Calculemus
(Vol. 2: Demostraciones con Lean4)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 10 de julio de 2023 (versión del 16 de junio de 2024)

Índice general

Capítulo 1

Introducción

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

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

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

Las soluciones del libro están en este repositorio de GitHub.

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

Este libro es una continuación de

- DAO (Demostración Asistida por Ordenador) con Lean que es una introducción a la demostración con Lean3 y
- Calculemus (Vol. 1: Demostraciones con Isabelle/HOL y Lean3) que es la recopilación de la primera parte de los ejercicios del blog con demostraciones en Isabelle/HOL y Lean3.

Capítulo 2

Demostraciones de una propiedad de los números enteros

2.1. ∀ m n ∈ ℕ, Even n → Even (m * n)

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

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

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

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

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

open Nat
```

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

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

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

-- 6ª demostración
-- ===============

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

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

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

-- 8ª demostración
-- ===============

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

-- 9ª demostración
-- ===============

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

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

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

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

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

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

Capítulo 3

Propiedades elementales de los números reales

3.1. En ℝ, (ab)c = b(ac)

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

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

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

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

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

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

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

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

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

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

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

3.2. En ℝ, (cb)a = b(ac)

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

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

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

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

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

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

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

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

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

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

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

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

3.3. En ℝ, a(bc) = b(ac)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-- Demostraciones con Lean4
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.7. En ℝ, (a+b)(a+b) = aa+2ab+bb

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

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

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

-- Demostraciones con Lean4
```

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

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

variable (a b c : ℝ)

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

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

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

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

-- 5ª demostración
example :
```

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

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

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

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

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

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

3.8. En ℝ, (a+b)(c+d) = ac+ad+bc+bd

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

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

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

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

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

variable (a b c d : ℝ)

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

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

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

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

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

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

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

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

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

3.9. En ℝ, (a+b)(a-b) = a^2-b^2

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

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

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

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

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

variable (a b : ℝ)

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

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

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

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

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

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

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

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

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

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

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

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

variable (a b c d : ℝ)

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

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

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

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

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

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

-- 6ª demostración
example
```

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

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

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

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

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

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

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

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

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

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

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

variable (a b c : ℝ)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
  ring

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

3.13. En $\mathbb{R}$, $x^2 + y^2 = 0 \leftrightarrow x = 0 \land y = 0$

```
-- ---------------------------------------------------------------------
-- Demostrar que si x, y ∈ ℝ, entonces
--    x^2 + y^2 = 0 ↔ x = 0 ∧ y = 0
-- ----------------------------------------------------------------------

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

-- En la demostración usaremos el siguiente lema auxiliar
--    (∀ x, y ∈ ℝ)[x² + y² = 0 → x = 0]
--
-- Para la primera implicación, supongamos que
--    x² + y² = 0                                                  (1)
-- Entonces, por el lema auxiliar,
--    x = 0                                                        (2)
-- Además, aplicando la conmutativa a (1), se tiene
--    y² + x² = 0
-- y, por el lema auxiliar,
--    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
--
-- En la demostración del lema auxiliar 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, para demostrar el lema auxiliar 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.

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

import Mathlib.Data.Real.Basic
variable {x y : ℝ}

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

example
  (h : x^2 + y^2 = 0)
  : x = 0 :=
by
  have h’ : x^2 = 0 := by
  { apply le_antisymm
    . show x ^ 2 ≤ 0
      calc x ^ 2 ≤ x^2 + y^2 := by simp [le_add_of_nonneg_right,
                                         pow_two_nonneg]
               _ = 0         := by exact h
    . show 0 ≤ x ^ 2
      apply pow_two_nonneg }
```

```
  show x = 0
  exact pow_eq_zero h'

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

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 }
  exact pow_eq_zero h'

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

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'

-- 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

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

-- #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 : ∀ {n : ℕ}, x ^ n = 0 → x = 0)
-- #check (pow_two_nonneg x : 0 ≤ x ^ 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

3.14. En ℝ, $x^2 = 1 \to x = 1 \lor x = -1$

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

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

-- 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

-- 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 (sub_eq_zero : x - y = 0 ↔ x = y)
-- #check (sub_self x : x - x = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

3.15. En ℝ, x² = y² → x = y ∨ x = -y

```
-- ---------------------------------------------------------------------
-- Demostrar que 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

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

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

-- 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_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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

3.16. En ℝ, |a| = |a - b + b|

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    |a| = |a - b + b|
-- ----------------------------------------------------------------------

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

-- 1ª demostración
-- ===============

example
  : |a| = |a - b + b| :=
by
  congr
  -- a = a - b + b
  ring

-- Comentario: La táctica congr 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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

Capítulo 4

Propiedades elementales de los monoides

4.1. En los monoides, los inversos a la izquierda y a la derecha son iguales

#+INCLUDE ”../src/En_los_monoides_los_inversos_a_la_izquierda_y_a_la_derecha_so lean :lines ”7-”Se puede interactuar con las pruebas anteriores en Lean 4 Web.

4.2. Producto de potencias de la misma base en monoides

```
-- ---------------------------------------------------------------------
-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponentes naturales. En Lean la potencia x^n se
-- se caracteriza por los siguientes lemas:
--    pow_zero : x^0 = 1
--    pow_succ : x^(succ n) = x * x^n
--
-- Demostrar que
--    x^(m + n) = x^m * x^n
-- ---------------------------------------------------------------------

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

-- Por inducción en m.
```

```
--
-- Caso base:
--    x^(0 + n) = x^n
--              = 1 * x^n
--              = x^0 * x^n            [por pow_zero]
--
-- Paso: Supongamos que
--    x^(m + n) = x^m * x^n                                        (HI)
-- Entonces
--    x^((m+1) + n) = x^((m + n) + 1)
--                  = x * x^(m + n)    [por pow_succ]
--                  = x * (x^m * x^n)  [por HI]
--                  = (x * x^m) * x^n
--                  = x^(m+1) * x^n    [por pow_succ]

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

import Mathlib.Algebra.Group.Defs
import Mathlib.Algebra.GroupPower.Basic
open Nat

variable {M : Type} [Monoid M]
variable (x : M)
variable (m n : ℕ)

-- 1ª demostración
-- ===============

example :
  x^(m + n) = x^m * x^n :=
by
  induction' m with m HI
  . calc x^(0 + n)
       = x^n               := congrArg (x ^ .) (Nat.zero_add n)
     _ = 1 * x^n           := (Monoid.one_mul (x^n)).symm
     _ = x^0 * x^n         := congrArg (. * (x^n)) (pow_zero x).symm
  . calc x^(succ m + n)
       = x^succ (m + n)    := congrArg (x ^.) (succ_add m n)
     _ = x * x^(m + n)     := pow_succ x (m + n)
     _ = x * (x^m * x^n)   := congrArg (x * .) HI
     _ = (x * x^m) * x^n   := (mul_assoc x (x^m) (x^n)).symm
     _ = x^succ m * x^n    := congrArg (. * x^n) (pow_succ x m).symm

-- 2ª demostración
```

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

example :
  x^(m + n) = x^m * x^n :=
by
  induction' m with m HI
  . calc x^(0 + n)
         = x^n              := by simp only [Nat.zero_add]
       _ = 1 * x^n          := by simp only [Monoid.one_mul]
       _ = x^0 * x^n        := by simp only [_root_.pow_zero]
  . calc x^(succ m + n)
         = x^succ (m + n)   := by simp only [succ_add]
       _ = x * x^(m + n)    := by simp only [_root_.pow_succ]
       _ = x * (x^m * x^n) := by simp only [HI]
       _ = (x * x^m) * x^n := (mul_assoc x (x^m) (x^n)).symm
       _ = x^succ m * x^n   := by simp only [_root_.pow_succ]

-- 3ª demostración
-- ===============

example :
  x^(m + n) = x^m * x^n :=
by
  induction' m with m HI
  . calc x^(0 + n)
         = x^n              := by simp [Nat.zero_add]
       _ = 1 * x^n          := by simp
       _ = x^0 * x^n        := by simp
  . calc x^(succ m + n)
         = x^succ (m + n)   := by simp [succ_add]
       _ = x * x^(m + n)    := by simp [_root_.pow_succ]
       _ = x * (x^m * x^n) := by simp [HI]
       _ = (x * x^m) * x^n := (mul_assoc x (x^m) (x^n)).symm
       _ = x^succ m * x^n   := by simp [_root_.pow_succ]

-- 4ª demostración
-- ===============

example :
  x^(m + n) = x^m * x^n :=
pow_add x m n

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

```
-- variable (y z : M)
-- #check (Monoid.one_mul x : 1 * x = x)
-- #check (Nat.zero_add n : 0 + n = n)
-- #check (mul_assoc x y z : (x * y) * z = x * (y * z))
-- #check (pow_add x m n : x^(m + n) = x^m * x^n)
-- #check (pow_succ x n : x ^ succ n = x * x ^ n)
-- #check (pow_zero x : x ^ 0 = 1)
-- #check (succ_add n m : succ n + m = succ (n + m))
```

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

4.3. Equivalencia de inversos iguales al neutro

```
-- ---------------------------------------------------------------------
-- Sea M un monoide y a, b ∈ M tales que a * b = 1. Demostrar que a = 1
-- si y sólo si b = 1.
-- ---------------------------------------------------------------------

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

-- Demostraremos las dos implicaciones.
--
-- (⟹) Supongamos que a = 1. Entonces,
--    b = 1·b    [por neutro por la izquierda]
--      = a·b    [por supuesto]
--      = 1      [por hipótesis]
--
-- (⟸) Supongamos que b = 1. Entonces,
--    a = a·1    [por neutro por la derecha]
--      = a·b    [por supuesto]
--      = 1      [por hipótesis]

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

import Mathlib.Algebra.Group.Basic

variable {M : Type} [Monoid M]
variable {a b : M}

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

```
example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
by
  constructor
  . -- ⊢ a = 1 → b = 1
    intro a1
    -- a1 : a = 1
    -- ⊢ b = 1
    calc b = 1 * b := (one_mul b).symm
         _ = a * b := congrArg (. * b) a1.symm
         _ = 1     := h
  . -- ⊢ b = 1 → a = 1
    intro b1
    -- b1 : b = 1
    -- ⊢ a = 1
    calc a = a * 1 := (mul_one a).symm
         _ = a * b := congrArg (a * .) b1.symm
         _ = 1     := h

-- 2ª demostración
-- ===============

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
by
  constructor
  . -- ⊢ a = 1 → b = 1
    intro a1
    -- a1 : a = 1
    -- ⊢ b = 1
    rw [a1] at h
    -- h : 1 * b = 1
    rw [one_mul] at h
    -- h : b = 1
    exact h
  . -- ⊢ b = 1 → a = 1
    intro b1
    -- b1 : b = 1
    -- ⊢ a = 1
    rw [b1] at h
    -- h : a * 1 = 1
    rw [mul_one] at h
```

```
    -- h : a = 1
    exact h

-- 3ª demostración
-- ===============

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
by
  constructor
  . -- ⊢ a = 1 → b = 1
    rintro rfl
    -- h : 1 * b = 1
    simpa using h
  . -- ⊢ b = 1 → a = 1
    rintro rfl
    -- h : a * 1 = 1
    simpa using h

-- 4ª demostración
-- ===============

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
by constructor <;> (rintro rfl; simpa using h)

-- 5ª demostración
-- ===============

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
eq_one_iff_eq_one_of_mul_eq_one h

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

-- #check (eq_one_iff_eq_one_of_mul_eq_one : a * b = 1 → (a = 1 ↔ b = 1))
-- #check (mul_one a : a * 1 = a)
-- #check (one_mul a : 1 * a = a)
```

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

4.4. Unicidad de inversos en monoides

```
-- ---------------------------------------------------------------------
-- Demostrar que en los monoides conmutativos, si un elemento tiene un
-- inverso por la derecha, dicho inverso es único.
-- ---------------------------------------------------------------------

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

-- Por la siguiente cadena de igualdades
--    y = 1·y          [por neutro a la izquierda]
--      = (x·z)·y      [por hipótesis]
--      = (z·x)·y      [por la conmutativa]
--      = z·(x·y)      [por la asociativa]
--      = z·1          [por hipótesis]
--      = z            [por neutro a la derecha]

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

import Mathlib.Algebra.Group.Basic

variable {M : Type} [CommMonoid M]
variable {x y z : M}

-- 1ª demostración
-- ===============

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y       := (one_mul y).symm
     _ = (x * z) * y := congrArg (. * y) hz.symm
     _ = (z * x) * y := congrArg (. * y) (mul_comm x z)
     _ = z * (x * y) := mul_assoc z x y
     _ = z * 1       := congrArg (z * .) hy
     _ = z           := mul_one z

-- 2ª demostración
-- ===============

example
```

```
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y      := by simp only [one_mul]
   _ = (x * z) * y := by simp only [hz]
   _ = (z * x) * y := by simp only [mul_comm]
   _ = z * (x * y) := by simp only [mul_assoc]
   _ = z * 1       := by simp only [hy]
   _ = z           := by simp only [mul_one]

-- 3ª demostración
-- ===============

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y      := by simp
   _ = (x * z) * y := by simp [hz]
   _ = (z * x) * y := by simp [mul_comm]
   _ = z * (x * y) := by simp [mul_assoc]
   _ = z * 1       := by simp [hy]
   _ = z           := by simp

-- 4ª demostración
-- ===============

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
by
  apply left_inv_eq_right_inv _ hz
  -- ⊢ y * x = 1
  rw [mul_comm]
  -- ⊢ x * y = 1
  exact hy

-- 5ª demostración
-- ===============

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
```

```
inv_unique hy hz

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

-- #check (inv_unique : x * y = 1 → x * z = 1 → y = z)
-- #check (left_inv_eq_right_inv : y * x = 1 → x * z = 1 → y = z)
-- #check (mul_assoc x y z : (x * y) * z = x * (y * z))
-- #check (mul_comm x y : x * y = y * x)
-- #check (mul_one x : x * 1 = x)
-- #check (one_mul x : 1 * x = x)
```

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

4.5. Caracterización de producto igual al primer factor

```
-- ---------------------------------------------------------------------
-- Un monoide cancelativo por la izquierda es un monoide M que cumple la
-- propiedad cancelativa por la izquierda; es decir, para todo a, b ∈ M
--    a * b = a * c ↔ b = c.
--
-- En Lean4 la clase de los monoides cancelativos por la izquierda es
-- LeftCancelMonoid y la propiedad cancelativa por la izquierda es
--    mul_left_cancel : a * b = a * c → b = c
--
-- Demostrar que si M es un monoide cancelativo por la izquierda y
-- a, b ∈ M, entonces
--    a * b = a ↔ b = 1
-- ---------------------------------------------------------------------

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

-- Demostraremos las dos implicaciones.
--
-- (⟹) Supongamos que
--    a·b = a                                                       (1)
-- Por la propiedad del neutro por la derecha tenemos
--    a·1 = a                                                       (2)
-- Por tanto,
--    a·b = a·1
```

```
-- y, por la propiedad cancelativa,
--    b = 1
--
-- (⟸) Supongamos que b = 1. Entonces,
--    a·b = a·1
--        = a      [por el neutro por la derecha]

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

import Mathlib.Algebra.Group.Basic

variable {M : Type} [LeftCancelMonoid M]
variable {a b : M}

-- 1ª demostración
-- ===============

example : a * b = a ↔ b = 1 :=
by
  constructor
  . -- ⊢ a * b = a → b = 1
    intro h
    -- h : a * b = a
    -- ⊢ b = 1
    have h1 : a * b = a * 1 := by
      calc a * b = a     := by exact h
               _ = a * 1 := (mul_one a).symm
    exact mul_left_cancel h1
  . -- ⊢ b = 1 → a * b = a
    intro h
    -- h : b = 1
    -- ⊢ a * b = a
    rw [h]
    -- ⊢ a * 1 = a
    exact mul_one a

-- 2ª demostración
-- ===============

example : a * b = a ↔ b = 1 :=
calc a * b = a
     ↔ a * b = a * 1 := by rw [mul_one]
   _ ↔ b = 1         := mul_left_cancel_iff
```

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

example : a * b = a ↔ b = 1 :=
mul_right_eq_self

-- 4ª demostración
-- ===============

example : a * b = a ↔ b = 1 :=
by aesop

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

-- variable (c : M)
-- #check (mul_left_cancel : a * b = a * c → b = c)
-- #check (mul_one a : a * 1 = a)
-- #check (mul_right_eq_self : a * b = a ↔ b = 1)
```

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

4.6. Si M es un monoide, a ∈ M y m, n ∈ ℕ, entonces $a^{m \cdot n} = (a^m)^n$

```
-- ---------------------------------------------------------------------
-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponentes naturales. En Lean4, la potencia x^n se
-- se caracteriza por los siguientes lemas:
--    pow_zero : x^0 = 1
--    pow_succ' : x^(succ n) = x * x^n
--
-- Demostrar que si M es un monoide, a ∈ M y m, n ∈ ℕ, entonces
--    a^(m·n) = (a^m)^n
-- ---------------------------------------------------------------------

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

-- Por inducción en n.
--
-- Caso base: Supongamos que n = 0. Entonces,
```

```
--    a^(m·0) = a^0
--            = 1         [por pow_zero]
--            = (a^m)^0   [por pow_zero]
--
-- Paso de indución: Supogamos que se verifica para n; es decir,
--    a^(m·n) = (a^m)^n
-- Entonces,
--    a^(m·(n+1)) = a^(m·n + m)
--                = a^(m·n)·a^m
--                = (a^m)^n·a^m    [por HI]
--                = (a^m)^(n+1)    [por pow_succ']

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

import Mathlib.Algebra.GroupPower.Basic
open Nat

variable {M : Type} [Monoid M]
variable (a : M)
variable (m n : ℕ)

-- 1ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . calc a^(m * 0)
         = a^0             := congrArg (a ^ .) (Nat.mul_zero m)
       _ = 1               := pow_zero a
       _ = (a^m)^0         := (pow_zero (a^m)).symm
  . calc a^(m * succ n)
         = a^(m * n + m)   := congrArg (a ^ .) (Nat.mul_succ m n)
       _ = a^(m * n) * a^m := pow_add a (m * n) m
       _ = (a^m)^n * a^m   := congrArg (. * a^m) HI
       _ = (a^m)^(succ n)  := (pow_succ' (a^m) n).symm

-- 2ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . calc a^(m * 0)
```

```
        = a^0              := by simp only [Nat.mul_zero]
      _ = 1                := by simp only [_root_.pow_zero]
      _ = (a^m)^0          := by simp only [_root_.pow_zero]
  . calc a^(m * succ n)
        = a^(m * n + m)    := by simp only [Nat.mul_succ]
      _ = a^(m * n) * a^m := by simp only [pow_add]
      _ = (a^m)^n * a^m    := by simp only [HI]
      _ = (a^m)^succ n     := by simp only [_root_.pow_succ']

-- 3ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . calc a^(m * 0)
        = a^0              := by simp [Nat.mul_zero]
      _ = 1                := by simp
      _ = (a^m)^0          := by simp
  . calc a^(m * succ n)
        = a^(m * n + m)    := by simp [Nat.mul_succ]
      _ = a^(m * n) * a^m := by simp [pow_add]
      _ = (a^m)^n * a^m    := by simp [HI]
      _ = (a^m)^succ n     := by simp [_root_.pow_succ']

-- 4ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . simp [Nat.mul_zero]
  . simp [Nat.mul_succ,
          pow_add,
          HI,
          _root_.pow_succ']

-- 5ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . -- ⊢ a ^ (m * zero) = (a ^ m) ^ zero
    rw [Nat.mul_zero]
```

```
    -- ⊢ a ^ 0 = (a ^ m) ^ zero
    rw [_root_.pow_zero]
    -- ⊢ 1 = (a ^ m) ^ zero
    rw [_root_.pow_zero]
  . -- ⊢ a ^ (m * succ n) = (a ^ m) ^ succ n
    rw [Nat.mul_succ]
    -- ⊢ a ^ (m * n + m) = (a ^ m) ^ succ n
    rw [pow_add]
    -- ⊢ a ^ (m * n) * a ^ m = (a ^ m) ^ succ n
    rw [HI]
    -- ⊢ (a ^ m) ^ n * a ^ m = (a ^ m) ^ succ n
    rw [_root_.pow_succ']

-- 6ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
by
  induction' n with n HI
  . rw [Nat.mul_zero, _root_.pow_zero, _root_.pow_zero]
  . rw [Nat.mul_succ, pow_add, HI, _root_.pow_succ']

-- 7ª demostración
-- ===============

example : a^(m * n) = (a^m)^n :=
pow_mul a m n

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

-- #check (Nat.mul_succ n m : n * succ m = n * m + n)
-- #check (Nat.mul_zero m : m * 0 = 0)
-- #check (pow_add a m n : a ^ (m + n) = a ^ m * a ^ n)
-- #check (pow_mul a m n : a ^ (m * n) = (a ^ m) ^ n)
-- #check (pow_succ' a n : a ^ (n + 1) = a ^ n * a)
-- #check (pow_zero a : a ^ 0 = 1)
```

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

4.7. Los monoides booleanos son conmutativos

```
-- ---------------------------------------------------------------------
-- Un monoide es un conjunto junto con una operación binaria que es
-- asociativa y tiene elemento neutro.
--
-- Un monoide M es booleano si
--    ∀ x ∈ M, x * x = 1
-- y es conmutativo si
--    ∀ x y ∈ M, x * y = y * x
--
-- En Lean4, está definida la clase de los monoides (como ‘Monoid‘) y sus
-- propiedades características son
--    mul_assoc : (a * b) * c = a * (b * c)
--    one_mul :   1 * a = a
--    mul_one :   a * 1 = a
--
-- Demostrar que los monoides booleanos son conmutativos.
-- ---------------------------------------------------------------------

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

-- Sean a, b ∈ M. Se verifica la siguiente cadena de igualdades
--    a·b = (a·b)·1               [por mul_one]
--        = (a·b)·(a·a)           [por hipótesis, a·a = 1]
--        = ((a·b)·a)·a           [por mul_assoc]
--        = (a·(b·a))·a           [por mul_assoc]
--        = (1·(a·(b·a)))·a       [por one_mul]
--        = ((b·b)·(a·(b·a)))·a   [por hipótesis, b·b = 1]
--        = (b·(b·(a·(b·a))))·a   [por mul_assoc]
--        = (b·((b·a)·(b·a)))·a   [por mul_assoc]
--        = (b·1)·a               [por hipótesis, (b·a)·(b·a) = 1]
--        = b·a                   [por mul_one]

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

import Mathlib.Algebra.Group.Basic

variable {M : Type} [Monoid M]

-- 1ª demostración
```

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

example
  (h : ∀ x : M, x * x = 1)
  : ∀ x y : M, x * y = y * x :=
by
  intros a b
  calc a * b
       = (a * b) * 1
         := (mul_one (a * b)).symm
     _ = (a * b) * (a * a)
         := congrArg ((a*b) * .) (h a).symm
     _ = ((a * b) * a) * a
         := (mul_assoc (a*b) a a).symm
     _ = (a * (b * a)) * a
         := congrArg (. * a) (mul_assoc a b a)
     _ = (1 * (a * (b * a))) * a
         := congrArg (. * a) (one_mul (a*(b*a))).symm
     _ = ((b * b) * (a * (b * a))) * a
         := congrArg (. * a) (congrArg (. * (a*(b*a))) (h b).symm)
     _ = (b * (b * (a * (b * a)))) * a
         := congrArg (. * a) (mul_assoc b b (a*(b*a)))
     _ = (b * ((b * a) * (b * a))) * a
         := congrArg (. * a) (congrArg (b * .) (mul_assoc b a (b*a)).symm)
     _ = (b * 1) * a
         := congrArg (. * a) (congrArg (b * .) (h (b*a)))
     _ = b * a
         := congrArg (. * a) (mul_one b)

-- 2ª demostración
-- ===============

example
  (h : ∀ x : M, x * x = 1)
  : ∀ x y : M, x * y = y * x :=
by
  intros a b
  calc a * b
       = (a * b) * 1                 := by simp only [mul_one]
     _ = (a * b) * (a * a)           := by simp only [h a]
     _ = ((a * b) * a) * a           := by simp only [mul_assoc]
     _ = (a * (b * a)) * a           := by simp only [mul_assoc]
     _ = (1 * (a * (b * a))) * a     := by simp only [one_mul]
     _ = ((b * b) * (a * (b * a))) * a := by simp only [h b]
     _ = (b * (b * (a * (b * a)))) * a := by simp only [mul_assoc]
```

```
    _ = (b * ((b * a) * (b * a))) * a := by simp only [mul_assoc]
    _ = (b * 1) * a                   := by simp only [h (b*a)]
    _ = b * a                         := by simp only [mul_one]

-- 3ª demostración
-- ===============

example
  (h : ∀ x : M, x * x = 1)
  : ∀ x y : M, x * y = y * x :=
by
  intros a b
  calc a * b
      = (a * b) * 1                   := by simp only [mul_one]
    _ = (a * b) * (a * a)             := by simp only [h a]
    _ = (a * (b * a)) * a             := by simp only [mul_assoc]
    _ = (1 * (a * (b * a))) * a       := by simp only [one_mul]
    _ = ((b * b) * (a * (b * a))) * a := by simp only [h b]
    _ = (b * ((b * a) * (b * a))) * a := by simp only [mul_assoc]
    _ = (b * 1) * a                   := by simp only [h (b*a)]
    _ = b * a                         := by simp only [mul_one]

-- 4ª demostración
-- ===============

example
  (h : ∀ x : M, x * x = 1)
  : ∀ x y : M, x * y = y * x :=
by
  intros a b
  calc a * b
      = (a * b) * 1                   := by simp
    _ = (a * b) * (a * a)             := by simp only [h a]
    _ = (a * (b * a)) * a             := by simp only [mul_assoc]
    _ = (1 * (a * (b * a))) * a       := by simp
    _ = ((b * b) * (a * (b * a))) * a := by simp only [h b]
    _ = (b * ((b * a) * (b * a))) * a := by simp only [mul_assoc]
    _ = (b * 1) * a                   := by simp only [h (b*a)]
    _ = b * a                         := by simp

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

-- variable (a b c : M)
-- #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)
```

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

Capítulo 5

Propiedades elementales de los grupos

5.1. Unicidad del elemento neutro en los grupos

```
-- ---------------------------------------------------------------------
-- Demostrar que un grupo sólo posee un elemento neutro.
-- ---------------------------------------------------------------------

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

-- Sea e ∈ G tal que
--    (∀ x)[x·e = x]                                                 (1)
-- Entonces,
--    e = 1.e    [porque 1 es neutro]
--      = 1      [por (1)]

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

import Mathlib.Algebra.Group.Basic

variable {G : Type} [Group G]

-- 1ª demostración
-- ===============

example
```

```
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
calc e = 1 * e := (one_mul e).symm
     _ = 1     := h 1

-- 2ª demostración
-- ===============

example
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
by
  have h1 : e = e * e := (h e).symm
  exact self_eq_mul_left.mp h1

-- 3ª demostración
-- ===============

example
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
self_eq_mul_left.mp (h e).symm

-- 4ª demostración
-- ===============

example
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
by aesop

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

-- variable (a b : G)
-- #check (one_mul a : 1 * a = a)
-- #check (self_eq_mul_left : b = a * b ↔ a = 1)
```

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

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

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

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

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

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

import Mathlib.Algebra.Group.Defs

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

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

```
        _ = ((a⁻¹)⁻¹ * 1) * a⁻¹          := by rw [mul_left_inv]
        _ = (a⁻¹)⁻¹ * (1 * a⁻¹)          := by rw [mul_assoc]
        _ = (a⁻¹)⁻¹ * a⁻¹                := by rw [one_mul]
        _ = 1                            := by rw [mul_left_inv]

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

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

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

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

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

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

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

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

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

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

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

import Mathlib.Algebra.Group.Defs

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

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

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

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

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

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

-- variable (c : G)
```

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

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

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

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

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

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

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

import Mathlib.Algebra.Group.Defs

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

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

```
    _ = a⁻¹ * (a * b) := by rw [h]
    _ = (a⁻¹ * a) * b := by rw [mul_assoc]
    _ = 1 * b         := by rw [mul_left_inv]
    _ = b             := by rw [one_mul]

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

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

-- 4º demostración
example
  (h : a * b = 1)
  : a⁻¹ = b :=
by exact inv_eq_of_mul_eq_one_right h

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

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

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

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

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

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

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

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

import Mathlib.Algebra.Group.Defs

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

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

-- 1ª demostración
-- ===============

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

```
  have h1 : (a * b) * (b⁻¹ * a⁻¹) = 1 :=
    aux a b
  show (a * b)⁻¹ = b⁻¹ * a⁻¹
  exact inv_eq_of_mul_eq_one_right h1

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by exact mul_inv_rev a b

-- 6ª demostración
-- ===============

example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by simp

-- Lemas usados
```

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

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

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

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

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es un elemento de un grupo G, entonces a tiene un
-- único inverso; es decir, si b es un elemento de G tal que a * b = 1,
-- entonces a⁻¹ = b.
-- ---------------------------------------------------------------------

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

-- Por la siguiente cadena de igualdades
--    a⁻¹ = a⁻¹·1          [porque 1 es neutro]
--        = a⁻¹·(a·b)      [por hipótesis]
--        = (a⁻¹·a)·b      [por la asociativa]
--        = 1·b            [porque a⁻¹ es el inverso de a]
--        = b              [porque 1 es neutro]

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

import Mathlib.Algebra.Group.Basic

variable {G : Type} [Group G]
variable {a b : G}

-- 1ª demostración
-- ===============

example
```

```
  (h : a * b = 1)
  : a⁻¹ = b :=
calc a⁻¹ = a⁻¹ * 1  := (mul_one a⁻¹).symm
  _ = a⁻¹ * (a * b) := congrArg (a⁻¹ * .) h.symm
  _ = (a⁻¹ * a) * b := (mul_assoc a⁻¹ a b).symm
  _ = 1 * b         := congrArg (. * b) (inv_mul_self a)
  _ = b             := one_mul b

-- 2ª demostración
-- ===============

example
  (h : a * b = 1)
  : a⁻¹ = b :=
calc a⁻¹ = a⁻¹ * 1       := by simp only [mul_one]
       _ = a⁻¹ * (a * b) := by simp only [h]
       _ = (a⁻¹ * a) * b := by simp only [mul_assoc]
       _ = 1 * b         := by simp only [inv_mul_self]
       _ = b             := by simp only [one_mul]

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

example
```

```
  (h : b * a = 1)
  : b = a⁻¹ :=
eq_inv_iff_mul_eq_one.mpr h

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

-- variable (c : G)
-- #check (eq_inv_iff_mul_eq_one : a = b⁻¹ ↔ a * b = 1)
-- #check (inv_mul_self 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 (one_mul a : 1 * a = a)
```

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

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

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

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

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

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

import Mathlib.Algebra.Group.Defs
```

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

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

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

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

-- 5ª demostración
-- ===============

example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by exact mul_inv_rev a b

-- 6ª demostración
-- ===============

example : (a * b)⁻¹ = b⁻¹ * a⁻¹ :=
by simp

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

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

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

5.8. Si G un grupo y a ∈ G, entonces $(a^{-1})^{-1} = a$

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

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

-- Por la siguiente cadena de igualdades
--    (a⁻¹)⁻¹ = (a⁻¹)⁻¹·1          [porque 1 es neutro]
```

```
--             = (a⁻¹)⁻¹·(a⁻¹·a)    [porque a⁻¹ es el inverso de a]
--             = ((a⁻¹)⁻¹·a⁻¹)·a    [por la asociativa]
--             = 1·a                [porque (a⁻¹)⁻¹ es el inverso de a⁻¹]
--             = a                  [porque 1 es neutro]

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

import Mathlib.Algebra.Group.Basic

variable {G : Type} [Group G]
variable {a : G}

-- 1ª demostración
-- ===============

example : (a⁻¹)⁻¹ = a :=
calc (a⁻¹)⁻¹
     = (a⁻¹)⁻¹ * 1          := (mul_one (a⁻¹)⁻¹).symm
   _ = (a⁻¹)⁻¹ * (a⁻¹ * a)  := congrArg ((a⁻¹)⁻¹ * .) (inv_mul_self a).symm
   _ = ((a⁻¹)⁻¹ * a⁻¹) * a  := (mul_assoc _ _ _).symm
   _ = 1 * a                := congrArg (. * a) (inv_mul_self a⁻¹)
   _ = a                    := one_mul a

-- 2ª demostración
-- ===============

example : (a⁻¹)⁻¹ = a :=
calc (a⁻¹)⁻¹
     = (a⁻¹)⁻¹ * 1          := by simp only [mul_one]
   _ = (a⁻¹)⁻¹ * (a⁻¹ * a)  := by simp only [inv_mul_self]
   _ = ((a⁻¹)⁻¹ * a⁻¹) * a  := by simp only [mul_assoc]
   _ = 1 * a                := by simp only [inv_mul_self]
   _ = a                    := by simp only [one_mul]

-- 3ª demostración
-- ===============

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

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

example : (a⁻¹)⁻¹ = a :=
by
  apply mul_eq_one_iff_inv_eq.mp
  -- ⊢ a⁻¹ * a = 1
  exact mul_left_inv a

-- 5ª demostración
-- ===============

example : (a⁻¹)⁻¹ = a :=
mul_eq_one_iff_inv_eq.mp (mul_left_inv a)

-- 6ª demostración
-- ===============

example : (a⁻¹)⁻¹ = a:=
inv_inv a

-- 7ª demostración
-- ===============

example : (a⁻¹)⁻¹ = a:=
by simp

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

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

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

5.9. Si G es un grupo y a, b, c ∈ G tales que a·b = a·c, entonces b = c

```
-- ---------------------------------------------------------------------
-- Sea G un grupo y a,b,c ∈ G. Demostrar que si a·b = a·c, entonces
-- b = c.
-- ---------------------------------------------------------------------

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

-- Por la siguiente cadena de igualdades
--    b = 1·b           [porque 1 es neutro]
--      = (a⁻¹·a)·b     [porque a⁻¹ es el inverso de a]
--      = a⁻¹·(a·b)     [por la asociativa]
--      = a⁻¹·(a·c)     [por la hipótesis]
--      = (a⁻¹·a)·c     [por la asociativa]
--      = 1·c           [porque a⁻¹ es el inverso de a]
--      = c             [porque 1 es neutro]

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

import Mathlib.Algebra.Group.Basic

variable {G : Type} [Group G]
variable {a b c : G}

-- 1ª demostración
-- ===============

example
  (h: a * b = a  * c)
  : b = c :=
calc b = 1 * b         := (one_mul b).symm
     _ = (a⁻¹ * a) * b := congrArg (. * b) (inv_mul_self a).symm
     _ = a⁻¹ * (a * b) := mul_assoc a⁻¹ a b
     _ = a⁻¹ * (a * c) := congrArg (a⁻¹ * .) h
     _ = (a⁻¹ * a) * c := (mul_assoc a⁻¹ a c).symm
     _ = 1 * c         := congrArg (. * c) (inv_mul_self a)
     _ = c             := one_mul c

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

```
example
  (h: a * b = a  * c)
  : b = c :=
calc b = 1 * b          := by rw [one_mul]
     _ = (a⁻¹ * a) * b := by rw [inv_mul_self]
     _ = a⁻¹ * (a * b) := by rw [mul_assoc]
     _ = a⁻¹ * (a * c) := by rw [h]
     _ = (a⁻¹ * a) * c := by rw [mul_assoc]
     _ = 1 * c          := by rw [inv_mul_self]
     _ = c              := by rw [one_mul]

-- 3ª demostración
-- ===============

example
  (h: a * b = a  * c)
  : b = c :=
calc b = 1 * b          := by simp
     _ = (a⁻¹ * a) * b := by simp
     _ = a⁻¹ * (a * b) := by simp
     _ = a⁻¹ * (a * c) := by simp [h]
     _ = (a⁻¹ * a) * c := by simp
     _ = 1 * c          := by simp
     _ = c              := by simp

-- 4ª demostración
-- ===============

example
  (h: a * b = a  * c)
  : b = c :=
calc b = a⁻¹ * (a * b) := by simp
     _ = a⁻¹ * (a * c) := by simp [h]
     _ = c              := by simp

-- 4ª demostración
-- ===============

example
  (h: a * b = a  * c)
  : b = c :=
mul_left_cancel h

-- 5ª demostración
```

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

example
  (h: a * b = a  * c)
  : b = c :=
by aesop

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

-- #check (inv_mul_self a : a⁻¹ * a = 1)
-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (mul_left_cancel : a * b = a * c → b = c)
-- #check (one_mul a : 1 * a = a)
```

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

Capítulo 6

Propiedades elementales de los anillos

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

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

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

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

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

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

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

variable (a b : R)

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

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

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

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

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

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

import Mathlib.Algebra.Ring.Defs

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

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

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

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

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

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

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

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

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

```
  add_neg_cancel_right a b

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-- 2ª demostración
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
--      = c

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

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

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

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

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

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

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

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

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

```
  rw [h]
  rw [add_neg_cancel_right]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

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

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

-- 5ª demostración
-- ===============

example : a * 0 = 0 :=
  mul_zero a

-- 6ª demostración
-- ===============

example : a * 0 = 0 :=
by simp

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-- 4ª demostración
```

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

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

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

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

6.11. Si R es un anillo, entonces -0 = 0

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

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

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

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

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

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

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

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

variable {R : Type _} [Ring R]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
by simp

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

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

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

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

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

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

-- Por la definición de la resta.

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.15. En los anillos, 1 + 1 = 2

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

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

-- Por cálculo.

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

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

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

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

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

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

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

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

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

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

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

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

Capítulo 7

Propiedades de orden en los números reales

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

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

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

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

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

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

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

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

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

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

import Mathlib.Data.Real.Basic

variable (a b c d e : ℝ)

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

example
```

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

-- 3ª demostración
-- ===============

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

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

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

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

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

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

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

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

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

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

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

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

```
import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

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

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

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

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

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

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

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

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

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

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b d : ℝ)

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

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

-- 3ª demostración
example
```

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

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

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

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

7.4. En ℝ, si $a \leq b$ y $c < d$, entonces $a + e^c + f \leq b + e^d + f$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

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

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

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

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

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

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

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

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

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

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

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

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b c : ℝ)

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

-- 2ª demostración
```

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.8. En ℝ, 2ab ≤ a² + b²

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

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

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.9. En ℝ, |ab| ≤ (a²+b²)/2]

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

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

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

```
--   a² + b² ≥ -2ab

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

-- 1ª demostración
-- ===============

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

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

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

-- 3ª demostración
-- ===============

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.10. En ℝ, min(a,b) = min(b,a)

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

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

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

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

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

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

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

-- 1ª demostración
```

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

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.11. En ℝ, max(a,b) = max(b,a)

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

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

-- Es consecuencia de la siguiente propiedad
```

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

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

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

lemma aux : max a b ≤ max b a :=
```

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.12. En ℝ, min(min(a,b),c) = min(a,min(b,c))

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

-- Demostración en lenguaje natural
```

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

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

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

import Mathlib.Data.Real.Basic

variable {a b c : ℝ}

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

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

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

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

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

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

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

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

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

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

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

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

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.13. En ℝ, min(a,b)+c = min(a+c,b+c)

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

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

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

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

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

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

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

import Mathlib.Data.Real.Basic

variable {a b c : ℝ}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.14. En ℝ, |a| - |b| ≤ |a - b|

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

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

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

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

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

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

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

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

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

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

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

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

import Mathlib.Data.Real.Basic

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

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

-- 3ª demostración
-- ===============

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.16. En ℝ, a <b → ¬(b <a)

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

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

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.17. Hay algún número real entre 2 y 3

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

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

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

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

import Mathlib.Data.Real.Basic

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

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

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

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

Se puede interactuar con las pruebas anteriores en Lean 4 Web

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

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

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

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

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

import Mathlib.Data.Real.Basic

variable (x : ℝ)

-- 1ª demostración
-- ===============

example
  (h : ∀ ε > 0, x ≤ ε)
  : x ≤ 0 :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > 0
  intro hx0
  -- hx0 : x > 0
  -- ⊢ False
  apply absurd h
  -- ⊢ ¬∀ (ε : ℝ), ε > 0 → x ≤ ε
  push_neg
  -- ⊢ ∃ ε, ε > 0 ∧ ε < x
  use x /2
  -- ⊢ x / 2 > 0 ∧ x / 2 < x
  constructor
  { show x / 2 > 0
    exact half_pos hx0 }
  { show x / 2 < x
    exact half_lt_self hx0 }

-- 2ª demostración
```

```
-- ===============

example
  (x : ℝ)
  (h : ∀ ε > 0, x ≤ ε)
  : x ≤ 0 :=
by
  contrapose! h
  -- ⊢ ∃ ε, ε > 0 ∧ ε < x
  use x / 2
  -- ⊢ x / 2 > 0 ∧ x / 2 < x
  constructor
  { show x / 2 > 0
    exact half_pos h }
  { show x / 2 < x
    exact half_lt_self h }

