⋃ᵢ Bᵢ]

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

-- 1ª demostración
-- ===============

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' ⋃ (i : I), B i ↔ x ∈ ⋃ (i : I), f ⁻¹' B i
  constructor
  . -- ⊢ x ∈ f ⁻¹' ⋃ (i : I), B i → x ∈ ⋃ (i : I), f ⁻¹' B i
    intro hx
    -- hx : x ∈ f ⁻¹' ⋃ (i : I), B i
    -- ⊢ x ∈ ⋃ (i : I), f ⁻¹' B i
    rw [mem_preimage] at hx
    -- hx : f x ∈ ⋃ (i : I), B i
    rw [mem_iUnion] at hx
    -- hx : ∃ i, f x ∈ B i
    cases' hx with i fxBi
    -- i : I
    -- fxBi : f x ∈ B i
    rw [mem_iUnion]
    -- ⊢ ∃ i, x ∈ f ⁻¹' B i
    use i
    -- ⊢ x ∈ f ⁻¹' B i
    apply mem_preimage.mpr
    -- ⊢ f x ∈ B i
    exact fxBi
  . -- ⊢ x ∈ ⋃ (i : I), f ⁻¹' B i → x ∈ f ⁻¹' ⋃ (i : I), B i
    intro hx
    -- hx : x ∈ ⋃ (i : I), f ⁻¹' B i
    -- ⊢ x ∈ f ⁻¹' ⋃ (i : I), B i
    rw [mem_preimage]
    -- ⊢ f x ∈ ⋃ (i : I), B i
    rw [mem_iUnion]
    -- ⊢ ∃ i, f x ∈ B i
    rw [mem_iUnion] at hx
    -- hx : ∃ i, x ∈ f ⁻¹' B i
```

```
    cases' hx with i xBi
    -- i : I
    -- xBi : x ∈ f ⁻¹' B i
    use i
    -- ⊢ f x ∈ B i
    rw [mem_preimage] at xBi
    -- xBi : f x ∈ B i
    exact xBi

-- 2ª demostración
-- ===============

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
preimage_iUnion

-- 3ª demostración
-- ===============

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
by  simp

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i)
-- ----------------------------------------------------------------------

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

-- Se demuestra mediante la siguiente cadena de equivalencias
--    x ∈ f⁻¹[⋂ᵢ Bᵢ] ↔ f x ∈ ⋂ᵢ Bᵢ
--                     ↔ (∀ i) f(x) ∈ Bᵢ
--                     ↔ (∀ i) x ∈ f⁻¹[Bᵢ]
--                     ↔ x ∈ ⋂ᵢ f⁻¹[Bᵢ]

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

-- 1ª demostración
-- ===============

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
by
  ext x
  -- x : α
```

```
  -- ⊢ x ∈ f ⁻¹' ⋂ (i : I), B i ↔ x ∈ ⋂ (i : I), f ⁻¹' B i
  calc  (x ∈ f ⁻¹' ⋂ i, B i)
     ↔ f x ∈ ⋂ i, B i       := mem_preimage
   _ ↔ (∀ i, f x ∈ B i)     := mem_iInter
   _ ↔ (∀ i, x ∈ f ⁻¹' B i) := iff_of_eq rfl
   _ ↔ x ∈ ⋂ i, f ⁻¹' B i   := mem_iInter.symm

