

# Calculus

## (Vol. 2: Demonstrations with 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 30 de septiembre de 2023)

Esta obra está bajo una licencia Reconocimiento–NoComercial–CompartirIgual 2.5 Spain de Creative Commons.

**Se permite:**

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

**Bajo las condiciones siguientes:**

**Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor.



**No comercial.** No puede utilizar esta obra para fines comerciales.



**Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Algunas de estas condiciones pueden no aplicarse si se obtiene el permiso del titular de los derechos de autor.

Esto es un resumen del texto legal (la licencia completa). Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-sa/2.5/es/> o envíe una carta a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

# Índice general

<b>1. Introducción</b>	<b>7</b>
<b>2. Demostraciones de una propiedad de los números enteros</b>	<b>9</b>
2.1. $\forall m, n \in \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n)$	9
<b>3. Propiedades elementales de los números reales</b>	<b>13</b>
3.1. En $\mathbb{R}$ , $(ab)c = b(ac)$	13
3.2. En $\mathbb{R}$ , $(cb)a = b(ac)$	14
3.3. En $\mathbb{R}$ , $a(bc) = b(ac)$	15
3.4. En $\mathbb{R}$ , si $ab = cd$ y $e = f$ , entonces $a(be) = c(df)$	17
3.5. En $\mathbb{R}$ , si $bc = ef$ , entonces $((ab)c)d = ((ae)f)d$	18
3.6. En $\mathbb{R}$ , si $c = ba-d$ y $d = ab$ , entonces $c = 0$	20
3.7. En $\mathbb{R}$ , $(a+b)(a+b) = aa+2ab+bb$	21
3.8. En $\mathbb{R}$ , $(a+b)(c+d) = ac+ad+bc+bd$	23
3.9. En $\mathbb{R}$ , $(a+b)(a-b) = a^2-b^2$	25
3.10. En $\mathbb{R}$ , si $c = da+b$ y $b = ad$ , entonces $c = 2ad$	28
3.11. En $\mathbb{R}$ , si $a+b = c$ , entonces $(a+b)(a+b) = ac+bc$	30
<b>4. Propiedades elementales de los anillos</b>	<b>33</b>
4.1. Si $R$ es un anillo y $a \in R$ , entonces $a + 0 = a$	33
4.2. Si $R$ es un anillo y $a \in R$ , entonces $a + -a = 0$	34
4.3. Si $R$ es un anillo y $a, b \in R$ , entonces $-a + (a + b) = b$	36
4.4. Si $R$ es un anillo y $a, b \in R$ , entonces $(a + b) + -b = a$	38
4.5. Si $R$ es un anillo y $a, b, c \in R$ tales que $a+b=a+c$ , entonces $b=c$	39
4.6. Si $R$ es un anillo y $a, b, c \in R$ tales que $a+b=c+b$ , entonces $a=c$	42
4.7. Si $R$ es un anillo y $a \in R$ , entonces $a.0 = 0$	44
4.8. Si $R$ es un anillo y $a \in R$ , entonces $0.a = 0$	46
4.9. Si $R$ es un anillo y $a, b \in R$ tales que $a+b=0$ , entonces $-a=b$	48
4.10. Si $R$ es un anillo y $a, b \in R$ tales que $a+b=0$ , entonces $a=-b$	50

4.11. Si $R$ es un anillo, entonces $-0 = 0$ . . . . .	52
4.12. Si $R$ es un anillo y $a \in R$ , entonces $-(-a) = a$ . . . . .	54
4.13. Si $R$ es un anillo y $a, b \in R$ , entonces $a - b = a + -b$ . . . . .	55
4.14. Si $R$ es un anillo y $a \in R$ , entonces $a - a = 0$ . . . . .	56
4.15. En los anillos, $1 + 1 = 2$ . . . . .	57
4.16. Si $R$ es un anillo y $a \in R$ , entonces $2a = a+a$ . . . . .	57
<b>5. Propiedades elementales de los grupos</b> . . . . .	<b>59</b>
5.1. Si $G$ es un grupo y $a \in G$ , entonces $aa^{-1} = 1$ . . . . .	59
5.2. Si $G$ es un grupo y $a \in G$ , entonces $a \cdot 1 = a$ . . . . .	61
5.3. Si $G$ es un grupo y $a, b \in G$ tales que $ab = 1$ entonces $a^{-1} = b$ . . . . .	62
5.4. Si $G$ es un grupo y $a, b \in G$ , entonces $(ab)^{-1} = b^{-1}a^{-1}$ . . . . .	64
<b>6. Propiedades de orden en los números reales</b> . . . . .	<b>67</b>
6.1. En $\mathbb{R}$ , si $a \leq b$ , $b < c$ , $c \leq d$ y $d < e$ , entonces $a < e$ . . . . .	67
6.2. En $\mathbb{R}$ , si $2a \leq 3b$ , $1 \leq a$ y $d = 2$ , entonces $d + a \leq 5b$ . . . . .	70
6.3. En $\mathbb{R}$ , si $1 \leq a$ y $b \leq d$ , entonces $2 + a + e^b \leq 3a + e^d$ . . . . .	71
6.4. En $\mathbb{R}$ , si $a \leq b$ y $c < d$ , entonces $a + e^c + f \leq b + e^d + f$ . . . . .	73
6.5. En $\mathbb{R}$ , si $d \leq f$ , entonces $c + e^{(a+d)} \leq c + e^{(a+f)}$ . . . . .	75
6.6. En $\mathbb{R}$ , si $a \leq b$ , entonces $\log(1+e^a) \leq \log(1+e^b)$ . . . . .	77
6.7. En $\mathbb{R}$ , si $a \leq b$ , entonces $c - e^b \leq c - e^a$ . . . . .	79
6.8. En $\mathbb{R}$ , $2ab \leq a^2 + b^2$ . . . . .	80
6.9. En $\mathbb{R}$ , $ ab  \leq (a^2+b^2)/2$ . . . . .	82
6.10. En $\mathbb{R}$ , $\min(a,b) = \min(b,a)$ . . . . .	84
6.11. En $\mathbb{R}$ , $\max(a,b) = \max(b,a)$ . . . . .	86
6.12. En $\mathbb{R}$ , $\min(\min(a,b),c) = \min(a,\min(b,c))$ . . . . .	88
6.13. En $\mathbb{R}$ , $\min(a,b)+c = \min(a+c,b+c)$ . . . . .	92
6.14. En $\mathbb{R}$ , $ a  -  b  \leq  a - b $ . . . . .	96
<b>7. Divisibilidad</b> . . . . .	<b>99</b>
7.1. Si $x,y,z \in \mathbb{N}$ , entonces $x \mid yxz$ . . . . .	99
7.2. Si $x$ divide a $w$ , también divide a $y(xz)+x^2+w^2$ . . . . .	100
7.3. Conmutatividad del máximo común divisor . . . . .	102
<b>8. Retículos</b> . . . . .	<b>105</b>
8.1. En los retículos, $x \sqcap y = y \sqcap x$ . . . . .	105
8.2. En los retículos, $x \sqcup y = y \sqcup x$ . . . . .	107

8.3. En los retículos, $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$ . . . . .	109
8.4. En los retículos, $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$ . . . . .	113
8.5. En los retículos, $x \sqcap (x \sqcup y) = x$ . . . . .	118
8.6. En los retículos, $x \sqcup (x \sqcap y) = x$ . . . . .	121
8.7. En los retículos, una distributiva del ínfimo implica la otra . . .	123
8.8. En los retículos, una distributiva del supremos implica la otra .	124
<b>9. Anillos ordenados</b>	<b>127</b>
9.1. En los anillos ordenados, $a \leq b \rightarrow 0 \leq b - a$ . . . . .	127
9.2. En los anillos ordenados, $0 \leq b - a \rightarrow a \leq b$ . . . . .	128
9.3. En los anillos ordenados, $\{a \leq b, 0 \leq c\} \vdash ac \leq bc$ . . . . .	130
<b>Bibliografía</b>	<b>132</b>
<b>Lemas usados</b>	<b>137</b>