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- variable (p q : Prop)
-- #check (le_of_not_gt : ¬a > b → a ≤ b)
-- #check (half_lt_self : 0 < a → a / 2 < a)
-- #check (half_pos : 0 < a → 0 < a / 2)
-- #check (absurd : p → ¬p → q)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.19. Si 0 <0, entonces a >37 para cualquier número a

```
-- ---------------------------------------------------------------------
-- Demostrar que si 0 < 0, entonces a > 37 para cualquier número a.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Basta demostrar una contradicción, 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 : ℝ)

-- 1ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by
  exfalso
  -- ⊢ False
  show False
  exact lt_irrefl 0 h

-- 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

-- 5ª demostración
-- ===============

example
  (h : 0 < 0)
  : a > 37 :=
by linarith

-- Lemas usados
-- ============

-- variable (p q : Prop)
-- #check (lt_irrefl a : ¬a < a)
-- #check (absurd : p → ¬p → q)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.20. {x ≤ y, y ≰ x} ⊢ x ≤ y ∧ x ≠ y

```
-- ---------------------------------------------------------------------
-- 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

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

-- 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)⟩

-- 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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.21. $x \leq y \land x \neq y \vdash y \not\leq x$

```
-- ---------------------------------------------------------------------
-- 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
  cases' 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
  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
```

```
  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'

-- 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

-- 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’)

-- 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.22. (∃x ∈ ℝ)[2 <x <3]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic

-- 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⟩
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.23. Si $(\exists z \in \mathbb{R})[x < z < y]$, entonces $x < y$

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
variable (x y : ℝ)

-- 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

-- Lemas usados
-- ============

-- variable (a b c : ℝ)
-- #check (lt_trans : a < b → b < c → a < c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.24. En ℝ, x ≤ y ∧ x ≠ y → x ≤ y ∧ y ≰ x

```
-- ---------------------------------------------------------------------
-- Demostrar que, en ℝ, 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
-- ========================

import Mathlib.Data.Real.Basic
variable (x y : ℝ)

-- 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
-- ============

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.25. En $\mathbb{R}$, si $x \leq y$, entonces $y \nleq x \leftrightarrow x \neq y$

```
-- ---------------------------------------------------------------------
-- 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
  . show ¬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 }
  . show 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
  . show ¬y ≤ x → x ≠ y
    { intro h1
      -- h1 : ¬y ≤ x
      -- ⊢ x ≠ y
```

```
      intro h2
      -- h2 : x = y
      -- ⊢ False
      show False
      exact h1 (by rw [h2]) }
  . show 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
  . show ¬y ≤ x → x ≠ y
    { intro h1 h2
      exact h1 (by rw [h2]) }
  . show x ≠ y → ¬y ≤ x
    { intro h1 h2
      exact h1 (le_antisymm h h2) }

-- 4ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . intro h1 h2
    exact h1 (by rw [h2])
  . intro h1 h2
    exact h1 (le_antisymm h h2)

-- 5ª demostración
-- ===============
```

```
example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . exact fun h1 h2 ↦ h1 (by rw [h2])
  . exact fun h1 h2 ↦ h1 (le_antisymm h h2)

-- 6ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
  ⟨fun h1 h2 ↦ h1 (by rw [h2]),
   fun h1 h2 ↦ h1 (le_antisymm h h2)⟩

-- 7ª demostración
-- ===============

example
  (h : x ≤ y)
  : ¬y ≤ x ↔ x ≠ y :=
by
  constructor
  . show ¬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 }
  . show 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 }
```

```
-- 8ª 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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.26. Si |x + 3| <5, entonces -8 <x <2

```
-- ---------------------------------------------------------------------
-- Demostrar que si
--    |x + 3| < 5
-- entonces
--    -8 < x < 2
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que
--    |x + 3| < 5
-- entonces
--    -5 < x + 3 < 5
-- por tanto
--    -8 < x < 2

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic
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]
  intro h
  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]
  exact fun _ ↦ ⟨by linarith, by linarith⟩

-- Lemas usados
-- ============
```

```
-- #check (abs_lt: |x| < y ↔ -y < x ∧ x < y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.27. En ℝ, y >x² ⊢ y >0 ∨ y <-1

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
variable {x y : ℝ}

-- 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] }

-- Lema usado
-- ==========

-- #check (pow_two_nonneg x : 0 ≤ x ^ 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.28. En ℝ, -y >x² + 1 ⊢ y >0 ∨ y <-1

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
variable {x y : ℝ}

-- 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] }

-- Lemas usados
-- ============
```

```
-- variable (a b c : ℝ)
-- #check (add_le_add_right : b ≤ c → ∀ (a : ℝ),  b + a ≤ c + a)
-- #check (lt_neg : a < -b ↔ b < -a)
-- #check (pow_two_nonneg a : 0 ≤ a ^ 2)
-- #check (zero_add a : 0 + a = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.29. En ℝ, si x <|y|, entonces x <y ó x <-y

```
-- ---------------------------------------------------------------------
-- 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 y, por tanto,
-- x < y.
--
-- Segundo caso: Supongamos que y < 0. Entonces, |y| = -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
  cases' 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

-- Lemas usados
-- ============

-- #check (le_or_gt x y : x ≤ y ∨ x > y)
-- #check (abs_of_nonneg : 0 ≤ x → abs x = x)
-- #check (abs_of_neg : x < 0 → abs x = -x)
-- #check (lt_abs : x < |y| ↔ x < y ∨ x < -y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.30. En ℝ, x ≤ |x|

```
-- ---------------------------------------------------------------------
-- Demostrar que en ℝ,
--    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
-- ========================

import Mathlib.Data.Real.Basic
variable {x : ℝ}

-- 1ª demostración
-- ===============

example : x ≤ |x| :=
by
  cases' 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
  cases' 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

-- Lemas usados
-- ============

-- variable (y : ℝ)
-- #check (Left.self_le_neg : x ≤ 0 → x ≤ -x)
-- #check (abs_of_neg : x < 0 → |x| = -x)
-- #check (abs_of_nonneg : 0 ≤ x → |x| = x)
-- #check (le_abs_self x : x ≤ |x|)
-- #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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.31. En ℝ, -x ≤ |x|

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
variable {x : ℝ}

-- 1ª demostración
-- ===============

example : -x ≤ |x| :=
by
  cases' (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
  cases' (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_self x

-- Lemas usados
-- ============

-- variable (y : ℝ)
-- #check (abs_of_neg : x < 0 → |x| = -x)
-- #check (abs_of_nonneg : 0 ≤ x → |x| = x)
-- #check (le_or_gt x y : x ≤ y ∨ x > y)
-- #check (le_refl x : x ≤ x)
-- #check (neg_le_abs_self x : -x ≤ |x|)
-- #check (neg_le_self : 0 ≤ x → -x ≤ x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.32. En ℝ, |x + y| ≤ |x| + |y|

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
variable {x y : ℝ}

-- 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_self x) (neg_le_abs_self 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_self x) (neg_le_abs_self 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_self x, neg_le_abs_self y]

-- 3ª demostración
-- ===============

example : |x + y| ≤ |x| + |y| :=
  abs_add x y

-- Lemas usados
-- ============

-- variable (a b c d : ℝ)
-- #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_or_gt x y : x ≤ y ∨ x > y)
-- #check (neg_add x y : -(x + y) = -x + -y)
```

```
-- #check (neg_le_abs_self x : -x ≤ |x|)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.33. En ℝ, si x ≠ 0 entonces x <0 ó x >0

```
-- ---------------------------------------------------------------------
-- 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

-- Lemas usados
-- ============

-- variable (y : ℝ)
-- #check (lt_trichotomy x y : x < y ∨ x = y ∨ y < x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.34. Si (∃ x, y ∈ ℝ)[z = x² + y² ∨ z = x² + y² + 1], entonces z ≥ 0

```
-- ---------------------------------------------------------------------
-- 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

-- 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.35. En ℝ, si 1 <a, entonces a <aa

```
-- ---------------------------------------------------------------------
-- 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[1a = 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 = 1a   [por L2]
--      < aa   [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 _).2 h
  . -- ⊢ a = 1 * a
    rw [one_mul]
  . -- ⊢ 0 < a
    exact lt_trans zero_lt_one h

-- Lemas usados
```

```
-- ============

-- variables (a b c : ℝ)
-- #check (mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c))
-- #check (one_mul a : 1 * a = a)
-- #check (lt_trans : a < b → b < c → a < c)
-- #check (zero_lt_one : 0 < 1)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

7.36. Si x, y ∈ ℝ tales que (∀ z)[y <z → x ≤ z], entonces x ≤ y

```
-- ---------------------------------------------------------------------
-- Sean x, y números reales tales que
--    ∀ z, y < z → x ≤ z
-- Demostrar que x ≤ y.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Lo demostraremos por reducción al absurdo. Para ello, supongamos que
--    x ≰ y.
-- Entonces
--    y < x
-- y, por la densidad de ℝ, existe un a tal que
--    y < a < x
-- Puesto que y < a, por la hipótesis se tiene que
--    x ≤ a
-- en contradicción con
--    a < x.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable {x y : ℝ}

-- 1ª demostración
-- ===============
```

```
example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
by
  by_contra h1
  -- h1 : ¬x ≤ y
  -- ⊢ False
  have hxy : x > y := not_le.mp h1
  -- ⊢ ¬x > y
  cases' (exists_between hxy) with a ha
  -- a : ℝ
  -- ha : y < a ∧ a < x
  apply (lt_irrefl a)
  -- ⊢ a < a
  calc a
       < x := ha.2
     _ ≤ a := h a ha.1

-- 2ª demostración
-- ===============

example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > y
  intro hxy
  -- hxy : x > y
  -- ⊢ False
  cases' (exists_between hxy) with a ha
  -- a : ℝ
  -- ha : y < a ∧ a < x
  apply (lt_irrefl a)
  -- ⊢ a < a
  calc a
       < x := ha.2
     _ ≤ a := h a ha.1

-- 3ª demostración
-- ===============

example
  (h : ∀ z, y < z → x ≤ z) :
```

```
  x ≤ y :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > y
  intro hxy
  -- hxy : x > y
  -- ⊢ False
  cases' (exists_between hxy) with a ha
  -- ha : y < a ∧ a < x
  apply (lt_irrefl a)
  -- ⊢ a < a
  exact lt_of_lt_of_le ha.2 (h a ha.1)

-- 4ª demostración
-- ===============

example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > y
  intro hxy
  -- hxy : x > y
  -- ⊢ False
  cases' (exists_between hxy) with a ha
  -- a : ℝ
  -- ha : y < a ∧ a < x
  exact (lt_irrefl a) (lt_of_lt_of_le ha.2 (h a ha.1))

-- 5ª demostración
-- ===============

example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
by
  apply le_of_not_gt
  -- ⊢ ¬x > y
  intro hxy
  -- hxy : x > y
  -- ⊢ False
  rcases (exists_between hxy) with ⟨a, hya, hax⟩
  -- a : ℝ
  -- hya : y < a
```

```
  -- hax : a < x
  exact (lt_irrefl a) (lt_of_lt_of_le hax (h a hya))

-- 6ª demostración
-- ===============

example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
le_of_forall_le_of_dense h

-- Lemas usados
-- ============

-- variable (z : ℝ)
-- #check (exists_between : x < y → ∃ z, x < z ∧ z < y)
-- #check (le_of_forall_le_of_dense : (∀ z, y < z → x ≤ z) → x ≤ y)
-- #check (le_of_not_gt : ¬x > y → x ≤ y)
-- #check (lt_irrefl x : ¬x < x)
-- #check (lt_of_lt_of_le : x < y → y ≤ z → x < z)
-- #check (not_le : ¬x ≤ y ↔ y < x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 8

Divisibilidad

8.1. Si x,y,z ∈ ℕ, entonces x | yxz

```
-- ---------------------------------------------------------------------
-- Demostrar que si x, y, z ∈ ℕ, entonces
--    x ∣ y * x * z
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la transitividad de la divisibilidad aplicada a las relaciones
--    x ∣ yx
--    yx ∣ yxz

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (x y z : ℕ)

-- 1ª demostración
-- ===============

example : x ∣ y * x * z :=
by
  have h1 : x ∣ y * x :=
    dvd_mul_left x y
  have h2 : (y * x) ∣ (y * x * z) :=
    dvd_mul_right (y * x) z
  show x ∣ y * x * z
```

```
  exact dvd_trans h1 h2

-- 2ª demostración
-- ===============

example : x ∣ y * x * z :=
dvd_trans (dvd_mul_left x y) (dvd_mul_right (y * x) z)

-- 3ª demostración
-- ===============

example : x ∣ y * x * z :=
by
  apply dvd_mul_of_dvd_left
  apply dvd_mul_left

-- Lemas usados
-- ============

-- #check (dvd_mul_left x y : x ∣ y * x)
-- #check (dvd_mul_right x y : x ∣ x * y)
-- #check (dvd_trans : x ∣ y → y ∣ z → x ∣ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.2. Si x divide a w, también divide a y(xz)+x²+w²

```
-- ---------------------------------------------------------------------
-- Demostrar que si
--    x ∣ w
-- entonces
--    x ∣ y * (x * z) + x^2 + w^2
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por la divisibilidad de la suma basta probar que
--    x ∣ yxz                                                        (1)
--    x ∣ x²                                                         (2)
--    x ∣ w²                                                         (3)
--
-- Para demostrar (1), por la divisibilidad del producto se tiene
```

```
--    x | xz
-- y, de nuevo por la divisibilidad del producto,
--    x | y(xz).
--
-- La propiedad (2) se tiene por la definición de cuadrado y la
-- divisibilidad del producto.
--
-- La propiedad (3) se tiene por la definición de cuadrado, la hipótesis
-- y la divisibilidad del producto.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (w x y z : ℕ)

-- 1ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  have h1 : x ∣ x * z :=
    dvd_mul_right x z
  have h2 : x ∣ y * (x * z) :=
    dvd_mul_of_dvd_right h1 y
  have h3 : x ∣ x^2 := by
    apply dvd_mul_left
  have h4 : x ∣ w * w :=
    dvd_mul_of_dvd_left h w
  have h5 : x ∣ w^2 := by
    rwa [← pow_two w] at h4
  have h6 : x ∣ y * (x * z) + x^2 :=
    dvd_add h2 h3
  show x ∣ y * (x * z) + x^2 + w^2
  exact dvd_add h6 h5

-- 2ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  apply dvd_add
  { apply dvd_add
    { apply dvd_mul_of_dvd_right
      apply dvd_mul_right }
```

```
    { rw [pow_two]
      apply dvd_mul_right }}
  { rw [pow_two]
    apply dvd_mul_of_dvd_left h }

-- 3ª demostración
example
  (h : x ∣ w)
  : x ∣ y * (x * z) + x^2 + w^2 :=
by
  repeat' apply dvd_add
  { apply dvd_mul_of_dvd_right
    apply dvd_mul_right }
  { rw [pow_two]
    apply dvd_mul_right }
  { rw [pow_two]
    apply dvd_mul_of_dvd_left h }

-- Lemas usados
-- ============

-- #check (dvd_add : x ∣ y → x ∣ z → x ∣ y + z)
-- #check (dvd_mul_left x y : x ∣ y * x)
-- #check (dvd_mul_right x y : x ∣ x * y)
-- #check (dvd_mul_of_dvd_left : x ∣ y → ∀ (c : ℕ), x ∣ y * c)
-- #check (dvd_mul_of_dvd_right : x ∣ y → ∀ (c : ℕ), x ∣ c * y)
-- #check (pow_two x : x ^ 2 = x * x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.3. Transitividad de la divisibilidad

```
-- ---------------------------------------------------------------------
-- Demostrar que la relación de divisibilidad es transitiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que a ∣ b y b ∣ c. Entonces, existen d y e tales que
--    b = ad                                                        (1)
--    c = be                                                        (2)
-- Por tanto,
```

```
--    c = be        [por (2)]
--      = (ad)e     [por (1)]
--      = a(de)
-- Por consiguiente, a | c.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

variable {a b c : ℕ}

-- 1ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  rcases divab with ⟨d, beq : b = a * d⟩
  rcases divbc with ⟨e, ceq : c = b * e⟩
  have h1 : c = a * (d * e) :=
    calc c = b * e       := ceq
         _ = (a * d) * e := congrArg (. * e) beq
         _ = a * (d * e) := mul_assoc a d e
  show a ∣ c
  exact Dvd.intro (d * e) h1.symm

-- 2ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  rcases divab with ⟨d, beq : b = a * d⟩
  rcases divbc with ⟨e, ceq : c = b * e⟩
  use (d * e)
  -- ⊢ c = a * (d * e)
  rw [ceq, beq]
  -- ⊢ (a * d) * e = a * (d * e)
  exact mul_assoc a d e

-- 3ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
```

```
  a ∣ c :=
by
  rcases divbc with ⟨e, rfl⟩
  -- ⊢ a ∣ b * e
  rcases divab with ⟨d, rfl⟩
  -- ⊢ a ∣ a * d * e
  use (d * e)
  -- ⊢ a * d * e = a * (d * e)
  ring

-- 4ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by
  cases' divab with d beq
  -- d : ℕ
  -- beq : b = a * d
  cases' divbc with e ceq
  -- e : ℕ
  -- ceq : c = b * e
  rw [ceq, beq]
  -- ⊢ a ∣ a * d * e
  use (d * e)
  -- ⊢ (a * d) * e = a * (d * e)
  exact mul_assoc a d e

-- 5ª demostración
example
  (divab : a ∣ b)
  (divbc : b ∣ c) :
  a ∣ c :=
by exact dvd_trans divab divbc

-- Lemas usados
-- ============

-- #check (mul_assoc a b c : (a * b) * c = a * (b * c))
-- #check (Dvd.intro c : a * c = b → a ∣ b)
-- #check (dvd_trans : a ∣ b → b ∣ c → a ∣ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.4. Si a divide a b y a c, entonces divide a b+c

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es un divisor de b y de c, tambien lo es de b + c.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Puesto que a divide a b y a c, existen d y e tales que
--    b = ad                                                          (1)
--    c = ae                                                          (2)
-- Por tanto,
--    b + c = ad + c      [por (1)]
--          = ad + ae     [por (2)]
--          = a(d + e)    [por la distributiva]
-- Por consiguiente, a divide a b + c.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

variable {a b c : ℕ}

-- 1ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  have h1 : b + c = a * (d + e) :=
    calc b + c
         = (a * d) + c       := congrArg (. + c) beq
       _ = (a * d) + (a * e) := congrArg ((a * d) + .) ceq
       _ = a * (d + e)       := by rw [← mul_add]
  show a ∣ (b + c)
  exact Dvd.intro (d + e) h1.symm

-- 2ª demostración
example
  (h1 : a ∣ b)
```

```
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  have h1 : b + c = a * (d + e) := by linarith
  show a ∣ (b + c)
  exact Dvd.intro (d + e) h1.symm

-- 3ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, beq : b = a * d⟩
  rcases h2 with ⟨e, ceq: c = a * e⟩
  show a ∣ (b + c)
  exact Dvd.intro (d + e) (by linarith)

-- 4ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  cases' h1 with d beq
  -- d : ℕ
  -- beq : b = a * d
  cases' h2 with e ceq
  -- e : ℕ
  -- ceq : c = a * e
  rw [ceq, beq]
  -- ⊢ a ∣ a * d + a * e
  use (d + e)
  -- ⊢ a * d + a * e = a * (d + e)
  ring

-- 5ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
by
  rcases h1 with ⟨d, rfl⟩
```

```
  -- ⊢ a ∣ a * d + c
  rcases h2 with ⟨e, rfl⟩
  -- ⊢ a ∣ a * d + a * e
  use (d + e)
  -- ⊢ a * d + a * e = a * (d + e)
  ring

-- 6ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ (b + c) :=
dvd_add h1 h2

-- Lemas usados
-- ============

-- #check (Dvd.intro c : a * c = b → a ∣ b)
-- #check (dvd_add : a ∣ b →  a ∣ c → a ∣ (b + c))
-- #check (mul_add a b c : a * (b + c) = a * b + a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.5. Conmutatividad del máximo común divisor

```
-- ---------------------------------------------------------------------
-- Demostrar que si m y n son números naturales, entonces
--    gcd m n = gcd n m
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ x, y ∈ ℕ)[gcd(x,y) | gcd(y,x)]                             (1)
-- En efecto, sustituyendo en (1) x por m e y por n, se tiene
--    gcd(m, n) | gcd(n, m)                                         (2)
-- y sustituyendo en (1) x por n e y por m, se tiene
--    gcd(n, m) | gcd(m, n)                                         (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
```

```
--    gcd(m, n) = gcd(n, m)
--
-- Para demostrar (1), por la definición del máximo común divisor, basta
-- demostrar las siguientes relaciones
--    gcd(m, n) | n
--    gcd(m, n) | m
-- y ambas se tienen por la definición del máximo común divisor.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (k m n : ℕ)

open Nat

-- 1ª demostración del lema auxiliar
lemma aux : gcd m n ∣ gcd n m :=
by
  have h1 : gcd m n ∣ n :=
    gcd_dvd_right m n
  have h2 : gcd m n ∣ m :=
    gcd_dvd_left m n
  show gcd m n ∣ gcd n m
  exact dvd_gcd h1 h2

-- 2ª demostración del lema auxiliar
example : gcd m n ∣ gcd n m :=
dvd_gcd (gcd_dvd_right m n) (gcd_dvd_left m n)

-- 1ª demostración
example : gcd m n = gcd n m :=
by
  have h1 : gcd m n ∣ gcd n m := aux m n
  have h2 : gcd n m ∣ gcd m n := aux n m
  show gcd m n = gcd n m
  exact _root_.dvd_antisymm h1 h2

-- 2ª demostración
example : gcd m n = gcd n m :=
by
  apply _root_.dvd_antisymm
  { exact aux m n }
  { exact aux n m }
```

```
-- 3ª demostración
example : gcd m n = gcd n m :=
_root_.dvd_antisymm (aux m n) (aux n m)

-- 4ª demostración
example : gcd m n = gcd n m :=
-- by apply?
gcd_comm m n

-- Lemas usados
-- ============

-- #check (_root_.dvd_antisymm : m ∣ n → n ∣ m → m = n)
-- #check (dvd_gcd : k ∣ m → k ∣ n → k ∣ gcd m n)
-- #check (gcd_comm m n : gcd m n = gcd n m)
-- #check (gcd_dvd_left  m n: gcd m n ∣ m)
-- #check (gcd_dvd_right m n : gcd m n ∣ n)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.6. Si (m ∣ n ∧ m ≠ n), entonces (m ∣ n ∧ ¬(n ∣ m))

```
-- ---------------------------------------------------------------------
-- 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
  . show m ∣ n
    exact h.left
  . show ¬n ∣ m
    { intro (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
  . exact h.left
  . intro (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
  cases' 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
  rcases h with ⟨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

-- Lemas usados
-- ============

-- #check (dvd_antisymm : m ∣ n → n ∣ m → m = n)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.7. Existen números primos m y n tales que 4 <m <n <10

```
-- ---------------------------------------------------------------------
-- 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
-- ======================

import Mathlib.Data.Nat.Prime
import Mathlib.Tactic

example :
  ∃ m n : ℕ, 4 < m ∧ m < n ∧ n < 10 ∧ Nat.Prime m ∧ Nat.Prime n :=
by
  use 5, 7
  norm_num
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.8. 3 divide al máximo común divisor de 6 y 15

```
-- ---------------------------------------------------------------------
-- 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.Data.Nat.GCD.Basic

open Nat

-- 1ª demostración
-- ===============

example : 3 ∣ 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 ∣ 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 ∣ gcd m n ↔ k ∣ m ∧ k ∣ n)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.9. Si m divide a n o a k, entonces m divide a nk

```
-- ---------------------------------------------------------------------
-- Demostrar que si m divide a n o a k, entonces m 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.10. Existen infinitos números primos

```
-- ---------------------------------------------------------------------
-- Demostrar que hay infinitos números primos.
-- ---------------------------------------------------------------------
```

```
-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas de los números naturales
--    n ≠ 1 → el menor factor primo de n es primo                      (L1)
--    n! > 0                                                           (L2)
--    0 < k → n < k + n                                                (L3)
--    k < n → n ≠ k                                                    (L4)
--    k ≱ n → k ≤ n                                                    (L5)
--    0 < k → k ≤ n → k ∣ n!                                           (L6)
--    0 < minFac(n)                                                    (L7)
--    k ∣ m → (k ∣ n ↔ k ∣ m + n)                                      (L8)
--    minFac(n) ∣ n                                                    (L9)
--    Prime(n) → ¬n ∣ 1                                                (L10)
--
-- Sea p el menor factor primo de n! + 1. Tenemos que demostrar que n ≤
-- p y que p es primo.
--
-- Para demostrar que p es primo, por el lema L1, basta demostrar que
--    n! + 1 ≠ 1
-- Su demostración es
--    n ! > 0           [por L2]
--    ==> n ! + 1 > 1   [por L3]
--    ==> n ! + 1 ≠ 1   [por L4]
--
-- Para demostrar n ≤ p, por el lema L5, basta demostrar que
--    n ≱ p
-- Su demostración es
--    n ≥ p
--    ==> p ∣ n!    [por L6 y L7]
--    ==> p ∣ 1     [por L8 y (p ∣ n! + 1) por L9]
--    ==> Falso     [por L10 y p es primo]

-- Demostración con Lean4
-- ======================

import Mathlib.Tactic
import Mathlib.Data.Nat.Prime
open Nat

-- 1ª demostración
-- ===============

example
  (n : ℕ) :
```

```
  ∃ p, n ≤ p ∧ Nat.Prime p :=
by
  let p := minFac (n !  + 1)
  have h1 : Nat.Prime p := by
    apply minFac_prime
    -- ⊢ n ! + 1 ≠ 1
    have h3 : n ! > 0     := factorial_pos n
    have h4 : n ! + 1 > 1 := Nat.lt_add_of_pos_left h3
    show n ! + 1 ≠ 1
    exact Nat.ne_of_gt h4
  use p
  constructor
  . -- ⊢ n ≤ p
    apply le_of_not_ge
    -- ⊢ ¬n ≥ p
    intro h5
    -- h5 : n ≥ p
    -- ⊢ False
    have h6 : p ∣ n ! := dvd_factorial (minFac_pos _) h5
    have h7 : p ∣ 1   := (Nat.dvd_add_iff_right h6).mpr (minFac_dvd _)
    show False
    exact (Nat.Prime.not_dvd_one h1) h7
  . -- ⊢ Nat.Prime p
    exact h1
  done

-- 2ª demostración
-- ===============

example
  (n : ℕ) :
  ∃ p, n ≤ p ∧ Nat.Prime p :=
exists_infinite_primes n

-- Lemas usados
-- ============

-- variable (k m n : ℕ)
-- #check (Nat.Prime.not_dvd_one : Nat.Prime n → ¬n ∣ 1)
-- #check (Nat.dvd_add_iff_right : k ∣ m → (k ∣ n ↔ k ∣ m + n))
-- #check (Nat.dvd_one : n ∣ 1 ↔ n = 1)
-- #check (Nat.lt_add_of_pos_left : 0 < k → n < k + n)
-- #check (Nat.ne_of_gt : k < n → n ≠ k)
-- #check (dvd_factorial : 0 < k → k ≤ n → k ∣ n !)
-- #check (factorial_pos n: n ! > 0)
```

```
-- #check (le_of_not_ge : ¬k ≥ n → k ≤ n)
-- #check (minFac_dvd n : minFac n ∣ n)
-- #check (minFac_pos n : 0 < minFac n)
-- #check (minFac_prime : n ≠ 1 → Nat.Prime (minFac n))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.11. Si n² es par, entonces n es par

```
-- ---------------------------------------------------------------------
-- Demostrar que si n² es par, entonces n es par.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usara el siguiente lema: "Si p es primo, entonces
--    (∀ a, b ∈ ℕ)[p ∣ ab ↔ p ∣ a ∨ p ∣ b].
--
-- Si n² es par, entonces 2 divide a n.n y, por el lema, 2 divide a n.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
open Nat
variable (n : ℕ)

-- 1ª demostración
-- ===============

example
  (h : 2 ∣ n ^ 2)
  : 2 ∣ n :=
by
  rw [pow_two] at h
  -- h : 2 ∣ n * n
  have h1 : Nat.Prime 2 := prime_two
  have h2 : 2 ∣ n ∨ 2 ∣ n := (Prime.dvd_mul h1).mp h
  rcases h2 with h3 | h4
  · -- h3 : 2 ∣ n
    exact h3
  · -- h4 : 2 ∣ n
```

```
    exact h4
  done

-- 2ª demostración
-- ===============

example
  (h : 2 ∣ n ^ 2)
  : 2 ∣ n :=
by
  rw [pow_two] at h
  -- h : 2 ∣ n * n
  have h2 : 2 ∣ n ∨ 2 ∣ n := (Prime.dvd_mul prime_two).mp h
  rcases h2 with h3 | h4
  · exact h3
  · exact h4
  done

-- 3ª demostración
-- ===============

example
  (h : 2 ∣ n ^ 2)
  : 2 ∣ n :=
by
  rw [pow_two] at h
  -- h : 2 ∣ n * n
  have h2 : 2 ∣ n ∨ 2 ∣ n := (Prime.dvd_mul prime_two).mp h
  tauto
  done

-- Lemas usados
-- ============

-- variable (p a b : ℕ)
-- #check (prime_two : Nat.Prime 2)
-- #check (Prime.dvd_mul : Nat.Prime p → (p ∣ a * b ↔ p ∣ a ∨ p ∣ b))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.12. La raíz cuadrada de 2 es irracional

```
-- ---------------------------------------------------------------------
-- Demostrar que la raíz cuadrada de 2 es irracional; es decir, que no
-- existen m, n ∈ ℕ tales que m y n son coprimos (es decir, que no
-- factores comunes distintos de uno) y m² = 2n².
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos el lema del ejercicio anterior:
--    (∀ n ∈ ℕ)[2 ∣ n² → 2 ∣ n]
--
-- Supongamos que existen existen m, n ∈ ℕ tales que m y n son coprimos y
-- m² = 2n² y tenemos que demostrar una contradicción. Puesto que 2 no
-- divide a 1, para tener la contradicción basta demostrar que 2 divide
-- a 1 y (puesto que m y n son coprimos), para ello es suficiente
-- demostrar que 2 divide al máximo común divisor de m y n. En
-- definitiva, basta demostrar que 2 divide a m y a n.
--
-- La demostración de que 2 divide a m es
--    m² = 2n² ⟹ 2 ∣ m²
--             ⟹ 2 ∣ m    [por el lema]
--
-- Para demostrar que 2 divide a n, observamos que, puesto que 2 divide
-- a m, existe un k ∈ ℕ tal que m = 2k. Sustituyendo en
--    m² = 2n²
-- se tiene
--    (2k)² = 2n²
-- Simplificando, queda
--    2k = n²
-- Por tanto, 2 divide a n² y, por el lema, 2 divide a n.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
import Mathlib.Data.Nat.Prime
import Std.Data.Nat.Gcd
open Nat
variable {m n : ℕ}

lemma par_si_cuadrado_par
```

```
  (h : 2 ∣ n ^ 2)
  : 2 ∣ n :=
by
  rw [pow_two] at h
  -- h : 2 ∣ n * n
  have h2 : 2 ∣ n ∨ 2 ∣ n := (Prime.dvd_mul prime_two).mp h
  tauto

example : ¬∃ m n, coprime m n ∧ m ^ 2 = 2 * n ^ 2 :=
by
  rintro ⟨m, n, ⟨h1, h2⟩⟩
  -- m n : ℕ
  -- h1 : coprime m n
  -- h2 : m ^ 2 = 2 * n ^ 2
  -- ⊢ False
  have h3 : ¬(2 ∣ 1) := by norm_num
  have h4 : 2 ∣ 1 := by
    have h5 : Nat.gcd m n = 1 := h1
    rw [← h5]
    -- ⊢ 2 ∣ Nat.gcd m n
    have h6 : 2 ∣ m := by
      apply par_si_cuadrado_par
      -- ⊢ 2 ∣ m ^ 2
      rw [h2]
      -- ⊢ 2 ∣ 2 * n ^ 2
      exact Nat.dvd_mul_right 2 (n ^ 2)
    have h7 : 2 ∣ n := by
      have h8 : ∃ k, m = 2 * k := h6
      rcases h8 with ⟨k, h9⟩
      -- k : ℕ
      -- h9 : m = 2 * k
      have h10 : 2 * k ^ 2 = n ^ 2 := by
        have h10a : 2 * (2 * k ^ 2) = 2 * n ^ 2 := calc
          2 * (2 * k ^ 2) = (2 * k) ^ 2 := by nlinarith
                        _ = m ^ 2       := by rw [← h9]
                        _ = 2 * n ^ 2   := h2
        show 2 * k ^ 2 = n ^ 2
        exact (mul_right_inj' (by norm_num : 2 ≠ 0)).mp h10a
      have h11 : 2 ∣ n ^ 2 := by
        rw [← h10]
        -- ⊢ 2 ∣ 2 * k ^ 2
        exact Nat.dvd_mul_right 2 (k ^ 2)
      show 2 ∣ n
      exact par_si_cuadrado_par h11
    show 2 ∣ Nat.gcd m n
```

```
    exact Nat.dvd_gcd h6 h7
  show False
  exact h3 h4

-- Lemas usados
-- ============

-- variable (p k : ℕ)
-- #check (pow_two n : n ^ 2 = n * n)
-- #check (Prime.dvd_mul : Nat.Prime p → (p ∣ m * n ↔ p ∣ m ∨ p ∣ n))
-- #check (prime_two : Nat.Prime 2)
-- #check (Nat.dvd_gcd : k ∣ m → k ∣ n → k ∣ Nat.gcd m n)
-- #check (Nat.dvd_mul_right m n :  m ∣ m * n)
-- #check (mul_right_inj' : k ≠ 0 → (k * m = k * n ↔ m = n))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

8.13. Un número es par si y solo si lo es su cuadrado

```
-- ---------------------------------------------------------------------
-- Demostrar que un número es par si y solo si lo es su cuadrado.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea n ∈ ℤ. Tenemos que demostrar que n² es par si y solo si n es
-- par. Lo haremos probando las dos implicaciones.
--
-- (⟹) Lo demostraremos por contraposición. Para ello, supongamos que n
-- no es par. Entonces, existe un k ∈ Z tal que
--    n = 2k+1                                                      (1)
-- Luego,
--    n² = (2k+1)²          [por (1)]
--       = 4k²+4k+1
--       = 2(2k(k+1))+1
-- Por tanto, n² es impar.
--
-- (⟸) Supongamos que n es par. Entonces, existe un k ∈ ℤ tal que
--    n = 2k                                                        (2)
-- Luego,
```

```
--    n² = (2k)²          [por (2)]
--        = 2(2k²)
-- Por tanto, n² es par.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Int.Parity
import Mathlib.Tactic
open Int

variable (n : ℤ)

-- 1ª demostración
-- ===============

example :
  Even (n^2) ↔ Even n :=
by
  constructor
  . -- ⊢ Even (n ^ 2) → Even n
    contrapose
    -- ⊢ ¬Even n → ¬Even (n ^ 2)
    intro h
    -- h : ¬Even n
    -- ⊢ ¬Even (n ^ 2)
    rw [←odd_iff_not_even] at *
    -- h : Odd n
    -- ⊢ Odd (n ^ 2)
    cases' h with k hk
    -- k : ℤ
    -- hk : n = 2 * k + 1
    use 2*k*(k+1)
    -- ⊢ n ^ 2 = 2 * (2 * k * (k + 1)) + 1
    calc n^2
         = (2*k+1)^2        := by rw [hk]
       _ = 4*k^2+4*k+1      := by ring
       _ = 2*(2*k*(k+1))+1 := by ring
  . -- ⊢ Even n → Even (n ^ 2)
    intro h
    -- h : Even n
    -- ⊢ Even (n ^ 2)
    cases' h with k hk
    -- k : ℤ
    -- hk : n = k + k
```

```
    use 2*k^2
    -- ⊢ n ^ 2 = 2 * k ^ 2 + 2 * k ^ 2
    calc n^2
         = (k + k)^2      := by rw [hk]
       _ = 2*k^2 + 2*k^2 := by ring

-- 2ª demostración
-- ===============

example :
  Even (n^2) ↔ Even n :=
by
  constructor
  . -- ⊢ Even (n ^ 2) → Even n
    contrapose
    -- ⊢ ¬Even n → ¬Even (n ^ 2)
    rw [←odd_iff_not_even]
    -- ⊢ Odd n → ¬Even (n ^ 2)
    rw [←odd_iff_not_even]
    -- ⊢ Odd n → Odd (n ^ 2)
    unfold Odd
    -- ⊢ (∃ k, n = 2 * k + 1) → ∃ k, n ^ 2 = 2 * k + 1
    intro h
    -- h : ∃ k, n = 2 * k + 1
    -- ⊢ ∃ k, n ^ 2 = 2 * k + 1
    cases' h with k hk
    -- k : ℤ
    -- hk : n = 2 * k + 1
    use 2*k*(k+1)
    -- ⊢ n ^ 2 = 2 * (2 * k * (k + 1)) + 1
    rw [hk]
    -- ⊢ (2 * k + 1) ^ 2 = 2 * (2 * k * (k + 1)) + 1
    ring
  . -- ⊢ Even n → Even (n ^ 2)
    unfold Even
    -- ⊢ (∃ r, n = r + r) → ∃ r, n ^ 2 = r + r
    intro h
    -- h : ∃ r, n = r + r
    -- ⊢ ∃ r, n ^ 2 = r + r
    cases' h with k hk
    -- k : ℤ
    -- hk : n = k + k
    use 2*k^2
    -- ⊢ n ^ 2 = 2 * k ^ 2 + 2 * k ^ 2
    rw [hk]
```

```
    -- ⊢ (k + k) ^ 2 = 2 * k ^ 2 + 2 * k ^ 2
    ring

-- 3ª demostración
-- ===============

example :
  Even (n^2) ↔ Even n :=
by
  constructor
  . -- ⊢ Even (n ^ 2) → Even n
    contrapose
    -- ⊢ ¬Even n → ¬Even (n ^ 2)
    rw [←odd_iff_not_even]
    -- ⊢ Odd n → ¬Even (n ^ 2)
    rw [←odd_iff_not_even]
    -- ⊢ Odd n → Odd (n ^ 2)
    rintro ⟨k, rfl⟩
    -- k : ℤ
    -- ⊢ Odd ((2 * k + 1) ^ 2)
    use 2*k*(k+1)
    -- ⊢ (2 * k + 1) ^ 2 = 2 * (2 * k * (k + 1)) + 1
    ring
  . -- ⊢ Even n → Even (n ^ 2)
    rintro ⟨k, rfl⟩
    -- k : ℤ
    -- ⊢ Even ((k + k) ^ 2)
    use 2*k^2
    -- ⊢ (k + k) ^ 2 = 2 * k ^ 2 + 2 * k ^ 2
    ring

-- 4ª demostración
-- ===============

example :
  Even (n^2) ↔ Even n :=
calc Even (n^2)
     ↔ Even (n * n)      := iff_of_eq (congrArg Even (sq n))
   _ ↔ (Even n ∨ Even n) := even_mul
   _ ↔ Even n            := or_self_iff (Even n)

-- 5ª demostración
-- ===============

example :
```

```
  Even (n^2) ↔ Even n :=
calc Even (n^2)
     ↔ Even (n * n)       := by ring_nf
   _ ↔ (Even n ∨ Even n) := even_mul
   _ ↔ Even n            := by simp

-- Lemas usados
-- ============

-- variable (a b : Prop)
-- variable (m : ℤ)
-- #check (even_mul : Even (m * n) ↔ Even m ∨ Even n)
-- #check (iff_of_eq : a = b → (a ↔ b))
-- #check (odd_iff_not_even : Odd n ↔ ¬Even n)
-- #check (or_self_iff a : a ∨ a ↔ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 9

Retículos

9.1. En los retículos, x ⊓ y = y ⊓ x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--     x ⊓ y = y ⊓ x
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ a, b)[a ⊓ b ≤ b ⊓ a]                                       (1)
-- En efecto, sustituyendo en (1) a por x y b por y, se tiene
--    x ⊓ y ≤ y ⊓ x                                                 (2)
-- y sustituyendo en (1) a por y y b por x, se tiene
--    y ⊓ x ≤ x ⊓ y                                                 (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
--    x ⊓ y = y ⊓ x
--
-- Para demostrar (1), por la definición del ínfimo, basta demostrar
-- las siguientes relaciones
--    y ⊓ x ≤ x
--    y ⊓ x ≤ y
-- y ambas se tienen por la definición del ínfimo.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
```

```
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración del lema auxiliar
lemma aux : x ⊓ y ≤ y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y :=
    inf_le_right
  have h2 : x ⊓ y ≤ x :=
    inf_le_left
  show x ⊓ y ≤ y ⊓ x
  exact le_inf h1 h2

-- 2ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
by
  apply le_inf
  { apply inf_le_right }
  { apply inf_le_left }

-- 3ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
le_inf inf_le_right inf_le_left