-- 2ª demostración
-- ===============

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' ⋂ (i : I), B i ↔ x ∈ ⋂ (i : I), f ⁻¹' B i
  constructor
  . -- ⊢ x ∈ f ⁻¹' ⋂ (i : I), B i → x ∈ ⋂ (i : I), f ⁻¹' B i
    intro hx
    -- hx : x ∈ f ⁻¹' ⋂ (i : I), B i
    -- ⊢ x ∈ ⋂ (i : I), f ⁻¹' B i
    apply mem_iInter_of_mem
    -- ⊢ ∀ (i : I), x ∈ f ⁻¹' B i
    intro i
    -- i : I
    -- ⊢ x ∈ f ⁻¹' B i
    rw [mem_preimage]
    -- ⊢ f x ∈ B i
    rw [mem_preimage] at hx
    -- hx : f x ∈ ⋂ (i : I), B i
    rw [mem_iInter] at hx
    -- hx : ∀ (i : I), f x ∈ B i
    exact hx i
  . -- ⊢ x ∈ ⋂ (i : I), f ⁻¹' B i → x ∈ f ⁻¹' ⋂ (i : I), B i
    intro hx
    -- hx : x ∈ ⋂ (i : I), f ⁻¹' B i
    -- ⊢ x ∈ f ⁻¹' ⋂ (i : I), B i
    rw [mem_preimage]
    -- ⊢ f x ∈ ⋂ (i : I), B i
    rw [mem_iInter]
    -- ⊢ ∀ (i : I), f x ∈ B i
    intro i
    -- i : I
    -- ⊢ f x ∈ B i
    rw [←mem_preimage]
    -- ⊢ x ∈ f ⁻¹' B i
```

```
    rw [mem_iInter] at hx
    -- hx : ∀ (i : I), x ∈ f ⁻¹' B i
    exact hx i

-- 3ª demostración
-- ===============

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
by
  ext x
  -- ⊢ x ∈ f ⁻¹' ⋂ (i : I), B i ↔ x ∈ ⋂ (i : I), f ⁻¹' B i
  simp

-- 4ª demostración
-- ===============

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
by { ext ; simp }

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

variable (A : I → Set α)
variable (a b : Prop)
variable (i : I)
variable (s : Set α)
variable (v : Set β)
variable (x : α)
variable (y : β)
#check (iff_of_eq : a = b → (a ↔ b))
#check (image_iInter_subset A f : f '' ⋂ i, A i ⊆ ⋂ i, f '' A i)
#check (image_iUnion : f '' ⋃ i, A i = ⋃ i, f '' A i)
#check (mem_iInter : x ∈ ⋂ i, A i ↔ ∀ i, x ∈ A i)
#check (mem_iInter_of_mem : (∀ i, x ∈ A i) → x ∈ ⋂ i, A i)
#check (mem_iUnion : x ∈ ⋃ i, A i ↔ ∃ i, x ∈ A i)
#check (mem_iUnion_of_mem i : x ∈ A i → x ∈ ⋃ i, A i)
#check (mem_image f s y : (y ∈ f '' s ↔ ∃ x, x ∈ s ∧ f x = y))
#check (mem_image_of_mem f : x ∈ s → f x ∈ f '' s)
#check (mem_preimage : x ∈ f ⁻¹' v ↔ f x ∈ v)
#check (preimage_iUnion : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i))
```

4.2.7. Definición de inyectiva

```
import Mathlib.Data.Set.Function

open Set

universe u v
variable {α : Type u}
variable {β : Type v}
variable (f : α → β)
variable (s : Set α)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que f es inyectiva sobre s syss
--    ∀ x1 ∈ s, ∀ x2 ∈ s, f x1 = f x2 → x1 = x2
-- ----------------------------------------------------------------------

example :
  InjOn f s ↔ ∀ x1 ∈ s, ∀ x2 ∈ s, f x1 = f x2 → x1 = x2 :=
Iff.refl _

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

variable (a : Prop)
#check (Iff.refl a : a ↔ a)
```

4.2.8. Inyectividad del logaritmo

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que lo función logarítmica es inyectiva sobre
-- los números positivos.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Function
import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Set Real

example : InjOn log { x | x > 0 } :=
by
  intro x hx y hy
  -- x : ℝ
```

```
  -- hx : x ∈ {x | x > 0}
  -- y : ℝ
  -- hy : y ∈ {x | x > 0}
  -- ⊢ log x = log y → x = y
  intro e
  -- e : log x = log y
  -- ⊢ x = y
  calc
    x = exp (log x) := by rw [exp_log hx]
    _ = exp (log y) := by rw [e]
    _ = y           := by rw [exp_log hy]

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

variable (x : ℝ)
#check (exp_log : 0 < x → exp (log x) = x)
```