# 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. $\forall m n \in \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n)$

```
-----  
-- 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 :  $\forall m n : \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n) := \text{by}$ 
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  exact mul_add m k k

-- 8ª demostración
-- =====

example :  $\forall m n : \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n) := \text{by}$ 
  intros m n hn
  unfold Even at *
  cases hn with
  | intro k hk =>
    use m * k
    rw [hk, mul_add]

-- 9ª demostración
-- =====

example :  $\forall m n : \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n) := \text{by}$ 
  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 :  $\forall m n : \mathbb{N}, \text{Even } n \rightarrow \text{Even } (m * n) := \text{by}$ 
  intros; simp [*, parity_simps]

-- Lemas usados
-- =====

-- #check (mul_add :  $\forall a b c : \mathbb{N}, a * (b + c) = a * b + a * c$ )

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).



# Capítulo 3

## Propiedades elementales de los números reales

### 3.1. En $\mathbb{R}$ , $(ab)c = b(ac)$

```
-- 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 $\mathbb{R}$ , $(cb)a = b(ac)$

```

-----
-- 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 $\mathbb{R}$ , $a(bc) = b(ac)$

```

-- -----
-- 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 $\mathbb{R}$ , 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 lenguaje 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 $\mathbb{R}$ , 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 $\mathbb{R}$ , 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 $\mathbb{R}$ , $(a+b)(a+b) = aa+2ab+bb$

```

-----
-- 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 $\mathbb{R}$ , $(a+b)(c+d) = ac+ad+bc+bd$

```

-----
-- 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 $\mathbb{R}$ , $(a+b)(a-b) = a^2-b^2$

```

-----
-- 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 $\mathbb{R}$ , 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 $\mathbb{R}$ , 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](#).





# Capítulo 4

## Propiedades elementales de los anillos

### 4.1. Si $R$ es un anillo y $a \in R$ , entonces $a + 0 = a$

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

```
-- 2ª demostración
```

```
example : a + 0 = a :=
```

```
by
```

```
  rw [add_comm]
```

```
  rw [zero_add]
```

```
-- 3ª demostración
```

```
example : a + 0 = a :=
```

```
by rw [add_comm, zero_add]
```

```
-- 4ª demostración
```

```
example : a + 0 = a :=
```

```
by exact add_zero a
```

```
-- 5ª demostración
```

```
example : a + 0 = a :=
```

```
  add_zero a
```

```
-- 5ª demostración
```

```
example : a + 0 = a :=
```

```
by simp
```

```
-- Lemas usados
```

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

```
variable (a b : R)
```

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

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

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

```

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

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

import Mathlib.Algebra.Ring.Defs

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

-- 1ª demostración
-- =====

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

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

-- 4ª demostración
-- =====

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

-- 5ª demostración
-- =====

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

```

```

-- 6ª demostración
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-----
-- Demostrar en Lean4 que si R es un anillo, entonces
--    $\forall a, b : R, -a + (a + b) = b$ 
-- -----

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

```

```

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

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

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

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-----
-- Demostrar en Lean4 que si  $R$  es un anillo, entonces
--    $\forall a, b : R, (a + b) + -b = a$ 
-----

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

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

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

```

```

add_neg_cancel_right a b

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-- -----
-- Demostrar que si  $R$  es un anillo y  $a, b, c \in R$  tales que
--    $a + b = a + c$ 
-- entonces
--    $b = c$ 
-- -----

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

-- 1ª demostración en LN
-- =====

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

```

```

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

-- 2ª demostración en LN
-- =====

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

-- 3ª demostración en LN
-- =====

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

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

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

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

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

-- 2ª demostración

```



```

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

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

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

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

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

```

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

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

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

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

-- 1ª demostración en LN
-- =====

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

-- 2ª demostración en LN
-- =====

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

```

--      = c

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

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

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

-- 1ª demostración con Lean4
-- =====

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

-- 2ª demostración con Lean4
-- =====

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

-- 3ª demostración con Lean4
-- =====

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

```

```

rw [h]
rw [add_neg_cancel_right]

-- 4ª demostración con Lean4
-- =====

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

-- 5ª demostración con Lean4
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 4.7. Si $R$ es un anillo y $a \in R$ , entonces $a \cdot 0 = 0$

```

-- -----
-- Demostrar que si  $R$  es un anillo y  $a \in R$ , entonces
--    $a \cdot 0 = 0$ 
-- -----

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

-- Basta aplicar la propiedad cancelativa a
--    $a \cdot 0 + a \cdot 0 = a \cdot 0 + 0$ 

```

```

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

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

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

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

-- 1ª demostración
-- =====

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

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

```

```

-- 4ª demostración
-- =====

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

-- 5ª demostración
-- =====

example : a * 0 = 0 :=
  mul_zero a

-- 6ª demostración
-- =====

example : a * 0 = 0 :=
by simp

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 4.8. Si $R$ es un anillo y $a \in R$ , entonces $0.a = 0$

```

-----
-- Demostrar que si  $R$  es un anillo y  $a \in R$ , entonces
--    $0 * a = 0$ 
-----

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

```

```

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

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

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

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

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

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

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

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

```

```

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-- -----
-- Demostrar que si es un anillo y  $a, b \in R$  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 1ª 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 1ª 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 2ª en LN)
example
  (h : a + b = 0)
  : -a = b :=
by
  have h1 : -a + (a + b) = -a + 0 := congrArg (HAdd.hAdd (-a)) h
  have h2 : -a + (a + b) = b := neg_add_cancel_left a b

```

```

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-----
-- Demostrar que si  $R$  es un anillo y  $a, b \in R$  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 conelativa]
--    $= 0 + -b$               [por la hipótesis]
--    $= -b$                   [por la suma con cero]

-- 2ª demostración en LN
-- -----

```

```

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

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

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

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

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

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

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

-- 4ª demostración

```

**example**

```

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

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

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

-- 1ª demostración en LN
-- =====

-- Por la suma con cero se tiene
--    $0 + 0 = 0$ 
-- Aplicándole la propiedad
--    $\forall a b \in R, a + b = 0 \rightarrow -a = b$ 
-- se obtiene
--    $-0 = 0$ 

-- 2ª demostración en LN
-- =====

-- Puesto que
--    $\forall a b \in R, a + b = 0 \rightarrow -a = b$ 
-- basta demostrar que
--    $0 + 0 = 0$ 
-- que es cierta por la suma con cero.

```

```

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

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

variable {R : Type _} [Ring R]

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

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-----
-- Demostrar que si  $R$  es un anillo y  $a \in R$ , entonces
--  $-(-a) = a$ 
-----

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

-- Es consecuencia de las siguientes propiedades demostradas en
-- ejercicios anteriores:
--  $\forall a, b \in R, a + b = 0 \rightarrow -a = b$ 
--  $\forall a \in R, -a + a = 0$ 

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

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

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

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

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

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

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

```

```
by simp

-- Lemas usados
-- =====

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

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

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

```
-- Por la definición de la resta.
```

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

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

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

```
variable (a b : R)
```

```
example : a - b = a + -b :=
```

```
-- by exact?
```

```
sub_eq_add_neg a b
```

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

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 4.14. Si $R$ es un anillo y $a \in R$ , entonces $a - a = 0$

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

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

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

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

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

```
-- 1ª demostración
```

```
example : a - a = 0 :=
```

```
calc
```

```
  a - a = a + -a := by rw [sub_eq_add_neg a a]
    _ = 0       := by rw [add_right_neg]
```

```
-- 2ª demostración
```

```
example : a - a = 0 :=
```

```
sub_self a
```

```
-- 3ª demostración
```

```
example : a - a = 0 :=
```

```
by simp
```

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

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).