-- 1ª demostración
example : x ⊓ y = y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y ⊓ x :=
    aux x y
  have h2 : y ⊓ x ≤ x ⊓ y :=
    aux y x
  show x ⊓ y = y ⊓ x
  exact le_antisymm h1 h2

-- 2ª demostración
example : x ⊓ y = y ⊓ x :=
by
  apply le_antisymm
  { apply aux }
  { apply aux }

-- 3ª demostración
example : x ⊓ y = y ⊓ x :=
le_antisymm (aux x y) (aux y x)
```

```
-- 4ª demostración
example : x ⊓ y = y ⊓ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊓ y = y ⊓ x :=
-- by apply?
inf_comm

-- Lemas usados
-- ============

-- #check (inf_comm : x ⊓ y = y ⊓ x)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_le_right : x ⊓ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.2. En los retículos, x ⊔ y = y ⊔ x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊔ y = y ⊔ x
-- para todo x e y en el retículo.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Es consecuencia del siguiente lema auxiliar
--    (∀ a, b)[a ⊔ b ≤ b ⊔ a]                                        (1)
-- En efecto, sustituyendo en (1) a por x y b por y, se tiene
--    x ⊔ y ≤ y ⊔ x                                                  (2)
-- y sustituyendo en (1) a por y y b por x, se tiene
--    y ⊔ x ≤ x ⊔ y                                                  (3)
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad
-- a (2) y (3), se tiene
--    x ⊔ y = y ⊔ x
--
-- Para demostrar (1), por la definición del supremo, basta demostrar
-- las siguientes relaciones
```

```
--    x ≤ y ⊔ x
--    y ≤ y ⊔ x
-- y ambas se tienen por la definición del supremo.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración del lema auxiliar
lemma aux : x ⊔ y ≤ y ⊔ x :=
by
  have h1 : x ≤ y ⊔ x :=
    le_sup_right
  have h2 : y ≤ y ⊔ x :=
    le_sup_left
  show x ⊔ y ≤ y ⊔ x
  exact sup_le h1 h2

-- 2ª demostración del lema auxiliar
example : x ⊔ y ≤ y ⊔ x :=
by
  apply sup_le
  { apply le_sup_right }
  { apply le_sup_left }

-- 3ª demostración del lema auxiliar
example : x ⊔ y ≤ y ⊔ x :=
sup_le le_sup_right le_sup_left

-- 1ª demostración
example : x ⊔ y = y ⊔ x :=
by
  have h1 : x ⊔ y ≤ y ⊔ x :=
    aux x y
  have h2 : y ⊔ x ≤ x ⊔ y :=
    aux y x
  show x ⊔ y = y ⊔ x
  exact le_antisymm h1 h2

-- 2ª demostración
example : x ⊔ y = y ⊔ x :=
by
```

```
  apply le_antisymm
  { apply aux }
  { apply aux }

-- 3ª demostración
example : x ⊔ y = y ⊔ x :=
le_antisymm (aux x y) (aux y x)

-- 4ª demostración
example : x ⊔ y = y ⊔ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊔ y = y ⊔ x :=
-- by apply?
sup_comm

-- Lemas usados
-- ============

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (sup_comm : x ⊔ y = y ⊔ x)
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.3. En los retículos, (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--     (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm : x ≤ y → y ≤ x → x = y
--    le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y
--    inf_le_left : x ⊓ y ≤ x
--    inf_le_right : x ⊓ y ≤ y)
```

```
--
-- Por le_antisym, es suficiente demostrar las siguientes relaciones:
--    (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)                                         (1)
--    x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z                                         (2)
--
-- Para demostrar (1), por le_inf, basta probar que
--    (x ⊓ y) ⊓ z ≤ x                                                  (1a)
--    (x ⊓ y) ⊓ z ≤ y ⊓ z                                              (1b)
--
-- La (1a) se demuestra por la siguiente cadena de desigualdades
--    (x ⊓ y) ⊓ z ≤ x ⊓ y   [por inf_le_left]
--                ≤ x       [por inf_le_left]
--
-- Para demostrar (1b), por le_inf, basta probar que
--    (x ⊓ y) ⊓ z ≤ y                                                 (1b1)
--    (x ⊓ y) ⊓ z ≤ z                                                 (1b2)
--
-- La (1b1) se demuestra por la siguiente cadena de desigualdades
--    (x ⊓ y) ⊓ z ≤ x ⊓ y   [por inf_le_left]
--                ≤ y       [por inf_le_right]
--
-- La (1b2) se tiene por inf_le_right.
--
-- Para demostrar (2), por le_inf, basta probar que
--    x ⊓ (y ⊓ z) ≤ x ⊓ y                                              (2a)
--    x ⊓ (y ⊓ z) ≤ z                                                  (2b)
--
-- Para demostrar (2a), por le_inf, basta probar que
--    x ⊓ (y ⊓ z) ≤ x                                                 (2a1)
--    x ⊓ (y ⊓ z) ≤ y                                                 (2a2)
--
-- La (2a1) se tiene por inf_le_left.
--
-- La (2a2) se demuestra por la siguiente cadena de desigualdades
--    x ⊓ (y ⊓ z) ≤ y ⊓ z   [por inf_le_right]
--                ≤ y       [por inf_le_left]
--
-- La (2b) se demuestra por la siguiente cadena de desigualdades
--    x ⊓ (y ⊓ z) ≤ y ⊓ z   [por inf_le_right]
--                ≤ z       [por inf_le_right]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
```

```
variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  have h1 : (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z) := by
  { have h1a : (x ⊓ y) ⊓ z ≤ x := calc
      (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
                _ ≤ x     := by exact inf_le_left
    have h1b : (x ⊓ y) ⊓ z ≤ y ⊓ z := by
    { have h1b1 : (x ⊓ y) ⊓ z ≤ y := calc
        (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
                  _ ≤ y     := by exact inf_le_right
      have h1b2 : (x ⊓ y) ⊓ z ≤ z :=
        inf_le_right
      show (x ⊓ y) ⊓ z ≤ y ⊓ z
      exact le_inf h1b1 h1b2 }
    show (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)
    exact le_inf h1a h1b }
  have h2 : x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z := by
  { have h2a : x ⊓ (y ⊓ z) ≤ x ⊓ y := by
    { have h2a1 : x ⊓ (y ⊓ z) ≤ x :=
        inf_le_left
      have h2a2 : x ⊓ (y ⊓ z) ≤ y := calc
        x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
                  _ ≤ y     := by exact inf_le_left
      show x ⊓ (y ⊓ z) ≤ x ⊓ y
      exact le_inf h2a1 h2a2 }
    have h2b : x ⊓ (y ⊓ z) ≤ z := by calc
      x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
                _ ≤ z     := by exact inf_le_right
    show x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z
    exact le_inf h2a h2b }
  show (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊓ y ⊓ z = x ⊓ (y ⊓ z) := by
  apply le_antisymm
  · apply le_inf
```

```
    · apply le_trans
      apply inf_le_left
      apply inf_le_left
    . apply le_inf
      · apply le_trans
        apply inf_le_left
        apply inf_le_right
      . apply inf_le_right
  . apply le_inf
    · apply le_inf
      · apply inf_le_left
      . apply le_trans
        apply inf_le_right
        apply inf_le_left
    . apply le_trans
      apply inf_le_right
      apply inf_le_right

-- 3ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  apply le_antisymm
  . apply le_inf
    . apply inf_le_of_left_le inf_le_left
    . apply le_inf (inf_le_of_left_le inf_le_right) inf_le_right
  . apply le_inf
    . apply le_inf inf_le_left (inf_le_of_right_le inf_le_left)
    . apply inf_le_of_right_le inf_le_right

-- 4ª demostración
-- ===============

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
le_antisymm
  (le_inf
    (inf_le_of_left_le inf_le_left)
    (le_inf (inf_le_of_left_le inf_le_right) inf_le_right))
  (le_inf
    (le_inf inf_le_left (inf_le_of_right_le inf_le_left))
    (inf_le_of_right_le inf_le_right))

-- 5ª demostración
-- ===============
```

```
example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
-- by apply?
inf_assoc

-- Lemas usados
-- ============

-- #check (inf_assoc : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z))
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_le_of_left_le : x ≤ z → x ⊓ y ≤ z)
-- #check (inf_le_of_right_le : y ≤ z → x ⊓ y ≤ z)
-- #check (inf_le_right : x ⊓ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.4. En los retículos, (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm  : x ≤ y → y ≤ x → x = y
--    le_sup_left  : x ≤ x ⊔ y
--    le_sup_right : y ≤ x ⊔ y
--    sup_le       : x ≤ z → y ≤ z → x ⊔ y ≤ z
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)                                        (1)
--    x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z                                        (2)
--
-- Para demostrar (1), por sup_le, basta probar
--    x ⊔ y ≤ x ⊔ (y ⊔ z)                                             (1a)
--    z ≤ x ⊔ (y ⊔ z)                                                 (1b)
--
```

```
-- Para demostrar (1a), por sup_le, basta probar
--    x ≤ x ⊔ (y ⊔ z)                                                  (1a1)
--    y ≤ x ⊔ (y ⊔ z)                                                  (1a2)
--
-- La (1a1) se tiene por le_sup_left.
--
-- La (1a2) se tiene por la siguiente cadena de desigualdades:
--    y ≤ y ⊔ z          [por le_sup_left]
--      ≤ x ⊔ (y ⊔ z)    [por le_sup_right]
--
-- La (1b) se tiene por la siguiente cadena de desigualdades
--    z ≤ y ⊔ z          [por le_sup_right]
--      ≤ x ⊔ (y ⊔ z)    [por le_sup_right]
--
-- Para demostrar (2), por sup_le, basta probar
--    x ≤ (x ⊔ y) ⊔ z                                                   (2a)
--    y ⊔ z ≤ (x ⊔ y) ⊔ z                                               (2b)
--
-- La (2a) se demuestra por la siguiente cadena de desigualdades:
--    x ≤ x ⊔ y          [por le_sup_left]
--      ≤ (x ⊔ y) ⊔ z    [por le_sup_left]
--
-- Para demostrar (2b), por sup_le, basta probar
--    y ≤ (x ⊔ y) ⊔ z                                                  (2b1)
--    z ≤ (x ⊔ y) ⊔ z                                                  (2b2)
--
-- La (2b1) se demuestra por la siguiente cadena de desigualdades:
--    y ≤ x ⊔ y          [por le_sup_right]
--      ≤ (x ⊔ y) ⊔ z    [por le_sup_left]
--
-- La (2b2) se tiene por le_sup_right.

-- Demostraciones con Lean 4
-- =========================

import Mathlib.Order.Lattice

variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
```

```
  have h1 : (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z) := by
  { have h1a : x ⊔ y ≤ x ⊔ (y ⊔ z) := by
    { have h1a1 : x ≤ x ⊔ (y ⊔ z) := by exact le_sup_left
      have h1a2 : y ≤ x ⊔ (y ⊔ z) := calc
        y ≤ y ⊔ z         := by exact le_sup_left
        _ ≤ x ⊔ (y ⊔ z) := by exact le_sup_right
      show x ⊔ y ≤ x ⊔ (y ⊔ z)
      exact sup_le h1a1 h1a2 }
    have h1b : z ≤ x ⊔ (y ⊔ z) := calc
      z ≤ y ⊔ z         := by exact le_sup_right
      _ ≤ x ⊔ (y ⊔ z) := by exact le_sup_right
    show (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    exact sup_le h1a h1b }
  have h2 : x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z := by
  { have h2a : x ≤ (x ⊔ y) ⊔ z := calc
      x ≤ x ⊔ y         := by exact le_sup_left
      _ ≤ (x ⊔ y) ⊔ z := by exact le_sup_left
    have h2b : y ⊔ z ≤ (x ⊔ y) ⊔ z := by
    { have h2b1 : y ≤ (x ⊔ y) ⊔ z := calc
        y ≤ x ⊔ y         := by exact le_sup_right
        _ ≤ (x ⊔ y) ⊔ z := by exact le_sup_left
      have h2b2 : z ≤ (x ⊔ y) ⊔ z := by
        exact le_sup_right
      show  y ⊔ z ≤ (x ⊔ y) ⊔ z
      exact sup_le h2b1 h2b2 }
    show x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    exact sup_le h2a h2b }
  show (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊔ y ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  · -- (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    apply sup_le
    · -- x ⊔ y ≤ x ⊔ (y ⊔ z)
      apply sup_le
      . -- x ≤ x ⊔ (y ⊔ z)
        apply le_sup_left
      · -- y ≤ x ⊔ (y ⊔ z)
        apply le_trans
        . -- y ≤ y ⊔ z
```

```
          apply @le_sup_left _ _ y z
        . -- y ⊔ z ≤ x ⊔ (y ⊔ z)
          apply le_sup_right
    . -- z ≤ x ⊔ (y ⊔ z)
      apply le_trans
      . -- z ≤ x ⊔ (y ⊔ z)
        apply @le_sup_right _ _ y z
      . -- y ⊔ z ≤ x ⊔ (y ⊔ z)
        apply le_sup_right
  . -- x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    apply sup_le
    · -- x ≤ (x ⊔ y) ⊔ z
      apply le_trans
      . -- x ≤ x ⊔ y
        apply @le_sup_left _ _ x y
      . -- x ⊔ y ≤ (x ⊔ y) ⊔ z
        apply le_sup_left
    . -- y ⊔ z ≤ (x ⊔ y) ⊔ z
      apply sup_le
      · -- y ≤ (x ⊔ y) ⊔ z
        apply le_trans
        . -- y ≤ x ⊔ y
          apply @le_sup_right _ _ x y
        . -- x ⊔ y ≤ (x ⊔ y) ⊔ z
          apply le_sup_left
      . -- z ≤ (x ⊔ y) ⊔ z
        apply le_sup_right

-- 3ª demostración
-- ===============

example : x ⊔ y ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  · apply sup_le
    · apply sup_le
      . apply le_sup_left
      · apply le_trans
        . apply @le_sup_left _ _ y z
        . apply le_sup_right
    . apply le_trans
      . apply @le_sup_right _ _ y z
      . apply le_sup_right
  . apply sup_le
    · apply le_trans
```

```
      . apply @le_sup_left _ _ x y
      . apply le_sup_left
    . apply sup_le
      · apply le_trans
        . apply @le_sup_right _ _ x y
        . apply le_sup_left
      . apply le_sup_right

-- 4ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . -- (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    apply sup_le
    . -- x ⊔ y ≤ x ⊔ (y ⊔ z)
      apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . -- z ≤ x ⊔ (y ⊔ z)
      apply le_sup_of_le_right le_sup_right
  . -- x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    apply sup_le
    . -- x ≤ (x ⊔ y) ⊔ z
      apply le_sup_of_le_left le_sup_left
    . -- y ⊔ z ≤ (x ⊔ y) ⊔ z
      apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 5ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . apply sup_le
    . apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . apply le_sup_of_le_right le_sup_right
  . apply sup_le
    . apply le_sup_of_le_left le_sup_left
    . apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 6ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
le_antisymm
```

```
  (sup_le
    (sup_le le_sup_left (le_sup_of_le_right le_sup_left))
    (le_sup_of_le_right le_sup_right))
  (sup_le
    (le_sup_of_le_left le_sup_left)
    (sup_le (le_sup_of_le_left le_sup_right) le_sup_right))

-- 7ª demostración
-- ===============

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
-- by apply?
sup_assoc

-- Lemas usados
-- ============

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
-- #check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)
-- #check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.5. En los retículos, x ⊓ (x ⊔ y) = x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊓ (x ⊔ y) = x
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--    le_antisymm  : x ≤ y → y ≤ x → x = y
--    inf_le_left  : x ⊓ y ≤ x
--    le_inf       : z ≤ x → z ≤ y → z ≤ x ⊓ y
--    le_rfl       : x ≤ x
```

```
--    le_sup_left  : x ≤ x ⊔ y
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    x ⊓ (x ⊔ y) ≤ x                                                (1)
--    x ≤ x ⊓ (x ⊔ y)                                                (2)
--
-- La (1) se tiene por inf_le_left.
--
-- Para demostrar la (2), por le_inf, basta probar las relaciones:
--    x ≤ x                                                         (2a)
--    x ≤ x ⊔ y                                                     (2b)
--
-- La (2a) se tiene por le_rfl.
--
-- La (2b) se tiene por le_sup_left

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (x y : α)

-- 1ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  have h1 : x ⊓ (x ⊔ y) ≤ x := inf_le_left
  have h2 : x ≤ x ⊓ (x ⊔ y)
  { have h2a : x ≤ x := le_rfl
    have h2b : x ≤ x ⊔ y := le_sup_left
    show x ≤ x ⊓ (x ⊔ y)
    exact le_inf h2a h2b }
  show x ⊓ (x ⊔ y) = x
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  have h1 : x ⊓ (x ⊔ y) ≤ x := by simp
  have h2 : x ≤ x ⊓ (x ⊔ y) := by simp
  show x ⊓ (x ⊔ y) = x
```

```
  exact le_antisymm h1 h2

-- 3ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by
  apply le_antisymm
  . -- x ⊓ (x ⊔ y) ≤ x
    apply inf_le_left
  . -- x ≤ x ⊓ (x ⊔ y)
    apply le_inf
    . -- x ≤ x
      apply le_rfl
    . -- x ≤ x ⊔ y
      apply le_sup_left

-- 4ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
le_antisymm inf_le_left (le_inf le_rfl le_sup_left)

-- 5ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
-- by apply?
inf_sup_self

-- 6ª demostración
-- ===============

example : x ⊓ (x ⊔ y) = x :=
by simp

-- Lemas usados
-- ============

-- variable (z : α)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_sup_self : x ⊓ (x ⊔ y) = x)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
-- #check (le_rfl : x ≤ x)
```

```
-- #check (le_sup_left : x ≤ x ⊔ y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.6. En los retículos, x ⊔ (x ⊓ y) = x

```
-- ---------------------------------------------------------------------
-- Demostrar que en los retículos se verifica que
--    x ⊔ (x ⊓ y) = x
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- En la demostración se usarán los siguientes lemas
--   le_antisymm  : x ≤ y → y ≤ x → x = y
--   inf_le_left  : x ⊓ y ≤ x
--   le_rfl       : x ≤ x
--   le_sup_left  : x ≤ x ⊔ y
--   sup_le       : x ≤ z → y ≤ z → x ⊔ y ≤ z
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--    x ⊔ (x ⊓ y) ≤ x                                            (1)
--    x ≤ x ⊔ (x ⊓ y)    [que se tiene por le_sup_left]
--
-- Para demostrar (1), por sup_le, basta probar las relaciones:
--    x ≤ x              [que se tiene por le_rfl]
--    x ⊓ y ≤ x          [que se tiene por inf_le_left]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]--
variable (x y : α)

-- 1ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  have h1 : x ⊔ (x ⊓ y) ≤ x
  { have h1a : x ≤ x := le_rfl
```

```
    have h1b : x ⊓ y ≤ x := inf_le_left
    show x ⊔ (x ⊓ y) ≤ x
    exact sup_le h1a h1b }
  have h2 : x ≤ x ⊔ (x ⊓ y) := le_sup_left
  show x ⊔ (x ⊓ y) = x
  exact le_antisymm h1 h2

-- 2ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  have h1 : x ⊔ (x ⊓ y) ≤ x := by simp
  have h2 : x ≤ x ⊔ (x ⊓ y) := by simp
  show x ⊔ (x ⊓ y) = x
  exact le_antisymm h1 h2

-- 3ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by
  apply le_antisymm
  . -- x ⊔ (x ⊓ y) ≤ x
    apply sup_le
    . -- x ≤ x
      apply le_rfl
    . -- x ⊓ y ≤ x
      apply inf_le_left
  . -- x ≤ x ⊔ (x ⊓ y)
    apply le_sup_left

-- 4ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
-- by apply?
sup_inf_self

-- 5ª demostración
-- ===============

example : x ⊔ (x ⊓ y) = x :=
by simp
```

```
-- Lemas usados
-- ============

-- variable (z : α)
-- #check (le_rfl : x ≤ x)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (sup_inf_self : x ⊔ (x ⊓ y) = x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.7. En los retículos, una distributiva del ínfimo implica la otra

```
-- ---------------------------------------------------------------------
-- Demostrar que si α es un retículo tal que
--    ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
-- entonces
--    (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c)
-- para todos los elementos de α.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se demuestra por la siguiente cadena de igualdades
--    (a ⊔ b) ⊓ c = c ⊓ (a ⊔ b)          [por conmutatividad de ⊓]
--                = (c ⊓ a) ⊔ (c ⊓ b)    [por la hipótesis]
--                = (a ⊓ c) ⊔ (c ⊓ b)    [por conmutatividad de ⊓]
--                = (a ⊓ c) ⊔ (b ⊓ c)    [por conmutatividad de ⊓]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (a b c : α)

-- 1ª demostración
example
```

```
  (h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
  : (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
calc
  (a ⊔ b) ⊓ c = c ⊓ (a ⊔ b)       := by rw [inf_comm]
            _ = (c ⊓ a) ⊔ (c ⊓ b) := by rw [h]
            _ = (a ⊓ c) ⊔ (c ⊓ b) := by rw [@inf_comm _ _ c a]
            _ = (a ⊓ c) ⊔ (b ⊓ c) := by rw [@inf_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
  : (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
by simp [h, inf_comm]

-- Lemas usados
-- ============

-- #check (inf_comm : a ⊓ b = b ⊓ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

9.8. En los retículos, una distributiva del supremos implica la otra

```
-- ---------------------------------------------------------------------
-- Demostrar que si α es un retículo tal que
--    ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z)
-- entonces
--    (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se demuestra por la siguiente cadena de igualdades
--    (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)          [por la conmutatividad de ⊔]
--                = (c ⊔ a) ⊓ (c ⊔ b)    [por la hipótesis]
--                = (a ⊔ c) ⊓ (c ⊔ b)    [por la conmutatividad de ⊔]
--                = (a ⊔ c) ⊓ (b ⊔ c)    [por la conmutatividad de ⊔]

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Order.Lattice
variable {α : Type _} [Lattice α]
variable (a b c : α)

-- 1ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
calc
  (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)       := by rw [sup_comm]
            _ = (c ⊔ a) ⊓ (c ⊔ b) := by rw [h]
            _ = (a ⊔ c) ⊓ (c ⊔ b) := by rw [@sup_comm _ _ c a]
            _ = (a ⊔ c) ⊓ (b ⊔ c) := by rw [@sup_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
by simp [h, sup_comm]

-- Lemas usados
-- ============

-- #check (sup_comm : a ⊔ b = b ⊔ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

Capítulo 10

Relaciones de orden

10.1. En los órdenes parciales, a <b ↔ a ≤ b ∧ a ≠ b

```
-- ---------------------------------------------------------------------
-- 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 (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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

10.2. Si ≤ es un preorden, entonces <es irreflexiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si ≤ es un preorden, entonces < 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
-- ========================

import Mathlib.Tactic
variable {α : Type _} [Preorder α]
variable (a : α)

-- 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
```

```
-- Lemas usados
-- ============

-- variable (b : α)
-- #check (lt_iff_le_not_le : a < b ↔ a ≤ b ∧ ¬b ≤ a)
-- #check (irrefl a : ¬a < a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

10.3. Si ≤ es un preorden, entonces <es transitiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si ≤ es un preorden, entonces < 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).

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Tactic
variable {α : Type _} [Preorder α]
variable (a b c : α)

-- 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 (lt_iff_le_not_le : a < b ↔ a ≤ b ∧ ¬b ≤ a)
-- #check (le_trans : a ≤ b → b ≤ c → a ≤ c)
-- #check (lt_trans : a < b → b < c → a < c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

Capítulo 11

Relaciones de equivalencia

11.1. La congruencia módulo 2 es una relación de equivalencia

```
-- ---------------------------------------------------------------------
-- Se define la relación R entre los números enteros de forma que x está
-- relacionado con y si x-y es divisible por 2. Demostrar que R es una
-- relación de equivalencia.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que R es reflexiva, simétrica y transitiva.
--
-- Para demostrar que R es reflexiva, sea x ∈ ℤ. Entonces, x - x = 0 que
-- es divisible por 2. Luego, xRx.
--
-- Para demostrar que R es simétrica, sean x, y ∈ ℤ tales que
-- xRy. Entonces, x - y es divisible por 2. Luego, existe un a ∈ ℤ tal
-- que
--    x - y = 2·a
-- Por tanto,
--    y - x = 2·(-a)
-- Por lo que y - x es divisible por 2 y yRx.
--
-- Para demostrar que R es transitiva, sean x, y, z ∈ ℤ tales que xRy y
-- yRz. Entonces, tanto x - y como y - z son divibles por 2. Luego,
-- existen a, b ∈ ℤ tales que
--    x - y = 2·a
```

```
--    y - z = 2·b
-- Por tanto,
--    x - z = 2·(a + b)
-- Por lo que x - z es divisible por 2 y xRz.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Int.Basic
import Mathlib.Tactic

def R (m n : ℤ) := 2 ∣ (m - n)

-- 1ª demostración
-- ===============

example : Equivalence R :=
by
  repeat' constructor
  . -- ⊢ ∀ (x : ℤ), R x x
    intro x
    -- x : ℤ
    -- ⊢ R x x
    unfold R
    -- ⊢ 2 ∣ x - x
    rw [sub_self]
    -- ⊢ 2 ∣ 0
    exact dvd_zero 2
  . -- ⊢ ∀ {x y : ℤ}, R x y → R y x
    intros x y hxy
    -- x y : ℤ
    -- hxy : R x y
    -- ⊢ R y x
    unfold R at *
    -- hxy : 2 ∣ x - y
    -- ⊢ 2 ∣ y - x
    cases' hxy with a ha
    -- a : ℤ
    -- ha : x - y = 2 * a
    use -a
    -- ⊢ y - x = 2 * -a
    calc y - x
        = -(x - y) := (neg_sub x y).symm
      _ = -(2 * a) := by rw [ha]
      _ = 2 * -a   := neg_mul_eq_mul_neg 2 a
```

```
  . -- ⊢ ∀ {x y z : ℤ}, R x y → R y z → R x z
    intros x y z hxy hyz
    -- x y z : ℤ
    -- hxy : R x y
    -- hyz : R y z
    -- ⊢ R x z
    cases' hxy with a ha
    -- a : ℤ
    -- ha : x - y = 2 * a
    cases' hyz with b hb
    -- b : ℤ
    -- hb : y - z = 2 * b
    use a + b
    -- ⊢ x - z = 2 * (a + b)
    calc x - z
         = (x - y) + (y - z) := (sub_add_sub_cancel x y z).symm
       _ = 2 * a + 2 * b     := congrArg₂ (. + .) ha hb
       _ = 2 * (a + b)       := (mul_add 2 a b).symm

-- 2ª demostración
-- ===============

example : Equivalence R :=
by
  repeat' constructor
  . -- ⊢ ∀ (x : ℤ), R x x
    intro x
    -- x : ℤ
    -- ⊢ R x x
    simp [R]
  . -- ⊢ ∀ {x y : ℤ}, R x y → R y x
    rintro x y ⟨a, ha⟩
    -- x y a : ℤ
    -- ha : x - y = 2 * a
    -- ⊢ R y x
    use -a
    -- ⊢ y - x = 2 * -a
    linarith
  . -- ⊢ ∀ {x y z : ℤ}, R x y → R y z → R x z
    rintro x y z ⟨a, ha⟩ ⟨b, hb⟩
    -- x y z a : ℤ
    -- ha : x - y = 2 * a
    -- b : ℤ
    -- hb : y - z = 2 * b
    -- ⊢ R x z
```

```
    use a + b
    -- ⊢ x - z = 2 * (a + b)
    linarith

-- Lemas usados
-- ============

-- variable (a b c x y x' y' : ℤ)
-- #check (congrArg₂ (. + .) : x = x' → y = y' → x + y = x' + y')
-- #check (dvd_zero a : a ∣ 0)
-- #check (mul_add a b c : a * (b + c) = a * b + a * c)
-- #check (neg_mul_eq_mul_neg a b : -(a * b) = a * -b)
-- #check (neg_sub a b : -(a - b) = b - a)
-- #check (sub_add_sub_cancel a b c : a - b + (b - c) = a - c)
-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 12

Anillos ordenados

12.1. En los anillos ordenados, a ≤ b → 0 ≤ b - a

```
-- ---------------------------------------------------------------------
-- Demostrar que en los anillos ordenados se verifica que
--    a ≤ b → 0 ≤ b - a
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    sub_self         : a - a = 0
--    sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c
--
-- Supongamos que
--    a ≤ b                                                         (1)
-- La demostración se tiene por la siguiente cadena de desigualdades:
--    0 = a - a    [por sub_self]
--      ≤ b - a    [por (1) y sub_le_sub_right]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
```

```
example : a ≤ b → 0 ≤ b - a :=
by
  intro h
  calc
    0 = a - a := (sub_self a).symm
    _ ≤ b - a := sub_le_sub_right h a

-- 2ª demostración
example : a ≤ b → 0 ≤ b - a :=
sub_nonneg.mpr

-- 3ª demostración
example : a ≤ b → 0 ≤ b - a :=
by simp

-- Lemas usados
-- ============

-- #check (sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

12.2. En los anillos ordenados, 0 ≤ b - a → a ≤ b

```
-- ---------------------------------------------------------------------
-- Demostrar que en los anillos ordenados
--    0 ≤ b - a → a ≤ b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    zero_add a : 0 + a = a
--    add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a
--    sub_add_cancel a b : a - b + b = -a
-- Supongamos que
--    0 ≤ b - a                                                     (1)
-- La demostración se tiene por la siguiente cadena de desigualdades:
```

```
--    a = 0 + a          [por zero_add]
--      ≤ (b - a) + a    [por (1) y add_le_add_right]
--      = b              [por sub_add_cancel]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
by
  intro h
  calc
    a = 0 + a       := (zero_add a).symm
    _ ≤ (b - a) + a := add_le_add_right h a
    _ = b           := sub_add_cancel b a

-- 2ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
-- by apply?
sub_nonneg.mp

-- 3ª demostración
-- ===============

example : 0 ≤ b - a → a ≤ b :=
by simp

-- Lemas usados
-- ============

-- #check (zero_add a : 0 + a = a)
-- #check (add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a)
-- #check (sub_add_cancel a b : a - b + b = a)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

12.3. En los anillos ordenados, {a ≤ b, 0 ≤ c} ⊢ ac ≤ bc

```
-- ---------------------------------------------------------------------
-- Demostrar que, en los anillos ordenados, si
--    a ≤ b
--    0 ≤ c
-- entonces
--    a * c ≤ b * c
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    sub_nonneg                 : 0 ≤ a - b ↔ b ≤ a)
--    mul_nonneg                 : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
--    sub_mul a b c              : (a - b) * c = a * c - b * c)
--
-- Supongamos que
--    a ≤ b                                                            (1)
--    0 ≤ c
-- De (1), por sub_nonneg, se tiene
--    0 ≤ b - a
-- y con (2), por mul_nonneg, se tiene
--    0 ≤ (b - a) * c
-- que, por sub_mul, da
--    0 ≤ b * c - a * c
-- y, aplicándole sub_nonneg, se tiene
--    a * c ≤ b * c

-- Demostraciones con Lean4
-- ========================

import Mathlib.Algebra.Order.Ring.Defs
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)

-- 1ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
```

```
  : a * c ≤ b * c :=
by
  have h3 : 0 ≤ b - a :=
    sub_nonneg.mpr h1
  have h4 : 0 ≤ b * c - a * c := calc
    0 ≤ (b - a) * c   := mul_nonneg h3 h2
    _ = b * c - a * c := sub_mul b a c
  show a * c ≤ b * c
  exact sub_nonneg.mp h4

-- 2ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  have h3 : 0 ≤ b - a := sub_nonneg.mpr h1
  have h4 : 0 ≤ (b - a) * c := mul_nonneg h3 h2
  -- h4 : 0 ≤ b * c - a * c
  rw [sub_mul] at h4
  -- a * c ≤ b * c
  exact sub_nonneg.mp h4

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  -- 0 ≤ b * c - a * c
  apply sub_nonneg.mp
  -- 0 ≤ (b - a) * c
  rw [← sub_mul]
  apply mul_nonneg
  . -- 0 ≤ b - a
    exact sub_nonneg.mpr h1
  . -- 0 ≤ c
    exact h2

-- 4ª demostración
-- ===============
```

```
example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
by
  apply sub_nonneg.mp
  rw [← sub_mul]
  apply mul_nonneg (sub_nonneg.mpr h1) h2

-- 5ª demostración
example
  (h1 : a ≤ b)
  (h2 : 0 ≤ c)
  : a * c ≤ b * c :=
-- by apply?
mul_le_mul_of_nonneg_right h1 h2

-- Lemas usados
-- ============

-- #check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
-- #check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
-- #check (sub_mul a b c : (a - b) * c = a * c - b * c)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

Capítulo 13

Espacios métricos

13.1. En los espacios métricos, dist(x,y) ≥ 0

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los espacios métricos
--    0 ≤ dist x y
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usarán los siguientes lemas:
--    dist_comm x y             : dist x y = dist y x
--    dist_self x               : dist x x = 0
--    dist_triangle x y z       : dist x z ≤ dist x y + dist y z
--    mul_two a                 : a * 2 = a + a
--    nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a
--    zero_lt_two               : 0 < 2
--
-- Por nonneg_of_mul_nonneg_left es suficiente demostrar las siguientes
-- desigualdades:
--    0 ≤ dist x y * 2                                                (1)
--    0 < 2                                                           (2)
--
-- La (1) se demuestra por las siguiente cadena de desigualdades:
--    0 = dist x x               [por dist_self]
--      ≤ dist x y + dist y x    [por dist_triangle]
--      = dist x y + dist x y    [por dist_comm]
--      = dist x y * 2           [por mul_two]
--
-- La (2) se tiene por zero_lt_two.
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Topology.MetricSpace.Basic
variable {X : Type _} [MetricSpace X]
variable (x y : X)

-- 1ª demostración
example : 0 ≤ dist x y :=
by
  have h1 : 0 ≤ dist x y * 2 := calc
    0 = dist x x            := (dist_self x).symm
    _ ≤ dist x y + dist y x := dist_triangle x y x
    _ = dist x y + dist x y := by rw [dist_comm x y]
    _ = dist x y * 2        := (mul_two (dist x y)).symm
  show 0 ≤ dist x y
  exact nonneg_of_mul_nonneg_left h1 zero_lt_two

-- 2ª demostración
example : 0 ≤ dist x y :=
by
  apply nonneg_of_mul_nonneg_left
  . -- 0 ≤ dist x y * 2
    calc 0 = dist x x            := by simp only [dist_self]
         _ ≤ dist x y + dist y x := by simp only [dist_triangle]
         _ = dist x y + dist x y := by simp only [dist_comm]
         _ = dist x y * 2        := by simp only [mul_two]
  . -- 0 < 2
    exact zero_lt_two

-- 3ª demostración
example : 0 ≤ dist x y :=
by
  have : 0 ≤ dist x y + dist y x := by
    rw [← dist_self x]
    apply dist_triangle
  linarith [dist_comm x y]

-- 3ª demostración
example : 0 ≤ dist x y :=
-- by apply?
dist_nonneg

-- Lemas usados
```

```
-- ============

-- variable (a b : ℝ)
-- variable (z : X)
-- #check (dist_comm x y : dist x y = dist y x)
-- #check (dist_nonneg : 0 ≤ dist x y)
-- #check (dist_self x : dist x x = 0)
-- #check (dist_triangle x y z : dist x z ≤ dist x y + dist y z)
-- #check (mul_two a : a * 2 = a + a)
-- #check (nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a)
-- #check (zero_lt_two : 0 < 2)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

Capítulo 14

Funciones reales

14.1. La suma de una cota superior de f y una cota superior de g es una cota superior de f+g

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de una cota superior de f y una cota superior
-- de g es una cota superior de f + g.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
-- Por la definición de cota superior, hay que demostrar que
--    (∀ x ∈ ℝ) [f(x) + g(x) ≤ a + b]                                (1)
-- Para ello, sea x ∈ R. Puesto que es a es una cota superior de f, se
-- tiene que
--    f(x) ≤ a                                                        (2)
-- y, puesto que b es una cota superior de g, se tiene que
--    g(x) ≤ b                                                        (3)
-- De (2) y (3), por add_le_add, se tiene que
--    f(x) + g(x) ≤ a + b
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- 1ª demostración
-- ===============

example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  have h1 : ∀ x, (f + g) x  ≤ a + b := by
  { intro x
    have h2 : f x ≤ a := hfa x
    have h3 : g x ≤ b := hgb x
    show (f + g) x ≤ a + b
    exact add_le_add h2 h3 }
  show CotaSuperior (f + g) (a + b)
  exact h1

-- 2ª demostración
-- ===============

example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  have h1 : ∀ x, (f + g) x ≤ a + b := by
  { intro x
    show (f + g) x ≤ a + b
    exact add_le_add (hfa x) (hgb x) }
  show CotaSuperior (f + g) (a + b)
  exact h1

-- 3ª demostración
-- ===============

example
```

```
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
by
  intro x
  dsimp
  apply add_le_add
  . apply hfa
  . apply hgb

-- 4ª demostración
-- ===============

theorem sumaCotaSup
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  : CotaSuperior (f + g) (a + b) :=
λ x ↦ add_le_add (hfa x) (hgb x)

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.2. La suma de una cota inferior de f y una cota inferior de g es una cota inferior de f+g

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de una cota inferior de f y una  cota inferior
-- de g es una cota inferior de f + g.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
```

```
-- Por la definición de cota inferior, hay que demostrar que
--    (∀ x ∈ ℝ) [a + b ≤ f(x) + g(x)]                                (1)
-- Para ello, sea x ∈ R. Puesto que es a es una cota inferior de f, se
-- tiene que
--    a ≤ f(x)                                                       (2)
-- y, puesto que b es una cota inferior de g, se tiene que
--    b ≤ g(x)                                                       (3)
-- De (2) y (3), por add_le_add, se tiene que
--    a + b ≤ f(x) + g(x)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable {f g : ℝ → ℝ}
variable {a b : ℝ}

-- 1ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  have h1 : ∀ x, a + b ≤ f x + g x
  { intro x
    have h1a : a ≤ f x := hfa x
    have h1b : b ≤ g x := hgb x
    show a + b ≤ f x + g x
    exact add_le_add h1a h1b }
  show CotaInferior (f + g) (a + b)
  exact h1

-- 2ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  have h1 : ∀ x, a + b ≤ f x + g x
```

```
  { intro x
    show a + b ≤ f x + g x
    exact add_le_add (hfa x) (hgb x) }
  show CotaInferior (f + g) (a + b)
  exact h1

-- 3ª demostración
example
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
by
  intro x
  dsimp
  apply add_le_add
  . apply hfa
  . apply hgb

-- 4ª demostración
theorem sumaCotaInf
  (hfa : CotaInferior f a)
  (hgb : CotaInferior g b)
  : CotaInferior (f + g) (a + b) :=
λ x ↦ add_le_add (hfa x) (hgb x)

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.3. El producto de funciones no negativas es no negativo

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de dos funciones no negativas es no
-- negativa.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- =================================

-- Se usará el siguiente lema
--    mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b
--
-- Hay que demostrar que
--    (∀ x ∈ ℝ) [0 ≤ f(x) * g(x)]                                        (1)
-- Para ello, sea x ∈ R. Puesto que f es no negatica, se tiene que
--    0 ≤ f(x)                                                           (2)
-- y, puesto que g es no negativa, se tiene que
--    0 ≤ g(x)                                                           (3)
-- De (2) y (3), por mul_nonneg, se tiene que
--    0 ≤ f(x) * g(x)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable (f g : ℝ → ℝ)

-- 1ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x
  { intro x
    have h2: 0 ≤ f x := nnf x
    have h3: 0 ≤ g x := nng x
    show 0 ≤ f x * g x
    exact mul_nonneg h2 h3 }
  show CotaInferior (f * g) 0
  exact h1

-- 2ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
```

```
  : CotaInferior (f * g) 0 :=
by
  have h1 : ∀x, 0 ≤ f x * g x
  { intro x
    show 0 ≤ f x * g x
    exact mul_nonneg (nnf x) (nng x) }
  show CotaInferior (f * g) 0
  exact h1

-- 3ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
by
  intro x
  dsimp
  apply mul_nonneg
  . apply nnf
  . apply nng

-- 4ª demostración
example
  (nnf : CotaInferior f 0)
  (nng : CotaInferior g 0)
  : CotaInferior (f * g) 0 :=
λ x ↦ mul_nonneg (nnf x) (nng x)