4.2.9. Rango de la exponencial

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que el rango de la función exponencial es el
-- conjunto de los números positivos,
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Function
import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Set Real

example : range exp = { y | y > 0 } := by
  ext y
  -- y : ℝ
  -- ⊢ y ∈ range rexp ↔ y ∈ {y | y > 0}
  constructor
  · -- ⊢ y ∈ range rexp → y ∈ {y | y > 0}
    rintro ⟨x, rfl⟩
    -- x : ℝ
    -- ⊢ rexp x ∈ {y | y > 0}
    apply exp_pos
  . -- ⊢ y ∈ {y | y > 0} → y ∈ range rexp
    intro hy
    -- hy : y ∈ {y | y > 0}
```

```
    -- ⊢ y ∈ range rexp
    use log y
    -- ⊢ rexp (log y) = y
    rw [exp_log hy]

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

variable (x : ℝ)
#check (exp_log : 0 < x → exp (log x) = x)
```

4.2.10. Inyectividad del cuadrado

```
import Mathlib.Data.Real.Basic
import Mathlib.Data.Real.Sqrt

open Set Real

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que la función raíz cuadrada es inyectiva sobre
-- los números no negativos.
-- ----------------------------------------------------------------------

example : InjOn sqrt { x | x ≥ 0 } :=
by
  intro x hx y hy
  -- x : ℝ
  -- hx : x ∈ {x | x ≥ 0}
  -- y : ℝ
  -- hy : y ∈ {x | x ≥ 0}
  intro e
  -- e : √x = √y
  -- ⊢ x = y
  calc
    x = sqrt x ^ 2 := by rw [sq_sqrt hx]
    _ = sqrt y ^ 2 := by rw [e]
    _ = y := by rw [sq_sqrt hy]

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que la función cuadrado es inyectiva sobre
-- los números no negativos.
-- ----------------------------------------------------------------------
```

```
example : InjOn (fun x ↦ x ^ 2) { x : ℝ | x ≥ 0 } :=
by
  intro x hx y hy
  -- x : ℝ
  -- hx : x ∈ {x | x ≥ 0}
  -- y : ℝ
  -- hy : y ∈ {x | x ≥ 0}
  -- ⊢ (fun x => x ^ 2) x = (fun x => x ^ 2) y → x = y
  intro e
  -- e : (fun x => x ^ 2) x = (fun x => x ^ 2) y
  -- ⊢ x = y
  dsimp at *
  -- hx : x ≥ 0
  -- hy : y ≥ 0
  -- e : x ^ 2 = y ^ 2
  -- ⊢ x = y
  calc
    x = sqrt (x ^ 2) := by rw [sqrt_sq hx]
    _ = sqrt (y ^ 2) := by rw [e]
    _ = y := by rw [sqrt_sq hy]

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

variable (x : ℝ)
#check (sq_sqrt : 0 ≤ x → √x ^ 2 = x)
#check (sqrt_sq : 0 ≤ x → √(x ^ 2) = x)
```

4.2.11. Rango del cuadrado