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

```

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

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

--  Por cálculo.

--  Demostración con Lean4
--  =====

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

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

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

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

--  Lemas usados
--  =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 4.16. Si $R$ es un anillo y $a \in R$ , entonces $2a = a + a$

```

-----
-- Demostrar que si R es un anillo y  $a \in R$ , entonces
--    $2 * a = a + a$ 
-----

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

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

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

import Mathlib.Algebra.Ring.Defs

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## Capítulo 5

# Propiedades elementales de los grupos

### 5.1. Si $G$ es un grupo y $a \in G$ , 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 :     $\forall a b c : G, a * b * c = a * (b * c)$   
--   one_mul :       $\forall a : G, 1 * a = a$   
--   mul_left_inv :  $\forall a : G, a^{-1} * a = 1$   
--  
-- Demostrar que si  $G$  es un grupo y  $a \in G$ , entonces  
--    $a * a^{-1} = 1$   
-- -----  
  
-- Demostración en lenguaje natural  
-- =====  
  
-- Por la siguiente cadena de igualdades  
--    $a \cdot a^{-1} = 1 \cdot (a \cdot a^{-1})$  [por producto con uno]  
--    $= (1 \cdot a) \cdot a^{-1}$  [por asociativa]  
--    $= (((a^{-1})^{-1} \cdot a^{-1}) \cdot a) \cdot a^{-1}$  [por producto con inverso]  
--    $= ((a^{-1})^{-1} \cdot (a^{-1} \cdot a)) \cdot a^{-1}$  [por asociativa]  
--    $= ((a^{-1})^{-1} \cdot 1) \cdot a^{-1}$  [por producto con inverso]  
--    $= (a^{-1})^{-1} \cdot (1 \cdot a^{-1})$  [por asociativa]  
--    $= (a^{-1})^{-1} \cdot a^{-1}$  [por producto con uno]
```

```

--          = 1                                     [por producto con inverso]

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

import Mathlib.Algebra.Group.Defs

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

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

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

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

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

-- Lemas usados
-- =====

```

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 5.2. Si $G$ es un grupo y $a \in G$ , entonces $a \cdot 1 = a$

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

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

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

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

```
import Mathlib.Algebra.Group.Defs
```

```
variable {G : Type _} [Group G]
```

```
variable (a b : G)
```

```
-- 1ª demostración
```

```
example : a * 1 = a :=
```

```
calc
```

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

```
-- 2ª demostración
```

```
example : a * 1 = a :=
```

```
calc
```

```

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

### 5.3. Si $G$ es un grupo y $a, b \in G$ tales que $ab = 1$ entonces $a^{-1} = b$

```

-- -----
-- Demostrar que si  $G$  es un grupo y  $a, b \in G$ , tales que
--    $a * b = 1$ 
-- entonces
--    $a^{-1} = b$ 
-- -----

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

-- Se tiene a partir de la siguiente cadena de igualdades
--    $a^{-1} = a^{-1} * 1$            [por producto por uno]
--    $= a^{-1} * (a * b)$          [por hipótesis]
--    $= (a^{-1} * a) * b$          [por asociativa]

```

```

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

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

import Mathlib.Algebra.Group.Defs

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

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

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

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

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

```

```

by exact inv_eq_of_mul_eq_one_right h

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

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

```

-----
-- Demostrar que si  $G$  es un grupo y  $a, b \in G$ , entonces
--  $(a * b)^{-1} = b^{-1} * a^{-1}$ 
--
-----

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

-- Teniendo en cuenta la propiedad
--  $\forall a, b \in G, ab = 1 \rightarrow a^{-1} = b$ ,
-- basta demostrar que
--  $(a \cdot b) \cdot (b^{-1} \cdot a^{-1}) = 1$ .
-- La identidad anterior se demuestra mediante la siguiente cadena de
-- igualdades
--  $(a \cdot b) \cdot (b^{-1} \cdot a^{-1}) = a \cdot (b \cdot (b^{-1} \cdot a^{-1}))$  [por la asociativa]
--  $= a \cdot ((b \cdot b^{-1}) \cdot a^{-1})$  [por la asociativa]
--  $= a \cdot (1 \cdot a^{-1})$  [por producto con inverso]
--  $= a \cdot a^{-1}$  [por producto con uno]
--  $= 1$  [por producto con inverso]

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

import Mathlib.Algebra.Group.Defs

```



```

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

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

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

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

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

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

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

```

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

-- Lemas usados
-- =====

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

# Capítulo 6

## Propiedades de orden en los números reales

**6.1. En  $\mathbb{R}$ , si  $a \leq b$ ,  $b < c$ ,  $c \leq d$  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 ( $a \leq b$ ) y 2 ( $b < c$ ) se tiene
--    $a < c$ 
-- que, junto la hipótesis 3 ( $c \leq d$ ) 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 ( $a \leq b$ ) se reduce a probar
--    $b < e$ 
-- que, por la hipótesis 2 ( $b < c$ ), se reduce a
--    $c < e$ 
-- que, por la hipótesis 3 ( $c \leq d$ ), 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 :  $a \leq b$ )
  (h2 :  $b < c$ )
  (h3 :  $c \leq d$ )
  (h4 :  $d < e$ ) :
   $a < e$  :=
calc
   $a \leq b$  := h1
  _ < c := h2
  _  $\leq$  d := h3
  _ < e := h4

-- 2ª demostración
-- =====

example

```

```

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

```

```

-- 3ª demostración
-- =====

```

```

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

```

```

-- El desarrollo de la prueba es
--
--   a b c d e : ℝ,
--   h1 : a ≤ b,
--   h2 : b < c,
--   h3 : c ≤ d,
--   h4 : d < e
--   ⊢ a < e
--   apply lt_of_le_of_lt h1,
--   ⊢ b < e
--   apply lt_trans h2,
--   ⊢ c < e
--   apply lt_of_le_of_lt h3,
--   ⊢ d < e
--   exact h4,
--   no goals
--
-- 4ª demostración
-- =====

```

```

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 6.2. En $\mathbb{R}$ , si $2a \leq 3b$ , $1 \leq a$ y $d = 2$ , entonces $d + a \leq 5b$

```

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

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

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

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

```

```

import Mathlib.Data.Real.Basic

variable (a b c : ℝ)

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

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

### 6.3. En $\mathbb{R}$ , si $1 \leq a$ y $b \leq d$ , entonces $2 + a + e^b \leq 3a + e^d$

```

-----
-- 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

```

```

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

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

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b d : ℝ)

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

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

-- 3ª demostración
example

```



```

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#).

## 6.4. En $\mathbb{R}$ , 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 + e^c + f ≤ b + e^d + 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)
--    $f \leq f$  (2)
--
-- La (1), de nuevo por la monotonía de la suma, se reduce a las
-- siguientes dos:
--    $a \leq b$  (1.1)
--    $e^c < e^d$  (1.2)
--
-- La (1.1) se tiene por la hipótesis 1.
--
-- La (1.2) se tiene aplicando la monotonía de la exponencial a la
-- hipótesis 2.
--
-- La (2) se tiene por la propiedad reflexiva.

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

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

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

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

```

```

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.5. En $\mathbb{R}$ , si $d \leq f$ , entonces $c + e^{(a + d)} \leq c + e^{(a + f)}$

```

-----
-- 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 monotonía de la suma, se tiene
--    $a + d \leq a + f$ 
-- que, por la monotonía de la exponencial, da
--    $\exp(a + d) \leq \exp(a + f)$ 
-- y, por la monotonía de la suma, se tiene
--    $c + \exp(a + d) \leq c + \exp(a + f)$ 