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.4. Si a es una cota superior no negativa de f y b es es una cota superior de la función no negativa g, entonces ab es una cota superior de fg

```
-- ---------------------------------------------------------------------

-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de f, b es una cota superior
-- de g, a es no negativa y g es no negativa, entonces ab es una cota
-- superior de fg.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema
--    mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d
--
-- Hay que demostrar que
--    (∀ x ∈ ℝ) [f x * g x ≤ a * b]                                    (1)
-- Para ello, sea x ∈ R. Puesto que a es una cota superior de f, se tiene que
--    f(x) ≤ a                                                          (2)
-- puesto que b es una cota superior de g, se tiene que
--    g(x) ≤ b                                                          (3)
-- puesto que g es no negativa, se tiene que
--    0 ≤ g(x)                                                          (4)
-- y, puesto que a es no negativa, se tiene que
--    0 ≤ a                                                             (5)
-- De (2), (3), (4) y (5), por mul_le_mul, se tiene que
--    f x * g x ≤ a * b
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a
```

```
-- (CotaInferior f a) expresa que a es una cota inferior de f.
def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

variable (f g : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  have h1 : ∀ x, f x * g x ≤ a * b
  { intro x
    have h2 : f x ≤ a := hfa x
    have h3 : g x ≤ b := hgb x
    have h4 : 0 ≤ g x := nng x
    show f x * g x ≤ a * b
    exact mul_le_mul h2 h3 h4 nna }
  show CotaSuperior (f * g) (a * b)
  exact h1

-- 2ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  dsimp
  apply mul_le_mul
  . apply hfa
  . apply hgb
  . apply nng
  . apply nna

-- 3ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
```

```
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  have h1:= hfa x
  have h2:= hgb x
  have h3:= nng x
  exact mul_le_mul h1 h2 h3 nna

-- 4ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
by
  intro x
  specialize hfa x
  specialize hgb x
  specialize nng x
  exact mul_le_mul hfa hgb nng nna

-- 5ª demostración
example
  (hfa : CotaSuperior f a)
  (hgb : CotaSuperior g b)
  (nng : CotaInferior g 0)
  (nna : 0 ≤ a)
  : CotaSuperior (f * g) (a * b) :=
λ x ↦ mul_le_mul (hfa x) (hgb x) (nng x) nna

-- Lemas usados
-- ============

-- variable (c d : ℝ)
-- #check (mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.5. La suma de dos funciones acotadas superiormente también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones acotadas superiormente también
-- lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Del ejercicio ”La suma de una cota superior de f y una cota superior
-- de g es una cota superior de f+g” (que se encuentra en
-- https://bit.ly/3QauluK ) usaremos la definición de cota superior
-- (CotaSuperior) y el lema sumaCotaSup.
--
-- Puesto que f está acotada superiormente, tiene una cota superior. Sea
-- a una de dichas cotas. Análogamentte, puesto que g está acotada
-- superiormente, tiene una cota superior. Sea b una de dichas
-- cotas. Por el lema sumaCotaSup, a+b es una cota superior de f+g. or
-- consiguiente, f+g está acotada superiormente.

-- Demostraciones con Lean4
-- ========================

import src.Suma_de_cotas_superiores

variable {f g : ℝ → ℝ}

-- (acotadaSup f) afirma que f tiene cota superior.
def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

-- 1ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  cases' hg with b hb
  -- b : ℝ
```

```
  -- hb : CotaSuperior g b
  have h1 : CotaSuperior (f + g) (a + b) :=
    sumaCotaSup ha hb
  have h2 : ∃ z, CotaSuperior (f+g) z :=
    Exists.intro (a + b) h1
  show acotadaSup (f + g)
  exact h2

-- 2ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : FnUb f a
  cases' hg with b hb
  -- b : ℝ
  -- hb : FnUb g b
  use a + b
  apply sumaCotaSup ha hb

-- 4ª demostración
example
  (hf : acotadaSup f)
  (hg : acotadaSup g)
  : acotadaSup (f + g) :=
by
  rcases hf with ⟨a, ha⟩
  rcases hg with ⟨b, hb⟩
  exact ⟨a + b, sumaCotaSup ha hb⟩

-- 5ª demostración
example :
  acotadaSup f → acotadaSup g → acotadaSup (f + g) :=
by
  rintro ⟨a, ha⟩ ⟨b, hb⟩
  exact ⟨a + b, sumaCotaSup ha hb⟩

-- 6ª demostración
example :
  acotadaSup f → acotadaSup g → acotadaSup (f + g) :=
fun ⟨a, ha⟩ ⟨b, hb⟩ ↦ ⟨a + b, sumaCotaSup ha hb⟩
```

```
-- Lemas usados
-- ============

-- #check (sumaCotaSup : CotaSuperior f a → CotaSuperior g b → CotaSuperior (f + g) (a +
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.6. La suma de dos funciones acotadas inferiormente también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones acotadas inferiormente también
-- lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Del ejercicio ”La suma de una cota inferior de f y una cota inferior
-- de g es una cota inferior de f+g” usaremos la definición de cota
-- inferior (CotaInferior) y el lema sumaCotaInf.
--
-- Puesto que f está acotada inferiormente, tiene una cota inferior. Sea
-- a una de dichas cotas. Análogamentte, puesto que g está acotada
-- inferiormente, tiene una cota inferior. Sea b una de dichas
-- cotas. Por el lema FnLb_add, a+b es una cota inferior de f+g. Por
-- consiguiente, f+g está acotada inferiormente.

-- Demostraciones con Lean4
-- ========================

import src.Suma_de_cotas_inferiores
variable {f g : ℝ → ℝ}

-- (acotadaInf f) afirma que f tiene cota inferior.
def acotadaInf (f : ℝ → ℝ) :=
  ∃ a, CotaInferior f a

-- 1ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
```

```
  : acotadaInf (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaInferior f a
  cases' hg with b hb
  -- b : ℝ
  -- hb : CotaInferior g b
  have h1 : CotaInferior (f + g) (a + b) := sumaCotaInf ha hb
  have h2 : ∃ z, CotaInferior (f + g) z :=
    Exists.intro (a + b) h1
  show acotadaInf (f + g)
  exact h2

-- 2ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
  : acotadaInf (f + g) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : FnLb f a
  cases' hg with b hgb
  -- b : ℝ
  -- hgb : FnLb g b
  use a + b
  -- ⊢ FnLb (f + g) (a + b)
  apply sumaCotaInf ha hgb

-- 3ª demostración
example
  (hf : acotadaInf f)
  (hg : acotadaInf g)
  : acotadaInf (f + g) :=
by
  rcases hf with ⟨a, ha⟩
  -- a : ℝ
  -- ha : FnLb f a
  rcases hg with ⟨b, hb⟩
  -- b : ℝ
  -- hb : FnLb g b
  exact ⟨a + b, sumaCotaInf ha hb⟩

-- 4ª demostración
```

```
example :
  acotadaInf f → acotadaInf g → acotadaInf (f + g) :=
by
  rintro ⟨a, ha⟩ ⟨b, hb⟩
  -- a : ℝ
  -- ha : FnLb f a
  -- b : ℝ
  -- hb : FnLb g b
  exact ⟨a + b, sumaCotaInf ha hb⟩

-- 5ª demostración
example :
  acotadaInf f → acotadaInf g → acotadaInf (f + g) :=
fun ⟨a, ha⟩ ⟨b, hb⟩ ↦ ⟨a + b, sumaCotaInf ha hb⟩

-- Lemas usados
-- ============

-- #check (sumaCotaInf : FnLb f a → FnLb g b → FnLb (f + g) (a + b))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.7. Si a es una cota superior de f y c ≥ 0, entonces ca es una cota superior de cf

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de f y c ≥ 0,
-- entonces c * a es una cota superior de c * f.
-- ----------------------------------------------------------------------

-- Demostración en lenguaj natural
-- ===============================

-- Se usará el lema
--    {b ≤ c, 0 ≤ a} ⊢ ab ≤ ac                                        (L1)
--
-- Tenemos que demostrar que
--    (∀ y ∈ ℝ) cf(y) ≤ ca.
-- Sea y ∈ R. Puesto que a es una cota de f, se tiene que
--    f(y) ≤ a
-- que, junto con c ≥ 0, por el lema L1 nos da
--    cf(y) ≤ ca
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

-- (CotaSuperior f a) se verifica si a es una cota superior de f.
def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

variable {f : ℝ → ℝ}
variable {c : ℝ}

-- Demostraciones con Lean4
-- ========================

-- 1ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  -- y : ℝ
  -- ⊢ (fun x => c * f x) y ≤ c * a
  have ha : f y ≤ a := hfa y
  calc (fun x => c * f x) y
       = c * f y := by rfl
     _ ≤ c * a   := mul_le_mul_of_nonneg_left ha h

-- 2ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  calc (fun x => c * f x) y
       = c * f y := by rfl
     _ ≤ c * a   := mul_le_mul_of_nonneg_left (hfa y) h

-- 3ª demostración
example
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
```

```
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
by
  intro y
  show (fun x => c * f x) y ≤ c * a
  exact mul_le_mul_of_nonneg_left (hfa y) h

-- 4ª demostración
lemma CotaSuperior_mul
  (hfa : CotaSuperior f a)
  (h : c ≥ 0)
  : CotaSuperior (fun x ↦ c * f x) (c * a) :=
fun y ↦ mul_le_mul_of_nonneg_left (hfa y) h

-- Lemas usados
-- ============

-- variable (c : ℝ)
-- #check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.8. Si c ≥ 0 y f está acotada superiormente, entonces c·f también lo está

```
-- ---------------------------------------------------------------------
-- Demostrar que si c ≥ 0 y f está acotada superiormente, entonces c * f
-- también lo está.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos el siguiente lema:
--    CotaSuperior_mul : CotaSuperior f a → c ≥ 0 → CotaSuperior (fun x ↦ c * f x) (c *
--
-- Puesto que f está acotada superiormente, tiene una cota superior. Sea
-- a una de dichas cotas. Entonces, por el lema CotaSuperior_mul, ca es una cota
-- superior de cf. Por consiguiente, cf está acotada superiormente.

-- Demostraciones con Lean4
-- ========================
```

```
import src.Cota_superior_de_producto_por_escalar

variable {f : ℝ → ℝ}
variable {c : ℝ}

-- (acotadaSup f) afirma que f tiene cota superior.
def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

-- 1ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  have h1 : CotaSuperior (fun x ↦ c * f x) (c * a) :=
    CotaSuperior_mul ha hc
  have h2 : ∃ z, ∀ x, (fun x ↦ c * f x) x ≤ z :=
    Exists.intro (c * a) h1
  show acotadaSup (fun x ↦ c * f x)
  exact h2

-- 2ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  cases' hf with a ha
  -- a : ℝ
  -- ha : CotaSuperior f a
  use c * a
  -- ⊢ CotaSuperior (fun x => c * f x) (c * a)
  apply CotaSuperior_mul ha hc

-- 3ª demostración
example
  (hf : acotadaSup f)
  (hc : c ≥ 0)
  : acotadaSup (fun x ↦ c * f x) :=
by
  rcases hf with ⟨a, ha⟩
```

```
  -- a : ℝ
  -- ha : CotaSuperior f a
  exact ⟨c * a, CotaSuperior_mul ha hc⟩

-- 4ª demostración
example
  (hc : c ≥ 0)
  : acotadaSup f → acotadaSup (fun x ↦ c * f x) :=
by
  rintro ⟨a, ha⟩
  -- a : ℝ
  -- ha : CotaSuperior f a
  exact ⟨c * a, CotaSuperior_mul ha hc⟩

-- 5ª demostración
example
  (hc : c ≥ 0)
  : acotadaSup f → acotadaSup (fun x ↦ c * f x) :=
fun ⟨a, ha⟩ ↦ ⟨c * a, CotaSuperior_mul ha hc⟩

-- Lemas usados
-- ============

-- #check (CotaSuperior_mul : CotaSuperior f a → c ≥ 0 → CotaSuperior (fun x ↦ c * f x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.9. Si para cada a existe un x tal que f(x) >a, entonces f no tiene cota superior

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función de ℝ en ℝ tal que para cada a,
-- existe un x tal que f x > a, entonces f no tiene cota superior.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f tiene cota superior. Sea b una de dichas cotas
-- superiores. Por la hipótesis, existe un x tal que f(x) > b. Además,
-- como b es una cota superior de f, f(x) ≤ b que contradice la
-- desigualdad anterior.
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

def acotadaSup (f : ℝ → ℝ) : Prop :=
  ∃ a, CotaSuperior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
example
  (h : ∀ a, ∃ x, f x > a)
  : ¬ acotadaSup f :=
by
  intros hf
  -- hf : acotadaSup f
  -- ⊢ False
  cases' hf with b hb
  -- b : ℝ
  -- hb : CotaSuperior f b
  cases' h b with x hx
  -- x : ℝ
  -- hx : f x > b
  have : f x ≤ b := hb x
  linarith

-- 2ª demostración
theorem sinCotaSup
  (h : ∀ a, ∃ x, f x > a)
  : ¬ acotadaSup f :=
by
  intros hf
  -- hf : acotadaSup f
  -- ⊢ False
  rcases hf with ⟨b, hb : CotaSuperior f b⟩
  rcases h b with ⟨x, hx : f x > b⟩
  have : f x ≤ b := hb x
  linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.10. Si para cada a existe un x tal que f(x) <a, entonces f no tiene cota inferior

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función de ℝ en ℝ tal que para cada a,
-- existe un x tal que f x < a, entonces f no tiene cota inferior.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f tiene cota inferior. Sea b una de dichas cotas
-- inferiores. Por la hipótesis, existe un x tal que f(x) < b. Además,
-- como b es una cota inferior de f, b ≤ f(x) que contradice la
-- desigualdad anterior.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

def CotaInferior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, a ≤ f x

def acotadaInf (f : ℝ → ℝ) : Prop :=
  ∃ a, CotaInferior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ acotadaInf f :=
by
  intros hf
  -- hf : acotadaInf f
  -- ⊢ False
  cases' hf with b hb
  -- b : ℝ
  -- hb : CotaInferior f b
  cases' h b with x hx
  -- x : ℝ
  -- hx : f x < b
  have : b ≤ f x := hb x
```

```
  linarith

-- 2ª demostración
example
  (h : ∀ a, ∃ x, f x < a)
  : ¬ acotadaInf f :=
by
  intros hf
  -- hf : acotadaInf f
  -- ⊢ False
  rcases hf with ⟨b, hb : CotaInferior f b⟩
  rcases h b with ⟨x, hx : f x < b⟩
  have : b ≤ f x := hb x
  linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.11. La función identidad no está acotada superiormente

```
-- ---------------------------------------------------------------------
-- Demostrar que la función identidad no está acotada superiormente.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usamos el lema de ejercicio anterior (que afirma que si para cada a,
-- existe un x tal que f x > a, entonces f no tiene cota superior) basta
-- demostrar que
--    (∀a ∈ ℝ)(∃x ∈ ℝ) [x > a]
-- Sea a ∈ ℝ. Entonces a + 1 > a y, por tanto, (∃x ∈ ℝ) [x > a].

-- Demostraciones con Lean4
-- ========================

import src.Funcion_no_acotada_superiormente

-- 1ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
```

```
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  intro a
  -- a : ℝ
  -- ⊢ ∃ x, x > a
  use a + 1
  -- ⊢ a + 1 > a
  linarith

-- 2ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  intro a
  -- a : ℝ
  -- ⊢ ∃ x, x > a
  exact ⟨a + 1, by linarith⟩

-- 3ª demostración
example : ¬ acotadaSup (fun x ↦ x) :=
by
  apply sinCotaSup
  -- ⊢ ∀ (a : ℝ), ∃ x, x > a
  exact fun a ↦ ⟨a + 1, by linarith⟩
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.12. Si f no está acotada superiormente, entonces (∀a)(∃x)[f(x) >a]

```
-- ---------------------------------------------------------------------
-- Sea f una función de ℝ en ℝ. Demostrar que si f no está acotada
-- superiormente, entonces (∀a)(∃x)[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 CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ¬acotadaSup 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
    have h3a : ¬ f x > a := h2 x
    show f x ≤ a
    exact le_of_not_gt h3a
  have h4 : CotaSuperior f a := h3
  have h5 : ∃ b, CotaSuperior f b := ⟨a, h4⟩
  have h6 : acotadaSup f := h5
  show False
  exact h h6

-- 2ª demostración
-- ===============

example
  (h : ¬acotadaSup 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
  -- ⊢ acotadaSup f
  use a
  -- ⊢ CotaSuperior 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
  exact h2

-- 3ª demostración
-- ===============

example
  (h : ¬acotadaSup f)
  : ∀ a, ∃ x, f x > a :=
by
  unfold acotadaSup at h
  -- h : ¬∃ a, CotaSuperior f a
  unfold CotaSuperior 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 : ¬acotadaSup f)
  : ∀ a, ∃ x, f x > a :=
by
  simp only [acotadaSup, CotaSuperior] 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 : ¬acotadaSup f) :
  ∀ a, ∃ x, f x > a :=
by
  contrapose h
  -- h : ¬∀ (a : ℝ), ∃ x, f x > a
  -- ⊢ ¬¬acotadaSup f
  push_neg at *
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
```

```
  -- ⊢ acotadaSup f
  exact h

-- 6ª demostración
-- ===============

example
  (h : ¬acotadaSup f) :
  ∀ a, ∃ x, f x > a :=
by
  contrapose! h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  -- ⊢ acotadaSup f
  exact h

-- Lemas usados
-- ============

-- variable {α : Type _}
-- variable (P : α → Prop)
-- #check (forall_not_of_not_exists : (¬∃ x, P x) → ∀ x, ¬P x)
--
-- variable (a b : ℝ)
-- #check (le_of_not_gt : ¬a > b → a ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.13. Si ¬(∀a)(∃x)[f(x) >a], entonces f está acotada superiormente

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic

def CotaSuperior (f : ℝ → ℝ) (a : ℝ) : Prop :=
  ∀ x, f x ≤ a

def acotadaSup (f : ℝ → ℝ) :=
  ∃ a, CotaSuperior f a

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ¬∀ a, ∃ x, f x > a)
  : acotadaSup f :=
by
  unfold acotadaSup
  -- ⊢ ∃ a, CotaSuperior f a
  unfold CotaSuperior
  -- ⊢ ∃ 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)
  : acotadaSup f :=
by
  unfold acotadaSup CotaSuperior
  -- ⊢ ∃ 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)
  : acotadaSup f :=
by
  push_neg at h
  -- h : ∃ a, ∀ (x : ℝ), f x ≤ a
  exact h
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.14. Suma de funciones monótonas

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones monótonas es monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el siguiente lema:
--    add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
--
-- Supongamos que f y g son monótonas y teneno que demostrar que f+g
-- también lo es; que
--    ∀ a b, a ≤ b → (f + g)(a) ≤ (f + g)(b)
-- Sean a, b ∈ ℝ tales que
--    a ≤ b                                                         (1)
-- Entonces, por ser f y g monótonas se tiene
--    f(a) ≤ f(b)                                                   (2)
--    g(a) ≤ g(b)                                                   (3)
-- Entonces,
--    (f + g)(a) = f(a) + g(a)
--               ≤ f(b) + g(b)    [por add_le_add, (2) y (3)]
--               = (f + g)(b)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- 1ª demostración
```

```
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    have h2 : f a ≤ f b := mf hab
    have h3 : g a ≤ g b := mg hab
    calc (f + g) a
          = f a + g a := rfl
        _ ≤ f b + g b := add_le_add h2 h3
        _ = (f + g) b := rfl }
  show Monotone (f + g)
  exact h1

-- 2ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    calc (f + g) a
          = f a + g a := rfl
        _ ≤ f b + g b := add_le_add (mf hab) (mg hab)
        _ = (f + g) b := rfl }
  show Monotone (f + g)
  exact h1

-- 3ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  have h1 : ∀ a b, a ≤ b → (f + g) a ≤ (f + g) b
  { intros a b hab
    show (f + g) a ≤ (f + g) b
    exact add_le_add (mf hab) (mg hab) }
  show Monotone (f + g)
  exact h1

-- 4ª demostración
```

```
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  apply add_le_add
  . -- f a ≤ f b
    apply mf hab
  . --  g a ≤ g b
    apply mg hab

-- 5ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f + g) :=
λ _ _ hab ↦ add_le_add (mf hab) (mg hab)

-- Lemas usados
-- ============

-- variable (a b c d : ℝ)
-- #check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.15. Si c es no negativo y f es monótona, entonces cf es monótona.

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es no negativo y f es monótona, entonces cf es
-- monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el Lema
--    mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c
```

```
--
-- Tenemos que demostrar que
--    (∀ a, b ∈ ℝ) [a ≤ b → (cf)(a) ≤ (cf)(b)]
-- Sean a, b ∈ ℝ tales que a ≤ b. Puesto que f es monótona, se tiene
--    f(a) ≤ f(b).
-- y, junto con la hipótesis de que c es no negativo, usando el lema
-- mul_le_mul_of_nonneg_left, se tiene que
--    cf(a) ≤ cf(b)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f : ℝ → ℝ)
variable {c : ℝ}

-- 1ª demostración
example
  (mf : Monotone f)
  (nnc : 0 ≤ c)
  : Monotone (fun x ↦ c * f x) :=
by
  have h1 : ∀ a b, a ≤ b → (fun x ↦ c * f x) a ≤ (fun x ↦ c * f x) b
  { intros a b hab
    have h2 : f a ≤ f b := mf hab
    show (fun x ↦ c * f x) a ≤ (fun x ↦ c * f x) b
    exact mul_le_mul_of_nonneg_left h2 nnc }
  show Monotone (fun x ↦ c * f x)
  exact h1

-- 2ª demostración
example
  (mf : Monotone f)
  (nnc : 0 ≤ c)
  : Monotone (fun x ↦ c * f x) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  -- (fun x => c * f x) a ≤ (fun x => c * f x) b
  apply mul_le_mul_of_nonneg_left
  . -- f a ≤ f b
    apply mf hab
```

```
  . -- 0 ≤ c
    apply nnc

-- 3ª demostración
example (mf : Monotone f) (nnc : 0 ≤ c) :
  Monotone (fun x ↦ c * f x) :=
λ _ _ hab ↦ mul_le_mul_of_nonneg_left (mf hab) nnc

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.16. La composición de dos funciones monótonas es monótona

```
-- ---------------------------------------------------------------------
-- Demostrar que la composición de dos funciones monótonas es monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sean f y g dos funciones monótonas de ℝ en ℝ. Tenemos que demostrar
-- que f ∘ g es monótona; es decir, que
--    (∀ a, b ∈ ℝ) [a ≤ b → (f ∘ g)(a) ≤ (f ∘ g)(b)]
-- Sean a, b ∈ ℝ tales que a ≤ b. Por ser g monótona, se tiene
--    g(a) ≤ g(b)
-- y, por ser f monótona, se tiene
--    f(g(a)) ≤ f(g(b))
-- Finalmente, por la definición de composición,
--    (f ∘ g)(a) ≤ (f ∘ g)(b)
-- que es lo que había que demostrar.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
```

```
variable (f g : ℝ → ℝ)

-- 1ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  have h1 : ∀ a b, a ≤ b → (f ∘ g) a ≤ (f ∘ g) b
  { intros a b hab
    have h1 : g a ≤ g b := mg hab
    show (f ∘ g) a ≤ (f ∘ g) b
    exact mf h1 }
  show Monotone (f ∘ g)
  exact h1

-- 2ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  have h1 : ∀ a b, a ≤ b → (f ∘ g) a ≤ (f ∘ g) b
  { intros a b hab
    show (f ∘ g) a ≤ (f ∘ g) b
    exact mf (mg hab) }
  show Monotone (f ∘ g)
  exact h1

-- 3ª demostración
example
  (mf : Monotone f)
  (mg : Monotone g)
  : Monotone (f ∘ g) :=
by
  -- a b : ℝ
  -- hab : a ≤ b
  intros a b hab
  -- (f ∘ g) a ≤ (f ∘ g) b
  apply mf
  -- g a ≤ g b
  apply mg
  -- a ≤ b
  apply hab
```

```
-- 4ª demostración
example (mf : Monotone f) (mg : Monotone g) :
  Monotone (f ∘ g) :=
λ _ _ hab ↦ mf (mg hab)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.17. Si f es monótona y f(a) <f(b), entonces a <b

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es monótona y f(a) < f(b), entonces a < b
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos los lemas
--    a ≱ b → a < b                                                   (L1)
--    a ≥ b → a ≮ b                                                   (L2)
--
-- Usando el lema L1, basta demostrar que a ≱ b. Lo haremos por
-- reducción al absurdo. Para ello, supongamos que a ≥ b. Como f es
-- monótona, se tiene que f(a) ≥ f(b) y, aplicando el lema L2,
-- f(a) ≮ f(b), que contradice a la hipótesis.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable (f : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
```

```
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  have h4 : f a ≥ f b := h1 h3
  have h5 : ¬ f a < f b := not_lt_of_ge h4
  exact h5 h2

-- 2ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  have h5 : ¬ f a < f b := not_lt_of_ge (h1 h3)
  exact h5 h2

-- 3ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  intro h3
  -- h3 : a ≥ b
  -- ⊢ False
  exact (not_lt_of_ge (h1 h3)) h2

-- 4ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
```

```
  : a < b :=
by
  apply lt_of_not_ge
  -- ⊢ ¬a ≥ b
  exact fun h3 ↦ (not_lt_of_ge (h1 h3)) h2

-- 5ª demostración
-- ===============

example
  (h1 : Monotone f)
  (h2 : f a < f b)
  : a < b :=
lt_of_not_ge (fun h3 ↦ (not_lt_of_ge (h1 h3)) h2)

-- Lemas usados
-- ============

-- #check (lt_of_not_ge : ¬ a ≥ b → a < b)
-- #check (not_lt_of_ge : a ≥ b → ¬ a < b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.18. Si a, b ∈ ℝ tales que a ≤ b y f(b) <f(a), entonces f no es monótona

```
-- ---------------------------------------------------------------------
-- Demostrar que si a, b ∈ ℝ tales que (a ≤ b) y (f b < f a), entonces f
-- no es monótona.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Usaremos el lema
--    a ≥ b → a ≮ b                                           (L1)
--
-- Lo demostraremos por reducción al absurdo. Para ello, supongamos que
-- f es monótona. Entonces, como a ≤ b, se tiene f(a) ≤ f(b) y, por el
-- lema L1, f b ≮ f a, en contradicción con la hipótesis.

-- Demostraciones con Lean4
```

```
-- ========================

import Mathlib.Data.Real.Basic
variable (f : ℝ → ℝ)
variable (a b : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
  -- ⊢ False
  have h4 : f a ≤ f b := h3 h1
  have h5 : ¬(f b < f a) := not_lt_of_ge h4
  exact h5 h2

-- 2ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
  -- ⊢ False
  have h5 : ¬(f b < f a) := not_lt_of_ge (h3 h1)
  exact h5 h2

-- 3ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
by
  intro h3
  -- h3 : Monotone f
```

```
  -- ⊢ False
  exact (not_lt_of_ge (h3 h1)) h2

-- 4ª demostración
-- ===============

example
  (h1 : a ≤ b)
  (h2 : f b < f a)
  : ¬ Monotone f :=
fun h3 ↦ (not_lt_of_ge (h3 h1)) h2

-- Lemas usados
-- ============

-- #check (not_lt_of_ge : a ≥ b → ¬a < b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.19. No para toda f : ℝ → ℝ monótona, (∀a, b)[f(a) ≤ f(b) → a ≤ b]

```
-- ---------------------------------------------------------------------
-- Demostrar que no para toda f : ℝ → ℝ monótona,
--    (∀ a b)[f(a) ≤ f(b) → a ≤ b]
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que
--    (∀f)[f es monótona → (∀a, b)[f(a) ≤ f(b) → a ≤ b]]            (1)
-- Sea f : ℝ → ℝ la función constante igual a cero (es decir,
--    (∀x ∈ ℝ)[f(x) = 0]
-- Entonces, f es monótona y f(1) ≤ f(0) (ya que
-- f(1) = 0 ≤ 0 = f(0)). Luego, por (1), 1 ≤ 0 que es una
-- contradicción.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
```

```
-- 1ª demostración
-- ===============

example :
  ¬∀ {f : ℝ → ℝ}, Monotone f → ∀ {a b}, f a ≤ f b → a ≤ b :=
by
  intro h1
  -- h1 : ∀ {f : ℝ → ℝ}, Monotone f → ∀ {a b : ℝ}, f a ≤ f b → a ≤ b
  -- ⊢ False
  let f := fun _ : ℝ ↦ (0 : ℝ)
  have h2 : Monotone f := monotone_const
  have h3 : f 1 ≤ f 0 := le_refl 0
  have h4 : 1 ≤ 0 := h1 h2 h3
  linarith

-- Lemas usados
-- ============

-- variable (a c : ℝ)
-- #check (le_refl a : a ≤ a)
-- #check (monotone_const : Monotone fun _ : ℝ ↦ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.20. Si f no es monótona, entonces ∃x∃y[x ≤ y ∧ f(y) <f(x)]

```
-- ---------------------------------------------------------------------
-- 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 : ¬∀ y, a ≤ y → f a ≤ f y⟩
  have h3 : ∃ y, ¬(a ≤ y → f a ≤ f y) := not_forall.mp ha
  rcases h3 with ⟨b, hb : ¬(a ≤ b → f a ≤ f b)⟩
  have h4 : a ≤ b ∧ ¬(f a ≤ f b) := 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
  exact h5

-- 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 (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
-- #check (not_imp : ¬(p → q) ↔ p ∧ ¬q)
-- #check (lt_of_not_le : ¬b ≤ a → a < b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.21. f: ℝ → ℝ no es monótona syss (∃x,y)(x ≤ y ∧ f(x) >f(y))

```
-- ---------------------------------------------------------------------
-- Demostrar que f : ℝ → ℝ no es 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
-- ========================

import Mathlib.Data.Real.Basic
variable {f : ℝ → ℝ}

-- 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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.22. La función x ↦ -x no es monótona creciente

```
-- ---------------------------------------------------------------------
-- 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
-- ======================

import Mathlib.Data.Real.Basic
import src.CNS_de_no_monotona

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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.23. La suma de dos funciones pares es par

```
-- ---------------------------------------------------------------------
-- Demostrar que la suma de dos funciones pares es par.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Supongamos que f y g son funciones pares. Tenemos que demostrar que
-- f+g es par; es decir, que
--    (∀ x ∈ ℝ) (f + g)(x) = (f + g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f + g) x = f x + g x
--              = f (-x) + g x    [porque f es par]
--              = f (-x) + g (-x) [porque g es par]
--              = (f + g) (-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- 1ª demostración
-- ===============

example
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
  intro x
  have h1 : f x = f (-x) := h1 x
  have h2 : g x = g (-x) := h2 x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g x    := congrArg (. + g x) h1
     _ = f (-x) + g (-x) := congrArg (f (-x) + .) h2
     _ = (f + g) (-x)    := rfl

-- 2ª demostración
-- ===============

example
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
```

```
  intro x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g x    := congrArg (. + g x) (h1 x)
     _ = f (-x) + g (-x) := congrArg (f (-x) + .) (h2 x)
     _ = (f + g) (-x)    := rfl

-- 3ª demostración
-- ===============

example
  (h1 : esPar f)
  (h2 : esPar g)
  : esPar (f + g) :=
by
  intro x
  calc (f + g) x
       = f x + g x       := rfl
     _ = f (-x) + g (-x) := by rw [h1, h2]
     _ = (f + g) (-x)    := rfl
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.24. El producto de dos funciones impares es par

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de dos funciones impares es par.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f y g son funciones impares. Tenemos que demostrar que
-- f·g es par; es decir, que
--    (∀ x ∈ ℝ) (f·g)(x) = (f·g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f·g) x = f(x)g(x)
--            = (-f(-x))g(x)      [porque f es impar]
--            = (-f(-x)(-g(-x))   [porque g es impar]
--            = f(-x)g(-x))
--            = (f·g)(-x)
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  have h1 : f x = -f (-x) := h1 x
  have h2 : g x = -g (-x) := h2 x
  calc (f * g) x
       = f x * g x              := rfl
     _ = (-f (-x)) * g x        := congrArg (. * g x) h1
     _ = (-f (-x)) * (-g (-x)) := congrArg ((-f (-x)) * .) h2
     _ = f (-x) * g (-x)        := neg_mul_neg (f (-x)) (g (-x))
     _ = (f * g) (-x)           := rfl

-- 2ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x              := rfl
     _ = (-f (-x)) * g x        := congrArg (. * g x) (h1 x)
     _ = (-f (-x)) * (-g (-x)) := congrArg ((-f (-x)) * .) (h2 x)
     _ = f (-x) * g (-x)        := neg_mul_neg (f (-x)) (g (-x))
     _ = (f * g) (-x)           := rfl
```

```
-- 3ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x         := rfl
     _ = -f (-x) * -g (-x) := by rw [h1, h2]
     _ = f (-x) * g (-x)   := by rw [neg_mul_neg]
     _ = (f * g) (-x)      := rfl

-- 4ª demostración
example
  (h1 : esImpar f)
  (h2 : esImpar g)
  : esPar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x       := rfl
     _ = f (-x) * g (-x) := by rw [h1, h2, neg_mul_neg]
     _ = (f * g) (-x)    := rfl

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (neg_mul_neg a b : -a * -b = a * b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.25. El producto de una función par por una impar es impar

```
-- ---------------------------------------------------------------------
-- Demostrar que el producto de una función par por una impar es impar.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Supongamos que f es una función par y g lo es impar. Tenemos que
-- demostrar que f·g es imppar; es decir, que
--    (∀ x ∈ ℝ) (f·g)(x) = -(f·g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f·g) x = f(x)g(x)
--            = f(-x)g(x)       [porque f es par]
--            = f(-x)(-g(-x))   [porque g es impar]
--            = -f(-x)g(-x))
--            = -(f·g)(-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  have h1 : f x = f (-x) := h1 x
  have h2 : g x = -g (-x) := h2 x
  calc (f * g) x
       = f x * g x             := rfl
     _ = (f (-x)) * g x        := congrArg (. * g x) h1
     _ = (f (-x)) * (-g (-x)) := congrArg (f (-x) * .) h2
     _ = -(f (-x) * g (-x))    := mul_neg (f (-x)) (g (-x))
     _ = -(f * g) (-x)         := rfl

-- 2ª demostración
example
  (h1 : esPar f)
```

```
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x            := rfl
     _ = f (-x) * -g (-x)     := by rw [h1, h2]
     _ = -(f (-x) * g (-x))   := by rw [mul_neg]
     _ = -(f * g) (-x)        := rfl

-- 3ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esImpar (f * g) :=
by
  intro x
  calc (f * g) x
       = f x * g x            := rfl
     _ = -(f (-x) * g (-x))   := by rw [h1, h2, mul_neg]
     _ = -((f * g) (-x))      := rfl

-- Lemas usados
-- ===========

-- variable (a b : ℝ)
-- #check (mul_neg a b : a * -b = -(a * b))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.26. Si f es par y g es impar, entonces (f ∘ g) es par

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es par y g es impar, entonces f ∘ g es par.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f es una función par y g lo es impar. Tenemos que
-- demostrar que (f ∘ g) es par; es decir, que
```

```
--    (∀ x ∈ ℝ) (f ∘ g)(x) = (f ∘ g)(-x)
-- Sea x ∈ ℝ. Entonces,
--    (f ∘ g)(x) = f(g(x))
--               = f(-g(-x))    [porque g es impar]
--               = f(g(-x))     [porque f es par]
--               = (f ∘ g)(-x)

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

-- (esPar f) expresa que f es par.
def esPar (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = f (-x)

-- (esImpar f) expresa que f es impar.
def esImpar  (f : ℝ → ℝ) : Prop :=
  ∀ x, f x = - f (-x)

-- 1ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (-g (-x))  := congr_arg f (h2 x)
     _ = f (g (-x))   := (h1 (g (-x))).symm
     _ = (f ∘ g) (-x) := rfl

-- 2ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
       = f (g x)      := rfl
     _ = f (-g (-x))  := by rw [h2]
```

```
      _ = f (g (-x))   := by rw [← h1]
      _ = (f ∘ g) (-x) := rfl

-- 3ª demostración
example
  (h1 : esPar f)
  (h2 : esImpar g)
  : esPar (f ∘ g) :=
by
  intro x
  calc (f ∘ g) x
         = f (g x)      := rfl
       _ = f (g (-x))   := by rw [h2, ← h1]
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.27. Para cualquier conjunto s, s ⊆ s

```
-- ---------------------------------------------------------------------
-- Demostrar que para cualquier conjunto s, s ⊆ s.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x) [x ∈ s → x ∈ s]
-- Sea x tal que
--    x ∈ s                                                         (1)
-- Entonces, por (1), se tiene que
--    x ∈ s
-- que es lo que teníamos que demostrar.

-- Demostraciones con Lean 4
-- =========================

import Mathlib.Tactic

variable {α : Type _}
variable (s : Set α)

-- 1ª demostración
example : s ⊆ s :=
```

```
by
  intro x xs
  exact xs

-- 2ª demostración
example : s ⊆ s :=
  fun (x : α) (xs : x ∈ s) ↦ xs

-- 3ª demostración
example : s ⊆ s :=
  fun _ xs ↦ xs

-- 4ª demostración
example : s ⊆ s :=
  -- by exact?
  rfl.subset

-- 5ª demostración
example : s ⊆ s :=
by rfl
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.28. Las funciones f(x,y) = (x + y)² y g(x,y) = x² + 2xy + y² son iguales

```
-- ---------------------------------------------------------------------
-- 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

-- 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 }
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

14.29. La composición de una función creciente y una decreciente es decreciente

```
-- ---------------------------------------------------------------------
-- Sea una función f de ℝ en ℝ. Se dice que f es creciente si para todo
-- x e y tales que x ≤ y se tiene que f(x) ≤ f(y). Se dice que f es
-- decreciente si para todo x e y tales que x ≤ y se tiene que
-- f(x) ≥ f(y).
--
-- Demostrar con Lean4 que si f es creciente y g es decreciente,
-- entonces (g ∘ f) es decreciente.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sean x, y ∈ ℝ tales que x ≤ y. Entonces, por ser f creciente,
--    f(x) ≥ f(y)
-- y, por ser g decreciente,
--    g(f(x)) ≤ g(f(y)).
-- Por tanto,
--    (g ∘ f)(x) ≤ (g ∘ f)(y).

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (f g : ℝ → ℝ)

def creciente (f : ℝ → ℝ) : Prop :=
```

```
  ∀ {x y}, x ≤ y → f x ≤ f y

def decreciente (f : ℝ → ℝ) : Prop :=
  ∀ {x y}, x ≤ y → f x ≥ f y

-- 1ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  intro x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  have h1 : f x ≤ f y := hf h
  show (g ∘ f) x ≥ (g ∘ f) y
  exact hg h1

-- 2ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  intro x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  show (g ∘ f) x ≥ (g ∘ f) y
  exact hg (hf h)

-- 3ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
fun {_ _} h ↦ hg (hf h)
```

```
-- 4ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  intros x y hxy
  calc (g ∘ f) x
       = g (f x)   := rfl
     _ ≥ g (f y)   := hg (hf hxy)
     _ = (g ∘ f) y := rfl

-- 5ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  unfold creciente decreciente at *
  -- hf : ∀ {x y : ℝ}, x ≤ y → f x ≤ f y
  -- hg : ∀ {x y : ℝ}, x ≤ y → g x ≥ g y
  -- ⊢ ∀ {x y : ℝ}, x ≤ y → (g ∘ f) x ≥ (g ∘ f) y
  intros x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  unfold Function.comp
  -- ⊢ g (f x) ≥ g (f y)
  apply hg
  -- ⊢ f x ≤ f y
  apply hf
  -- ⊢ x ≤ y
  exact h

-- 6ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
```

```
by
  intros x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  apply hg
  -- ⊢ f x ≤ f y
  apply hf
  -- ⊢ x ≤ y
  exact h

-- 7ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  intros x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  apply hg
  -- ⊢ f x ≤ f y
  exact hf h

-- 8ª demostración
-- ===============

example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by
  intros x y h
  -- x y : ℝ
  -- h : x ≤ y
  -- ⊢ (g ∘ f) x ≥ (g ∘ f) y
  exact hg (hf h)

-- 9ª demostración
-- ===============

example
```

```
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
by tauto
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

14.30. Si una función es creciente e involutiva, entonces es la identidad

```
-- ---------------------------------------------------------------------
-- Sea una función f de ℝ en ℝ.
-- + Se dice que f es creciente si para todo x e y tales que x ≤ y se
--   tiene que f(x) ≤ f(y).
-- + Se dice que f es involutiva si para todo x se tiene que f(f(x)) = x.
--
-- En Lean4 que f sea creciente se representa por ‘Monotone f‘ y que sea
-- involutiva por ‘Involutive f‘
--
-- Demostrar con Lean4 que si f es creciente e involutiva, entonces f es
-- la identidad.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que para todo x ∈ ℝ, f(x) = x. Sea x ∈ ℝ.
-- Entonces, por ser f involutiva, se tiene que
--    f(f(x)) = x                                                  (1)
-- Además, por las propiedades del orden, se tiene que f(x) ≤ x ó
-- x ≤ f(x). Demostraremos que f(x) = x en los dos casos.
--
-- Caso 1: Supongamos que
--    f(x) ≤ x                                                     (2)
-- Entonces, por ser f creciente, se tiene que
--    f(f(x)) ≤ f(x)                                               (3)
-- Sustituyendo (1) en (3), se tiene
--    x ≤ f(x)
-- que junto con (1) da
--    f(x) = x
--
-- Caso 2: Supongamos que
```