```
import Mathlib.Data.Set.Function
import Mathlib.Data.Real.Basic
import Mathlib.Data.Real.Sqrt

open Set Real

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    sqrt '' { x | x ≥ 0 } = {y | y ≥ 0}
-- ----------------------------------------------------------------------

example : sqrt '' { x | x ≥ 0 } = {y | y ≥ 0} :=
by
```

```
  ext y
  -- ⊢ y ∈ sqrt '' {x | x ≥ 0} ↔ y ∈ {y | y ≥ 0}
  constructor
  · -- ⊢ y ∈ sqrt '' {x | x ≥ 0} → y ∈ {y | y ≥ 0}
    rintro ⟨x, ⟨hx, rfl⟩⟩
    -- x : ℝ
    -- hx : x ∈ {x | x ≥ 0}
    -- ⊢ √x ∈ {y | y ≥ 0}
    apply sqrt_nonneg
  . -- ⊢ y ∈ {y | y ≥ 0} → y ∈ sqrt '' {x | x ≥ 0}
    intro hy
    -- hy : y ∈ {y | y ≥ 0}
    -- ⊢ y ∈ sqrt '' {x | x ≥ 0}
    dsimp at hy
    -- hy : y ≥ 0
    simp
    -- ⊢ ∃ x, 0 ≤ x ∧ √x = y
    use y ^ 2
    -- ⊢ 0 ≤ y ^ 2 ∧ √(y ^ 2) = y
    constructor
    . -- ⊢ 0 ≤ y ^ 2
      apply pow_nonneg hy
    . -- ⊢ √(y ^ 2) = y
      apply sqrt_sq
      -- ⊢ 0 ≤ y
      assumption

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (range fun x ↦ x ^ 2) = {y | y ≥ 0}
-- ----------------------------------------------------------------------

example : (range fun x ↦ x ^ 2) = { y : ℝ | y ≥ 0 } :=
by
  ext y
  -- ⊢ (y ∈ range fun x => x ^ 2) ↔ y ∈ {y | y ≥ 0}
  constructor
  · -- ⊢ (y ∈ range fun x => x ^ 2) → y ∈ {y | y ≥ 0}
    rintro ⟨x, rfl⟩
    -- x : ℝ
    -- ⊢ (fun x => x ^ 2) x ∈ {y | y ≥ 0}
    dsimp at *
    -- ⊢ x ^ 2 ≥ 0
    apply pow_two_nonneg
  . -- ⊢ y ∈ {y | y ≥ 0} → y ∈ range fun x => x ^ 2
```

```
    intro hy
    -- hy : y ∈ {y | y ≥ 0}
    -- ⊢ y ∈ range fun x => x ^ 2
    simp
    -- ⊢ ∃ y_1, y_1 ^ 2 = y
    use sqrt y
    -- ⊢ √y ^ 2 = y
    exact sq_sqrt hy

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

variable (x : ℝ)
#check (pow_nonneg : 0 ≤ x → ∀ n, 0 ≤ x ^ n)
#check (pow_two_nonneg x : 0 ≤ x ^ 2)
#check (sq_sqrt : 0 ≤ x → √x ^ 2 = x)
#check (sqrt_nonneg x : 0 ≤ √x)
#check (sqrt_sq : 0 ≤ x → √(x ^ 2) = x)
```

4.2.12. Valor por defecto y elección de valores

```
-- ---------------------------------------------------------------------
-- Ejercicio. Declarar α como una variables de tipos habitados.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Lattice

variable {α : Type u} [Inhabited α]

-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--     default α
-- ----------------------------------------------------------------------

#check (default : α)

-- ---------------------------------------------------------------------
-- Ejercicio. Declarar P como un predicado sobre α tal que existe algún
-- elemento que verifica P.
-- ----------------------------------------------------------------------

variable (P : α → Prop) (h : ∃ x, P x)
```

```
-- ---------------------------------------------------------------------
-- Ejercicio. Calcular el tipo de
--    classical.some h
-- ----------------------------------------------------------------------

#check (Classical.choose h : α)

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    P (classical.some h)
-- ----------------------------------------------------------------------

example : P (Classical.choose h) :=
  Classical.choose_spec h

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

#check (Classical.choose_spec : (∃ x, P x) → P (Classical.choose h))
```

4.2.13. Función inversa