-- 2ª demostración en LN
-- =====

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

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

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

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

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

```

```

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.6. En $\mathbb{R}$ , si $a \leq b$ , entonces $\log(1+e^a) \leq \log(1+e^b)$

```

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

-- Lemas usados
-- =====

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

```

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.7. En $\mathbb{R}$ , si $a \leq b$ , entonces $c - e^b \leq c - e^a$

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

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

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

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

import Mathlib.Analysis.SpecialFunctions.Log.Basic

open Real

variable (a b c : ℝ)

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

-- 2ª demostración
```

```

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

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

```

-- Sean a y b números reales. Demostrar que
--   2ab ≤ a² + b²
--
-- Demostración en lenguaje natural
-- =====
--
-- Puesto que los cuadrados son positivos, se tiene
--   (a - b)² ≥ 0
-- Desarrollando el cuadrado, se obtiene
--   a² - 2ab + b² ≥ 0
-- Sumando 2ab a ambos lados, queda
--   a² + b² ≥ 2ab

```



```

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

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

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

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

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

-- Lemas usados
-- =====

```

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.9. En $\mathbb{R}$ , $|ab| \leq (a^2 + b^2)/2$

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

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

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

```

--  $a^2 + b^2 \geq -2ab$ 

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lemas auxiliares
-- =====

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

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

-- 1ª demostración
-- =====

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

-- 2ª demostración
-- =====

```

```

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

-- 3ª demostración
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.10. En $\mathbb{R}$ , $\min(a,b) = \min(b,a)$

```

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

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

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

```

```

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lema auxiliar
-- =====

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

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

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

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

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

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

-- 1ª demostración

```

```

-- =====

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

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.11. En $\mathbb{R}$ , $\max(a,b) = \max(b,a)$

```

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

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

-- Es consecuencia de la siguiente propiedad

```

```

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

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- Lema auxiliar
-- =====

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

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

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

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

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

lemma aux : max a b ≤ max b a :=

```

```

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

-- 1ª demostración
-- =====

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

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.12. En $\mathbb{R}$ , $\min(\min(a,b),c) = \min(a,\min(b,c))$

```

-----
-- 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
--    $\min(\min(a, b), c) \leq \min(a, \min(b, c))$  (1)
--    $\min(a, \min(b, c)) \leq \min(\min(a, b), c)$  (2)
--
-- La (1) es consecuencia de las siguientes desigualdades
--    $\min(\min(a, b), c) \leq a$  (1a)
--    $\min(\min(a, b), c) \leq b$  (1b)
--    $\min(\min(a, b), c) \leq c$  (1c)
-- En efecto, de (1b) y (1c) se obtiene
--    $\min(\min(a, b), c) \leq \min(b, c)$ 
-- que, junto con (1a) da (1).
--
-- La (2) es consecuencia de las siguientes desigualdades
--    $\min(a, \min(b, c)) \leq a$  (2a)
--    $\min(a, \min(b, c)) \leq b$  (2b)
--    $\min(a, \min(b, c)) \leq c$  (2c)
-- En efecto, de (2a) y (2b) se obtiene
--    $\min(a, \min(b, c)) \leq \min(a, b)$ 
-- que, junto con (2c) da (2).
--
-- La demostración de (1a) es
--    $\min(\min(a, b), c) \leq \min(a, b) \leq a$ 
-- La demostración de (1b) es
--    $\min(\min(a, b), c) \leq \min(a, b) \leq b$ 
-- La demostración de (2b) es
--    $\min(a, \min(b, c)) \leq \min(b, c) \leq b$ 
-- La demostración de (2c) es
--    $\min(a, \min(b, c)) \leq \min(b, c) \leq c$ 
-- La (1c) y (2a) son inmediatas.

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

import Mathlib.Data.Real.Basic

variable {a b c : ℝ}

-- Lemas auxiliares
-- =====

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

```

```

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

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

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

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

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

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

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

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

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

```

```

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

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

-- 1ª demostración
-- =====

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

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

-- 4ª demostración
-- =====

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

```

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

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

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

### 6.13. En $\mathbb{R}$ , $\min(a,b)+c = \min(a+c,b+c)$

```
-- 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  $a \leq b$ .
--
-- 1º caso: Supongamos que  $a \leq b$ . Entonces,
--    $\min(a, b) + c = a + c$ 
--                    $= \min(a + c, b + c)$ 
--
-- 2º caso: Supongamos que  $a \not\leq b$ . 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 :  $\min a b + c \leq \min (a + c) (b + c)$ 
--   aux2 :  $\min (a + c) (b + c) \leq \min a b + c$ 
--   cuyas demostraciones se exponen a continuación.

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

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

```

```

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

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

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

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

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

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

```

```

apply le_antisymm
{ exact aux1 }
{ exact aux2 }

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

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 6.14. En $\mathbb{R}$ , $|a| - |b| \leq |a - b|$

```

-- Sean a y b números reales. Demostrar que
--    $|a| - |b| \leq |a - b|$ 
--
-- Demostraciones en lenguaje natural (LN)
-- =====

-- 1ª demostración en LN
-- =====

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

-- 2ª demostración en LN
-- =====

-- Por la desigualdad triangular
--    $|a - b + b| \leq |a - b| + |b|$ 
-- simplificando en la izquierda
--    $|a| \leq |a - b| + |b|$ 
-- y, pasando  $|b|$  a la izquierda
--    $|a| - |b| \leq |a - b|$ 

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

import Mathlib.Data.Real.Basic

variable (a b : ℝ)

-- 1ª demostración (basada en la 1ª en LN)
example :  $|a| - |b| \leq |a - b|$  :=
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](#)



# Capítulo 7

## Divisibilidad

### 7.1. Si $x, y, z \in \mathbb{N}$ , entonces $x \mid yxz$

```
-- Demostrar que si  $x, y, z \in \mathbb{N}$ , entonces
--  $x \mid y * x * z$ 
-- -----

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

-- Por la transitividad de la divisibilidad aplicada a las relaciones
--  $x \mid yx$ 
--  $yx \mid yxz$ 

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

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

-- 1ª demostración
-- =====

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

```

exact dvd_trans h1 h2

-- 2ª demostración
-- =====

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

-- 3ª demostración
-- =====

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 7.2. Si $x$ divide a $w$ , también divide a $y(xz)+x^2+w^2$

```

-- -----
-- 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 | x^2                                     (2)
--   x | w^2                                     (3)
--
-- Para demostrar (1), por la divisibilidad del producto se tiene

```

```

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

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

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

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

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

```

```

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

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

-- Lemas usados
-- =====

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

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 7.3. 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

```

```

--       $(\forall x, y \in \mathbb{N})[\gcd(x,y) \mid \gcd(y,x)]$  (1)
-- En efecto, sustituyendo en (1)  $x$  por  $m$  e  $y$  por  $n$ , se tiene
--       $\gcd(m, n) \mid \gcd(n, m)$  (2)
-- y sustituyendo en (1)  $x$  por  $n$  e  $y$  por  $m$ , se tiene
--       $\gcd(n, m) \mid \gcd(m, n)$  (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
--       $\gcd(m, n) \mid n$ 
--       $\gcd(m, n) \mid m$ 
-- 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 :  $\gcd m n \mid \gcd n m$  :=
by
  have h1 :  $\gcd m n \mid n$  :=
    gcd_dvd_right m n
  have h2 :  $\gcd m n \mid m$  :=
    gcd_dvd_left m n
  show  $\gcd m n \mid \gcd n m$ 
  exact dvd_gcd h1 h2

-- 2ª demostración del lema auxiliar
example :  $\gcd m n \mid \gcd n m$  :=
dvd_gcd (gcd_dvd_right m n) (gcd_dvd_left m n)