```
--    x ≤ f(x)                                                         (4)
-- Entonces, por ser f creciente, se tiene que
--    f(x) ≤ f(f(x))                                                   (5)
-- Sustituyendo (1) en (5), se tiene
--    f(x) ≤ x
-- que junto con (4) da
--    f(x) = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function

variable (f : ℝ → ℝ)

-- 1ª demostración
example
  (hc : Monotone f)
  (hi : Involutive f)
  : f = id :=
by
  funext x
  -- x : ℝ
  -- ⊢ f x = id x
  have h : f (f x) = x := hi x
  cases' (le_total (f x) x) with h1 h2
  . -- h1 : f x ≤ x
    have h1a : f (f x) ≤ f x := hc h1
    have h1b : x ≤ f x := by rwa [h] at h1a
    show f x = x
    exact antisymm h1 h1b
  . -- h2 : x ≤ f x
    have h2a : f x ≤ f (f x) := hc h2
    have h2b : f x ≤ x := by rwa [h] at h2a
    show f x = x
    exact antisymm h2b h2

-- 2ª demostración
example
  (hc : Monotone f)
  (hi : Involutive f)
  : f = id :=
by
  unfold Monotone Involutive at *
```

```
  -- hc : ∀ ⦃a b : ℝ⦄, a ≤ b → f a ≤ f b
  -- hi : ∀ (x : ℝ), f (f x) = x
  funext x
  -- x : ℝ
  -- ⊢ f x = id x
  unfold id
  -- ⊢ f x = x
  cases' (le_total (f x) x) with h1 h2
  . -- h1 : f x ≤ x
    apply antisymm h1
    -- ⊢ x ≤ f x
    have h3 : f (f x) ≤ f x := by
      apply hc
      -- ⊢ f x ≤ x
      exact h1
    rwa [hi] at h3
  . -- h2 : x ≤ f x
    apply antisymm _ h2
    -- ⊢ f x ≤ x
    have h4 : f x ≤ f (f x) := by
      apply hc
      -- ⊢ x ≤ f x
      exact h2
    rwa [hi] at h4

-- 3ª demostración
example
  (hc : Monotone f)
  (hi : Involutive f)
  : f = id :=
by
  funext x
  -- x : ℝ
  -- ⊢ f x = id x
  cases' (le_total (f x) x) with h1 h2
  . -- h1 : f x ≤ x
    apply antisymm h1
    -- ⊢ x ≤ f x
    have h3 : f (f x) ≤ f x := hc h1
    rwa [hi] at h3
  . -- h2 : x ≤ f x
    apply antisymm _ h2
    -- ⊢ f x ≤ x
    have h4 : f x ≤ f (f x) := hc h2
    rwa [hi] at h4
```

```
-- 4ª demostración
example
  (hc : Monotone f)
  (hi : Involutive f)
  : f = id :=
by
  funext x
  -- x : ℝ
  -- ⊢ f x = id x
  cases' (le_total (f x) x) with h1 h2
  . -- h1 : f x ≤ x
    apply antisymm h1
    -- ⊢ x ≤ f x
    calc x
         = f (f x) := (hi x).symm
       _ ≤ f x     := hc h1
  . -- h2 : x ≤ f x
    apply antisymm _ h2
    -- ⊢ f x ≤ x
    calc f x
         ≤ f (f x) := hc h2
       _ = x       := hi x

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (le_total a b : a ≤ b ∨ b ≤ a)
-- #check (antisymm : a ≤ b → b ≤ a → a = b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

14.31. Si ‘f(x) ≤ f(y) → x ≤ y‘, entonces f es inyectiva

```
-- ---------------------------------------------------------------------
-- Sea f una función de ℝ en ℝ tal que
--    ∀ x y, f(x) ≤ f(y) → x ≤ y
-- Demostrar que f es inyectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- ================================

-- Sean x, y ∈ ℝ tales que
--    f(x) = f(y)                                                 (1)
-- Tenemos que demostrar que x = y.
--
-- De (1), tenemos que
--    f(x) ≤ f(y)
-- y, por la hipótesis,
--    x ≤ y                                                       (2)
--
-- También de (1), tenemos que
--    f(y) ≤ f(x)
-- y, por la hipótesis,
--    y ≤ x                                                       (3)
-- De (2) y (3), tenemos que
--    x = y

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function

variable (f : ℝ → ℝ)

-- 1ª demostración
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
by
  intros x y hxy
  -- x y : ℝ
  -- hxy : f x = f y
  -- ⊢ x = y
  have h1 : f x ≤ f y := le_of_eq hxy
  have h2 : x ≤ y     := h h1
  have h3 : f y ≤ f x := ge_of_eq hxy
  have h4 : y ≤ x     := h h3
  show x = y
  exact le_antisymm h2 h4

-- 2ª demostración
```

```
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
by
  intros x y hxy
  -- x y : ℝ
  -- hxy : f x = f y
  -- ⊢ x = y
  have h1 : x ≤ y     := h (le_of_eq hxy)
  have h2 : y ≤ x     := h (ge_of_eq hxy)
  show x = y
  exact le_antisymm h1 h2

-- 3ª demostración
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
by
  intros x y hxy
  -- x y : ℝ
  -- hxy : f x = f y
  -- ⊢ x = y
  show x = y
  exact le_antisymm (h (le_of_eq hxy)) (h (ge_of_eq hxy))

-- 3ª demostración
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
fun _ _ hxy ↦ le_antisymm (h hxy.le) (h hxy.ge)

-- 4ª demostración
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
by
  intros x y hxy
```

```
  -- x y : ℝ
  -- hxy : f x = f y
  -- ⊢ x = y
  apply le_antisymm
  . -- ⊢ x ≤ y
    apply h
    -- ⊢ f x ≤ f y
    exact le_of_eq hxy
  . -- ⊢ y ≤ x
    apply h
    -- ⊢ f y ≤ f x
    exact ge_of_eq hxy

-- 5ª demostración
-- ===============

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : Injective f :=
by
  intros x y hxy
  -- x y : ℝ
  -- hxy : f x = f y
  -- ⊢ x = y
  apply le_antisymm
  . -- ⊢ x ≤ y
    exact h (le_of_eq hxy)
  . -- ⊢ y ≤ x
    exact h (ge_of_eq hxy)

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (ge_of_eq : a = b → a ≥ b)
-- #check (le_antisymm : a ≤ b → b ≤ a → a = b)
-- #check (le_of_eq : a = b → a ≤ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

14.32. Las funciones con inversa por la izquierda son inyectivas

```
-- ---------------------------------------------------------------------
-- En Lean, que g es una inversa por la izquierda de f está definido por
--    LeftInverse (g : β → α) (f : α → β) : Prop :=
--       ∀ x, g (f x) = x
-- y que f tenga inversa por la izquierda está definido por
--    HasLeftInverse (f : α → β) : Prop :=
--       ∃ finv : β → α, LeftInverse finv f
-- Finalmente, que f es inyectiva está definido por
--    Injective (f : α → β) : Prop :=
--       ∀ ⦃x y⦄, f x = f y → x = y
--
-- Demostrar que si f tiene inversa por la izquierda, entonces f es
-- inyectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea f: A → B que tiene inversa por la izquierda. Entonces, existe
-- una g: B → A tal que
--    (∀ x ∈ A)[g(f(x)) = x]                                         (1)
-- Para demostrar que f es inyectiva, sean x, y ∈ A tales que
--    f(x) = f(y)
-- Entonces,
--    g(f(x)) = g(f(y))
-- y, por (1),
--    x = y

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
open Function

universe u v
variable {α : Type u}
variable {β : Type v}
variable {f : α → β}

-- 1ª demostración
-- ===============
```

```
example
  (hf : HasLeftInverse f)
  : Injective f :=
by
  intros x y hxy
  -- x y : α
  -- hxy : f x = f y
  -- ⊢ x = y
  unfold HasLeftInverse at hf
  -- hf : ∃ finv, LeftInverse finv f
  unfold LeftInverse at hf
  -- hf : ∃ finv, ∀ (x : α), finv (f x) = x
  cases' hf with g hg
  -- g : β → α
  -- hg :
  calc x = g (f x) := (hg x).symm
       _ = g (f y) := congr_arg g hxy
       _ = y       := hg y

-- 2ª demostración
-- ===============

example
  (hf : HasLeftInverse f)
  : Injective f :=
by
  intros x y hxy
  -- x y : α
  -- hxy : f x = f y
  -- ⊢ x = y
  cases' hf with g hg
  -- g : β → α
  -- hg : LeftInverse g f
  calc x = g (f x) := (hg x).symm
       _ = g (f y) := congr_arg g hxy
       _ = y       := hg y

-- 3ª demostración
-- ===============

example
  (hf : HasLeftInverse f)
  : Injective f :=
by
```

```
  apply Exists.elim hf
  -- ⊢ ∀ (a : β → α), LeftInverse a f → Injective f
  intro g hg
  -- g : β → α
  -- hg : LeftInverse g f
  -- ⊢ Injective f
  exact LeftInverse.injective hg

-- 4ª demostración
-- ===============

example
  (hf : HasLeftInverse f)
  : Injective f :=
Exists.elim hf (fun _g hg ↦ LeftInverse.injective hg)

-- 5ª demostración
-- ===============

example
  (hf : HasLeftInverse f)
  : Injective f :=
HasLeftInverse.injective hf

-- Lemas usados
-- ============

-- variable (x y : α)
-- variable (p : α → Prop)
-- variable (b : Prop)
-- variable (g : β → α)
-- #check (Exists.elim : (∃ x, p x) → (∀ x, p x → b) → b)
-- #check (HasLeftInverse.injective : HasLeftInverse f → Injective f)
-- #check (LeftInverse.injective : LeftInverse g f → Injective f)
-- #check (congr_arg f : x = y → f x = f y)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

14.33. Si g ∘ f es inyectiva, entonces f es inyectiva

```
-- ---------------------------------------------------------------------
-- Sean f: X → Y y g: Y → Z. Demostrar que si g ∘ f es inyectiva,
-- entonces f es inyectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sean a, b ∈ X tales que
--    f(a) = f(b)
-- Entonces,
--    g(f(a)) = g(f(b))
-- Luego
--    (g ∘ f)(a) = (g ∘ f)(b)
-- y, como g ∘ f es inyectiva,
--    a = b

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Function

variable {X Y Z : Type}
variable {f : X → Y}
variable {g : Y → Z}

-- 1ª demostración
-- ===============

example
  (h : Injective (g ∘ f))
  : Injective f :=
by
  intro a b hab
  -- a b : X
  -- hab : f a = f b
  -- ⊢ a = b
  have h1 : (g ∘ f) a = (g ∘ f) b := by simp_all only [comp_apply]
  exact h h1
```

```
-- 2ª demostración
-- ===============

example
  (h : Injective (g ∘ f))
  : Injective f :=
by
  intro a b hab
  -- a b : X
  -- hab : f a = f b
  -- ⊢ a = b
  apply h
  -- ⊢ (g ∘ f) a = (g ∘ f) b
  change g (f a) = g (f b)
  -- ⊢ g (f a) = g (f b)
  rw [hab]

-- Lemas usados
-- ============

-- variable (x : X)
-- #check (Function.comp_apply : (g ∘ f) x = g (f x))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 15

Teoría de conjuntos

15.1. Si r ⊆ s y s ⊆ t, entonces r ⊆ t

```
-- ---------------------------------------------------------------------
-- Demostrar que si r ⊆ s y s ⊆ t, entonces r ⊆ t.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural (LN)
-- =====================================

-- 1ª demostración en LN
-- ---------------------

-- Tenemos que demostrar que
--    (∀ x) [x ∈ r → x ∈ t]
-- Sea x tal que
--    x ∈ r.
-- Puesto que r ⊆ s, se tiene que
--    x ∈ s
-- y, puesto que s ⊆ t, se tiene que
--    x ∈ t
-- que es lo que teníamos que demostrar.

-- 2ª demostración en LN
-- ---------------------

-- Tenemos que demostrar que
--    (∀ x) [x ∈ r → x ∈ t]
-- Sea x tal que
--    x ∈ r
-- Tenemos que demostrar que
```

```
--    x ∈ t
-- que, puesto que s ⊆ t, se reduce a
--    x ∈ s
-- que, puesto que r ⊆ s, se redece a
--    x ∈ r
-- que es lo que hemos supuesto.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Set

variable {α : Type _}
variable (r s t : Set α)

-- 1ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by
  intros x xr
  -- xr : x ∈ r
  have xs : x ∈ s := rs xr
  show x ∈ t
  exact st xs

-- 2ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by
  intros x xr
  -- x : α
  -- xr : x ∈ r
  apply st
  -- ⊢ x ∈ s
  apply rs
  -- ⊢ x ∈ r
  exact xr

-- 3ª demostración
```

```
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
fun _ xr ↦ st (rs xr)

-- 4ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
-- by exact?
Subset.trans rs st

-- 5ª demostración
example
  (rs : r ⊆ s)
  (st : s ⊆ t)
  : r ⊆ t :=
by tauto

-- Lemas usados
-- ============

-- #check (Subset.trans : r ⊆ s → s ⊆ t → r ⊆ t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.2. Si a es una cota superior de s y a ≤ b, entonces b es una cota superior de s

```
-- ---------------------------------------------------------------------
-- Demostrar que si a es una cota superior de s y a ≤ b, entonces b es
-- una cota superior de s.
-- ----------------------------------------------------------------------

import Mathlib.Tactic

variable {α : Type _} [PartialOrder α]
variable (s : Set α)
variable (a b : α)
```

```
-- (CotaSupConj s a) afirma que a es una cota superior del conjunto s.
def CotaSupConj (s : Set α) (a : α) :=
  ∀ {x}, x ∈ s → x ≤ a

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x) [x ∈ s → x ≤ b]
-- Sea x tal que x ∈ s. Entonces,
--    x ≤ a   [porque a es una cota superior de s]
--      ≤ b
-- Por tanto, x ≤ b.

-- 1ª demostración
example
  (h1 : CotaSupConj s a)
  (h2 : a ≤ b)
  : CotaSupConj s b :=
by
  intro x (xs : x ∈ s)
  have h3 : x ≤ a := h1 xs
  show x ≤ b
  exact le_trans h3 h2

-- 2ª demostración
example
  (h1 : CotaSupConj s a)
  (h2 : a ≤ b)
  : CotaSupConj s b :=
by
  intro x (xs : x ∈ s)
  calc x ≤ a := h1 xs
       _ ≤ b := h2
-
-- Lemas usados
-- ============

-- variable (c : α)
-- #check (le_trans : a ≤ b → b ≤ c → a ≤ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.3. Si s ⊆ t, entonces s ∩ u ⊆ t ∩ u

```
-- ---------------------------------------------------------------------
-- Demostrar que 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 (inter_subset_inter_left u : s ⊆ t → s ∩ u ⊆ t ∩ u)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.4. s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

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) ∪ (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]
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.5. $(s \setminus t) \setminus u \subseteq s \setminus (t \cup u)$

```
-- ---------------------------------------------------------------------
-- 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
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]

-- Lema usado
-- ==========

-- #check (diff_diff : (s \ t) \ u = s \ (t ∪ u))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.6. (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
open Set

variable {α : Type}
variable (s t u : Set α)
```

```
-- 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_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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.7. $s \setminus (t \cup u) \subseteq (s \setminus t) \setminus u$

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
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
  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]

-- Lema usado
-- ==========

-- #check (diff_diff : (s \ t) \ u = s \ (t ∪ u))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.8. s ∩ t = t ∩ s

```
-- ---------------------------------------------------------------------
-- 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.9. s ∩ (s ∪ t) = s

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    s ∩ (s ∪ t) = s
-- ----------------------------------------------------------------------

-- Demostación 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic
open Set

variable {α : Type}
variable (s t : Set α)

-- 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

-- Lemas usados
-- ============

-- variable (a b : Prop)
-- #check (And.left : a ∧ b → a)
-- #check (Or.inl : a → a ∨ b)
-- #check (inf_sup_self : s ∩ (s ∪ t) = s)
-- #check (subset_antisymm : s ⊆ t → t ⊆ s → s = t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.10. s ∪ (s ∩ t) = s

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
open Set

variable {α : Type}
variable (s t : Set α)

-- 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

-- Lemas usados
-- ============

-- variable (a b c : Prop)
-- #check (And.left : a ∧ b → a)
-- #check (Or.elim : a ∨ b → (a → c) → (b → c) → c)
-- #check (sup_inf_self : s ∪ (s ∩ t) = s)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.11. (s \ t) ∪ t = s ∪ t

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
open Set

variable {α : Type}
variable (s t : Set α)

-- 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_iff (x ∈ s ∨ x ∈ t)
    _ ↔ 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

-- Lemas usados
-- ============

-- variable (a b c : Prop)
-- variable (x : α)
-- #check (and_or_right : (a ∧ b) ∨ c ↔ (a ∨ c) ∧ (b ∨ c))
-- #check (and_true_iff a : a ∧ True ↔ a)
-- #check (diff_union_self : (s \ t) ∪ t = s ∪ t)
-- #check (em' a : ¬a ∨ a)
-- #check (mem_diff x : x ∈ s \ t ↔ x ∈ s ∧ x ∉ t)
-- #check (mem_union x s t : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.12. (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
open Set

variable {α : Type}
variable (s t : Set α)

-- 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_iff (x ∈ s ∨ x ∈ t),
                      true_and_iff (¬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 [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 (mem_union x s t : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
-- #check (mem_diff x : x ∈ s \ t ↔ x ∈ s ∧ ¬x ∈ t)
-- #check (and_or_right : (a ∧ b) ∨ c ↔ (a ∨ c) ∧ (b ∨ c))
-- #check (or_and_left : a ∨ (b ∧ c) ↔ (a ∨ b) ∧ (a ∨ c))
-- #check (em a : a ∨ ¬ a)
-- #check (em' a : ¬ a ∨ a)
-- #check (and_true_iff a : a ∧ True ↔ a)
-- #check (true_and_iff a : True ∧ a ↔ a)
-- #check (or_comm : a ∨ b ↔ b ∨ a)
-- #check (not_and_or : ¬(a ∧ b) ↔ ¬a ∨ ¬b)
-- #check (mem_inter_iff x s t : x ∈ s ∩ t ↔ x ∈ s ∧ x ∈ t)
-- #check (ext_iff : s = t ↔ ∀ (x : α), x ∈ s ↔ x ∈ t)
-- #check (iff_def : (a ↔ b) ↔ (a → b) ∧ (b → a))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.13. Pares ∪ Impares = Naturales

```
-- ---------------------------------------------------------------------
-- Los conjuntos de los números naturales, de los pares y de los impares
-- se definen por
--    def Naturales : Set ℕ := {n | True}
--    def Pares     : Set ℕ := {n | Even n}
--    def Impares   : Set ℕ := {n | ¬Even n}
--
-- 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
```

```
-- ========================

import Mathlib.Data.Nat.Parity
open Set

def Naturales : Set ℕ := {n | True}
def Pares     : Set ℕ := {n | Even n}
def Impares   : Set ℕ := {n | ¬Even n}

-- 1ª demostración
-- ===============

example : Pares ∪ Impares = Naturales :=
by
  unfold Pares Impares Naturales
  -- ⊢ {n | Even n} ∪ {n | ¬Even n} = {n | True}
  ext n
  -- ⊢ n ∈ {n | Even n} ∪ {n | ¬Even n} ↔ n ∈ {n | True}
  simp
  -- ⊢ Even 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 | Even n} ∪ {n | ¬Even n} ↔ n ∈ {n | True}
  tauto
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.14. Los primos mayores que 2 son impares

```
-- ---------------------------------------------------------------------
-- 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.Data.Nat.Parity
import Mathlib.Data.Nat.Prime
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 → ¬Even n
  intro hn
  -- hn : Nat.Prime n
  -- ⊢ 2 < n → ¬Even 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
    -- ⊢ ¬Even 2
    exfalso
    exact absurd h1 (lt_irrefl 2)
  . -- h : n % 2 = 1
    rw [even_iff]
    -- ⊢ 2 < n → ¬n % 2 = 0
    rw [h]
```

```
    -- ⊢ 2 < n → ¬1 = 0
    intro
    -- a : 2 < n
    -- ⊢ ¬1 = 0
    exact one_ne_zero

-- 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
    rw [even_iff]
    -- ⊢ ¬n % 2 = 0
    rw [h4]
    -- ⊢ ¬1 = 0
    exact one_ne_zero

-- 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
    rw [even_iff]
    -- ⊢ ¬n % 2 = 0
    linarith

-- 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 (one_ne_zero : 1 ≠ 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.15. s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Data.Set.Lattice
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 : ℕ), A i
     ↔ x ∈ s ∧ x ∈ ⋃ (i : ℕ), A i :=
        by simp only [mem_inter_iff]
   _ ↔ x ∈ s ∧ (∃ i : ℕ, x ∈ A i) :=
        by simp only [mem_iUnion]
   _ ↔ ∃ i : ℕ, x ∈ s ∧ x ∈ A i :=
        by simp only [exists_and_left]
   _ ↔ ∃ i : ℕ, x ∈ A i ∧ x ∈ s :=
        by simp only [and_comm]
   _ ↔ ∃ i : ℕ, x ∈ A i ∩ s :=
        by simp only [mem_inter_iff]
   _ ↔ x ∈ ⋃ (i : ℕ), 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 : ℕ), A i ↔ x ∈ ⋃ (i : ℕ), A i ∩ s
  constructor
  . -- ⊢ x ∈ s ∩ ⋃ (i : ℕ), A i → x ∈ ⋃ (i : ℕ), A i ∩ s
    intro h
    -- h : x ∈ s ∩ ⋃ (i : ℕ), A i
    -- ⊢ x ∈ ⋃ (i : ℕ), A i ∩ s
    rw [mem_iUnion]
    -- ⊢ ∃ i, x ∈ A i ∩ s
    rcases h with ⟨xs, xUAi⟩
    -- xs : x ∈ s
    -- xUAi : x ∈ ⋃ (i : ℕ), A i
    rw [mem_iUnion] at xUAi
    -- xUAi : ∃ i, x ∈ A i
    rcases xUAi with ⟨i, xAi⟩
    -- i : ℕ
    -- xAi : x ∈ A i
    use i
    -- ⊢ x ∈ A i ∩ s
    constructor
    . -- ⊢ x ∈ A i
      exact xAi
    . -- ⊢ x ∈ s
      exact xs
  . -- ⊢ x ∈ ⋃ (i : ℕ), A i ∩ s → x ∈ s ∩ ⋃ (i : ℕ), A i
    intro h
    -- h : x ∈ ⋃ (i : ℕ), A i ∩ s
    -- ⊢ x ∈ s ∩ ⋃ (i : ℕ), A i
    rw [mem_iUnion] at h
    -- h : ∃ i, x ∈ A i ∩ s
    rcases h with ⟨i, hi⟩
    -- i : ℕ
    -- hi : x ∈ A i ∩ s
    rcases hi with ⟨xAi, xs⟩
    -- xAi : x ∈ A i
    -- xs : x ∈ s
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ x ∈ ⋃ (i : ℕ), A i
      rw [mem_iUnion]
      -- ⊢ ∃ i, x ∈ A i
```

```
      use i
      -- ⊢ x ∈ A i
      exact xAi

-- 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
  -- ⊢ (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 : ℕ
    -- 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 : ℕ
    -- 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 : ℕ), A i ↔ x ∈ ⋃ (i : ℕ), A i ∩ s
  aesop

-- 4ª demostración
-- ===============

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by ext; aesop
```

```
-- Lemas usados
-- ============

-- variable (x : α)
-- variable (t : Set α)
-- variable (a b : Prop)
-- variable (p : α → Prop)
-- #check (mem_iUnion : x ∈ ⋃ i, A i ↔ ∃ i, x ∈ A i)
-- #check (mem_inter_iff x s t : x ∈ s ∩ t ↔ x ∈ s ∧ x ∈ t)
-- #check (exists_and_left : (∃ (x : α), b ∧ p x) ↔ b ∧ ∃ (x : α), p x)
-- #check (and_comm : a ∧ b ↔ b ∧ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.16. (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α : Type}
```

```
variable (A B : ℕ → Set α)

-- 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)
-- #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_inter_iff x a b : x ∈ a ∩ b ↔ x ∈ a ∧ x ∈ b)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.17. s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s)

```
-- ---------------------------------------------------------------------
-- 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 a b : x ∈ s ∪ t ↔ x ∈ s ∨ x ∈ t)
-- #check (or_comm : a ∨ b ↔ b ∨ a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.18. $f^{-1}[u \cap v] = f^{-1}[u] \cap f^{-1}[v]$

```
-- ---------------------------------------------------------------------
-- En Lean, la imagen inversa de un conjunto s (de elementos de tipo β)
-- por la función f (de tipo α → β) es el conjunto ‘f ⁻¹' s‘ de
-- elementos x (de tipo α) tales que ‘f x ∈ s‘.
--
-- 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_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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.19. f[s ∪ t] = f[s] ∪ f[t]

```
-- ---------------------------------------------------------------------
-- En Lean, la imagen de un conjunto s por una función f se representa
-- por ‘f '' s‘; es decir,
--    f '' s = {y | ∃ x, x ∈ s ∧ f x = y}
--
-- 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.20. s ⊆ f⁻¹[f[s]]

```
-- ---------------------------------------------------------------------
-- 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))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.21. f[s] ⊆ u ↔ s ⊆ f⁻¹[u]

```
-- ---------------------------------------------------------------------
-- 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)
-- #check (image_subset_iff : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u)
-- #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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.22. La función (x ↦ x + c) es inyectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que, para todo c la función
--    f(x) = x + c
-- es inyectiva
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Se usará el lema
--    (∀ a, b, c) [a + b = c + b → a = c]                              (L1)
-- Hay que demostrar que
--    (∀ x₁ x₂) [f(x₁) = f(x₂) → x₁ = x₂]
-- Sean x₁, x₂ tales que f(x₁) = f(x₂). Entonces,
--    x₁ + c = x₂ + c
-- y, por L1, x₁ = x₂.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function

variable {c : ℝ}

-- 1ª demostración
example : Injective ((. + c)) :=
by
  intro (x1 : ℝ) (x2 : ℝ) (h1 : x1 + c = x2 + c)
  show x1 = x2
  exact add_right_cancel h1

-- 2ª demostración
example : Injective ((. + c)) :=
by
  intro x1 x2 h1
  show x1 = x2
  exact add_right_cancel h1

-- 3ª demostración
example : Injective ((. + c)) :=
  fun _ _ h ↦ add_right_cancel h

-- Lemas usados
-- ============

-- variable {a b : ℝ}
-- #check (add_right_cancel : a + b = c + b → a = c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.23. Si c ≠ 0, entonces la función (x ↦ cx) es inyectiva

```
-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que para todo c distinto de cero la función
--    f(x) = c * x
-- es inyectiva
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se usará el lema
--    (∀ a, b, c) [a ≠ 0 → (a * b = a * c ↔ b = c))]             (L1)
-- Hay que demostrar que
--    (∀ x₁, x₂) [f(x₁) = f(x₂) → x₁ = x₂]
-- Sean x₁, x₂ tales que f(x₁) = f(x₂). Entonces,
--    cx₁ = cx₂
-- y, por L1 y puesto que c ≠ 0, se tiene que
--    x₁ = x₂.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
open Function
variable {c : ℝ}

-- 1ª demostración
example
  (h : c ≠ 0)
  : Injective ((c * .)) :=
by
  intro (x1 : ℝ) (x2 : ℝ) (h1 : c * x1 = c * x2)
  show x1 = x2
  exact (mul_right_inj' h).mp h1

-- 2ª demostración
example
  (h : c ≠ 0)
  : Injective ((c * .)) :=
fun _ _ h1 ↦ mul_left_cancel₀ h h1

-- Lemas usados
```

```
-- ============

-- variable (a b : ℝ)
-- #check (mul_right_inj' : a ≠ 0 → (a * b = a * c ↔ b = c))
-- #check (mul_left_cancel₀ : a ≠ 0 → a * b = a * c → b = c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.24. La composición de funciones inyectivas es inyectiva

```
-- Demostraciones en lenguaje natural (LN)
-- =======================================

-- 1ª demostración en LN
-- =====================

-- Tenemos que demostrar que
--    (∀ x, y) [(g ∘ f)(x) = (g ∘ f)(y) → x = y]
-- Sean x, y tales que
--    (g ∘ f)(x) = (g ∘ f)(y)
-- Entonces, por la definición de la composición,
--    g(f(x)) = g(f(y))
-- y, ser g inyectiva,
--    f(x) = f(y)
-- y, ser f inyectiva,
--    x = y

-- 2ª demostración en LN
-- =====================

-- Tenemos que demostrar que
--    (∀ x, y) [(g ∘ f)(x) = (g ∘ f)(y) → x = y]
-- Sean x, y tales que
--    (g ∘ f)(x) = (g ∘ f)(y)                                     (1)
-- y tenemos que demostrar que
--    x = y                                                       (2)
-- El objetivo (2), usando que f es inyectiva, se reduce a
--    f(x) = f(y)
-- que, usando que g es inyectiva, se reduce a
--    g(f(x)) = g(f(y))
-- que, por la definición de la composición, coincide con (1).
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Function

variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}

-- 1ª demostración (basada en la 1ª en LN)
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro (x : α) (y : α) (h1: (g ∘ f) x = (g ∘ f) y)
  have h2: g (f x) = g (f y) := h1
  have h3: f x = f y := hg h2
  show x = y
  exact hf h3

-- 2ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro (x : α) (y : α) (h1: (g ∘ f) x = (g ∘ f) y)
  have h2: f x = f y := hg h1
  show x = y
  exact hf h2

-- 3ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intro x y h
  exact hf (hg h)

-- 4ª demostración
example
```

```
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
fun _ _ h ↦ hf (hg h)

-- 5ª demostración (basada en la 2ª en LN)
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by
  intros x y h
  -- x y : α
  -- h : (g ∘ f) x = (g ∘ f) y
  apply hf
  -- ⊢ f x = f y
  apply hg
  -- ⊢ g (f x) = g (f y)
  apply h

-- 6ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
-- by exact?
Injective.comp hg hf

-- 7ª demostración
example
  (hg : Injective g)
  (hf : Injective f) :
  Injective (g ∘ f) :=
by tauto

-- Lemas usados
-- ============

-- #check (Injective.comp : Injective g → Injective f → Injective (g ∘ f))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.25. La función (x ↦ x + c) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que para todo número real c, la función
--    f(x) = x + c
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    (∀ x ∈ ℝ)(∃ y ∈ ℝ)[y+c = x]
-- Sea x ∈ ℝ. Entonces, y = x-c ∈ ℝ y
--    y + c = (x - c) + c
--          = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable {c : ℝ}

open Function

-- 1ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  dsimp
  -- ⊢ (x - c) + c = x
  exact sub_add_cancel x c

-- 2ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
```

```
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  change (x - c) + c = x
  -- ⊢ (x - c) + c = x
  exact sub_add_cancel x c

-- 3ª demostración
example : Surjective (fun x ↦ x + c) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => x + c) a = x
  use x - c
  -- ⊢ (fun x => x + c) (x - c) = x
  exact sub_add_cancel x c

-- 4ª demostración
example : Surjective (fun x ↦ x + c) :=
fun x ↦ ⟨x - c, sub_add_cancel x c⟩

-- 5ª demostración
example : Surjective (fun x ↦ x + c) :=
fun x ↦ ⟨x - c, by ring⟩

-- 6ª demostración
example : Surjective (fun x ↦ x + c) :=
add_right_surjective c

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (sub_add_cancel a b : (a - b) + b = a)
-- #check (add_right_surjective c : Surjective (fun x ↦ x + c))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.26. Si c ≠ 0, entonces la función (x ↦ cx) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es un número real no nulo, entonces la función
```

```
--    f(x) = c * x
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Hay que demostrar que
--    (∀ x ∈ ℝ)(∃ y ∈ ℝ)[cy = x]
-- Sea x ∈ ℝ. Entonces, y = x/c ∈ R y
--    cy = c(x/c)
--       = y

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable {c : ℝ}
open Function

-- 1ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x) a = x
  use (x / c)
  -- ⊢ (fun x => c * x) (x / c) = x
  dsimp
  -- ⊢ c * (x / c) = x
  rw [mul_comm]
  -- ⊢ (x / c) * c = x
  exact div_mul_cancel x h

-- 2ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x) a = x
  use (x / c)
```

```
  -- ⊢ (fun x => c * x) (x / c) = x
  exact mul_div_cancel' x h

-- 3ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
fun x ↦ ⟨x / c, mul_div_cancel' x h⟩

-- 4ª demostración
example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x) :=
mul_left_surjective₀ h

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (div_mul_cancel a : b ≠ 0 → (a / b) * b = a)
-- #check (mul_comm a b : a * b = b * a)
-- #check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
-- #check (mul_left_surjective₀ : c ≠ 0 → Surjective (fun x ↦ c * x))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.27. Si c ≠ 0, entonces la función (x ↦ cx + d) es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que si c es un número real no nulo, entonces la función
--    f(x) = c * x + d
-- es suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Hay que demostrar que
--    (∀x ∈ ℝ)(∃y ∈ ℝ)[cy+d = x]
-- Sea x ∈ ℝ. Entonces, y = (x-d)/c ∈ R y
--    cy = c((x-d)/c)+d
```

```
--       = (x-d)+d
--       = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable {c d : ℝ}
open Function

-- 1ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  dsimp
  -- ⊢ c * ((x - d) / c) + d = x
  show c * ((x - d) / c) + d = x
  calc c * ((x - d) / c) + d
         = (x - d) + d := congrArg (. + d) (mul_div_cancel' (x - d) h)
       _ = x           := sub_add_cancel x d

-- 2ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  dsimp
  -- ⊢ c * ((x - d) / c) + d = x
  simp [mul_div_cancel', h]
```

```
-- 3ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
by
  intro x
  -- x : ℝ
  -- ⊢ ∃ a, (fun x => c * x + d) a = x
  use ((x - d) / c)
  -- ⊢ (fun x => c * x + d) ((x - d) / c) = x
  simp [mul_div_cancel', h]

-- 4ª demostración
-- ===============

example
  (h : c ≠ 0)
  : Surjective (fun x ↦ c * x + d) :=
fun x ↦ ⟨(x - d) / c, by simp [mul_div_cancel', h]⟩

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
-- #check (sub_add_cancel a b : a - b + b = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.28. Si f: ℝ → ℝ es suprayectiva, entonces ∃x ∈ ℝ tal que f(x)² = 9

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es una función suprayectiva de ℝ en ℝ,
-- entonces existe un x tal que (f x)^2 = 9.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================
```

```
-- Al ser f suprayectiva, existe un y tal que f(y) = 3. Por tanto,
-- f(y)² = 9.

-- Demostración con Lean9
-- ======================

import Mathlib.Data.Real.Basic

open Function

example
  {f : ℝ → ℝ}
  (h : Surjective f)
  : ∃ x, (f x)^2 = 9 :=
by
  cases' h 3 with y hy
  -- y : ℝ
  -- hy : f y = 3
  use y
  -- ⊢ f y ^ 2 = 9
  rw [hy]
  -- ⊢ 3 ^ 2 = 9
  norm_num
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.29. La composición de funciones suprayectivas es suprayectiva

```
-- ---------------------------------------------------------------------
-- Demostrar que la composición de funciones suprayectivas es
-- suprayectiva.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos que f : A → B y g : B → C son suprayectivas. Tenemos que
-- demostrar que
--    (∀z ∈ C)(∃x ∈ A)[g(f(x)) = z]
-- Sea z ∈ C. Por ser g suprayectiva, existe un y ∈ B tal que
--    g(y) = z                                                       (1)
```

```
-- Por ser f suprayectiva, existe un x ∈ A tal que
--     f(x) = y                                                    (2)
-- Por tanto,
--     g(f(x)) = g(y)   [por (2)]
--             = z      [por (1)]

-- Demostraciones con lean4
-- ========================

import Mathlib.Tactic
open Function
variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}

-- 1ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  use x
  -- ⊢ (g ∘ f) x = z
  dsimp
  -- ⊢ g (f x) = z
  rw [hx]
  -- ⊢ g y = z
  exact hy

-- 2ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
```

```
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  use x
  -- ⊢ (g ∘ f) x = z
  dsimp
  -- ⊢ g (f x) = z
  rw [hx, hy]

-- 3ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  cases' hg z with y hy
  -- y : β
  -- hy : g y = z
  cases' hf y with x hx
  -- x : α
  -- hx : f x = y
  exact ⟨x, by dsimp ; rw [hx, hy]⟩

-- 4ª demostración
example
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
by
  intro z
  -- z : γ
  -- ⊢ ∃ a, (g ∘ f) a = z
  rcases hg z with ⟨y, hy : g y = z⟩
  rcases hf y with ⟨x, hx : f x = y⟩
  exact ⟨x, by dsimp ; rw [hx, hy]⟩

-- 5ª demostración
example
```

```
  (hg : Surjective g)
  (hf : Surjective f)
  : Surjective (g ∘ f) :=
Surjective.comp hg hf

-- Lemas usados
-- ============

-- #check (Surjective.comp : Surjective g → Surjective f → Surjective (g ∘ f))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