```
-- ---------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la teoría data.set.function
-- 2. Declarar u y v como universos.
-- 3. Declarar α como variable sobre tipo de u habitados.
-- 4. Declarar β como variable sobre tipo de v.
-- 5. Declarar la teoría como no computable.
-- 6. Usar la lógica clásica.
-- ----------------------------------------------------------------------

import Mathlib.Data.Set.Lattice
import Mathlib.Data.Set.Function

universe u v                               -- 2
variable {α : Type u} [Inhabited α]    -- 3
variable {β : Type v}                     -- 4
noncomputable section                      -- 5
open Classical

-- ---------------------------------------------------------------------
-- Ejercicio. Definir la inversa de una función
```

```
-- -----------------------------------------------------------------------

def inverse (f : α → β) : β → α := fun y : β ↦
  if h : ∃ x, f x = y
  then Classical.choose h
  else default

-- ----------------------------------------------------------------------
-- Ejercicio. Sea d una función de α en β e y un elemento de
-- β. Demostrar que si
--     ∃ x, f x = y
-- entonces
--     f (inverse f y) = y
-- -----------------------------------------------------------------------

theorem inverse_spec
  {f : α → β}
  (y : β)
  (h : ∃ x, f x = y)
  : f (inverse f y) = y :=
by
  rw [inverse, dif_pos h]
  -- ⊢ f (choose h) = y
  exact Classical.choose_spec h

-- Comentarios:
-- 1. La identidad (dif_pos h), cuando (h : e), reescribe la expresión
--     (if h : e then x else y) a x.

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

variable (P : α -> Prop)
variable (c : Prop)
variable {t : c → α}
variable {e : ¬c → α}
variable (hc : c)
variable (h : ∃ x, P x)
#check (Classical.choose : (∃ x, P x) → α)
#check (Classical.choose_spec : (∃ x, P x) → P (Classical.choose h))
#check (dif_pos hc : dite c t e = t hc)
#check (inverse : (α → β) → β → α)
```

4.2.14. Caracterización de las funciones inyectivas mediante la inversa por la izquierda

```
import src.Conjuntos.Funcion_inversa
import Mathlib.Data.Set.Function

universe u v
variable {α : Type u} [Inhabited α]
variable {β : Type v}
variable (f : α → β)
variable (g : β → α)
variable (x : α)

open Set Function

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que g es la inversa por la izquierda de f syss
--    ∀ x, g (f x) = x
-- ----------------------------------------------------------------------

example : LeftInverse g f ↔ ∀ x, g (f x) = x :=
by rw [LeftInverse]

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que las siguientes condiciones son equivalentes:
-- 1. f es inyectiva
-- 2. left_inverse (inverse f) f
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : Injective f ↔ LeftInverse (inverse f) f := by
  constructor
  · -- ⊢ Injective f → LeftInverse (inverse f) f
    intro h y
    -- h : Injective f
    -- y : α
    -- ⊢ inverse f (f y) = y
    apply h
    -- ⊢ f (inverse f (f y)) = f y
    apply inverse_spec
    -- ⊢ ∃ x, f x = f y
    use y
```

```
  · -- ⊢ LeftInverse (inverse f) f → Injective f
    intro h x1 x2 e
    -- x1 x2 : α
    -- e : f x1 = f x2
    -- ⊢ x1 = x2
    rw [← h x1, ← h x2, e]

-- 2ª demostración
-- ===============

example : Injective f ↔ LeftInverse (inverse f) f :=
  ⟨fun h y ↦ h (inverse_spec _ ⟨y, rfl⟩), fun h x1 x2 e ↦ by rw [← h x1, ← h x2, e]⟩

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

variable (y : β)
#check (LeftInverse : (β → α) → (α → β) → Prop)
#check (inverse_spec y : (∃ x, f x = y) → f (inverse f y) = y)
```

4.2.15. Caracterización de las funciones suprayectivas mediante la inversa por la derecha