-- 1ª demostración
example :  $\gcd m n = \gcd n m$  :=
by
  have h1 :  $\gcd m n \mid \gcd n m$  := aux m n
  have h2 :  $\gcd n m \mid \gcd m n$  := aux n m
  show  $\gcd m n = \gcd n m$ 
  exact _root_.dvd_antisymm h1 h2

```

```

-- 2ª demostración
example : gcd m n = gcd n m :=
by
  apply _root_.dvd_antisymm
  { exact aux m n }
  { exact aux n m }

-- 3ª demostración
example : gcd m n = gcd n m :=
_root_.dvd_antisymm (aux m n) (aux n m)

-- 4ª demostración
example : gcd m n = gcd n m :=
-- by apply?
gcd_comm m n

-- Lemas usados
-- =====

-- #check (_root_.dvd_antisymm : m | n → n | m → m = n)
-- #check (dvd_gcd : k | m → k | n → k | gcd m n)
-- #check (gcd_comm m n : gcd m n = gcd n m)
-- #check (gcd_dvd_left m n : gcd m n | m)
-- #check (gcd_dvd_right m n : gcd m n | n)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)



# Capítulo 8

## Retículos

### 8.1. En los retículos, $x \sqcap y = y \sqcap x$

```
-- -----  
-- Demostrar que en los retículos se verifica que  
--       $x \sqcap y = y \sqcap x$   
-- -----  
  
-- Demostración en lenguaje natural  
-- =====  
  
-- Es consecuencia del siguiente lema auxiliar  
--       $(\forall a, b)[a \sqcap b \leq b \sqcap a]$  (1)  
-- En efecto, sustituyendo en (1)  $a$  por  $x$  y  $b$  por  $y$ , se tiene  
--       $x \sqcap y \leq y \sqcap x$  (2)  
-- y sustituyendo en (1)  $a$  por  $y$  y  $b$  por  $x$ , se tiene  
--       $y \sqcap x \leq x \sqcap y$  (3)  
-- Finalmente, aplicando la propiedad antisimétrica de la divisibilidad  
-- a (2) y (3), se tiene  
--       $x \sqcap y = y \sqcap x$   
--  
-- Para demostrar (1), por la definición del ínfimo, basta demostrar  
-- las siguientes relaciones  
--       $y \sqcap x \leq x$   
--       $y \sqcap x \leq y$   
-- y ambas se tienen por la definición del ínfimo.  
  
-- Demostraciones con Lean4  
-- =====  
  
import Mathlib.Order.Lattice
```

```

variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración del lema auxiliar
lemma aux : x ⊓ y ≤ y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y :=
    inf_le_right
  have h2 : x ⊓ y ≤ x :=
    inf_le_left
  show x ⊓ y ≤ y ⊓ x
  exact le_inf h1 h2

-- 2ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
by
  apply le_inf
  { apply inf_le_right }
  { apply inf_le_left }

-- 3ª demostración del lema auxiliar
example : x ⊓ y ≤ y ⊓ x :=
le_inf inf_le_right inf_le_left

-- 1ª demostración
example : x ⊓ y = y ⊓ x :=
by
  have h1 : x ⊓ y ≤ y ⊓ x :=
    aux x y
  have h2 : y ⊓ x ≤ x ⊓ y :=
    aux y x
  show x ⊓ y = y ⊓ x
  exact le_antisymm h1 h2

-- 2ª demostración
example : x ⊓ y = y ⊓ x :=
by
  apply le_antisymm
  { apply aux }
  { apply aux }

-- 3ª demostración
example : x ⊓ y = y ⊓ x :=
le_antisymm (aux x y) (aux y x)

```

```

-- 4ª demostración
example : x ⊓ y = y ⊓ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊓ y = y ⊓ x :=
-- by apply?
inf_comm

-- Lemas usados
-- =====

-- #check (inf_comm : x ⊓ y = y ⊓ x)
-- #check (inf_le_left : x ⊓ y ≤ x)
-- #check (inf_le_right : x ⊓ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.2. En los retículos, $x \sqcup y = y \sqcup x$

```

-----
-- 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

```

```

--       $x \leq y \sqcup x$ 
--       $y \leq y \sqcup x$ 
-- 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 :  $x \sqcup y \leq y \sqcup x$  :=
by
  have h1 :  $x \leq y \sqcup x$  :=
    le_sup_right
  have h2 :  $y \leq y \sqcup x$  :=
    le_sup_left
  show  $x \sqcup y \leq y \sqcup x$ 
  exact sup_le h1 h2

-- 2ª demostración del lema auxiliar
example :  $x \sqcup y \leq y \sqcup x$  :=
by
  apply sup_le
  { apply le_sup_right }
  { apply le_sup_left }

-- 3ª demostración del lema auxiliar
example :  $x \sqcup y \leq y \sqcup x$  :=
sup_le le_sup_right le_sup_left

-- 1ª demostración
example :  $x \sqcup y = y \sqcup x$  :=
by
  have h1 :  $x \sqcup y \leq y \sqcup x$  :=
    aux x y
  have h2 :  $y \sqcup x \leq x \sqcup y$  :=
    aux y x
  show  $x \sqcup y = y \sqcup x$ 
  exact le_antisymm h1 h2

-- 2ª demostración
example :  $x \sqcup y = y \sqcup x$  :=
by

```

```

apply le_antisymm
{ apply aux }
{ apply aux }

-- 3ª demostración
example : x ⊔ y = y ⊔ x :=
le_antisymm (aux x y) (aux y x)

-- 4ª demostración
example : x ⊔ y = y ⊔ x :=
by apply le_antisymm; simp ; simp

-- 5ª demostración
example : x ⊔ y = y ⊔ x :=
-- by apply?
sup_comm

-- Lemas usados
-- =====

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (sup_comm : x ⊔ y = y ⊔ x)
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.3. En los retículos, $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$

```

-----
-- 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:
--    $(x \sqcap y) \sqcap z \leq x \sqcap (y \sqcap z)$  (1)
--    $x \sqcap (y \sqcap z) \leq (x \sqcap y) \sqcap z$  (2)
--
-- Para demostrar (1), por le_inf, basta probar que
--    $(x \sqcap y) \sqcap z \leq x$  (1a)
--    $(x \sqcap y) \sqcap z \leq y \sqcap z$  (1b)
--
-- La (1a) se demuestra por la siguiente cadena de desigualdades
--    $(x \sqcap y) \sqcap z \leq x \sqcap y$  [por inf_le_left]
--    $\leq x$  [por inf_le_left]
--
-- Para demostrar (1b), por le_inf, basta probar que
--    $(x \sqcap y) \sqcap z \leq y$  (1b1)
--    $(x \sqcap y) \sqcap z \leq z$  (1b2)
--
-- La (1b1) se demuestra por la siguiente cadena de desigualdades
--    $(x \sqcap y) \sqcap z \leq x \sqcap y$  [por inf_le_left]
--    $\leq y$  [por inf_le_right]
--
-- La (1b2) se tiene por inf_le_right.
--
-- Para demostrar (2), por le_inf, basta probar que
--    $x \sqcap (y \sqcap z) \leq x \sqcap y$  (2a)
--    $x \sqcap (y \sqcap z) \leq z$  (2b)
--
-- Para demostrar (2a), por le_inf, basta probar que
--    $x \sqcap (y \sqcap z) \leq x$  (2a1)
--    $x \sqcap (y \sqcap z) \leq y$  (2a2)
--
-- La (2a1) se tiene por inf_le_left.
--
-- La (2a2) se demuestra por la siguiente cadena de desigualdades
--    $x \sqcap (y \sqcap z) \leq y \sqcap z$  [por inf_le_right]
--    $\leq y$  [por inf_le_left]
--
-- La (2b) se demuestra por la siguiente cadena de desigualdades
--    $x \sqcap (y \sqcap z) \leq y \sqcap z$  [por inf_le_right]
--    $\leq z$  [por inf_le_right]
--
-- Demostraciones con Lean4
-- =====
import Mathlib.Order.Lattice