15.30. Si f es inyectiva, entonces f⁻¹[f[s]] ⊆ s

```
-- ---------------------------------------------------------------------
-- 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

open Set Function
```

```
variable {α β : Type _}
variable (f : α → β)
variable (s : Set α)

-- 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

-- Lemas usados
-- ============
```

```
-- variable (x : α)
-- variable (y : β)
-- variable (t : Set β)
-- #check (mem_image f s y : y ∈ f '' s ↔ ∃ (x : α), x ∈ s ∧ f x = y)
-- #check (mem_preimage : x ∈ f ⁻¹' t ↔ f x ∈ t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.31. f[f⁻¹[u]] ⊆ u

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
open Set

variable {α β : Type _}
variable (f : α → β)
variable (u : Set β)

-- 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

-- Lemas usados
-- ============

-- #check (image_preimage_subset f u : f '' (f⁻¹' u) ⊆ u)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.32. Si f es suprayectiva, entonces u ⊆ f[f⁻¹[u]]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
open Set Function
variable {α β : Type _}
variable (f : α → β)
variable (u : Set β)

-- 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

-- Lemas usados
-- ============

-- variable (x : α)
-- #check (mem_preimage : x ∈ f ⁻¹' u ↔ f x ∈ u)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.33. Si s ⊆ t, entonces f[s] ⊆ f[t]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set

variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)
```

```
-- 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

-- Lemas usados
-- ============

-- variable (y : β)
-- #check (mem_image f s y : y ∈ f '' s ↔ ∃ x, x ∈ s ∧ f x = y)
-- #check (image_subset f : s ⊆ t → f '' s ⊆ f '' t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.34. Si u ⊆ v, entonces f⁻¹[u] ⊆ f⁻¹[v]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
open Set

variable {α β : Type _}
variable (f : α → β)
variable (u v : Set β)
```

```
-- 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

-- Lemas usados
-- ============

-- variable (a : α)
-- #check (mem_preimage : a ∈ f ⁻¹' u ↔ f a ∈ u)
-- #check (preimage_mono : u ⊆ v → f ⁻¹' u ⊆ f ⁻¹' v)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.35. $f^{-1}[A \cup B] = f^{-1}[A] \cup f^{-1}[B]$

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que, para todo x,
--    x ∈ f⁻¹[A ∪ B] ↔ x ∈ f⁻¹[A] ∪ f⁻¹[B]
-- Lo haremos demostrando las dos implicaciones.
--
-- (⟹) Supongamos que x ∈ f⁻¹[A ∪ B]. Entonces, f(x) ∈ A ∪ B.
-- Distinguimos dos casos:
--
-- Caso 1: Supongamos que f(x) ∈ A. Entonces, x ∈ f⁻¹[A] y, por tanto,
-- x ∈ f⁻¹[A] ∪ f⁻¹[B].
--
-- Caso 2: Supongamos que f(x) ∈ B. Entonces, x ∈ f⁻¹[B] y, por tanto,
-- x ∈ f⁻¹[A] ∪ f⁻¹[B].
--
-- (⟸) Supongamos que x ∈ f⁻¹[A] ∪ f⁻¹[B]. Distinguimos dos casos.
--
-- Caso 1: Supongamos que x ∈ f⁻¹[A]. Entonces, f(x) ∈ A y, por tanto,
-- f(x) ∈ A ∪ B. Luego, x ∈ f⁻¹[A ∪ B].
--
```

```
-- Caso 2: Supongamos que x ∈ f⁻¹[B]. Entonces, f(x) ∈ B y, por tanto,
-- f(x) ∈ A ∪ B. Luego, x ∈ f⁻¹[A ∪ B].

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Set.Function

open Set

variable {α β : Type _}
variable (f : α → β)
variable (A B : Set β)

-- 1ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (A ∪ B) ↔ x ∈ f ⁻¹' A ∪ f ⁻¹' B
  constructor
  . -- ⊢ x ∈ f ⁻¹' (A ∪ B) → x ∈ f ⁻¹' A ∪ f ⁻¹' B
    intro h
    -- h : x ∈ f ⁻¹' (A ∪ B)
    -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B
    rw [mem_preimage] at h
    -- h : f x ∈ A ∪ B
    rcases h with fxA | fxB
    . -- fxA : f x ∈ A
      left
      -- ⊢ x ∈ f ⁻¹' A
      apply mem_preimage.mpr
      -- ⊢ f x ∈ A
      exact fxA
    . -- fxB : f x ∈ B
      right
      -- ⊢ x ∈ f ⁻¹' B
      apply mem_preimage.mpr
      -- ⊢ f x ∈ B
      exact fxB
  . -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B → x ∈ f ⁻¹' (A ∪ B)
    intro h
    -- h : x ∈ f ⁻¹' A ∪ f ⁻¹' B
```

```
    -- ⊢ x ∈ f ⁻¹' (A ∪ B)
    rw [mem_preimage]
    -- ⊢ f x ∈ A ∪ B
    rcases h with xfA | xfB
    . -- xfA : x ∈ f ⁻¹' A
      rw [mem_preimage] at xfA
      -- xfA : f x ∈ A
      left
      -- ⊢ f x ∈ A
      exact xfA
    . -- xfB : x ∈ f ⁻¹' B
      rw [mem_preimage] at xfB
      -- xfB : f x ∈ B
      right
      -- ⊢ f x ∈ B
      exact xfB

-- 2ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (A ∪ B) ↔ x ∈ f ⁻¹' A ∪ f ⁻¹' B
  constructor
  . -- ⊢ x ∈ f ⁻¹' (A ∪ B) → x ∈ f ⁻¹' A ∪ f ⁻¹' B
    intros h
    -- h : x ∈ f ⁻¹' (A ∪ B)
    -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B
    rcases h with fxA | fxB
    . -- fxA : f x ∈ A
      left
      -- ⊢ x ∈ f ⁻¹' A
      exact fxA
    . -- fxB : f x ∈ B
      right
      -- ⊢ x ∈ f ⁻¹' B
      exact fxB
  . -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B → x ∈ f ⁻¹' (A ∪ B)
    intro h
    -- h : x ∈ f ⁻¹' A ∪ f ⁻¹' B
    -- ⊢ x ∈ f ⁻¹' (A ∪ B)
    rcases h with xfA | xfB
    . -- xfA : x ∈ f ⁻¹' A
```

```
      left
      -- ⊢ f x ∈ A
      exact xfA
    . -- xfB : x ∈ f ⁻¹' B
      right
      -- ⊢ f x ∈ B
      exact xfB

-- 3ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (A ∪ B) ↔ x ∈ f ⁻¹' A ∪ f ⁻¹' B
  constructor
  . -- ⊢ x ∈ f ⁻¹' (A ∪ B) → x ∈ f ⁻¹' A ∪ f ⁻¹' B
    rintro (fxA | fxB)
    . -- fxA : f x ∈ A
      -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B
      exact Or.inl fxA
    . -- fxB : f x ∈ B
      -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B
      exact Or.inr fxB
  . -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B → x ∈ f ⁻¹' (A ∪ B)
    rintro (xfA | xfB)
    . -- xfA : x ∈ f ⁻¹' A
      -- ⊢ x ∈ f ⁻¹' (A ∪ B)
      exact Or.inl xfA
    . -- xfB : x ∈ f ⁻¹' B
      -- ⊢ x ∈ f ⁻¹' (A ∪ B)
      exact Or.inr xfB

-- 4ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (A ∪ B) ↔ x ∈ f ⁻¹' A ∪ f ⁻¹' B
  constructor
  . -- ⊢ x ∈ f ⁻¹' (A ∪ B) → x ∈ f ⁻¹' A ∪ f ⁻¹' B
    aesop
```

```
  . -- ⊢ x ∈ f ⁻¹' A ∪ f ⁻¹' B → x ∈ f ⁻¹' (A ∪ B)
    aesop

-- 5ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by
  ext x
  -- x : α
  -- ⊢ x ∈ f ⁻¹' (A ∪ B) ↔ x ∈ f ⁻¹' A ∪ f ⁻¹' B
  aesop

-- 6ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by ext ; aesop

-- 7ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by ext ; rfl

-- 8ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
rfl

-- 9ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
preimage_union

-- 10ª demostración
-- ===============

example : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B :=
by simp

-- Lemas usados
-- ============
```

```
-- variable (x : α)
-- variable (p q : Prop)
-- #check (Or.inl: p → p ∨ q)
-- #check (Or.inr: q → p ∨ q)
-- #check (mem_preimage : x ∈ f ⁻¹' A ↔ f x ∈ A)
-- #check (preimage_union : f ⁻¹' (A ∪ B) = f ⁻¹' A ∪ f ⁻¹' B)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.36. f[s ∩ t] ⊆ f[s] ∩ f[t]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set
```

```
variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)

-- 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
    -- ⊢ x ∈ s ∧ f x = y
    exact ⟨xs, fxy⟩
  . -- ⊢ y ∈ f '' t
    use x
    -- ⊢ x ∈ t ∧ f x = y
    exact ⟨xt, fxy⟩

-- 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

-- Lemas usados
-- ============

-- #check (image_inter_subset f s t : f '' (s ∩ t) ⊆ f '' s ∩ f '' t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.37. Si f es inyectiva, entonces f[s] ∩ f[t] ⊆ f[s ∩ t]

```
-- ---------------------------------------------------------------------
-- 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) ∈
-- y, por (4),
--    y ∈ f[s ∩ t]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Set.Function

open Set Function

variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)

-- 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
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.38. f[s] \ f[t] ⊆ f[s \ t]

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set

variable {α β : Type _}
variable (f : α → β)
variable (s t : Set α)
```

```
-- 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

-- Lemmas usados
-- =============

-- variable (x : α)
-- #check (mem_image_of_mem f : x  ∈ s → f x ∈ f '' s)
-- #check (subset_image_diff f s t : f '' s \ f '' t ⊆ f '' (s \ t))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.39. f[s] ∩ v = f[s ∩ f⁻¹[v]]

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que, para toda y,
--    y ∈ f[s] ∩ t ↔ y ∈ f[s ∩ f⁻¹[t]]
-- Lo haremos probando las dos implicaciones.
--
-- (⟹) Supongamos que y ∈ f[s] ∩ t. Entonces, se tiene que
--    y ∈ f[s]                                                       (1)
--    y ∈ t                                                          (2)
-- Por (1), existe un x tal que
--    x ∈ s                                                          (3)
--    f(x) = y                                                       (4)
-- Por (2) y (4),
--    f(x) ∈ t
-- y, por tanto,
--    x ∈ f⁻¹[t]
-- que, junto con (3), da
```

```
--    x ∈ s ∩ f⁻¹[t]
-- y, por tanto,
--    f(x) ∈ f[s ∩ f⁻¹[t]]
-- que, junto con (4), da
--    y ∈ f[s ∩ f⁻¹[t]]
--
-- (□) Supongamos que y ∈ f[s ∩ f⁻¹[t]]. Entonces, existe un x tal que
--    x ∈ s ∩ f⁻¹[t]                                                     (5)
--    f(x) = y                                                           (6)
-- Por (1), se tiene que
--    x ∈ s                                                              (7)
--    x ∈ f⁻¹[t]                                                         (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) ∈ t
-- y, junto con (6), se tiene que
--    y ∈ t                                                             (10)
-- Por (9) y (19), se tiene que
--    y ∈ f[s] ∩ t

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set

variable {α β : Type _}
variable (f : α → β)
variable (s : Set α)
variable (t : Set β)

-- 1ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ t ↔ y ∈ f '' (s ∩ f ⁻¹' t)
  have h1 : y ∈ f '' s ∩ t → y ∈ f '' (s ∩ f ⁻¹' t) := by
```

```
    intro hy
    -- hy : y ∈ f '' s ∩ t
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t)
    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 ∈ t := hy.2
    have h1e : f x ∈ t := by rwa [←h1c] at h1d
    have h1f : x ∈ s ∩ f ⁻¹' t := mem_inter h1b h1e
    have h1g : f x ∈ f '' (s ∩ f ⁻¹' t) := mem_image_of_mem f h1f
    show y ∈ f '' (s ∩ f ⁻¹' t)
    rwa [h1c] at h1g
  have h2 : y ∈ f '' (s ∩ f ⁻¹' t) → y ∈ f '' s ∩ t :=  by
    intro hy
    -- hy : y ∈ f '' (s ∩ f ⁻¹' t)
    -- ⊢ y ∈ f '' s ∩ t
    obtain ⟨x : α, hx : x ∈ s ∩ f ⁻¹' t ∧ 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' t := hx.1.2
    have h2e : f x ∈ t := mem_preimage.mp h2d
    have h2f : y ∈ t := by rwa [hx.2] at h2e
    show y ∈ f '' s ∩ t
    exact mem_inter h2c h2f
  show y ∈ f '' s ∩ t ↔ y ∈ f '' (s ∩ f ⁻¹' t)
  exact ⟨h1, h2⟩

-- 2ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ t ↔ y ∈ f '' (s ∩ f ⁻¹' t)
  constructor
  . -- ⊢ y ∈ f '' s ∩ t → y ∈ f '' (s ∩ f ⁻¹' t)
    intro hy
    -- hy : y ∈ f '' s ∩ t
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t)
    cases' hy with hyfs yt
    -- hyfs : y ∈ f '' s
    -- yt : y ∈ t
```

```
    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' t ∧ f x = y
    constructor
    . -- ⊢ x ∈ s ∩ f ⁻¹' t
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ f ⁻¹' t
        rw [mem_preimage]
        -- ⊢ f x ∈ t
        rw [fxy]
        -- ⊢ y ∈ t
        exact yt
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t) → y ∈ f '' s ∩ t
    intro hy
    -- hy : y ∈ f '' (s ∩ f ⁻¹' t)
    -- ⊢ y ∈ f '' s ∩ t
    cases' hy with x hx
    -- x : α
    -- hx : x ∈ s ∩ f ⁻¹' t ∧ 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 ∈ t
      cases' hx with hx1 fxy
      -- hx1 : x ∈ s ∩ f ⁻¹' t
      -- fxy : f x = y
      rw [←fxy]
      -- ⊢ f x ∈ t
      rw [←mem_preimage]
      -- ⊢ x ∈ f ⁻¹' t
```

```
      exact hx1.2

-- 3ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ t ↔ y ∈ f '' (s ∩ f ⁻¹' t)
  constructor
  . -- ⊢ y ∈ f '' s ∩ t → y ∈ f '' (s ∩ f ⁻¹' t)
    rintro ⟨⟨x, xs, fxy⟩, yt⟩
    -- yt : y ∈ t
    -- x : α
    -- xs : x ∈ s
    -- fxy : f x = y
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t)
    use x
    -- ⊢ x ∈ s ∩ f ⁻¹' t ∧ f x = y
    constructor
    . -- ⊢ x ∈ s ∩ f ⁻¹' t
      constructor
      . -- ⊢ x ∈ s
        exact xs
      . -- ⊢ x ∈ f ⁻¹' t
        rw [mem_preimage]
        -- ⊢ f x ∈ t
        rw [fxy]
        -- ⊢ y ∈ t
        exact yt
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t) → y ∈ f '' s ∩ t
    rintro ⟨x, ⟨xs, xt⟩, fxy⟩
    -- x : α
    -- fxy : f x = y
    -- xs : x ∈ s
    -- xt : x ∈ f ⁻¹' t
    -- ⊢ y ∈ f '' s ∩ t
    constructor
    . -- ⊢ y ∈ f '' s
      use x, xs
      -- ⊢ f x = y
      exact fxy
```

```
    . -- ⊢ y ∈ t
      rw [←fxy]
      -- ⊢ f x ∈ t
      rw [←mem_preimage]
      -- ⊢ x ∈ f ⁻¹' t
      exact xt

-- 4ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
by
  ext y
  -- y : β
  -- ⊢ y ∈ f '' s ∩ t ↔ y ∈ f '' (s ∩ f ⁻¹' t)
  constructor
  . -- ⊢ y ∈ f '' s ∩ t → y ∈ f '' (s ∩ f ⁻¹' t)
    rintro ⟨⟨x, xs, fxy⟩, yt⟩
    -- yt : y ∈ t
    -- x : α
    -- xs : x ∈ s
    -- fxy : f x = y
    -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t)
    aesop
  . -- ⊢ y ∈ f '' (s ∩ f ⁻¹' t) → y ∈ f '' s ∩ t
    rintro ⟨x, ⟨xs, xt⟩, fxy⟩
    -- x : α
    -- fxy : f x = y
    -- xs : x ∈ s
    -- xt : x ∈ f ⁻¹' t
    -- ⊢ y ∈ f '' s ∩ t
    aesop

-- 5ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
by ext ; constructor <;> aesop

-- 6ª demostración
-- ===============

example : (f '' s) ∩ t = f '' (s ∩ f ⁻¹' t) :=
(image_inter_preimage f s t).symm
```

```
-- Lemas usados
-- ============

-- variable (x : α)
-- variable (v : Set α)
-- #check (image_inter_preimage f s t : f '' (s ∩ f ⁻¹' t) = f '' s ∩ t)
-- #check (mem_image_of_mem f : x ∈ s → f x ∈ f '' s)
-- #check (mem_inter : x ∈ s → x ∈ v → x ∈ s ∩ v)
-- #check (mem_preimage : x ∈ f ⁻¹' t ↔ f x ∈ t)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.40. Unión con la imagen

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f '' (s ∪ f ⁻¹' v) ⊆ f '' s ∪ v
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea y ∈ f[s ∪ f⁻¹[v]]. Entonces, existe un x tal que
--    x ∈ s ∪ f⁻¹[v]                                               (1)
--    f(x) = y                                                     (2)
-- De (1), se tiene que x ∈ s ó x ∈ f⁻¹[v]. Vamos a demostrar en ambos
-- casos que
--    y ∈ f[s] ∪ v
--
-- Caso 1: Supongamos que x ∈ s. Entonces,
--    f(x) ∈ f[s]
-- y, por (2), se tiene que
--    y ∈ f[s]
-- Por tanto,
--    y ∈ f[s] ∪ v
--
-- Caso 2: Supongamos que x ∈ f⁻¹[v]. Entonces,
--    f(x) ∈ v
-- y, por (2), se tiene que
--    y ∈ v
-- Por tanto,
--    y ∈ f[s] ∪ v
```

```
-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set

variable (α β : Type _)
variable (f : α → β)
variable (s : Set α)
variable (v : Set β)

-- 1ª demostración
-- ===============

example : f '' (s ∪ f ⁻¹' v) ⊆ f '' s ∪ v :=
by
  intros y hy
  obtain ⟨x : α, hx : x ∈ s ∪ f ⁻¹' v ∧ f x = y⟩ := hy
  obtain ⟨hx1 : x ∈ s ∪ f ⁻¹' v, fxy : f x = y⟩ := hx
  cases' hx1 with xs xv
  . -- xs : x ∈ s
    have h1 : f x ∈ f '' s := mem_image_of_mem f xs
    have h2 : y ∈ f '' s := by rwa [fxy] at h1
    show y ∈ f '' s ∪ v
    exact mem_union_left v h2
  . -- xv : x ∈ f ⁻¹' v
    have h3 : f x ∈ v := mem_preimage.mp xv
    have h4 : y ∈ v := by rwa [fxy] at h3
    show y ∈ f '' s ∪ v
    exact mem_union_right (f '' s) h4

-- 1ª demostración
-- ===============

example : f '' (s ∪ f ⁻¹' v) ⊆ f '' s ∪ v :=
by
  intros y hy
  obtain ⟨x : α, hx : x ∈ s ∪ f ⁻¹' v ∧ f x = y⟩ := hy
  obtain ⟨hx1 : x ∈ s ∪ f ⁻¹' v, fxy : f x = y⟩ := hx
  cases' hx1 with xs xv
  . -- xs : x ∈ s
    left
    -- ⊢ y ∈ f '' s
```

```
    use x
    -- ⊢ x ∈ s ∧ f x = y
    constructor
    . -- ⊢ x ∈ s
      exact xs
    . -- ⊢ f x = y
      exact fxy
  . -- ⊢ y ∈ f '' s ∪ v
    right
    -- ⊢ y ∈ v
    rw [←fxy]
    -- ⊢ f x ∈ v
    exact xv

-- 2ª demostración
-- ===============

example : f '' (s ∪ f ⁻¹' v) ⊆ f '' s ∪ v :=
by
  rintro y ⟨x, xs | xv, fxy⟩
  -- y : β
  -- x : α
  . -- xs : x ∈ s
    -- ⊢ y ∈ f '' s ∪ v
    left
    -- ⊢ y ∈ f '' s
    use x, xs
    -- ⊢ f x = y
    exact fxy
  . -- xv : x ∈ f ⁻¹' v
    -- ⊢ y ∈ f '' s ∪ v
    right
    -- ⊢ y ∈ v
    rw [←fxy]
    -- ⊢ f x ∈ v
    exact xv

-- 3ª demostración
-- ===============

example : f '' (s ∪ f ⁻¹' v) ⊆ f '' s ∪ v :=
by
  rintro y ⟨x, xs | xv, fxy⟩ <;>
  aesop
```

```
-- Lemas usados
-- ============

-- variable (x : α)
-- variable (t : Set α)
-- #check (mem_image_of_mem f : x ∈ s → f x ∈ f '' s)
-- #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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.41. Intersección con la imagen inversa

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v)
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemmos que demostrar que, para todo y,
--    y ∈ f[s] ∩ v ↔ y ∈ f[s ∩ f⁻¹[v]]
-- Lo haremos demostrando las sod implicaciones.
--
-- (⟹) Supongamos que y ∈ f[s] ∩ v. Entonces,
--    y ∈ f[s]                                                      (1)
--    y ∈ v                                                         (2)
-- Por (1), existe un x tal que
--    x ∈ s                                                         (3)
--    f(x) = y                                                      (4)
-- De (2) y (4), se tiene que
--    f(x) ∈ v
-- y, por tanto,
--    x ∈ f⁻¹[v]                                                    (5)
-- De (3) y (5), se tiene que
--    x ∈ s ∩ f⁻¹[v]
-- Por tanto,
--    f(x) ∈ f[s ∩ f⁻¹[v]]
-- y, por (4),
--    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]                                                  (6)
--    f(x) = y                                                        (7)
-- Por (6), se tiene que
--    x ∈ s                                                           (8)
--    x ∈ f⁻¹[v]                                                      (9)
-- Por (8), se tiene que
--    f(x) ∈ f[s]
-- y, por (7),
--    y ∈ f[s]                                                       (10)
-- Por (9),
--    f(x) ∈ v
-- y, por (7),
--    y ∈ v                                                          (11)
-- Por (10) y (11),
--    y ∈ f[s] ∩ v

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Set.Function
import Mathlib.Tactic

open Set

variable {α β : Type _}
variable (f : α → β)
variable (s : Set α)
variable (v : Set β)

-- 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)
  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
    have h1 : f x ∈ v := by rwa [←fxy] at yv
    have h3 : x ∈ s ∩ f ⁻¹' v := mem_inter xs h1
    have h4 : f x ∈ f '' (s ∩ f ⁻¹' v) := mem_image_of_mem f h3
    show y ∈ f '' (s ∩ f ⁻¹' v)
    rwa [fxy] at h4
  . -- ⊢ 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
    cases' hx with hx1 fxy
    -- hx1 : x ∈ s ∩ f ⁻¹' v
    -- fxy : f x = y
    cases' hx1 with xs xfv
    -- xs : x ∈ s
    -- xfv : x ∈ f ⁻¹' v
    have h5 : f x ∈ f '' s := mem_image_of_mem f xs
    have h6 : y ∈ f '' s := by rwa [fxy] at h5
    have h7 : f x ∈ v := mem_preimage.mp xfv
    have h8 : y ∈ v := by rwa [fxy] at h7
    show y ∈ f '' s ∩ v
    exact mem_inter h6 h8

-- 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
    . exact fxy
  . 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
      -- ⊢ f x = y
```

```
      exact fxy
    . -- ⊢ 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
```

```
-- Lemas usados
-- ============

-- variable (x : α)
-- variable (a b : Set α)
-- #check (image_inter_preimage f s v : f '' (s ∩ f ⁻¹' v) = f '' s ∩ v)
-- #check (mem_image_of_mem  f : x ∈ a → f x ∈ f '' a)
-- #check (mem_inter : x ∈ a → x ∈ b → x ∈ a ∩ b)
-- #check (mem_preimage : x ∈ f ⁻¹' v ↔ f x ∈ v)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.42. Unión con la imagen inversa

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Function

open Set

variable {α β : Type _}
```

```
variable (f : α → β)
variable (s : Set α)
variable (v : Set β)

-- 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)
  cases' 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
  cases' 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)
  cases' 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 (t : Set α)
-- variable (a b c : Prop)
-- #check (Or.elim : a ∨ b → (a → c) → (b → c) → c)
```

```
-- #check (Or.inl : a → a ∨ b)
-- #check (Or.inr : b → a ∨ b)
-- #check (mem_image_of_mem f : x  ∈ s → f x ∈ f '' s)
-- #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 (union_preimage_subset s v f : s ∪ f ⁻¹' v ⊆ f ⁻¹' (f '' s ∪ v))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.43. Imagen de la unión general

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α β I : Type _}
variable (f : α → β)
variable (A : ℕ → Set α)

-- 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
    -- ⊢ x ∈ A i
    exact xAi

-- 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
```

```
    -- ⊢ f x = y
    exact fxy
  . -- ⊢ (∃ 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

-- Lemas usados
-- ============

-- variable (x : α)
-- variable (y : β)
-- variable (s : Set α)
-- variable (i : ℕ)
-- #check (image_iUnion : f '' ⋃ i, A i = ⋃ i, f '' 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.44. Imagen de la intersección general

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f[⋂ᵢ Aᵢ] ⊆ ⋂ᵢ f[Aᵢ]
-- ----------------------------------------------------------------------

-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α β I : Type _}
variable (f : α → β)
variable (A : I → Set α)

-- 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

-- 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
  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

-- 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
  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)

-- 3ª 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
  -- ⊢ f x = y
  exact fxy

-- 4ª demostración
-- ===============

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
image_iInter_subset A f

-- Lemas usados
-- ============

-- variable (x : α)
-- variable (s : Set α)
-- #check (image_iInter_subset A f : 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_image_of_mem f : x ∈ s → f x ∈ f '' s)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.45. Imagen de la intersección general mediante aplicaciones inyectivas

```
-- ---------------------------------------------------------------------
-- Demostrar que si f es inyectiva, entonces
--    ⋂ᵢf[Aᵢ] ⊆ f[⋂ᵢAᵢ]
-- ----------------------------------------------------------------------

-- 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
-- ========================

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)
```

```
-- 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

-- Lemas usados
-- ============

-- variable (x : α)
-- variable (s : Set α)
-- #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_image_of_mem f : x  ∈ s → f x ∈ f '' s)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.46. Imagen inversa de la unión general

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f⁻¹[⋃ᵢ Bᵢ] = ⋃ᵢ f⁻¹[Bᵢ]
-- ----------------------------------------------------------------------

-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α β I : Type _}
variable (f : α → β)
variable (B : I → Set β)

-- 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

-- Lemas usados
-- ============

-- variable (x : α)
-- variable (s : Set β)
-- variable (A : I → Set α)
-- #check (mem_iUnion : x ∈ ⋃ i, A i ↔ ∃ i, x ∈ A i)
-- #check (mem_preimage : x ∈ f ⁻¹' s ↔ f x ∈ s)
-- #check (preimage_iUnion : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.47. Imagen inversa de la intersección general

```
-- ---------------------------------------------------------------------
-- Demostrar que
--    f⁻¹[⋂ᵢ Bᵢ] = ⋂ᵢ f⁻¹[Bᵢ]
-- ----------------------------------------------------------------------

-- 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
-- ========================

import Mathlib.Data.Set.Basic
import Mathlib.Tactic

open Set

variable {α β I : Type _}
variable (f : α → β)
```

```
variable (B : I → Set β)

-- 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 (x : α)
-- variable (s : Set β)
-- variable (A : I → Set α)
-- variable (a b : Prop)
-- #check (iff_of_eq : a = b → (a ↔ b))
-- #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_preimage : x ∈ f ⁻¹' s ↔ f x ∈ s)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.48. 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
  cases' hf S with j hj
  -- j : α
  -- hj : f j = S
  by_cases j ∈ S
  . -- h : j ∈ S
    dsimp 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}
  cases' hf S with j hj
  -- j : α
  -- hj : f j = S
  by_cases 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}
  cases' 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 : α)
-- variable (p : α → Prop)
-- variable (a b : Prop)
-- #check (Set.mem_setOf_eq : (x ∈ {y : α | p y}) = p x)
-- #check (iff_of_eq : a = b → (a ↔ b))
-- #check (iff_not_self : ¬(a ↔ ¬a))
-- #check (cantor_surjective : ∀ f : α → Set α, ¬ Surjective f)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.49. Si g ∘ f es suprayectiva, entonces g es suprayectiva

```
-- ---------------------------------------------------------------------
-- Sean f: X → Y y g: Y → Z. Demostrar que si g ∘ f es suprayectiva,
-- entonces g es suprayectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se z ∈ Z. Entonces, por ser g ∘ f suprayectiva, existe un x ∈ X tal
-- que
--    (g ∘ f)(x) = z                                                (1)
-- Por tanto, existe y = f(x) ∈ Y tal que
--    g(y) = g(f(x))
--         = (g ∘ f)(x)
--         = z              [por (1)]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Function

variable {X Y Z : Type}
variable {f : X → Y}
variable {g : Y → Z}

-- 1ª demostración
-- ===============

example
  (h : Surjective (g ∘ f))
  : Surjective g :=
by
  intro z
  -- z : Z
  -- ⊢ ∃ a, g a = z
  cases' h z with x hx
  -- x : X
  -- hx : (g ∘ f) x = z
  use f x
```

```
  -- ⊢ g (f x) = z
  exact hx

-- 2ª demostración
-- ===============

example
  (h : Surjective (g ∘ f))
  : Surjective g :=
by
  intro z
  -- z : Z
  -- ⊢ ∃ a, g a = z
  cases' h z with x hx
  -- x : X
  -- hx : (g ∘ f) x = z
  exact ⟨f x, hx⟩
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.50. Las funciones inyectivas tienen inversa por la izquierda

```
-- ---------------------------------------------------------------------
-- En Lean4, que g es una inversa por la izquierda de f está definido por
--    LeftInverse (g : β → α) (f : α → β) : Prop :=
--       ∀ x, g (f x) = x
-- y que f tenga inversa por la izquierda está definido por
--    HasLeftInverse (f : α → β) : Prop :=
--       ∃ finv : β → α, LeftInverse finv f
-- Finalmente, que f es inyectiva está definido por
--    Injective (f : α → β) : Prop :=
--       ∀ ⦃x y⦄, f x = f y → x = y
--
-- Demostrar que si f es una función inyectiva con dominio no vacío,
-- entonces f tiene inversa por la izquierda.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea f: A → B inyectiva con A ≠ ∅. Entonces, existe un a ∈ A. Sea
```

```
-- g: B → A definida por
--    g(y) = + un x tal que f(x) = y, si (∃x)[f(x) = y]
--           + a, en caso contrario.
-- Vamos a demostrar que g es una inversa por la izquierda de f; es
-- decir,
--    (∀x)[g(f(x)) = x]
-- Para ello, sea x ∈ A. Entonces,
--    (∃x)[f(x) = f(x)]
-- Por la definición de g,
--    g(f(x)) = z                                          (1)
-- donde
--    f(z) = f(x).
-- Como f es inyectiva,
--    z = x
-- Y, por (1),
--    g(f(x)) = x

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

open Function Classical

variable {α β: Type _}
variable {f : α → β}

-- 1ª demostración
-- ===============

example
  [hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
by
  unfold HasLeftInverse
  -- ⊢ ∃ finv, LeftInverse finv f
  set g := fun y ↦ if h : ∃ x, f x = y then h.choose else Classical.arbitrary α
  use g
  unfold LeftInverse
  -- ⊢ ∀ (x : α), g (f x) = x
  intro a
  -- ⊢ g (f a) = a
  have h1 : ∃ x : α, f x = f a := Exists.intro a rfl
  dsimp at *
```

```
  -- ⊢ (if h : ∃ x, f x = f a then Exists.choose h else Classical.arbitrary α) = a
  simp [h1]
  -- ⊢ Exists.choose (_ : ∃ x, f x = f a) = a
  apply hf
  -- ⊢ f (Exists.choose (_ : ∃ x, f x = f a)) = f a
  exact Classical.choose_spec h1

-- 2ª demostración
-- ===============

example
  [hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
by
  set g := fun y ↦ if h : ∃ x, f x = y then h.choose else Classical.arbitrary α
  use g
  -- ⊢ LeftInverse g f
  intro a
  -- a : α
  -- ⊢ g (f a) = a
  have h1 : ∃ x : α, f x = f a := Exists.intro a rfl
  dsimp at *
  -- ⊢ (if h : ∃ x, f x = f a then Exists.choose h else Classical.arbitrary α) = a
  simp [h1]
  -- ⊢ Exists.choose (_ : ∃ x, f x = f a) = a
  exact hf (Classical.choose_spec h1)

-- 3ª demostración
-- ===============

example
  [hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
by
  unfold HasLeftInverse
  -- ⊢ ∃ finv, LeftInverse finv f
  use invFun f
  -- ⊢ LeftInverse (invFun f) f
  unfold LeftInverse
  -- ⊢ ∀ (x : α), invFun f (f x) = x
  intro x
  -- x : α
  -- ⊢ invFun f (f x) = x
```

```
  apply hf
  -- ⊢ f (invFun f (f x)) = f x
  apply invFun_eq
  -- ⊢ ∃ a, f a = f x
  use x

-- 4ª demostración
-- ===============

example
  [hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
by
  use invFun f
  -- ⊢ LeftInverse (invFun f) f
  intro x
  -- x : α
  -- ⊢ invFun f (f x) = x
  apply hf
  -- ⊢ f (invFun f (f x)) = f x
  apply invFun_eq
  -- ⊢ ∃ a, f a = f x
  use x

-- 5ª demostración
-- ===============

example
  [_hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
⟨invFun f, leftInverse_invFun hf⟩

-- 6ª demostración
-- ===============

example
  [_hα : Nonempty α]
  (hf : Injective f)
  : HasLeftInverse f :=
Injective.hasLeftInverse hf

-- Lemas usados
-- ============
```

```
-- variable (p : α → Prop)
-- variable (x : α)
-- variable (b : β)
-- variable (γ : Type _) [Nonempty γ]
-- variable (f1 : γ → β)
-- #check (Classical.choose_spec : (h : ∃ x, p x) → p (Classical.choose h))
-- #check (Exists.intro x: p x → ∃ y, p y)
-- #check (Injective.hasLeftInverse : Injective f1 → HasLeftInverse f1)
-- #check (invFun_eq : (∃ a, f1 a = b) → f1 (invFun f1 b) = b)
-- #check (leftInverse_invFun : Function.Injective f1 → LeftInverse (Function.invFun f1)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.51. Las funciones con inversa por la derecha son suprayectivas

```
-- ---------------------------------------------------------------------
-- En Lean4, que g es una inversa por la izquierda de f está definido por
--    LeftInverse (g : β → α) (f : α → β) : Prop :=
--       ∀ x, g (f x) = x
-- que g es una inversa por la derecha de f está definido por
--    RightInverse (g : β → α) (f : α → β) : Prop :=
--       LeftInverse f g
-- y que f tenga inversa por la derecha está definido por
--    HasRightInverse (f : α → β) : Prop :=
--       ∃ g : β → α, RightInverse g f
-- Finalmente, que f es suprayectiva está definido por
--    def Surjective (f : α → β) : Prop :=
--       ∀ b, ∃ a, f a = b
--
-- Demostrar que si la función f tiene inversa por la derecha, entonces
-- f es suprayectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea f: A → B y g: B → A una inversa por la derecha de f. Entonces,
--    (∀y ∈ B)[f(g(y)) = y]                                          (1)
--
-- Para demostrar que f es subprayectiva, sea b ∈ B. Entonces,
-- g(b) ∈ A y, por (1),
```

```
--    f(g(b)) = b

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
open Function

variable {α β: Type _}
variable {f : α → β}

-- 1ª demostración
-- ===============

example
  (hf : HasRightInverse f)
  : Surjective f :=
by
  unfold Surjective
  -- ⊢ ∀ (b : β), ∃ a, f a = b
  unfold HasRightInverse at hf
  -- hf : ∃ finv, Function.RightInverse finv f
  cases' hf with g hg
  -- g : β → α
  -- hg : Function.RightInverse g f
  intro b
  -- b : β
  -- ⊢ ∃ a, f a = b
  use g b
  -- ⊢ f (g b) = b
  exact hg b

-- 2ª demostración
-- ===============

example
  (hf : HasRightInverse f)
  : Surjective f :=
by
  intro b
  -- b : β
  -- ⊢ ∃ a, f a = b
  cases' hf with g hg
  -- g : β → α
  -- hg : Function.RightInverse g f
```

```
  use g b
  -- ⊢ f (g b) = b
  exact hg b

-- 3ª demostración
-- ===============

example
  (hf : HasRightInverse f)
  : Surjective f :=
by
  intro b
  -- b : β
  -- ⊢ ∃ a, f a = b
  cases' hf with g hg
  -- g : β → α
  -- hg : Function.RightInverse g f
  exact ⟨g b, hg b⟩

-- 4ª demostración
-- ===============

example
  (hf : HasRightInverse f)
  : Surjective f :=
HasRightInverse.surjective hf

-- Lemas usados
-- ============

-- #check (HasRightInverse.surjective : HasRightInverse f → Surjective f)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.52. Las funciones suprayectivas tienen inversa por la derecha

```
-- ---------------------------------------------------------------------
-- En Lean4, que g es una inversa por la izquierda de f está definido
-- por
--    LeftInverse (g : β → α) (f : α → β) : Prop :=
--       ∀ x, g (f x) = x
```

```
-- que g es una inversa por la derecha de f está definido por
--    RightInverse (g : β → α) (f : α → β) : Prop :=
--       LeftInverse f g
-- y que f tenga inversa por la derecha está definido por
--    HasRightInverse (f : α → β) : Prop :=
--       ∃ g : β → α, RightInverse g f
-- Finalmente, que f es suprayectiva está definido por
--    def Surjective (f : α → β) : Prop :=
--       ∀ b, ∃ a, f a = b
--
-- Demostrar que si f es una función suprayectiva, entonces f tiene
-- inversa por la derecha.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea f: A → B una función suprayectiva. Sea g: B → A la función
-- definida por
--    g(y) = x, donde x es un elemento tal que f(x) = y
--
-- Veamos que g es una inversa por la derecha de f; es decir,
--    (∀y ∈ B)[f(g(y)) = y
-- Para ello, sea b ∈ B. Entonces,
--    f(g(b)) = f(a)
-- donde a es un elemento tal que
--    f(a) = b
-- Por tanto,
--    f(g(b)) = b

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
open Function Classical

variable {α β: Type _}
variable {f : α → β}

-- 1ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
```

```
by
  unfold HasRightInverse
  -- ⊢ ∃ finv, Function.RightInverse finv f
  let g := fun y ↦ Classical.choose (hf y)
  use g
  -- ⊢ Function.RightInverse g f
  unfold Function.RightInverse
  -- ⊢ LeftInverse f g
  unfold Function.LeftInverse
  -- ⊢ ∀ (x : β), f (g x) = x
  intro b
  -- ⊢ f (g b) = b
  exact Classical.choose_spec (hf b)

-- 2ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
by
  let g := fun y ↦ Classical.choose (hf y)
  use g
  -- ⊢ Function.RightInverse g f
  intro b
  -- ⊢ f (g b) = b
  exact Classical.choose_spec (hf b)

-- 3ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
by
  use surjInv hf
  -- ⊢ Function.RightInverse (surjInv hf) f
  intro b
  -- ⊢ f (surjInv hf b) = b
  exact surjInv_eq hf b

-- 4ª demostración
-- ===============

example
```

```
  (hf : Surjective f)
  : HasRightInverse f :=
by
  use surjInv hf
  -- ⊢ Function.RightInverse (surjInv hf) f
  exact surjInv_eq hf

-- 5ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
⟨surjInv hf, surjInv_eq hf⟩

-- 6ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
⟨_, rightInverse_surjInv hf⟩

-- 7ª demostración
-- ===============

example
  (hf : Surjective f)
  : HasRightInverse f :=
Surjective.hasRightInverse hf

-- Lemas usados
-- ============

-- variable (p : α -> Prop)
-- #check (Classical.choose_spec : (h : ∃ x, p x) → p (Classical.choose h))
--
-- variable (h : Surjective f)
-- variable (b : β)
-- #check (surjInv_eq h b : f (surjInv h b) = b)
-- #check (rightInverse_surjInv h : RightInverse (surjInv h) f)
--
-- #check (Surjective.hasRightInverse : Surjective f → HasRightInverse f)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

15.53. Las funciones con inversa son biyectivas

```
-- ---------------------------------------------------------------------
-- En Lean4 se puede definir que g es una inversa de f por
--    def inversa (f : X → Y) (g : Y → X) :=
--      (∀ x, (g ∘ f) x = x) ∧ (∀ y, (f ∘ g) y = y)
-- y que f tiene inversa por
--    def tiene_inversa (f : X → Y) :=
--      ∃ g, inversa g f
--
-- Demostrar que si la función f tiene inversa, entonces f es biyectiva.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Puesto que f tiene inversa, existe una g: Y → X tal que
--    (∀x)[(g ∘ f)(x) = x]                                            (1)
--    (∀y)[(f ∘ g)(y) = y]                                            (2)
--
-- Para demostrar que f es inyectiva, sean a, b ∈ X tales que
--    f(a) = f(b)                                                     (3)
-- entonces
--    a = g(f(a))    [por (1)]
--      = g(f(b))    [por (3)]
--      = b          [por (1)]
--
-- Para demostrar que f es suprayectiva, sea y ∈ Y. Entonces, existe
-- a = g(y) ∈ X tal que
--    f(a) = f(g(y))
--         = y          [por (2)]
--
-- Como f es inyectiva y suprayectiva, entonces es biyectiva.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
open Function

variable {X Y : Type _}
variable (f : X → Y)
```

```
def inversa (f : X → Y) (g : Y → X) :=
  (∀ x, (g ∘ f) x = x) ∧ (∀ y, (f ∘ g) y = y)

def tiene_inversa (f : X → Y) :=
  ∃ g, inversa g f

-- 1ª demostración
-- ===============

example
  (hf : tiene_inversa f)
  : Bijective f :=
by
  rcases hf with ⟨g, ⟨h1, h2⟩⟩
  -- g : Y → X
  -- h1 : ∀ (x : Y), (f ∘ g) x = x
  -- h2 : ∀ (y : X), (g ∘ f) y = y
  constructor
  . -- ⊢ Injective f
    intros a b hab
    -- a b : X
    -- hab : f a = f b
    -- ⊢ a = b
    calc a = g (f a) := (h2 a).symm
         _ = g (f b) := congr_arg g hab
         _ = b       := h2 b
  . -- ⊢ Surjective f
    intro y
    -- y : Y
    -- ⊢ ∃ a, f a = y
    use g y
    -- ⊢ f (g y) = y
    exact h1 y

-- 2ª demostración
-- ===============

example
  (hf : tiene_inversa f)
  : Bijective f :=
by
  rcases hf with ⟨g, ⟨h1, h2⟩⟩
  -- g : Y → X
  -- h1 : ∀ (x : Y), (f ∘ g) x = x
  -- h2 : ∀ (y : X), (g ∘ f) y = y
```

```
  constructor
  . -- ⊢ Injective f
    intros a b hab
    -- a b : X
    -- hab : f a = f b
    -- ⊢ a = b
    calc a = g (f a) := (h2 a).symm
         _ = g (f b) := congr_arg g hab
         _ = b       := h2 b
  . -- ⊢ Surjective f
    intro y
    -- y : Y
    -- ⊢ ∃ a, f a = y
    exact ⟨g y, h1 y⟩

-- 3ª demostración
-- ===============

example
  (hf : tiene_inversa f)
  : Bijective f :=
by
  rcases hf with ⟨g, ⟨h1, h2⟩⟩
  constructor
  . exact LeftInverse.injective h2
  . exact RightInverse.surjective h1

-- 4ª demostración
-- ===============

example
  (hf : tiene_inversa f)
  : Bijective f :=
by
  rcases hf with ⟨g, ⟨h1, h2⟩⟩
  exact ⟨LeftInverse.injective h2,
         RightInverse.surjective h1⟩

-- 5ª demostración
-- ===============

example :
  tiene_inversa f → Bijective f :=
by
  rintro ⟨g, ⟨h1, h2⟩⟩
```

```
  exact ⟨LeftInverse.injective h2,
         RightInverse.surjective h1⟩

-- 6ª demostración
-- ===============

example :
  tiene_inversa f → Bijective f :=
fun ⟨_, ⟨h1, h2⟩⟩ ↦ ⟨LeftInverse.injective h2,
                     RightInverse.surjective h1⟩

-- Lemas usados
-- ============

-- variable (x y : X)
-- variable (g : Y → X)
-- #check (congr_arg f : x = y → f x = f y)
-- #check (LeftInverse.injective : LeftInverse g f → Injective f)
-- #check (RightInverse.surjective : RightInverse g f → Surjective f)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 16

Lógica

16.1. Si ¬(∃x)P(x), entonces (∀x)¬P(x)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Tactic
variable {α : Type _}
variable (P : α → Prop)

-- 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

-- 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
  exact h1

-- 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
  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

-- Lemas usados
-- ============

-- #check (not_exists : (¬∃ x, P x) ↔ ∀ (x : α), ¬P x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.2. Si (∀x)¬P(x), entonces ¬(∃x)P(x)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Tactic
variable {α : Type _}
variable (P : α → Prop)

-- 1ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by
  intro h1
  -- h1 : ∃ x, P x
  -- ⊢ False
  rcases h1 with ⟨y, 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 : P y⟩
  exact (h y) hy

-- 3ª demostración
-- ===============

example
  (h : ∀ x, ¬ P x)
  : ¬ ∃ x, P x :=
by
  rintro ⟨y, 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

-- Lemas usados
-- ============

-- variable (q : Prop)
-- #check (not_exists_of_forall_not : (∀ x, P x → q) → (∃ x, P x) → q)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.3. Si ¬(∀x)P(x), entonces (∃x)¬P(x)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Tactic
variable {α : Type _}
variable (P : α → Prop)

-- 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⟩

-- 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

-- Lemas usados
-- ============

-- #check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.4. Si (∃x)¬P(x), entonces ¬(∀x)P(x)

```
-- ---------------------------------------------------------------------
-- 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
-- ========================
```

```
import Mathlib.Tactic
variable {α : Type _}
variable (P : α → Prop)

-- 1ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by
  intro h1
  -- h1 : ∀ (x : α), P x
  -- ⊢ False
  cases' h with y hy
  -- y : α
  -- hy : ¬P y
  apply hy
  -- ⊢ P y
  exact (h1 y)

-- 2ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by
  intro h1
  -- h1 : ∀ (x : α), P x
  -- ⊢ False
  rcases h with ⟨y, 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 : ¬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

-- 6ª demostración
-- ===============

example
  (h : ∃ x, ¬ P x)
  : ¬ ∀ x, P x :=
by aesop

-- Lemas usados
-- ============

-- #check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
-- #check (not_forall_of_exists_not : (∃ x, ¬P x) → ¬∀ x, P x)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.5. ¬¬P → P

```
-- ---------------------------------------------------------------------
-- Demostrar que ¬¬P → P.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
```

```
-- ================================

-- Por reducción al absurdo. Supongamos ¬P. Entonces, tenemos una
-- contradicción con la hipótesis (¬¬P).

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
variable (P : Prop)

-- 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

-- Lemas usados
-- ============

-- #check (of_not_not : ¬¬P → P)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.6. P → ¬¬P

```
-- ---------------------------------------------------------------------
-- Demostrar que P → ¬¬P.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Supongamos ¬P. Entonces, tenemos una contradicción con la hipótesis
-- (P).