```
import src.Conjuntos.Funcion_inversa
import Mathlib.Data.Set.Function

universe u v
variable {α : Type u} [Inhabited α]
variable {β : Type v}
variable (f : α → β)
variable (g : β → α)
variable (x : α)

open Set Function

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que g es la inversa por la derecha de f syss
--    ∀ x, f (g x) = x
-- ----------------------------------------------------------------------

example : RightInverse g f ↔ ∀ x, f (g x) = x :=
by
```

```
  rw [RightInverse]
  -- ⊢ LeftInverse f g ↔ ∀ (x : β), f (g x) = x
  rw [LeftInverse]

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que las siguientes condiciones son equivalentes:
-- 1. f es suprayectiva
-- 2. right_inverse (inverse f) f
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : Surjective f ↔ RightInverse (inverse f) f := by
  constructor
  · -- ⊢ Surjective f → RightInverse (inverse f) f
    intro h y
    -- h : Surjective f
    -- y : β
    -- ⊢ f (inverse f y) = y
    apply inverse_spec
    -- ⊢ ∃ x, f x = y
    apply h
  . -- ⊢ RightInverse (inverse f) f → Surjective f
    intro h y
    -- h : RightInverse (inverse f) f
    -- y : β
    -- ⊢ ∃ a, f a = y
    use inverse f y
    -- ⊢ f (inverse f y) = y
    apply h

-- 2ª demostración
-- ===============

example : Surjective f ↔ RightInverse (inverse f) f :=
  ⟨fun h _ ↦ inverse_spec _ (h _), fun h y ↦ ⟨inverse f y, h _⟩⟩

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que f es suprayectiva syss tiene una inversa por
-- la izquierda.
-- ----------------------------------------------------------------------

example : Surjective f ↔ ∃ g, RightInverse g f :=
by
```

```
  constructor
  . -- ⊢ Surjective f → ∃ g, RightInverse g f
    intro h
    -- h : Surjective f
    -- ⊢ ∃ g, RightInverse g f
    dsimp [Surjective] at h
    -- ⊢ ∃ g, RightInverse g f
    choose g hg using h
    -- g : β → α
    -- hg : ∀ (b : β), f (g b) = b
    use g
    -- ⊢ RightInverse g f
    exact hg
  . -- ⊢ (∃ g, RightInverse g f) → Surjective f
    rintro ⟨g, hg⟩
    -- g : β → α
    -- hg : RightInverse g f
    -- ⊢ Surjective f
    intros b
    -- b : β
    -- ⊢ ∃ a, f a = b
    use g b
    -- ⊢ f (g b) = b
    exact hg b

-- Comentarios:
-- 1. La táctica (dsimp [e] at h) simplifica la hipótesis h con la
--    definición de e.
-- 2. La táctica (choose g hg using h) si h es de la forma
--    (∀ (b : β), ∃ (a : α), f a = b) quita la hipótesis h y añade las
--     hipótesis (g : β → α) y (hg : ∀ (b : β), f (g b) = b).

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

variable (y : β)
#check (LeftInverse : (β → α) → (α → β) → Prop)
#check (RightInverse : (β → α) → (α → β) → Prop)
#check (Surjective : (α → β) → Prop)
#check (inverse_spec y : (∃ x, f x = y) → f (inverse f y) = y)
```

4.2.16. Teorema de Cantor

```
-- ---------------------------------------------------------------------
-- Demostrar el teorema de Cantor:
--    ∀ f : α → Set α, ¬Surjective f
-- ----------------------------------------------------------------------

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

-- Sea f una función de α en el conjunto de los subconjuntos de
-- α. Tenemos que demostrar que f no es suprayectiva. Lo haresmos por
-- reducción al absurdo. Para ello, supongamos que f es suprayectiva y
-- consideremos el conjunto
--    S := {i ∈ α | i ∉ f(i)}                                       (1)
-- Entonces, tiene que existir un j ∈ α tal que
--   f(j) = S                                                       (2)
-- Se pueden dar dos casos: j ∈ S ó j ∉ S. Veamos que ambos son
-- imposibles.
--
-- Caso 1: Supongamos que j ∈ S. Entonces, por (1)
--    j ∉ f(j)
-- y, por (2),
--    j ∉ S
-- que es una contradicción.
--
-- Caso 2: Supongamos que j ∉ S. Entonces, por (1)
--    j ∈ f(j)
-- y, por (2),
--    j ∈ S
-- que es una contradicción.