```

```

variable {α : Type _} [Lattice α]
variable (x y z : α)

-- 1ª demostración
-- =====

example : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z) :=
by
  have h1 : (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z) := by
    { have h1a : (x ⊓ y) ⊓ z ≤ x := calc
      (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
      _ ≤ x := by exact inf_le_left
    have h1b : (x ⊓ y) ⊓ z ≤ y ⊓ z := by
      { have h1b1 : (x ⊓ y) ⊓ z ≤ y := calc
        (x ⊓ y) ⊓ z ≤ x ⊓ y := by exact inf_le_left
        _ ≤ y := by exact inf_le_right
      have h1b2 : (x ⊓ y) ⊓ z ≤ z :=
        inf_le_right
      show (x ⊓ y) ⊓ z ≤ y ⊓ z
      exact le_inf h1b1 h1b2 }
    show (x ⊓ y) ⊓ z ≤ x ⊓ (y ⊓ z)
    exact le_inf h1a h1b }
  have h2 : x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z := by
    { have h2a : x ⊓ (y ⊓ z) ≤ x ⊓ y := by
      { have h2a1 : x ⊓ (y ⊓ z) ≤ x :=
        inf_le_left
      have h2a2 : x ⊓ (y ⊓ z) ≤ y := calc
        x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
        _ ≤ y := by exact inf_le_left
      show x ⊓ (y ⊓ z) ≤ x ⊓ y
      exact le_inf h2a1 h2a2 }
    have h2b : x ⊓ (y ⊓ z) ≤ z := by calc
      x ⊓ (y ⊓ z) ≤ y ⊓ z := by exact inf_le_right
      _ ≤ z := by exact inf_le_right
    show x ⊓ (y ⊓ z) ≤ (x ⊓ y) ⊓ z
    exact le_inf h2a h2b }
  show (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z)
  exact le_antisymm h1 h2

-- 2ª demostración
-- =====

example : x ⊓ y ⊓ z = x ⊓ (y ⊓ z) := by
  apply le_antisymm
  · apply le_inf

```

```

    · apply le_trans
      apply inf_le_left
      apply inf_le_left
    · apply le_inf
      · apply le_trans
        apply inf_le_left
        apply inf_le_right
      · apply inf_le_right
  · apply le_inf
    · apply le_inf
      · apply inf_le_left
    · apply le_trans
      apply inf_le_right
      apply inf_le_left
  · apply le_trans
    apply inf_le_right
    apply inf_le_right

-- 3ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  · apply le_inf
    · apply inf_le_of_left_le inf_le_left
    · apply le_inf (inf_le_of_left_le inf_le_right) inf_le_right
  · apply le_inf
    · apply le_inf inf_le_left (inf_le_of_right_le inf_le_left)
    · apply inf_le_of_right_le inf_le_right

-- 4ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
le_antisymm
  (le_inf
    (inf_le_of_left_le inf_le_left)
    (le_inf (inf_le_of_left_le inf_le_right) inf_le_right))
  (le_inf
    (le_inf inf_le_left (inf_le_of_right_le inf_le_left))
    (inf_le_of_right_le inf_le_right))

-- 5ª demostración
-- =====

```



```

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
-- by apply?
inf_assoc

-- Lemas usados
-- =====

-- #check (inf_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
-- #check (inf_le_left : x ⊔ y ≤ x)
-- #check (inf_le_of_left_le : x ≤ z → x ⊔ y ≤ z)
-- #check (inf_le_of_right_le : y ≤ z → x ⊔ y ≤ z)
-- #check (inf_le_right : x ⊔ y ≤ y)
-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊔ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.4. En los retículos, $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$

```

-----
-- 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
--    $x \leq x \sqcup (y \sqcup z)$  (1a1)
--    $y \leq x \sqcup (y \sqcup z)$  (1a2)
--
-- La (1a1) se tiene por le_sup_left.
--
-- La (1a2) se tiene por la siguiente cadena de desigualdades:
--    $y \leq y \sqcup z$  [por le_sup_left]
--    $\leq x \sqcup (y \sqcup z)$  [por le_sup_right]
--
-- La (1b) se tiene por la siguiente cadena de desigualdades
--    $z \leq y \sqcup z$  [por le_sup_right]
--    $\leq x \sqcup (y \sqcup z)$  [por le_sup_right]
--
-- Para demostrar (2), por sup_le, basta probar
--    $x \leq (x \sqcup y) \sqcup z$  (2a)
--    $y \sqcup z \leq (x \sqcup y) \sqcup z$  (2b)
--
-- La (2a) se demuestra por la siguiente cadena de desigualdades:
--    $x \leq x \sqcup y$  [por le_sup_left]
--    $\leq (x \sqcup y) \sqcup z$  [por le_sup_left]
--
-- Para demostrar (2b), por sup_le, basta probar
--    $y \leq (x \sqcup y) \sqcup z$  (2b1)
--    $z \leq (x \sqcup y) \sqcup z$  (2b2)
--
-- La (2b1) se demuestra por la siguiente cadena de desigualdades:
--    $y \leq x \sqcup y$  [por le_sup_right]
--    $\leq (x \sqcup y) \sqcup z$  [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  $\sqcup$  y)  $\sqcup$  z = x  $\sqcup$  (y  $\sqcup$  z) :=
by

```

```

have h1 : (x  $\sqcup$  y)  $\sqcup$  z  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := by
{ have h1a : x  $\sqcup$  y  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := by
  { have h1a1 : x  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := by exact le_sup_left
    have h1a2 : y  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := calc
      y  $\leq$  y  $\sqcup$  z := by exact le_sup_left
      _  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := by exact le_sup_right
    show x  $\sqcup$  y  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
    exact sup_le h1a1 h1a2 }
  have h1b : z  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := calc
    z  $\leq$  y  $\sqcup$  z := by exact le_sup_right
    _  $\leq$  x  $\sqcup$  (y  $\sqcup$  z) := by exact le_sup_right
  show (x  $\sqcup$  y)  $\sqcup$  z  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
  exact sup_le h1a h1b }
have h2 : x  $\sqcup$  (y  $\sqcup$  z)  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := by
{ have h2a : x  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := calc
  x  $\leq$  x  $\sqcup$  y := by exact le_sup_left
  _  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := by exact le_sup_left
  have h2b : y  $\sqcup$  z  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := by
  { have h2b1 : y  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := calc
    y  $\leq$  x  $\sqcup$  y := by exact le_sup_right
    _  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := by exact le_sup_left
  have h2b2 : z  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z := by
    exact le_sup_right
  show y  $\sqcup$  z  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z
  exact sup_le h2b1 h2b2 }
  show x  $\sqcup$  (y  $\sqcup$  z)  $\leq$  (x  $\sqcup$  y)  $\sqcup$  z
  exact sup_le h2a h2b }
show (x  $\sqcup$  y)  $\sqcup$  z = x  $\sqcup$  (y  $\sqcup$  z)
exact le_antisymm h1 h2

-- 2ª demostración
-- =====

example : x  $\sqcup$  y  $\sqcup$  z = x  $\sqcup$  (y  $\sqcup$  z) :=
by
  apply le_antisymm
  · -- (x  $\sqcup$  y)  $\sqcup$  z  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
    apply sup_le
    · -- x  $\sqcup$  y  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
      apply sup_le
      · -- x  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
        apply le_sup_left
      · -- y  $\leq$  x  $\sqcup$  (y  $\sqcup$  z)
        apply le_trans
        · -- y  $\leq$  y  $\sqcup$  z

```