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic
variable (P : Prop)

-- 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.7. (P → Q) ↔ ¬P ∨ Q

```
-- ---------------------------------------------------------------------
-- 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
  done

-- 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
  done

-- 3ª demostración
-- ===============

example
  (P Q : Prop)
```

```
  : (P → Q) ↔ ¬P ∨ Q :=
imp_iff_not_or

-- 4ª demostración
-- ===============

example
  (P Q : Prop)
  : (P → Q) ↔ ¬P ∨ Q :=
by tauto
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

16.8. La paradoja del barbero

```
-- ---------------------------------------------------------------------
-- Demostrar la paradoja del barbero https://bit.ly/3eWyvVw es decir,
-- que no existe un hombre que afeite a todos los que no se afeitan a sí
-- mismo y sólo a los que no se afeitan a sí mismo.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que
--    ¬((∃ x)(∀ y)[afeita(x,y) ↔ ¬afeita(y,y)])
-- Para ello, supongamos que
--    (∃ x)(∀ y)[afeita(x,y) ↔ ¬afeita(y,y)]                         (1)
-- y tenemos que llegar a una contradicción.
--
-- Sea b un elemento que verifica (1); es decir,
--    (∀ y)[afeita(b,y) ↔ ¬afeita(y,y)]
-- Entonces,
--    afeita(b,b) ↔ ¬afeita(b,b)
-- que es una contradicción.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Tactic

variable (Hombre : Type)
variable (afeita : Hombre → Hombre → Prop)
```

```
-- 1ª demostración
-- ===============

example :
  ¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
by
  intro h
  -- h : ∃ x, ∀ (y : Hombre), afeita x y ↔ ¬afeita y y
  -- ⊢ False
  cases' h with b hb
  -- b : Hombre
  -- hb : ∀ (y : Hombre), afeita b y ↔ ¬afeita y y
  specialize hb b
  -- hb : afeita b b ↔ ¬afeita b b
  by_cases (afeita b b)
  . -- h : afeita b b
    apply absurd h
    -- ⊢ ¬afeita b b
    exact hb.mp h
  . -- h : ¬afeita b b
    apply h
    -- ⊢ afeita b b
    exact hb.mpr h

-- 2ª demostración
-- ===============

example :
  ¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
by
  intro h
  -- h : ∃ x, ∀ (y : Hombre), afeita x y ↔ ¬afeita y y
  -- ⊢ False
  cases' h with b hb
  -- b : Hombre
  -- hb : ∀ (y : Hombre), afeita b y ↔ ¬afeita y y
  specialize hb b
  -- hb : afeita b b ↔ ¬afeita b b
  by_cases (afeita b b)
  . -- h : afeita b b
    exact (hb.mp h) h
  . -- h : ¬afeita b b
    exact h (hb.mpr h)
```

```
-- 3ª demostración
-- ===============

example :
  ¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
by
  intro h
  -- h : ∃ x, ∀ (y : Hombre), afeita x y ↔ ¬afeita y y
  -- ⊢ False
  cases' h with b hb
  -- b : Hombre
  -- hb : ∀ (y : Hombre), afeita b y ↔ ¬afeita y y
  exact iff_not_self (hb b)

-- 4ª demostración
-- ===============

example :
  ¬ (∃ x : Hombre,  ∀ y : Hombre, afeita x y ↔ ¬ afeita y y ) :=
by
  rintro ⟨b, hb⟩
  -- b : Hombre
  -- hb : ∀ (y : Hombre), afeita b y ↔ ¬afeita y y
  -- ⊢ False
  exact iff_not_self (hb b)

-- 5ª demostración
-- ===============

example :
  ¬ (∃ x : Hombre,  ∀ y : Hombre, afeita x y ↔ ¬ afeita y y ) :=
fun ⟨b, hb⟩ ↦ iff_not_self (hb b)

-- Lemas usados
-- ============

-- variable (p q : Prop)
-- #check (absurd : p → (¬p → q))
-- #check (iff_not_self : ¬(p ↔ ¬p))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Capítulo 17

Límites de sucesiones

17.1. La sucesión constante $s_n = c$ converge a c

```
-- ---------------------------------------------------------------------
-- Demostrar que, para todo a ∈ ℝ, la sucesión constante
--    s(n) = a
-- converge a a.
-- ----------------------------------------------------------------------

-- 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

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

def limite (s : ℕ → ℝ) (a : ℝ) :=
  ∀ ε > 0, ∃ N, ∀ n ≥ N, |s n - a| < ε

-- 1ª demostración
```

```
-- ===============

example : limite (fun _ : ℕ ↦ c) c :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun _ => c) n - c| < ε
  use 0
  -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(fun _ => c) n - c| < ε
  intros n _hn
  -- n : ℕ
  -- hn : n ≥ 0
  -- ⊢ |(fun _ => c) n - c| < ε
  show |(fun _ => c) n - c| < ε
  calc |(fun _ => c) n - c| = |c - c|  := by dsimp
                          _ = |0|      := by {congr ; exact sub_self c}
                          _ = 0        := abs_zero
                          _ < ε        := hε

-- 2ª demostración
-- ===============

example : limite (fun _ : ℕ ↦ c) c :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun _ => c) n - c| < ε
  use 0
  -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(fun _ => c) n - c| < ε
  intros n _hn
  -- n : ℕ
  -- hn : n ≥ 0
  -- ⊢ |(fun _ => c) n - c| < ε
  show |(fun _ => c) n - c| < ε
  calc |(fun _ => c) n - c| = 0        := by simp
                          _ < ε        := hε

-- 3ª demostración
-- ===============

example : limite (fun _ : ℕ ↦ c) c :=
by
  intros ε hε
```

```
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun _ => c) n - c| < ε
  aesop

-- 4ª demostración
-- ===============

example : limite (fun _ : ℕ ↦ c) c :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun _ => c) n - c| < ε
  aesop

-- 5ª demostración
-- ===============

example : limite (fun _ : ℕ ↦ c) c :=
  fun ε hε ↦ by aesop

-- Lemas usados
-- ============

-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

17.2. Si la sucesión s converge a b y la t a c, entonces s+t converge a b+c

```
-- ---------------------------------------------------------------------
-- Demostrar que si la sucesión u converge a a y la v a b, entonces u+v
-- converge a a+b
-- ----------------------------------------------------------------------

-- 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 para todo ε ∈ ℝ, si
--    ε > 0                                                            (1)
-- entonces
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |(u + v)(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 u es a, se tiene que
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |u(n) - a| < ε/2]
-- Sea N₁ ∈ ℕ tal que
--    (∀n ∈ ℕ)[n ≥ N₁ → |u(n) - a| < ε/2]                              (4)
-- Por (3) y porque el límite de v es b, se tiene que
--    (∃N ∈ ℕ)(∀n ∈ ℕ)[n ≥ N → |v(n) - b| < ε/2]
-- Sea N₂ ∈ ℕ tal que
--    (∀n ∈ ℕ)[n ≥ N₂ → |v(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
--    |u(n) - a| < ε/2                                                 (6)
--    |v(n) - b| < ε/2                                                 (7)
-- Finalmente,
--    |(u + v)(n) - (a + b)| = |(u(n) + v(n)) - (a + b)|
--                           = |(u(n) - a) + (v(n) - b)|
--                           ≤ |u(n) - a| + |v(n) - b|      [por L4]
--                           < ε / 2 + ε / 2                [por (6) y (7)
--                           = ε                            [por L5]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
variable {s t : ℕ → ℝ} {a b c : ℝ}

def limite (s : ℕ → ℝ) (a : ℝ) :=
  ∀ ε > 0, ∃ N, ∀ n ≥ N, |s n - a| < ε

-- 1ª demostración
-- ===============

example
```

```
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(u + v) n - (a + b)| < ε
  have hε2 : 0 < ε / 2 := half_pos hε
  cases' hu (ε / 2) hε2 with Nu hNu
  -- Nu : ℕ
  -- hNu : ∀ (n : ℕ), n ≥ Nu → |u n - a| < ε / 2
  cases' hv (ε / 2) hε2 with Nv hNv
  -- Nv : ℕ
  -- hNv : ∀ (n : ℕ), n ≥ Nv → |v n - b| < ε / 2
  clear hu hv hε2 hε
  let N := max Nu Nv
  use N
  -- ⊢ ∀ (n : ℕ), n ≥ N → |(s + t) n - (a + b)| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ N
  have nNu : n ≥ Nu := le_of_max_le_left hn
  specialize hNu n nNu
  -- hNu : |u n - a| < ε / 2
  have nNv : n ≥ Nv := le_of_max_le_right hn
  specialize hNv n nNv
  -- hNv : |v n - b| < ε / 2
  clear hn nNu nNv
  calc |(u + v) n - (a + b)|
        = |(u n + v n) - (a + b)|  := rfl
      _ = |(u n - a) + (v n - b)|  := by { congr; ring }
      _ ≤ |u n - a| + |v n - b|    := by apply abs_add
      _ < ε / 2 + ε / 2            := by linarith [hNu, hNv]
      _ = ε                        := by apply add_halves

-- 2ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε hε
```

```
  cases' hu (ε/2) (by linarith) with Nu hNu
  cases' hv (ε/2) (by linarith) with Nv hNv
  use max Nu Nv
  intros n hn
  have hn₁ : n ≥ Nu := le_of_max_le_left hn
  specialize hNu n hn₁
  have hn₂ : n ≥ Nv := le_of_max_le_right hn
  specialize hNv n hn₂
  calc |(u + v) n - (a + b)|
       = |(u n + v n) - (a + b)|  := by rfl
     _ = |(u n - a) + (v n -  b)| := by {congr; ring}
     _ ≤ |u n - a| + |v n -  b|   := by apply abs_add
     _ < ε / 2 + ε / 2            := by linarith
     _ = ε                        := by apply add_halves

-- 3ª demostración
-- ===============

lemma max_ge_iff
  {α : Type _}
  [LinearOrder α]
  {p q r : α}
  : r ≥ max p q  ↔ r ≥ p ∧ r ≥ q :=
max_le_iff

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε hε
  cases' hu (ε/2) (by linarith) with Nu hNu
  cases' hv (ε/2) (by linarith) with Nv hNv
  use max Nu Nv
  intros n hn
  cases' max_ge_iff.mp hn with hn₁ hn₂
  have cota₁ : |u n - a| < ε/2 := hNu n hn₁
  have cota₂ : |v n - b| < ε/2 := hNv n hn₂
  calc |(u + v) n - (a + b)|
       = |(u n + v n) - (a + b)| := by rfl
     _ = |(u n - a) + (v n - b)| := by { congr; ring }
     _ ≤ |u n - a| + |v n - b|   := by apply abs_add
     _ < ε                       := by linarith

-- 4ª demostración
```

```
-- ===============

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε hε
  cases' hu (ε/2) (by linarith) with Nu hNu
  cases' hv (ε/2) (by linarith) with Nv hNv
  use max Nu Nv
  intros n hn
  cases' max_ge_iff.mp hn with hn₁ hn₂
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|    := by rfl
     _ = |(u n - a) + (v n - b)| := by { congr; ring }
     _ ≤ |u n - a| + |v n - b|    := by apply abs_add
     _ < ε/2 + ε/2                := add_lt_add (hNu n hn₁) (hNv n hn₂)
     _ = ε                        := by simp

-- 5ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε hε
  cases' hu (ε/2) (by linarith) with Nu hNu
  cases' hv (ε/2) (by linarith) with Nv hNv
  use max Nu Nv
  intros n hn
  rw [max_ge_iff] at hn
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|    := by rfl
     _ = |(u n - a) + (v n - b)| := by { congr; ring }
     _ ≤ |u n - a| + |v n - b|    := by apply abs_add
     _ < ε                        := by linarith [hNu n (by linarith), hNv n (by linarith

-- 6ª demostración
-- ===============

example
  (hu : limite u a)
```

```
  (hv : limite v b)
  : limite (u + v) (a + b) :=
by
  intros ε Hε
  cases' hu (ε/2) (by linarith) with L HL
  cases' hv (ε/2) (by linarith) with M HM
  set N := max L M with _hN
  use N
  have HLN : N ≥ L := le_max_left _ _
  have HMN : N ≥ M := le_max_right _ _
  intros n Hn
  have H3 : |u n - a| < ε/2 := HL n (by linarith)
  have H4 : |v n - b| < ε/2 := HM n (by linarith)
  calc |(u + v) n - (a + b)|
       = |(u n + v n) - (a + b)|   := by rfl
     _ = |(u n - a) + (v n - b)|   := by {congr; ring }
     _ ≤ |(u n - a)| + |(v n - b)| := by apply abs_add
     _ < ε/2 + ε/2                 := by linarith
     _ = ε                         := by ring

-- Lemas usados
-- ============

-- variable (d : ℝ)
-- #check (abs_add a b : |a + b| ≤ |a| + |b|)
-- #check (add_halves a : a / 2 + a / 2 = a)
-- #check (add_lt_add : a < b → c < d → a + c < b + d)
-- #check (half_pos : a > 0 → a / 2 > 0)
-- #check (le_max_left a b : a ≤ max a b)
-- #check (le_max_right a b : b ≤ max 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 (max_le_iff : max a b ≤ c ↔ a ≤ c ∧ b ≤ c)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.3. Unicidad del límite de las sucesiones convergentes

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
```

```
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--    λ u a, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - a| < ε
-- donde se usa la notación |x| para el valor absoluto de x
--    notation `|`x`|` := abs x
--
-- Demostrar que cada sucesión tiene como máximo un límite.
-- ----------------------------------------------------------------------

import data.real.basic

variables {u : ℕ → ℝ}
variables {a b : ℝ}

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

-- 1ª demostración
-- ===============

lemma aux
  (ha : limite u a)
  (hb : limite u b)
  : b ≤ a :=
begin
  by_contra h,
  set ε := b - a with hε,
  cases ha (ε/2) (by linarith) with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,
  linarith,
end

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
```

```
le_antisymm (aux hb ha) (aux ha hb)

-- 2ª demostración
-- ===============

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contra h,
  wlog hab : a < b,
  { have : a < b ∨ a = b ∨ b < a := lt_trichotomy a b,
    tauto },
  set ε := b - a with hε,
  specialize ha (ε/2),
  have hε2 : ε/2 > 0 := by linarith,
  specialize ha hε2,
  cases ha with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,
  linarith,
end

-- 3ª demostración
-- ===============

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contra h,
  wlog hab : a < b,
  { have : a < b ∨ a = b ∨ b < a := lt_trichotomy a b,
    tauto },
  set ε := b - a with hε,
  cases ha (ε/2) (by linarith) with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
```

```
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,
  linarith,
end
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.4. Si el límite de la sucesión uₙ es a y c ∈ ℝ, entonces el límite de uₙ+c es a+c

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--      fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
--
-- Demostrar que si el límite de la sucesión uₙ es a y c ∈ ℝ, entonces
-- el límite de uₙ+c es a+c.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos que demostrar que
--    (∃ N)(∀ n ≥ N)[|(u(n) + c) - (a + c)| < ε]                   (1)
-- Puesto que el límite de la sucesión u(i) es a, existe un k tal que
--    (∀ n ≥ k)[|u(n) - a| < ε]                                    (2)
-- Veamos que con k se verifica (1); es decir, que
--    (∀ n ≥ k)[|(u(n) + c) - (a + c)| < ε]
-- Sea n ≥ k. Entonces, por (2),
--    |u(n) - a| < ε                                               (3)
-- y, por consiguiente,
--    |(u(n) + c) - (a + c)| = |u(n) - a|
--                           < ε            [por (3)]

-- Demostraciones con Lean4
-- ========================
```

```
import Mathlib.Data.Real.Basic
import Mathlib.Tactic
variable {u : ℕ → ℝ}
variable {a c : ℝ}

def limite : (ℕ → ℝ) → ℝ → Prop :=
  fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

-- 1ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun i ↦ u i + c) (a + c) :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun i => u i + c) n - (a + c)| < ε
  dsimp
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |u n + c - (a + c)| < ε
  cases' h ε hε with k hk
  -- k : ℕ
  -- hk : ∀ (n : ℕ), n ≥ k → |u n - a| < ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n + c - (a + c)| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  calc |u n + c - (a + c)|
       = |u n - a|         := by norm_num
     _ < ε                 := hk n hn

-- 2ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun i ↦ u i + c) (a + c) :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun i => u i + c) n - (a + c)| < ε
```

```
  dsimp
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |u n + c - (a + c)| < ε
  cases' h ε hε with k hk
  -- k : ℕ
  -- hk : ∀ (n : ℕ), n ≥ k → |u n - a| < ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n + c - (a + c)| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n + c - (a + c)| < ε
  convert hk n hn using 2
  -- ⊢ u n + c - (a + c) = u n - a
  ring

-- 3ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun i ↦ u i + c) (a + c) :=
by
  intros ε hε
  dsimp
  convert h ε hε using 6
  ring

-- 4ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun i ↦ u i + c) (a + c) :=
  fun ε hε ↦ (by convert h ε hε using 6; ring)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.5. Si el límite de la sucesión u_n es a y c ∈ ℝ, entonces el límite de cu_n es ca

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--      fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
--
-- 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 Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (u v : ℕ → ℝ)
variable (a c : ℝ)

def limite : (ℕ → ℝ) → ℝ → Prop :=
  fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
```

```
-- 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)| := congr_arg 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_cancel' 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_iff' hc']
    -- ⊢ |u n - a| < ε / |c|
    exact hN
```

```
-- 3ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun n ↦ c * (u n)) (c * a) :=
by
  by_cases hc : c = 0
  . subst hc
    intros ε hε
    aesop
  . intros ε hε
    have hc' : 0 < |c| := by aesop
    have hεc : 0 < ε / |c| := div_pos hε hc'
    cases' h (ε/|c|) hεc with N hN
    use N
    intros n hn
    specialize hN n hn
    dsimp only
    rw [← mul_sub, abs_mul, ← lt_div_iff' hc']
    exact hN

-- Lemas usados
-- ============

-- variable (b c : ℝ)
-- #check (abs_mul a b : |a * b| = |a| * |b|)
-- #check (abs_pos.mpr : a ≠ 0 → 0 < |a|)
-- #check (div_pos : 0 < a → 0 < b → 0 < a / b)
-- #check (lt_div_iff' : 0 < c → (a < b / c ↔ c * a < b))
-- #check (mul_div_cancel' 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)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.6. El límite de u es a syss el de u-a es 0

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
```

```
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--     fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
--
-- Demostrar que el límite de uₙ es a si y solo si el de uₙ-a es 0.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Se prueba por la siguiente cadena de equivalencias
--    limite u a ↔ (∀ε>0)(∃N)(∀n≥N)[|u(n) - a| < ε]
--               ↔ (∀ε>0)(∃N)(∀n≥N)[|(u(n) - a) - 0| < ε]
--               ↔ limite (fun n ↦ u(n) - a) 0

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable  {u : ℕ → ℝ}
variable {a c x : ℝ}

def limite : (ℕ → ℝ) → ℝ → Prop :=
 fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

-- 1ª demostración
-- ===============

example
  : limite u a ↔ limite (fun i ↦ u i - a) 0 :=
by
  rw [iff_eq_eq]
  calc limite u a
       = ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - a| < ε       := rfl
     _ = ∀ ε > 0, ∃ N, ∀ n ≥ N, |(u n - a) - 0| < ε := by simp
     _ = limite (fun i ↦ u i - a) 0                := rfl

-- 2ª demostración
-- ===============

example
  : limite u a ↔ limite (fun i ↦ u i - a) 0 :=
by
  constructor
```

```
  . -- ⊢ limite u a → limite (fun i => u i - a) 0
    intros h ε hε
    -- h : limite u a
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun i => u i - a) n - 0| < ε
    convert h ε hε using 2
    -- x : ℕ
    -- ⊢ (∀ (n : ℕ), n ≥ x → |(fun i => u i - a) n - 0| < ε) ↔ ∀ (n : ℕ), n ≥ x → |u n -
    norm_num
  . -- ⊢ limite (fun i => u i - a) 0 → limite u a
    intros h ε hε
    -- h : limite (fun i => u i - a) 0
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε
    convert h ε hε using 2
    -- x : ℕ
    -- ⊢ (∀ (n : ℕ), n ≥ x → |u n - a| < ε) ↔ ∀ (n : ℕ), n ≥ x → |(fun i => u i - a) n -
    norm_num

-- 3ª demostración
-- ===============

example
  : limite u a ↔ limite (fun i ↦ u i - a) 0 :=
by
  constructor <;>
  { intros h ε hε
    convert h ε hε using 2
    norm_num }

-- 4ª demostración
-- ===============

lemma limite_con_suma
  (c : ℝ)
  (h : limite u a)
  : limite (fun i ↦ u i + c) (a + c) :=
  fun ε hε ↦ (by convert h ε hε using 2; norm_num)

lemma CNS_limite_con_suma
  (c : ℝ)
  : limite u a ↔ limite (fun i ↦ u i + c) (a + c) :=
by
```

```
  constructor
  . -- ⊢ limite u a → limite (fun i => u i + c) (a + c)
    apply limite_con_suma
  . -- ⊢ limite (fun i => u i + c) (a + c) → limite u a
    intro h
    -- h : limite (fun i => u i + c) (a + c)
    -- ⊢ limite u a
    convert limite_con_suma (-c) h using 2
    . -- ⊢ u x = u x + c + -c
      simp
    . -- ⊢ a = a + c + -c
      simp

example
  (u : ℕ → ℝ)
  (a : ℝ)
  : limite u a ↔ limite (fun i ↦ u i - a) 0 :=
by
  convert CNS_limite_con_suma (-a) using 2
  -- ⊢ 0 = a + -a
  simp

-- Lemas usados
-- ============

-- variable (p q : Prop)
-- #check (iff_eq_eq : (p ↔ q) = (p = q))
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.7. Si u_n y v_n convergen a 0, entonces $u_n v_n$ converge a 0

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--      fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
--
-- Demostrar que si las sucesiones u(n) y v(n) convergen a cero,
```

```
-- entonces u(n)·v(n) también converge a cero.
-- ----------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos ue demostrar que
--    (∃N ∈ ℕ)(∀n ≥ N)[|(uv)(n) - 0| < ε]                         (1)
-- Puesto que el límite de uₙ es 0, existe un U ∈ ℕ tal que
--    (∀n ≥ U)[|u(n) - 0| < ε]                                        (2)
-- y, puesto que el límite de vₙ es 0, existe un V ∈ ℕ tal que
--    (∀n ≥ V)[|v(n) - 0| < 1]                                        (3)
-- Entonces, N = max(U, V) cumple (1). En efecto, sea n ≥ N. Entonces,
-- n ≥ U y n ≥ V y, aplicando (2) y (3), se tiene
--    |u(n) - 0| < ε                                                  (4)
--    |v(n) - 0| < 1                                                  (5)
-- Por tanto,
--    |(u·v)(n) - 0| = |u(n)·v(n)|
--                   = |u(n)|·|v n|
--                   < ε·1            [por (4) y (5)]
--                   = ε

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable {u v : ℕ → ℝ}

def limite : (ℕ → ℝ) → ℝ → Prop :=
  fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

-- 1ª demostración
-- ===============

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
```

```
  cases' hu ε hε with U hU
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - 0| < ε
  cases' hv 1 zero_lt_one with V hV
  -- V : ℕ
  -- hV : ∀ (n : ℕ), n ≥ V → |v n - 0| < 1
  let N := max U V
  use N
  -- ⊢ ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ N
  -- ⊢ |(u * v) n - 0| < ε
  specialize hU n (le_of_max_le_left hn)
  -- hU : |u n - 0| < ε
  specialize hV n (le_of_max_le_right hn)
  -- hV : |v n - 0| < 1
  rw [sub_zero] at *
  -- hU : |u n - 0| < ε
  -- hV : |v n - 0| < 1
  -- ⊢ |(u * v) n - 0| < ε
  calc |(u * v) n|
       = |u n * v n|   := rfl
     _ = |u n| * |v n| := abs_mul (u n) (v n)
     _ < ε * 1         := mul_lt_mul'' hU hV (abs_nonneg (u n)) (abs_nonneg (v n))
     _ = ε             := mul_one ε

-- 2ª demostración
-- ===============

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
  cases' hu ε hε with U hU
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - 0| < ε
  cases' hv 1 (by linarith) with V hV
  -- V : ℕ
  -- hV : ∀ (n : ℕ), n ≥ V → |v n - 0| < 1
```

```
  let N := max U V
  use N
  -- ⊢ ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ N
  -- ⊢ |(u * v) n - 0| < ε
  specialize hU n (le_of_max_le_left hn)
  -- hU : |u n - 0| < ε
  specialize hV n (le_of_max_le_right hn)
  -- hV : |v n - 0| < 1
  rw [sub_zero] at *
  -- hU : |u n| < ε
  -- hV : |v n| < 1
  -- ⊢ |(u * v) n| < ε
  calc |(u * v) n|
       = |u n * v n|    := rfl
     _ = |u n| * |v n| := abs_mul (u n) (v n)
     _ < ε * 1          := by { apply mul_lt_mul'' hU hV <;> simp [abs_nonneg] }
     _ = ε              := mul_one ε

-- 3ª demostración
-- ===============

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
  cases' hu ε hε with U hU
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - 0| < ε
  cases' hv 1 (by linarith) with V hV
  -- V : ℕ
  -- hV : ∀ (n : ℕ), n ≥ V → |v n - 0| < 1
  let N := max U V
  use N
  -- ⊢ ∀ (n : ℕ), n ≥ N → |(u * v) n - 0| < ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ N
```

```
  -- ⊢ |(u * v) n - 0| < ε
  have hUN : U ≤ N := le_max_left U V
  have hVN : V ≤ N := le_max_right U V
  specialize hU n (by linarith)
  -- hU : |u n - 0| < ε
  specialize hV n (by linarith)
  -- hV : |v n - 0| < 1
  rw [sub_zero] at *
  -- hU : |u n| < ε
  -- hV : |v n| < 1
  -- ⊢ |(u * v) n| < ε
  rw [Pi.mul_apply]
  -- ⊢ |u n * v n| < ε
  rw [abs_mul]
  -- ⊢ |u n| * |v n| < ε
  convert mul_lt_mul'' hU hV _ _ using 2 <;> simp

-- Lemas usados
-- ============

-- variable (a b c d : ℝ)
-- variable (I : Type _)
-- variable (f : I → Type _)
-- #check (zero_lt_one : 0 < 1)
-- #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 (sub_zero a : a - 0 = a)
-- #check (abs_mul a b : |a * b| = |a| * |b|)
-- #check (mul_lt_mul'' : a < c → b < d → 0 ≤ a → 0 ≤ b → a * b < c * d)
-- #check (abs_nonneg a : 0 ≤ |a|)
-- #check (mul_one a : a * 1 = a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.8. Teorema del emparedado

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--      fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε
```

```
--
-- Demostrar que si para todo n, u(n) ≤ v(n) ≤ w(n) y u(n) tiene el
-- mismo límite que w(n), entonces v(n) también tiene dicho límite.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Tenemos que demostrar que para cada ε > 0, existe un N ∈ ℕ tal que
--    (∀ n ≥ N)[|v(n) - a| ≤ ε]                                     (1)
--
-- Puesto que el límite de u es a, existe un U ∈ ℕ tal que
--    (∀ n ≥ U)[|u(n) - a| ≤ ε]                                     (2)
-- y, puesto que el límite de w es a, existe un W ∈ ℕ tal que
--    (∀ n ≥ W)[|w(n) - a| ≤ ε]                                     (3)
-- Sea N = máx(U, W). Veamos que se verifica (1). Para ello, sea
-- n ≥ N. Entonces, n ≥ U y n ≥ W. Por (2) y (3), se tiene que
--     |u(n) - a| ≤ ε                                               (4)
--     |w(n) - a| ≤ ε                                               (5)
-- Para demostrar que
--     |v(n) - a| ≤ ε
-- basta demostrar las siguientes desigualdades
--     -ε ≤ v(n) - a                                                (6)
--      v(n) - a ≤ ε                                                (7)
-- La demostración de (6) es
--    -ε ≤ u(n) - a    [por (4)]
--       ≤ v(n) - a    [por hipótesis]
-- La demostración de (7) es
--    v(n) - a ≤ w(n) - a    [por hipótesis]
--             ≤ ε           [por (5)]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (u v w : ℕ → ℝ)
variable (a : ℝ)

def limite : (ℕ → ℝ) → ℝ → Prop :=
  fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- Nota. En la demostración se usará el siguiente lema:
lemma max_ge_iff
  {p q r : ℕ}
```

```
  : r ≥ max p q ↔ r ≥ p ∧ r ≥ q :=
  max_le_iff

-- 1ª demostración
-- ===============

example
  (hu : limite u a)
  (hw : limite w a)
  (h1 : ∀ n, u n ≤ v n)
  (h2 : ∀ n, v n ≤ w n) :
  limite v a :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |v n - a| ≤ ε
  rcases hu ε hε with ⟨U, hU⟩
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - a| ≤ ε
  clear hu
  rcases hw ε hε with ⟨W, hW⟩
  -- W : ℕ
  -- hW : ∀ (n : ℕ), n ≥ W → |w n - a| ≤ ε
  clear hw hε
  use max U W
  intros n hn
  -- n : ℕ
  -- hn : n ≥ max U W
  -- ⊢ |v n - a| ≤ ε
  rw [max_ge_iff] at hn
  -- hn : n ≥ U ∧ n ≥ W
  specialize hU n hn.1
  -- hU : |u n - a| ≤ ε
  specialize hW n hn.2
  -- hW : |w n - a| ≤ ε
  specialize h1 n
  -- h1 : u n ≤ v n
  specialize h2 n
  -- h2 : v n ≤ w n
  clear hn
  rw [abs_le] at *
  -- ⊢ -ε ≤ v n - a ∧ v n - a ≤ ε
  constructor
  . -- ⊢ -ε ≤ v n - a
```

```
    calc -ε
         ≤ u n - a := hU.1
       _ ≤ v n - a := by linarith
  . -- ⊢ v n - a ≤ ε
    calc v n - a
         ≤ w n - a := by linarith
       _ ≤ ε       := hW.2

-- 2ª demostración
example
  (hu : limite u a)
  (hw : limite w a)
  (h1 : ∀ n, u n ≤ v n)
  (h2 : ∀ n, v n ≤ w n) :
  limite v a :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |v n - a| ≤ ε
  rcases hu ε hε with ⟨U, hU⟩
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - a| ≤ ε
  clear hu
  rcases hw ε hε with ⟨W, hW⟩
  -- W : ℕ
  -- hW : ∀ (n : ℕ), n ≥ W → |w n - a| ≤ ε
  clear hw hε
  use max U W
  intros n hn
  -- n : ℕ
  -- hn : n ≥ max U W
  rw [max_ge_iff] at hn
  -- hn : n ≥ U ∧ n ≥ W
  specialize hU n (by linarith)
  -- hU : |u n - a| ≤ ε
  specialize hW n (by linarith)
  -- hW : |w n - a| ≤ ε
  specialize h1 n
  -- h1 : u n ≤ v n
  specialize h2 n
  -- h2 : v n ≤ w n
  rw [abs_le] at *
  -- ⊢ -ε ≤ v n - a ∧ v n - a ≤ ε
  constructor
```

```
  . -- ⊢ -ε ≤ v n - a
    linarith
  . -- ⊢ v n - a ≤ ε
    linarith

-- 3ª demostración
example
  (hu : limite u a)
  (hw : limite w a)
  (h1 : ∀ n, u n ≤ v n)
  (h2 : ∀ n, v n ≤ w n) :
  limite v a :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |v n - a| ≤ ε
  rcases hu ε hε with ⟨U, hU⟩
  -- U : ℕ
  -- hU : ∀ (n : ℕ), n ≥ U → |u n - a| ≤ ε
  clear hu
  rcases hw ε hε with ⟨W, hW⟩
  -- W : ℕ
  -- hW : ∀ (n : ℕ), n ≥ W → |w n - a| ≤ ε
  clear hw hε
  use max U W
  intros n hn
  -- n : ℕ
  -- hn : n ≥ max U W
  -- ⊢ |v n - a| ≤ ε
  rw [max_ge_iff] at hn
  -- hn : n ≥ U ∧ n ≥ W
  specialize hU n (by linarith)
  -- hU : |u n - a| ≤ ε
  specialize hW n (by linarith)
  -- hW : |w n - a| ≤ ε
  specialize h1 n
  -- h1 : u n ≤ v n
  specialize h2 n
  -- h2 : v n ≤ w n
  rw [abs_le] at *
  -- hU : -ε ≤ u n - a ∧ u n - a ≤ ε
  -- hW : -ε ≤ w n - a ∧ w n - a ≤ ε
  -- ⊢ -ε ≤ v n - a ∧ v n - a ≤ ε
  constructor <;> linarith
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web

17.9. Los supremos de las sucesiones crecientes son sus límites

```
-- ---------------------------------------------------------------------
-- Sea u una sucesión creciente. Demostrar que si S es un supremo de u,
-- entonces el límite de u es S.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos que demostrar que
--    (∃ m ∈ ℕ)(∀ n ∈ ℕ)[n ≥ m → |uₙ - S| ≤ ε]                         (1)
--
-- Por ser S un supremo de u, existe un k ∈ ℕ tal que
--    uₖ ≥ S - ε                                                       (2)
-- Vamos a demostrar que k verifica la condición de (1); es decir, que
-- si n ∈ ℕ tal que n ≥ k, entonces
--    |uₙ - S| ≤ ε
-- o, equivalentemente,
--    -ε ≤ uₙ - S ≤ ε
--
-- La primera desigualdad se tiene por la siguente cadena:
--    -ε = (S - ε) - S
--       ≤ uₖ - S         [por (2)]
--       ≤ uₙ - S         [porque u es creciente y n ≥ k]
--
-- La segunda desigualdad se tiene por la siguente cadena:
--    uₙ - S ≤ S - S      [porque S es un supremo de u]
--           = 0
--           ≤ ε

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (u : ℕ → ℝ)
variable (S : ℝ)
```

```
-- (limite u c) expresa que el límite de u es c.
def limite (u : ℕ → ℝ) (c : ℝ) :=
  ∀ ε > 0, ∃ m, ∀ n ≥ m, |u n - c| ≤ ε

-- (supremo u S) expresa que el supremo de u es S.
def supremo (u : ℕ → ℝ) (S : ℝ) :=
  (∀ n, u n ≤ S) ∧ ∀ ε > 0, ∃ k, u k ≥ S - ε

-- 1ª demostración
-- ===============

example
  (hu : Monotone u)
  (hS : supremo u S)
  : limite u S :=
by
  unfold limite
  -- ⊢ ∀ (ε : ℝ), ε > 0 → ∃ m, ∀ (n : ℕ), n ≥ m → |u n - S| ≤ ε
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ m, ∀ (n : ℕ), n ≥ m → |u n - S| ≤ ε
  unfold supremo at hS
  -- hS : (∀ (n : ℕ), u n ≤ S) ∧ ∀ (ε : ℝ), ε > 0 → ∃ k, u k ≥ S - ε
  cases' hS with hS₁ hS₂
  -- hS₁ : ∀ (n : ℕ), u n ≤ S
  -- hS₂ : ∀ (ε : ℝ), ε > 0 → ∃ k, u k ≥ S - ε
  cases' hS₂ ε hε with k hk
  -- k : ℕ
  -- hk : u k ≥ S - ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n - S| ≤ ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n - S| ≤ ε
  rw [abs_le]
  -- ⊢ -ε ≤ u n - S ∧ u n - S ≤ ε
  constructor
  . -- ⊢ -ε ≤ u n - S
    unfold Monotone at hu
    -- hu : ∀ ⦃a b : ℕ⦄, a ≤ b → u a ≤ u b
    specialize hu hn
    -- hu : u k ≤ u n
    calc -ε
```

```
        = (S - ε) - S := by ring
      _ ≤ u k - S     := sub_le_sub_right hk S
      _ ≤ u n - S     := sub_le_sub_right hu S
  . calc u n - S
        ≤ S - S       := sub_le_sub_right (hS₁ n) S
      _ = 0           := sub_self S
      _ ≤ ε           := le_of_lt hε

-- 2ª demostración
-- ===============

example
  (hu : Monotone u)
  (hS : supremo u S)
  : limite u S :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ m, ∀ (n : ℕ), n ≥ m → |u n - S| ≤ ε
  cases' hS with hS₁ hS₂
  -- hS₁ : ∀ (n : ℕ), u n ≤ S
  -- hS₂ : ∀ (ε : ℝ), ε > 0 → ∃ k, u k ≥ S - ε
  cases' hS₂ ε hε with k hk
  -- k : ℕ
  -- hk : u k ≥ S - ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n - S| ≤ ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n - S| ≤ ε
  rw [abs_le]
  -- ⊢ -ε ≤ u n - S ∧ u n - S ≤ ε
  constructor
  . -- ⊢ -ε ≤ u n - S
    linarith [hu hn]
  . -- ⊢ u n - S ≤ ε
    linarith [hS₁ n]

-- 3ª demostración
-- ===============

example
  (hu : Monotone u)
```

```
  (hS : supremo u S)
  : limite u S :=
by
  intros ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ m, ∀ (n : ℕ), n ≥ m → |u n - S| ≤ ε
  cases' hS with hS₁ hS₂
  -- hS₁ : ∀ (n : ℕ), u n ≤ S
  -- hS₂ : ∀ (ε : ℝ), ε > 0 → ∃ k, u k ≥ S - ε
  cases' hS₂ ε hε with k hk
  -- k : ℕ
  -- hk : u k ≥ S - ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |u n - S| ≤ ε
  intros n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |u n - S| ≤ ε
  rw [abs_le]
  -- ⊢ -ε ≤ u n - S ∧ u n - S ≤ ε
  constructor <;> linarith [hu hn, hS₁ n]

-- Lemas usados
-- ============

-- variable (a b : ℝ)
-- #check (abs_le : |a| ≤ b ↔ -b ≤ a ∧ a ≤ b)
-- #check (le_of_lt : a < b → a ≤ b)
-- #check (sub_le_sub_right : a ≤ b → ∀ (c : ℝ), a - c ≤ b - c)
-- #check (sub_self a : a - a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

17.10. Las sucesiones convergentes están acotadas