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

import Mathlib.Data.Set.Basic

open Function

variable {α : Type}

-- 1ª demostración
-- ===============

example : ∀ f : α → Set α, ¬Surjective f :=
```

```
by
  intros f hf
  -- f : α → Set α
  -- hf : Surjective f
  -- ⊢ False
  let S := {i | i ∉ f i}
  unfold Surjective at hf
  -- hf : ∀ (b : Set α), ∃ a, f a = b
  rcases hf S with ⟨j, hj⟩
  -- j : α
  -- hj : f j = S
  by_cases h: j ∈ S
  . -- h : j ∈ S
    simp at h
    -- h : ¬j ∈ f j
    apply h
    -- ⊢ j ∈ f j
    rw [hj]
    -- ⊢ j ∈ S
    exact h
  . -- h : ¬j ∈ S
    apply h
    -- ⊢ j ∈ S
    rw [←hj] at h
    -- h : ¬j ∈ f j
    exact h

-- 2ª demostración
-- ===============

example : ∀ f : α → Set α, ¬ Surjective f :=
by
  intros f hf
  -- f : α → Set α
  -- hf : Surjective f
  -- ⊢ False
  let S := {i | i ∉ f i}
  rcases hf S with ⟨j, hj⟩
  -- j : α
  -- hj : f j = S
  by_cases h: j ∈ S
  . -- h : j ∈ S
    apply h
    -- ⊢ j ∈ f j
    rwa [hj]
```

```
  . -- h : ¬j ∈ S
    apply h
    rwa [←hj] at h

-- 3ª demostración
-- ===============

example : ∀ f : α → Set α, ¬ Surjective f :=
by
  intros f hf
  -- f : α → Set α
  -- hf : Surjective f
  -- ⊢ False
  let S := {i | i ∉ f i}
  rcases hf S with ⟨j, hj⟩
  -- j : α
  -- hj : f j = S
  have h : (j ∈ S) = (j ∉ S) :=
    calc  (j ∈ S)
       = (j ∉ f j) := Set.mem_setOf_eq
     _ = (j ∉ S)   := congrArg (j ∉ .) hj
  exact iff_not_self (iff_of_eq h)

-- 4ª demostración
-- ===============

example : ∀ f : α → Set α, ¬ Surjective f :=
cantor_surjective

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

variable (x y : α)
variable (p : α → Prop)
variable (a b : Prop)
variable (f : α → α)
#check (Set.mem_setOf_eq : (x ∈ {y : α | p y}) = p x)
#check (cantor_surjective : ∀ f : α → Set α, ¬ Surjective f)
#check (congrArg f : x = y → f x = f y)
#check (iff_not_self : ¬(a ↔ ¬a))
#check (iff_of_eq : a = b → (a ↔ b))
```

Capítulo 5

Bibliografía

- How to prove it with Lean. ~ Daniel Velleman .
- Lean 4 manual.
- Lean 4 cheatsheet. ~ Martin Dvořák.
- Mathematics in Lean. ~ Jeremy Avigad y Patrick Massot.
- The mechanics of proof. ~ Heather Macbeth.
- The natural number game. ~ Kevin Buzzard, Jon Eugster, and Mohammad Pedramfar.
- Theorem proving in Lean 4. ~ Jeremy Avigad, Leonardo de Moura, Soonho Kong y Sebastian Ullrich.
- Undergraduate mathematics in mathlib.