```

    apply @le_sup_left _ _ y z
  · --  $y \sqcup z \leq x \sqcup (y \sqcup z)$ 
    apply le_sup_right
  · --  $z \leq x \sqcup (y \sqcup z)$ 
    apply le_trans
  · --  $z \leq x \sqcup (y \sqcup z)$ 
    apply @le_sup_right _ _ y z
  · --  $y \sqcup z \leq x \sqcup (y \sqcup z)$ 
    apply le_sup_right
  · --  $x \sqcup (y \sqcup z) \leq (x \sqcup y) \sqcup z$ 
    apply sup_le
  · --  $x \leq (x \sqcup y) \sqcup z$ 
    apply le_trans
  · --  $x \leq x \sqcup y$ 
    apply @le_sup_left _ _ x y
  · --  $x \sqcup y \leq (x \sqcup y) \sqcup z$ 
    apply le_sup_left
  · --  $y \sqcup z \leq (x \sqcup y) \sqcup z$ 
    apply sup_le
  · --  $y \leq (x \sqcup y) \sqcup z$ 
    apply le_trans
  · --  $y \leq x \sqcup y$ 
    apply @le_sup_right _ _ x y
  · --  $x \sqcup y \leq (x \sqcup y) \sqcup z$ 
    apply le_sup_left
  · --  $z \leq (x \sqcup y) \sqcup z$ 
    apply le_sup_right

-- 3ª demostración
-- =====

example :  $x \sqcup y \sqcup z = x \sqcup (y \sqcup z) :=$ 
by
  apply le_antisymm
  · apply sup_le
    · apply sup_le
      · apply le_sup_left
      · apply le_trans
        · apply @le_sup_left _ _ y z
        · apply le_sup_right
    · apply le_trans
      · apply @le_sup_right _ _ y z
      · apply le_sup_right
  · apply sup_le
    · apply le_trans

```

```

    . apply @le_sup_left _ _ x y
    . apply le_sup_left
  . apply sup_le
  . apply le_trans
    . apply @le_sup_right _ _ x y
    . apply le_sup_left
    . apply le_sup_right

-- 4ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . -- (x ⊔ y) ⊔ z ≤ x ⊔ (y ⊔ z)
    apply sup_le
    . -- x ⊔ y ≤ x ⊔ (y ⊔ z)
      apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . -- z ≤ x ⊔ (y ⊔ z)
      apply le_sup_of_le_right le_sup_right
  . -- x ⊔ (y ⊔ z) ≤ (x ⊔ y) ⊔ z
    apply sup_le
    . -- x ≤ (x ⊔ y) ⊔ z
      apply le_sup_of_le_left le_sup_left
    . -- y ⊔ z ≤ (x ⊔ y) ⊔ z
      apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 5ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
by
  apply le_antisymm
  . apply sup_le
    . apply sup_le le_sup_left (le_sup_of_le_right le_sup_left)
    . apply le_sup_of_le_right le_sup_right
  . apply sup_le
    . apply le_sup_of_le_left le_sup_left
    . apply sup_le (le_sup_of_le_left le_sup_right) le_sup_right

-- 6ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
le_antisymm

```

```

(sup_le
  (sup_le le_sup_left (le_sup_of_le_right le_sup_left))
  (le_sup_of_le_right le_sup_right))
(sup_le
  (le_sup_of_le_left le_sup_left)
  (sup_le (le_sup_of_le_left le_sup_right) le_sup_right))

-- 7ª demostración
-- =====

example : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z) :=
-- by apply?
sup_assoc

-- Lemas usados
-- =====

-- #check (le_antisymm : x ≤ y → y ≤ x → x = y)
-- #check (le_sup_left : x ≤ x ⊔ y)
-- #check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
-- #check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
-- #check (le_sup_right : y ≤ x ⊔ y)
-- #check (le_trans : x ≤ y → y ≤ z → x ≤ z)
-- #check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
-- #check (sup_le : x ≤ z → y ≤ z → x ⊔ y ≤ z)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.5. En los retículos, $x \sqcap (x \sqcup y) = x$

```

-----
-- 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_refl     : x ≤ x

```

```

--      le_sup_left  :  $x \leq x \sqcup y$ 
--
-- Por le_antisymm, basta demostrar las siguientes relaciones:
--       $x \sqcap (x \sqcup y) \leq x$                                 (1)
--       $x \leq x \sqcap (x \sqcup y)$                             (2)
--
-- La (1) se tiene por inf_le_left.
--
-- Para demostrar la (2), por le_inf, basta probar las relaciones:
--       $x \leq x$                                               (2a)
--       $x \leq x \sqcup y$                                     (2b)
--
-- La (2a) se tiene por le_refl.
--
-- 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 :  $x \sqcap (x \sqcup y) = x$  :=
by
  have h1 :  $x \sqcap (x \sqcup y) \leq x$  := inf_le_left
  have h2 :  $x \leq x \sqcap (x \sqcup y)$ 
  { have h2a :  $x \leq x$  := le_refl
    have h2b :  $x \leq x \sqcup y$  := le_sup_left
    show  $x \leq x \sqcap (x \sqcup y)$ 
    exact le_inf h2a h2b }
  show  $x \sqcap (x \sqcup y) = x$ 
  exact le_antisymm h1 h2

-- 2ª demostración
-- =====

example :  $x \sqcap (x \sqcup y) = x$  :=
by
  have h1 :  $x \sqcap (x \sqcup y) \leq x$  := by simp
  have h2 :  $x \leq x \sqcap (x \sqcup y)$  := by simp
  show  $x \sqcap (x \sqcup y) = x$ 

```

```

exact le_antisymm h1 h2

-- 3ª demostración
-- =====

example : x  $\sqcap$  (x  $\sqcup$  y) = x :=
by
  apply le_antisymm
  . -- x  $\sqcap$  (x  $\sqcup$  y)  $\leq$  x
    apply inf_le_left
  . -- x  $\leq$  x  $\sqcap$  (x  $\sqcup$  y)
    apply le_inf
  . -- x  $\leq$  x
    apply le_refl
  . -- x  $\leq$  x  $\sqcup$  y
    apply le_sup_left

-- 4ª demostración
-- =====

example : x  $\sqcap$  (x  $\sqcup$  y) = x :=
le_antisymm inf_le_left (le_inf le_refl le_sup_left)

-- 5ª demostración
-- =====

example : x  $\sqcap$  (x  $\sqcup$  y) = x :=
-- by apply?
inf_sup_self

-- 6ª demostración
-- =====

example : x  $\sqcap$  (x  $\sqcup$  y) = x :=
by simp

-- Lemas usados
-- =====

-- variable (z :  $\alpha$ )
-- #check (inf_le_left : x  $\sqcap$  y  $\leq$  x)
-- #check (inf_sup_self : x  $\sqcap$  (x  $\sqcup$  y) = x)
-- #check (le_antisymm : x  $\leq$  y  $\rightarrow$  y  $\leq$  x  $\rightarrow$  x = y)
-- #check (le_inf : z  $\leq$  x  $\rightarrow$  z  $\leq$  y  $\rightarrow$  z  $\leq$  x  $\sqcap$  y)
-- #check (le_refl : x  $\leq$  x)