```
-- ---------------------------------------------------------------------
-- 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
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable {u : ℕ → ℝ}

-- (limite u c) expresa que el límite de u es c.
def limite (u : ℕ → ℝ) (c : ℝ) :=
  ∀ ε > 0, ∃ k, ∀ n ≥ k, |u n - c| ≤ ε

-- (convergente u) expresa que u es convergente.
def convergente (u : ℕ → ℝ) :=
  ∃ a, limite u a

-- 1ª demostración
-- ===============

example
  (h : convergente u)
  : ∃ k b, ∀ n, n ≥ k → |u n| ≤ b :=
by
  cases' h with a ua
  -- a : ℝ
  -- ua : limite u a
  cases' 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
  -- ⊢ |u n| ≤ 1 + |a|
  calc |u n|
       = |u n - a + a|   := congr_arg abs (eq_add_of_sub_eq rfl)
     _ ≤ |u n - a| + |a| := abs_add (u n - a) a
     _ ≤ 1 + |a|         := add_le_add_right h |a|

-- 2ª demostración
-- ===============

example
  (h : convergente u)
  : ∃ k b, ∀ n, n ≥ k → |u n| ≤ b :=
by
  cases' h with a ua
  -- a : ℝ
  -- ua : limite u a
  cases' 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)
-- #check (abs_add a b : |a + b| ≤ |a| + |b|)
```

```
-- #check (add_le_add_right : b ≤ c → ∀ a,  b + a ≤ c + a)
-- #check (eq_add_of_sub_eq :  a - c = b → a = b + c)
-- #check (zero_lt_one : 0 < 1)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

17.11. Si $(\forall n)[u_n \leq v_n]$, entonces $\lim u_n \leq \lim v_n$

```
-- ---------------------------------------------------------------------
-- En Lean, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a límite de la sucesión u, por
--    def limite (u : ℕ → ℝ) (c : ℝ) :=
--      ∀ ε > 0, ∃ k, ∀ n ≥ k, |u n - c| < ε
--
-- Demostrar que si (∀ n)[uₙ ≤ vₙ], a es límite de uₙ y c es límite de vₙ,
-- entonces a ≤ c.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Por reduccion al absurdo. Supongamos que a ≰ c. Entonces,
--    c < a                                                           (1)
-- Sea
--    ε = (a - c)/2                                                   (2)
-- Por (1),
--    ε > 0.
-- Por tanto, puesto que a es límite de uₙ, existe un p ∈ ℕ tal que
--    (∀ n)[n ≥ p → |uₙ - a| < ε]                                     (3)
-- Análogamente, puesto que c es límite de vₙ, existe un q ∈ ℕ tal
-- que
--    (∀ n)[n ≥ q → |vₙ - c| < ε]                                     (4)
-- Sea
--    k = max(p, q)
-- Entonces, k ≥ p y, por (3),
--   |uₖ - a| < ε                                                     (5)
-- Análogamente, k ≥ q y, por (4),
--   |vₖ - c| < ε                                                     (6)
-- Además, por la hipótesis,
--   uₖ ≤ vₖ                                                          (7)
-- Por tanto,
```

```
--    a - c = (a - uₖ) + (uₖ - c)
--          ≤ (a - uₖ) + (vₖ - c)      [por (7)]
--          ≤ |(a - uₖ) + (vₖ - c)|
--          ≤ |a - uₖ| + |vₖ - c|
--          = |uₖ - a| + |vₖ - c|
--          < ε + ε                    [por (5) y (6)]
--          = a - c                    [por (2)]
-- Luego,
--    a - c < a - c
-- que es una contradicción.

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic

variable (u v : ℕ → ℝ)
variable (a c : ℝ)

def limite (u : ℕ → ℝ) (c : ℝ) :=
  ∀ ε > 0, ∃ k, ∀ n ≥ k, |u n - c| < ε

-- 1ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v c)
  (huv : ∀ n, u n ≤ v n)
  : a ≤ c :=
by
  by_contra h
  -- h : ¬a ≤ c
  -- ⊢ False
  have hca : c < a := not_le.mp h
  set ε := (a - c) /2
  have hε : 0 < ε := half_pos (sub_pos.mpr hca)
  obtain ⟨ku, hku : ∀ n, n ≥ ku → |u n - a| < ε⟩ := hu ε hε
  obtain ⟨kv, hkv : ∀ n, n ≥ kv → |v n - c| < ε⟩ := hv ε hε
  let k := max ku kv
  have hku' : ku ≤ k := le_max_left ku kv
  have hkv' : kv ≤ k := le_max_right ku kv
  have ha : |u k - a| < ε := hku k hku'
  have hc : |v k - c| < ε := hkv k hkv'
  have hk : u k - c ≤ v k - c := sub_le_sub_right (huv k) c
```

```
  have hac1 : a - c < a - c := by
    calc a - c
         = (a - u k) + (u k - c)   := by ring
       _ ≤ (a - u k) + (v k - c)   := add_le_add_left hk (a - u k)
       _ ≤ |(a - u k) + (v k - c)| := le_abs_self ((a - u k) + (v k - c))
       _ ≤ |a - u k| + |v k - c|   := abs_add (a - u k) (v k - c)
       _ = |u k - a| + |v k - c|   := by simp only [abs_sub_comm]
       _ < ε + ε                   := add_lt_add ha hc
       _ = a - c                   := add_halves (a - c)
  have hac2 : ¬ a - c < a -c := lt_irrefl (a - c)
  show False
  exact hac2 hac1

-- 2ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v c)
  (huv : ∀ n, u n ≤ v n)
  : a ≤ c :=
by
  by_contra h
  -- h : ¬a ≤ c
  -- ⊢ False
  have hca : c < a := not_le.mp h
  set ε := (a - c) /2 with hε
  obtain ⟨ku, hku : ∀ n, n ≥ ku → |u n - a| < ε⟩ := hu ε (by linarith)
  obtain ⟨kv, hkv : ∀ n, n ≥ kv → |v n - c| < ε⟩ := hv ε (by linarith)
  let k := max ku kv
  have ha : |u k - a| < ε := hku k (le_max_left ku kv)
  have hc : |v k - c| < ε := hkv k (le_max_right ku kv)
  have hk : u k - c ≤ v k - c := sub_le_sub_right (huv k) c
  have hac1 : a - c < a -c := by
    calc a - c
         = (a - u k) + (u k - c)   := by ring
       _ ≤ (a - u k) + (v k - c)   := add_le_add_left hk (a - u k)
       _ ≤ |(a - u k) + (v k - c)| := le_abs_self ((a - u k) + (v k - c))
       _ ≤ |a - u k| + |v k - c|   := abs_add (a - u k) (v k - c)
       _ = |u k - a| + |v k - c|   := by simp only [abs_sub_comm]
       _ < ε + ε                   := add_lt_add ha hc
       _ = a - c                   := add_halves (a - c)
  have hac2 : ¬ a - c < a -c := lt_irrefl (a - c)
  show False
  exact hac2 hac1
```

```
-- 3ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v c)
  (huv : ∀ n, u n ≤ v n)
  : a ≤ c :=
by
  by_contra h
  -- h : ¬a ≤ c
  -- ⊢ False
  have hca : c < a := not_le.mp h
  set ε := (a - c) /2 with hε
  obtain ⟨ku, hku : ∀ n, n ≥ ku → |u n - a| < ε⟩ := hu ε (by linarith)
  obtain ⟨kv, hkv : ∀ n, n ≥ kv → |v n - c| < ε⟩ := hv ε (by linarith)
  let k := max ku kv
  have ha : |u k - a| < ε := hku k (le_max_left ku kv)
  have hc : |v k - c| < ε := hkv k (le_max_right ku kv)
  have hk : u k - c ≤ v k - c := sub_le_sub_right (huv k) c
  have hac1 : a - c < a -c := by
    calc a - c
         = (a - u k) + (u k - c)   := by ring
       _ ≤ (a - u k) + (v k - c)   := add_le_add_left hk (a - u k)
       _ ≤ |(a - u k) + (v k - c)| := by simp [le_abs_self]
       _ ≤ |a - u k| + |v k - c|   := by simp [abs_add]
       _ = |u k - a| + |v k - c|   := by simp [abs_sub_comm]
       _ < ε + ε                   := add_lt_add ha hc
       _ = a - c                   := by simp
  have hac2 : ¬ a - c < a -c := lt_irrefl (a - c)
  show False
  exact hac2 hac1

-- 4ª demostración
-- ===============

example
  (hu : limite u a)
  (hv : limite v c)
  (huv : ∀ n, u n ≤ v n)
  : a ≤ c :=
by
  apply le_of_not_lt
  -- ⊢ ¬c < a
```

```
  intro hca
  -- hca : c < a
  -- ⊢ False
  set ε := (a - c) /2 with hε
  cases' hu ε (by linarith) with ku hku
  -- ku : ℕ
  -- hku : ∀ (n : ℕ), n ≥ ku → |u n - a| < ε
  cases' hv ε (by linarith) with kv hkv
  -- kv : ℕ
  -- hkv : ∀ (n : ℕ), n ≥ kv → |v n - c| < ε
  let k := max ku kv
  have ha : |u k - a| < ε := hku k (le_max_left ku kv)
  have hc : |v k - c| < ε := hkv k (le_max_right ku kv)
  have hk : u k ≤ v k := huv k
  apply lt_irrefl (a - c)
  -- ⊢ a - c < a - c
  rw [abs_lt] at ha hc
  -- ha : -ε < u k - a ∧ u k - a < ε
  -- hc : -ε < v k - c ∧ v k - c < ε
  linarith

-- Lemas usados
-- ============

-- variable (b d : ℝ)
-- #check (abs_add a b : |a + b| ≤ |a| + |b|)
-- #check (abs_lt: |a| < b ↔ -b < a ∧ a < b)
-- #check (abs_sub_comm a b : |a - b| = |b - a|)
-- #check (add_halves a : a / 2 + a / 2 = a)
-- #check (add_le_add_left : b ≤ c → ∀ a, a + b ≤ a + c)
-- #check (add_lt_add : a < b → c < d → a + c < b + d)
-- #check (half_pos : 0 < a → 0 < a / 2)
-- #check (le_abs_self a : a ≤ |a|)
-- #check (le_max_left a b : a ≤ max a b)
-- #check (le_max_right a b : b ≤ max a b)
-- #check (le_of_not_lt :  ¬b < a → a ≤ b)
-- #check (lt_irrefl a : ¬a < a)
-- #check (not_le : ¬a ≤ b ↔ b < a)
-- #check (sub_le_sub_right : a ≤ b → ∀ c, a - c ≤ b - c)
-- #check (sub_pos : 0 < a - b ↔ b < a)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

17.12. Si u_n está acotada y lim $v_n = 0$, entonces lim $(u_n \cdot v_n) = 0$

```
-- ---------------------------------------------------------------------
-- Demostrar que si uₙ está acotada y lim vₙ = 0, entonces
-- lim (u·v)ₙ = 0.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos que demostrar
--    (∃ k)(∀ n)[n ≥ k → |(u·v)ₙ - 0| < ε]                          (1)
--
-- Puesto que la sucesión u está acotada, existe un B ∈ ℝ tal que
--    (∀ n ∈ ℕ) |uₙ| ≤ B                                             (2)
-- Luego B ≥ 0. Lo demostraremos por caso según que B = 0 o B > 0.
--
-- Caso 1: Supongamos que B = 0. Entonces, por (2),
--    (∀ n ∈ ℕ) |uₙ| ≤ 0
-- Luego,
--    (∀ n ∈ ℕ) uₙ = 0                                               (3)
-- Para demostrar (1), para basta tomar 0 como k, ya que si n ≥ 0,
-- entonces
--    |(u·v)ₙ - 0| = |uₙ·vₙ|
--                 = |0·vₙ|     [por (3)
--                 = 0
--                 < ε
--
-- Caso 2: Supongamos que B > 0. Entonces, ε/B > 0 y, puesto que
-- lim vₙ = 0, existe un k ∈ ℕ tal que
--    (∀ n)[n ≥ k → |vₙ - 0| < ε/B]                                  (4)
-- Para demostrar (1), para basta el mismo k, ya que si n ≥ k,
-- entonces
--    |(u·v)ₙ - 0| = |uₙ·vₙ|
--                 = |uₙ|·|vₙ|
--                 ≤ B·|vₙ|       [por (2)]
--                 < B·(ε/B)      [por (4)]
--                 = ε

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
```

```
import Mathlib.Tactic

variable (u v : ℕ → ℝ)
variable (a : ℝ)

def limite (u : ℕ → ℝ) (c : ℝ) :=
  ∀ ε > 0, ∃ k, ∀ n ≥ k, |u n - c| < ε

def acotada (a : ℕ → ℝ) :=
  ∃ B, ∀ n, |a n| ≤ B

-- 1ª demostración
-- ===============

example
  (hU : acotada u)
  (hV : limite v 0)
  : limite (u*v) 0 :=
by
  cases' hU with B hB
  -- B : ℝ
  -- hB : ∀ (n : ℕ), |u n| ≤ B
  have hBnoneg : 0 ≤ B :=
    calc 0 ≤ |u 0| := abs_nonneg (u 0)
         _ ≤ B     := hB 0
  by_cases hB0 : B = 0
  . -- hB0 : B = 0
    subst hB0
    -- hB : ∀ (n : ℕ), |u n| ≤ 0
    -- hBnoneg : 0 ≤ 0
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ k, ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    use 0
    -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(u * v) n - 0| < ε
    intros n _hn
    -- n : ℕ
    -- hn : n ≥ 0
    -- ⊢ |(u * v) n - 0| < ε
    simp_rw [sub_zero] at *
    -- ⊢ |(u * v) n| < ε
    calc |(u * v) n|
         = |u n * v n|   := congr_arg abs (Pi.mul_apply u v n)
       _ = |u n| * |v n| := abs_mul (u n) (v n)
```

```
        _ ≤ 0 * |v n|       := mul_le_mul_of_nonneg_right (hB n) (abs_nonneg (v n))
        _ = 0               := zero_mul (|v n|)
        _ < ε               := hε
  . -- hB0 : ¬B = 0
    change B ≠ 0 at hB0
    -- hB0 : B ≠ 0
    have hBpos : 0 < B := (Ne.le_iff_lt hB0.symm).mp hBnoneg
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ k, ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    cases' hV (ε/B) (div_pos hε hBpos) with k hk
    -- k : ℕ
    -- hk : ∀ (n : ℕ), n ≥ k → |v n - 0| < ε / B
    use k
    -- ⊢ ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    intros n hn
    -- n : ℕ
    -- hn : n ≥ k
    -- ⊢ |(u * v) n - 0| < ε
    simp_rw [sub_zero] at *
    -- ⊢ |(u * v) n| < ε
    calc |(u * v) n|
         = |u n * v n|       := congr_arg abs (Pi.mul_apply u v n)
       _ = |u n| * |v n|     := abs_mul (u n) (v n)
       _ ≤ B * |v n|         := mul_le_mul_of_nonneg_right (hB n) (abs_nonneg _)
       _ < B * (ε/B)         := mul_lt_mul_of_pos_left (hk n hn) hBpos
       _ = ε                 := mul_div_cancel' ε hB0

-- 2ª demostración
-- ===============

example
  (hU : acotada u)
  (hV : limite v 0)
  : limite (u*v) 0 :=
by
  cases' hU with B hB
  -- B : ℝ
  -- hB : ∀ (n : ℕ), |u n| ≤ B
  have hBnoneg : 0 ≤ B :=
    calc 0 ≤ |u 0| := abs_nonneg (u 0)
         _ ≤ B     := hB 0
  by_cases hB0 : B = 0
  . subst hB0
```

```
    -- hB : ∀ (n : ℕ), |u n| ≤ 0
    -- hBnoneg : 0 ≤ 0
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ k, ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    use 0
    -- ⊢ ∀ (n : ℕ), n ≥ 0 → |(u * v) n - 0| < ε
    intros n _hn
    -- n : ℕ
    -- _hn : n ≥ 0
    -- ⊢ |(u * v) n - 0| < ε
    simp_rw [sub_zero] at *
    -- ⊢ |(u * v) n| < ε
    calc |(u * v) n|
         = |u n| * |v n| := by aesop
       _ ≤ 0 * |v n|     := mul_le_mul_of_nonneg_right (hB n) (abs_nonneg (v n))
       _ = 0             := by ring
       _ < ε             := hε
  . -- hB0 : ¬B = 0
    change B ≠ 0 at hB0
    -- hB0 : B ≠ 0
    have hBpos : 0 < B := (Ne.le_iff_lt hB0.symm).mp hBnoneg
    intros ε hε
    -- ε : ℝ
    -- hε : ε > 0
    -- ⊢ ∃ k, ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    cases' hV (ε/B) (div_pos hε hBpos) with k hk
    -- k : ℕ
    -- hk : ∀ (n : ℕ), n ≥ k → |v n - 0| < ε / B
    use k
    -- ∀ (n : ℕ), n ≥ k → |(u * v) n - 0| < ε
    intros n hn
    -- n : ℕ
    -- hn : n ≥ k
    -- ⊢ |(u * v) n - 0| < ε
    simp_rw [sub_zero] at *
    -- ⊢ |(u * v) n| < ε
    calc |(u * v) n|
         = |u n| * |v n|  := by simp [Pi.mul_apply, abs_mul]
       _ ≤ B * |v n|      := mul_le_mul_of_nonneg_right (hB n) (abs_nonneg _)
       _ < B * (ε/B)      := by aesop
       _ = ε              := mul_div_cancel' ε hB0

-- Lemas usados
```

```
-- ============

-- #variable (n : ℕ)
-- #variable (a b c : ℝ)
-- #check (abs_nonneg a : 0 ≤ |a|)
-- #check (Ne.le_iff_lt : a ≠ b → (a ≤ b ↔ a < b))
-- #check (Pi.mul_apply u v n : (u * v) n = u n * v n)
-- #check (abs_mul a b : |a * b| = |a| * |b|)
-- #check (abs_nonneg a : 0 ≤ |a|)
-- #check (div_pos : 0 < a → 0 < b → 0 < a / b)
-- #check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
-- #check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
-- #check (mul_lt_mul_of_pos_left : b < c → 0 < a → a * b < a * c)
-- #check (sub_zero a : a - 0 = a)
-- #check (zero_mul a : 0 * a = 0)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

17.13. Si el límite de la sucesión u_n es a, entonces el límite de $-u_n$ es -a

```
-- ---------------------------------------------------------------------
-- En Lean4, una sucesión u₀, u₁, u₂, ... se puede representar mediante
-- una función (u : ℕ → ℝ) de forma que u(n) es uₙ.
--
-- Se define que a es el límite de la sucesión u, por
--    def limite : (ℕ → ℝ) → ℝ → Prop :=
--      fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε
--
-- Demostrar que que si el límite de uₙ es a, entonces el de
-- -uₙ es -a.
-- ---------------------------------------------------------------------

-- Demostración en lenguaje natural
-- ================================

-- Sea ε ∈ ℝ tal que ε > 0. Tenemos que demostrar que
--    (∃ N ∈ ℕ)(∀ n ≥ N)[|-uₙ - -a| < ε]                           (1)
-- 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,
--    |-uₙ - -a| = |-(uₙ - a)|
```

```
--                 = |uₙ - a|
--                 < ε            [por (2)]

-- Demostraciones con Lean4
-- ========================

import Mathlib.Data.Real.Basic
import Mathlib.Tactic

variable (u v : ℕ → ℝ)
variable (a c : ℝ)

def limite : (ℕ → ℝ) → ℝ → Prop :=
  fun u c ↦ ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

-- 1ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun n ↦ -u n) (-a) :=
by
  unfold limite at *
  -- h : ∀ (ε : ℝ), ε > 0 → ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε
  -- ⊢ ∀ (ε : ℝ), ε > 0 → ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => -u n) n - -a| < ε
  intro ε hε
  -- ε : ℝ
  -- hε : ε > 0
  -- ⊢ ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => -u n) n - -a| < ε
  specialize h ε hε
  -- h : ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε
  cases' h with k hk
  -- k : ℕ
  -- hk : ∀ (n : ℕ), n ≥ k → |u n - a| < ε
  use k
  -- ⊢ ∀ (n : ℕ), n ≥ k → |(fun n => -u n) n - -a| < ε
  intro n hn
  -- n : ℕ
  -- hn : n ≥ k
  -- ⊢ |(fun n => -u n) n - -a| < ε
  calc |(fun n => -u n) n - -a|
       = |(-u n - -a)|          := rfl
     _ = |(-(u n - a))|         := by congr ; ring
     _ = |(u n - a)|            := abs_neg (u n - a)
     _ < ε                      := hk n hn
```

```
-- 2ª demostración
-- ===============

example
  (h : limite u a)
  : limite (fun n ↦ -u n) (-a) :=
by
  unfold limite at *
  -- h : ∀ (ε : ℝ), ε > 0 → ∃ N, ∀ (n : ℕ), n ≥ N → |u n - a| < ε
  -- ⊢ ∀ (ε : ℝ), ε > 0 → ∃ N, ∀ (n : ℕ), n ≥ N → |(fun n => -u n) n - -a| < ε
  have h1 : ∀ n, |u n - a| = |(-u n - -a)| := by
    intro n
    -- n : ℕ
    -- ⊢ |u n - a| = |-u n - -a|
    rw [abs_sub_comm]
    -- ⊢ |a - u n| = |-u n - -a|
    congr 1
    -- ⊢ a - u n = -u n - -a
    ring
  simpa [h1] using h

-- Lemas usados
-- ============

-- variable (b : ℝ)
-- #check (abs_neg a : |(-a)| = |a|)
-- #check (abs_sub_comm a b : |a - b| = |b - a|)
```

Se puede interactuar con las pruebas anteriores en Lean 4 Web.

Bibliografía

[1] J. A. Alonso. Lean para matemáticos [1], 2021.

[2] J. A. Alonso. Matemáticas en Lean [2], 2021.

[3] J. A. Alonso. DAO (Demostración Asistida por Ordenador) con Lean [3], 2021.

[4] J. A. Alonso. Calculemus (Vol. 1: Demostraciones con Isabelle/HOL y Lean3) [4] , 2021.

[5] J. Avigad, L. de Moura, and S. Kong. Theorem Proving in Lean4 [5], 2021.

[6] J. Avigad, G. Ebner, and S. Ullrich. The Lean4 Manual [6], 2021.

[7] J. Avigad, M. J. H. Heule, and W. Nawrocki. Logic and mechanized reasoning [7] , 2023.

[8] J. Avigad, R. Y. Lewis, and F. van Doorn. Logic and proof [8] , 2021.

[9] J. Avigad and P. Massot. Mathematics in Lean [9] , 2023.

[10] A. Baanen, A. Bentkamp, J. Blanchette, J. Hölzl, and J. Limperg. The Hitchhiker's Guide to Logical Verification [10], 2020.

[1] https://github.com/jaalonso/Lean_para_matematicos
[2] https://github.com/jaalonso/Matematicas_en_Lean
[3] https://raw.githubusercontent.com/jaalonso/DAO_con_Lean/master/DAO_con_Lean.pdf
[4] https://raw.githubusercontent.com/jaalonso/Calculemus/master/Calculemus.pdf
[5] https://leanprover.github.io/theorem_proving_in_lean4/
[6] https://leanprover.github.io/lean4/doc/
[7] https://avigad.github.io/lamr/logic_and_mechanized_reasoning.pdf
[8] https://leanprover.github.io/logic_and_proof/logic_and_proof.pdf
[9] https://leanprover-community.github.io/mathematics_in_lean/
[10] https://raw.githubusercontent.com/blanchette/logical_verification_2020/master/hitchhikers_guide.pdf

[11] M. Ballard. Transition to advanced mathematics (Thinking and communicating like a mathematician) [11].

[12] K. Buzzard. Sets and logic (in Lean) [12].

[13] K. Buzzard. Functions and relations (in Lean) [13].

[14] K. Buzzard. Course on formalising mathematics [14], 2021.

[15] K. Buzzard. Course on formalising mathematics [15], 2023.

[16] K. Buzzard and M. Pedramfar. The Natural Number Game, version 1.3.3 [16].

[17] D. T. Christiansen. Functional programming in Lean [17], 2023.

[18] M. Community. Undergraduate mathematics in mathlib [18].

[19] M. Dvořák. Lean 4 Cheatsheet [19].

[20] S. Hazratpour. Introduction to proofs [20], 2022.

[21] S. Hazratpour. Introduction to proofs with Lean proof assistant [21], 2022.

[22] R. Lewis. Formal proof and verification, 2022 [22], 2022.

[23] R. Lewis. Discrete structures and probability [23], 2023.

[24] C. Löh. Exploring formalisation (A primer in human-readable mathematics in Lean 3 with examples from simplicial topology) [24], 2022.

[25] H. Macbeth. The mechanics of proof [25], 2023.

[11] https://300.f22.matthewrobertballard.com/
[12] https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C1.html
[13] https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C2.html
[14] https://github.com/ImperialCollegeLondon/formalising-mathematics
[15] https://github.com/ImperialCollegeLondon/formalising-mathematics-2023
[16] https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/
[17] https://leanprover.github.io/functional_programming_in_lean/
[18] https://leanprover-community.github.io/undergrad.html
[19] https://raw.githubusercontent.com/madvorak/lean4-cheatsheet/main/lean-tactics.pdf
[20] https://introproofs.github.io/s22/
[21] https://sinhp.github.io/teaching/2022-introduction-to-proofs-with-Lean
[22] https://github.com/BrownCS1951x/fpv2022
[23] https://github.com/brown-cs22/CS22-Lean-2023
[24] https://loeh.app.uni-regensburg.de/mapa/main.pdf
[25] https://hrmacbeth.github.io/math2001/index.html

[26] P. Massot. Introduction aux mathématiques formalisées [26].

[27] F. L. Roux. Code Lean contenant les preuves d'un cours standard sur les espaces métriques [27], 2020.

[28] W. Schulze. Learning LeanProver [28].

[29] Varios. LFTCM 2020: Lean for the Curious Mathematician 2020 [29].

[30] D. J. Velleman. How to prove it with Lean [30].

[26] https://www.imo.universite-paris-saclay.fr/~pmassot/enseignement/math114/
[27] https://github.com/FredericLeRoux/LEAN_ESPACES_METRIQUES
[28] https://youtube.com/playlist?list=PLYwF9EIrl42RFQgbmcR_LSCWRIx2WKbXs
[29] https://leanprover-community.github.io/lftcm2020/schedule.html
[30] https://djvelleman.github.io/HTPIwL/

Lemas usados

```
import Mathlib.Algebra.Group.Basic
import Mathlib.Algebra.Order.Ring.Defs
import Mathlib.Algebra.Ring.Defs
import Mathlib.Analysis.SpecialFunctions.Log.Basic
import Mathlib.Data.Real.Basic
import Mathlib.Order.Lattice
import Mathlib.Topology.MetricSpace.Basic

-- Números naturales
-- =================

section naturales
variable (x y z k m n : ℕ)
open Nat
#check (_root_.dvd_antisymm : m ∣ n → n ∣ m → m = n)
#check (dvd_add : x ∣ y → x ∣ z → x ∣ y + z)
#check (dvd_factorial : 0 < k → k ≤ n → k ∣ n !)
#check (dvd_gcd : k ∣ m → k ∣ n → k ∣ gcd m n)
#check (dvd_mul_left x y : x ∣ y * x)
#check (dvd_mul_of_dvd_left : x ∣ y → ∀ (c : ℕ), x ∣ y * c)
#check (dvd_mul_of_dvd_right : x ∣ y → ∀ (c : ℕ), x ∣ c * y)
#check (dvd_mul_right x y : x ∣ x * y)
#check (dvd_trans : x ∣ y → y ∣ z → x ∣ z)
#check (Dvd.intro k : m * k = n → m ∣ n)
#check (factorial_pos n: n ! > 0)
#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)
#check (minFac_dvd n : minFac n ∣ n)
#check (minFac_pos n : 0 < minFac n)
#check (minFac_prime : n ≠ 1 → Nat.Prime (minFac n))
#check (Nat.dvd_add_iff_right : k ∣ m → (k ∣ n ↔ k ∣ m + n))
#check (Nat.dvd_one : n ∣ 1 ↔ n = 1)
#check (Nat.lt_add_of_pos_left : 0 < k → n < k + n)
```

```
#check (Nat.ne_of_gt : k < n → n ≠ k)
#check (Nat.Prime.not_dvd_one : Nat.Prime n → ¬n ∣ 1)
end naturales

-- Números reales
-- ==============

section reales
open Real
variable (a b c d x y : ℝ)
#check (Left.self_le_neg : x ≤ 0 → x ≤ -x)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
#check (abs_le' : |a| ≤ b ↔ a ≤ b ∧ -a ≤ b)
#check (abs_lt: |a| < b ↔ -b < a ∧ a < b)
#check (abs_mul a b : |a * b| = |a| * |b|)
#check (abs_nonneg a : 0 ≤ |a|)
#check (abs_of_neg : x < 0 → |x| = -x)
#check (abs_of_nonneg : 0 ≤ x → |x| = x)
#check (abs_sub_abs_le_abs_sub a b : |a| - |b| ≤ |a - b|)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (add_le_add_left : b ≤ c → ∀ (a : ℝ), a + b ≤ a + c)
#check (add_le_add_right : b ≤ c → ∀ (a : ℝ),  b + a ≤ c + a)
#check (add_lt_add_of_le_of_lt : a ≤ b → c < d → a + c < b + d)
#check (add_lt_add_of_lt_of_le : a < b → c ≤ d → a + c < b + d)
#check (add_lt_add_right : b < c → ∀ (a : ℝ), b + a < c + a)
#check (add_neg_le_iff_le_add : a - b ≤ c ↔ a ≤ c + b)
#check (add_pos : 0 < a → 0 < b → 0 < a + b)
#check (add_sub_cancel a b : a + b - b = a)
#check (div_mul_cancel a : b ≠ 0 → (a / b) * b = a)
#check (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 (exp_le_exp : exp a ≤ exp b ↔ a ≤ b)
#check (exp_lt_exp : exp a < exp b ↔ a < b)
#check (exp_pos a : 0 < exp a)
#check (half_lt_self : 0 < a → a / 2 < a)
#check (half_pos : 0 < a → 0 < a / 2)
#check (le_abs_self x : x ≤ |x|)
#check (le_add_of_nonneg_right : 0 ≤ b → a ≤ a + b)
#check (le_antisymm : a ≤ b → b ≤ a → a = b)
#check (le_div_iff : 0 < c → (a ≤ b / c ↔ a * c ≤ b))
#check (le_max_left a b : a ≤ max a b)
#check (le_max_right a b : b ≤ max a b)
#check (le_min : c ≤ a → c ≤ b → c ≤ min a b)
#check (le_neg_self_iff : x ≤ -x ↔ x ≤ 0)
#check (le_of_eq : a = b → a ≤ b)
```

```
#check (le_of_lt : x < y → x ≤ y)
#check (le_of_not_ge : ¬x ≥ y → x ≤ y)
#check (le_of_not_gt : ¬a > b → a ≤ b)
#check (le_or_gt x y : x ≤ y ∨ x > y)
#check (le_refl a : a ≤ a)
#check (log_le_log' : 0 < a → a ≤ b → log a ≤ log b)
#check (lt_abs : x < |y| ↔ x < y ∨ x < -y)
#check (lt_asymm : a < b → ¬b < a)
#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_irrefl a : ¬a < a)
#check (lt_neg : a < -b ↔ b < -a)
#check (lt_of_le_of_ne : a ≤ b → a ≠ b → a < b)
#check (lt_of_lt_of_le : a < b → b ≤ c → a < c)
#check (lt_of_le_of_lt : a ≤ b → b < c → a < c)
#check (lt_of_le_of_ne : a ≤ b → a ≠ b → a < b)
#check (lt_of_not_ge : ¬ a ≥ b → a < b)
#check (lt_of_not_le : ¬b ≤ a → a < b)
#check (lt_trans : a < b → b < c → a < c)
#check (lt_trichotomy a b : a < b ∨ a = b ∨ b < a)
#check (max_comm a b : max a b = max b a)
#check (max_le : a ≤ c → b ≤ c → max a b ≤ c)
#check (min_add_add_right a b c : min (a + c) (b + c) = min a b + c)
#check (min_assoc a b c : min (min a b) c = min a (min b c))
#check (min_comm a b : min a b = min b a)
#check (min_eq_left : a ≤ b → min a b = a)
#check (min_eq_right : b ≤ a → min a b = b)
#check (min_le_left a b : min a b ≤ a)
#check (min_le_right a b : min a b ≤ b)
#check (mul_comm a b : a * b = b * a)
#check (mul_div_cancel' a : b ≠ 0 → b * (a / b) = a)
#check (mul_le_mul : a ≤ b → c ≤ d → 0 ≤ c → 0 ≤ b → a * c ≤ b * d)
#check (mul_le_mul_right : 0 < a → (b * a ≤ c * a ↔ b ≤ c))
#check (mul_left_cancel₀ : a ≠ 0 → a * b = a * c → b = c)
#check (mul_lt_mul_left : 0 < a → (a * b < a * c ↔ b < c))
#check (mul_lt_mul_right : 0 < a → (b * a < c * a ↔ b < c))
#check (mul_neg a b : a * -b = -(a * b))
#check (mul_right_inj' :  a ≠ 0 → (a * b = a * c ↔ b = c))
#check (mul_sub a b c : a * (b - c) = a * b - a * c)
#check (mul_two a : a * 2 = a + a)
#check (ne_comm : a ≠ b ↔ b ≠ a)
#check (neg_add x y : -(x + y) = -x + -y)
#check (neg_add_self a : -a + a = 0)
#check (neg_le_abs_self x : -x ≤ |x|)
#check (neg_mul_neg a b : -a * -b = a * b)
```

```
#check (nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a)
#check (not_lt_of_ge : a ≥ b → ¬ a < b)
#check (pow_eq_zero : ∀ {n : ℕ}, a ^ n = 0 → a = 0)
#check (pow_two a : a ^ 2 = a * a)
#check (pow_two_nonneg a : 0 ≤ a ^ 2)
#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 (sq_nonneg a : 0 ≤ a ^ 2)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_eq_zero : x - y = 0 ↔ x = y)
#check (sub_le_sub_left : a ≤ b → ∀ (c : ℝ), c - b ≤ c - a)
#check (sub_le_sub_right : a ≤ b → ∀ (c : ℝ), a - c ≤ b - c)
#check (sub_sq a b : (a - b) ^ 2 = a ^ 2 - 2 * a * b + b ^ 2)
#check (two_mul a : 2 * a = a + a)
#check (two_mul_le_add_sq a b : 2 * a * b ≤ a ^ 2 + b ^ 2)
#check (zero_lt_one : 0 < 1)
#check (zero_lt_two : 0 < 2)
end reales

-- Anillos
-- =======

section anillos
variable {R : Type _} [Ring R]
variable (a b c : R)
#check (add_assoc a b c : (a + b) + c = a + (b + c))
#check (add_comm a b : a + b = b + a)
#check (add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b)
#check (add_left_cancel : a + b = a + c → b = c)
#check (add_left_neg a : -a + a = 0)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
#check (add_neg_cancel_right a b : (a + b) + -b = a)
#check (add_neg_self a : a + -a = 0)
#check (add_right_cancel : a + b = c + b → a = c)
#check (add_right_neg a : a + -a = 0)
#check (add_zero a : a + 0 = a)
#check (mul_add a b c : a * (b + c) = a * b + a * c)
#check (mul_zero a : a * 0 = 0)
#check (neg_add_cancel_left a b : -a + (a + b) = b)
#check (neg_eq_iff_add_eq_zero : -a = b ↔ a + b = 0)
#check (neg_eq_of_add_eq_zero_left : a + b = 0 → -b = a)
#check (neg_eq_of_add_eq_zero_right : a + b = 0 → -a = b)
#check (neg_neg a : -(-a) = a)
#check (neg_zero : -0 = 0)
#check (one_add_one_eq_two : (1 : R) + 1 = 2)
```

```
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_eq_add_neg a b : a - b = a + -b)
#check (sub_mul a b c : (a - b) * c = a * c - b * c)
#check (sub_self a : a - a = 0)
#check (two_mul a : 2 * a = a + a)
#check (zero_add a : 0 + a = a)
#check (zero_mul a : 0 * a = 0)
end anillos

-- Grupos
-- ======

section grupos
variable {G : Type _} [Group G]
variable (a b c : G)
#check (inv_eq_of_mul_eq_one_right : a * b = 1 → a⁻¹ = b)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
#check (mul_inv_self a : a * a⁻¹ = 1)
#check (mul_inv_rev a b : (a * b)⁻¹ = b⁻¹ * a⁻¹)
#check (mul_left_inv a : a⁻¹  * a = 1)
#check (mul_one a : a * 1 = a)
#check (mul_right_inv a : a * a⁻¹ = 1)
#check (one_mul a : 1 * a = a)
end grupos

-- Retículos
-- =========

section reticulos
variable {α : Type _} [Lattice α]
variable (x y z : α)
#check (inf_assoc : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z))
#check (inf_comm : x ⊓ y = y ⊓ x)
#check (inf_le_left : x ⊓ y ≤ x)
#check (inf_le_of_left_le : x ≤ z → x ⊓ y ≤ z)
#check (inf_le_of_right_le : y ≤ z → x ⊓ y ≤ z)
#check (inf_le_right : x ⊓ y ≤ y)
#check (inf_sup_self : x ⊓ (x ⊔ y) = x)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)
#check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
#check (le_rfl : x ≤ x)
#check (le_sup_left : x ≤ x ⊔ y)
#check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
#check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
#check (le_sup_right : y ≤ x ⊔ y)
```

```
#check (le_trans : x ≤ y → y ≤ z → x ≤ z)
#check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
#check (sup_comm : x ⊔ y = y ⊔ x)
#check (sup_inf_self : x ⊔ (x ⊓ y) = x)
#check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)
end reticulos

-- AnillosOrdenados
-- ================

section AnillosOrdenados
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)
#check (add_le_add_right : b ≤ c → ∀ (a : R),  b + a ≤ c + a)
#check (mul_le_mul_of_nonneg_left : b ≤ c → 0 ≤ a → a * b ≤ a * c)
#check (mul_le_mul_of_nonneg_right : a ≤ b → 0 ≤ c → a * c ≤ b * c)
#check (mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b)
#check (sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c)
#check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
end AnillosOrdenados

-- Espacios métricos
-- =================

section EspacioMetrico
variable {X : Type _} [MetricSpace X]
variable (x y z : X)
#check (dist_comm x y : dist x y = dist y x)
#check (dist_nonneg : 0 ≤ dist x y)
#check (dist_self x : dist x x = 0)
#check (dist_triangle x y z : dist x z ≤ dist x y + dist y z)
end EspacioMetrico

-- Conjuntos
-- =========

section Conjuntos
open Set
variable {α : Type _}
variable (r s t : Set α)
#check (Subset.trans : r ⊆ s → s ⊆ t → r ⊆ t)
end Conjuntos

-- Órdenes parciales
-- =================
```

```
section OrdenParcial
variable {α : Type _} [PartialOrder α]
variable (a b c : α)
#check (irrefl a : ¬a < a)
#check (le_trans : a ≤ b → b ≤ c → a ≤ c)
#check (lt_trans : a < b → b < c → a < c)
#check (monotone_const : Monotone fun _ : ℝ ↦ c)
end OrdenParcial

-- Funciones
-- =========

section Funciones
open Function
variable {α : Type _} {β : Type _} {γ : Type _}
variable {f : α → β} {g : β → γ}
variable (c : ℝ)
#check (Injective.comp : Injective g → Injective f → Injective (g ∘ f))
#check (Surjective.comp : Surjective g → Surjective f → Surjective (g ∘ f))
#check (add_right_surjective c : Surjective (fun x ↦ x + c))
#check (mul_left_surjective₀ : c ≠ 0 → Surjective (fun x ↦ c * x))
end Funciones

-- Lógica
-- ======

section Logica
variable (p q : Prop)
variable {α : Type _}
variable (P : α → Prop)
#check (absurd : p → ¬p → q)
#check (forall_not_of_not_exists : (¬∃ x, P x) → ∀ x, ¬P x)
#check (not_exists : (¬∃ x, P x) ↔ ∀ (x : α), ¬P x)
#check (not_exists_of_forall_not : (∀ x, P x → q) → (∃ x, P x) → q)
#check (not_imp : ¬(p → q) ↔ p ∧ ¬q)
#check (not_forall : (¬∀ x, P x) ↔ ∃ x, ¬P x)
#check (not_forall_of_exists_not : (∃ x, ¬P x) → ¬∀ x, P x)
#check (not_not_intro : p → ¬¬p)
#check (of_not_not : ¬¬p → p)
#check (Or.inl : p → p ∨ q)
#check (Or.inr : q → p ∨ q)
end Logica
```