```



```
-- #check (le_sup_left : x ≤ x ∪ y)
```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.6. En los retículos, $x \sqcup (x \sqcap y) = x$

```
-- -----
-- 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_refl      : 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_refl]
--   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_refl
```

```

    have h1b :  $x \sqcap y \leq x$  := inf_le_left
    show  $x \sqcup (x \sqcap y) \leq x$ 
    exact sup_le h1a h1b }
  have h2 :  $x \leq x \sqcup (x \sqcap y)$  := le_sup_left
  show  $x \sqcup (x \sqcap y) = x$ 
  exact le_antisymm h1 h2

-- 2ª demostración
-- =====

example :  $x \sqcup (x \sqcap y) = x$  :=
by
  have h1 :  $x \sqcup (x \sqcap y) \leq x$  := by simp
  have h2 :  $x \leq x \sqcup (x \sqcap y)$  := by simp
  show  $x \sqcup (x \sqcap y) = x$ 
  exact le_antisymm h1 h2

-- 3ª demostración
-- =====

example :  $x \sqcup (x \sqcap y) = x$  :=
by
  apply le_antisymm
  . --  $x \sqcup (x \sqcap y) \leq x$ 
    apply sup_le
    . --  $x \leq x$ 
      apply le_refl
    . --  $x \sqcap y \leq x$ 
      apply inf_le_left
  . --  $x \leq x \sqcup (x \sqcap y)$ 
    apply le_sup_left

-- 4ª demostración
-- =====

example :  $x \sqcup (x \sqcap y) = x$  :=
-- by apply?
sup_inf_self

-- 5ª demostración
-- =====

example :  $x \sqcup (x \sqcap y) = x$  :=
by simp

```

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

-- variable (z :  $\alpha$ )
-- #check (le_rfl :  $x \leq x$ )
-- #check (inf_le_left :  $x \sqcap y \leq x$ )
-- #check (sup_le :  $x \leq z \rightarrow y \leq z \rightarrow x \sqcup y \leq z$ )
-- #check (le_sup_left :  $x \leq x \sqcup y$ )
-- #check (le_antisymm :  $x \leq y \rightarrow y \leq x \rightarrow x = y$ )
-- #check (sup_inf_self :  $x \sqcup (x \sqcap y) = x$ )
```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.7. En los retículos, una distributiva del ínfimo implica la otra

```
-- -----
-- Demostrar que si  $\alpha$  es un retículo tal que
--    $\forall x y z : \alpha, x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ 
-- entonces
--    $(a \sqcup b) \sqcap c = (a \sqcap c) \sqcup (b \sqcap c)$ 
-- para todos los elementos de  $\alpha$ .
-- -----

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

-- Se demuestra por la siguiente cadena de igualdades
--    $(a \sqcup b) \sqcap c = c \sqcap (a \sqcup b)$            [por conmutatividad de  $\sqcap$ ]
--    $= (c \sqcap a) \sqcup (c \sqcap b)$            [por la hipótesis]
--    $= (a \sqcap c) \sqcup (c \sqcap b)$            [por conmutatividad de  $\sqcap$ ]
--    $= (a \sqcap c) \sqcup (b \sqcap c)$            [por conmutatividad de  $\sqcap$ ]

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

import Mathlib.Order.Lattice
variable { $\alpha$  : Type _} [Lattice  $\alpha$ ]
variable (a b c :  $\alpha$ )

-- 1ª demostración
example
```

```

(h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
: (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
calc
  (a ⊔ b) ⊓ c = c ⊓ (a ⊔ b)           := by rw [inf_comm]
    _ = (c ⊓ a) ⊔ (c ⊓ b) := by rw [h]
    _ = (a ⊓ c) ⊔ (c ⊓ b) := by rw [@inf_comm _ _ c a]
    _ = (a ⊓ c) ⊔ (b ⊓ c) := by rw [@inf_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊓ (y ⊔ z) = (x ⊓ y) ⊔ (x ⊓ z))
  : (a ⊔ b) ⊓ c = (a ⊓ c) ⊔ (b ⊓ c) :=
by simp [h, inf_comm]

-- Lemas usados
-- =====

-- #check (inf_comm : a ⊓ b = b ⊓ a)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 8.8. En los retículos, una distributiva del supremos implica la otra

```

-- -----
-- Demostrar que si α es un retículo tal que
--   ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z)
-- entonces
--   (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c)
-- -----

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

-- Se demuestra por la siguiente cadena de igualdades
--   (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)           [por la conmutatividad de ⊔]
--               = (c ⊔ a) ⊓ (c ⊔ b)       [por la hipótesis]
--               = (a ⊔ c) ⊓ (c ⊔ b)       [por la conmutatividad de ⊔]
--               = (a ⊔ c) ⊓ (b ⊔ c)       [por la conmutatividad de ⊔]

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

```

```

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

-- 1ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
calc
  (a ⊓ b) ⊔ c = c ⊔ (a ⊓ b)           := by rw [sup_comm]
  _           = (c ⊔ a) ⊓ (c ⊔ b)      := by rw [h]
  _           = (a ⊔ c) ⊓ (c ⊔ b)      := by rw [@sup_comm _ _ c a]
  _           = (a ⊔ c) ⊓ (b ⊔ c)      := by rw [@sup_comm _ _ c b]

-- 2ª demostración
example
  (h : ∀ x y z : α, x ⊔ (y ⊓ z) = (x ⊔ y) ⊓ (x ⊔ z))
  : (a ⊓ b) ⊔ c = (a ⊔ c) ⊓ (b ⊔ c) :=
by simp [h, sup_comm]

-- Lemas usados
-- =====

-- #check (sup_comm : a ⊔ b = b ⊔ a)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)



# Capítulo 9

## Anillos ordenados

### 9.1. En los anillos ordenados, $a \leq b \rightarrow 0 \leq b - a$

```
-- -----
-- Demostrar que en los anillos ordenados se verifica que
--    $a \leq b \rightarrow 0 \leq b - a$ 
-- -----

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

-- Se usarán los siguientes lemas:
--   sub_self      :  $a - a = 0$ 
--   sub_le_sub_right :  $a \leq b \rightarrow \forall (c : R), a - c \leq b - c$ 
--
-- Supongamos que
--    $a \leq b$  (1)
-- La demostración se tiene por la siguiente cadena de desigualdades:
--    $0 = a - a$  [por sub_self]
--    $\leq b - a$  [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 :  $a \leq b \rightarrow 0 \leq b - a$  :=
by
```

```

intro h
calc
  0 = a - a := (sub_self a).symm
  _ ≤ b - a := sub_le_sub_right h a

-- 2ª demostración
example : a ≤ b → 0 ≤ b - a :=
sub_nonneg.mpr

-- 3ª demostración
example : a ≤ b → 0 ≤ b - a :=
by simp

-- Lemas usados
-- =====

-- #check (sub_le_sub_right : a ≤ b → ∀ (c : R), a - c ≤ b - c)
-- #check (sub_nonneg : 0 ≤ a - b ↔ b ≤ a)
-- #check (sub_self a : a - a = 0)

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

## 9.2. En los anillos ordenados, $0 \leq b - a \rightarrow a \leq b$

```

-- =====
-- 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 :  $0 \leq b - a \rightarrow a \leq b$  :=
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 :  $0 \leq b - a \rightarrow a \leq b$  :=
-- by apply?
sub_nonneg.mp

-- 3ª demostración
-- =====

example :  $0 \leq b - a \rightarrow a \leq b$  :=
by simp

-- Lemas usados
-- =====

-- #check (zero_add a :  $0 + a = a$ )
-- #check (add_le_add_right :  $b \leq c \rightarrow \forall (a : R), b + a \leq c + a$ )
-- #check (sub_add_cancel a b :  $a - b + b = a$ )
-- #check (sub_nonneg :  $0 \leq a - b \leftrightarrow b \leq a$ )

```

Se puede interactuar con las pruebas anteriores en [Lean 4 Web](#)

### 9.3. En los anillos ordenados, $\{a \leq b, 0 \leq c\} \vdash ac \leq bc$

```

-----
-- Demostrar que, en los anillos ordenados, si
--    $a \leq b$ 
--    $0 \leq c$ 
-- entonces
--    $a * c \leq b * c$ 
-----

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

-- Se usarán los siguientes lemas:
--   sub_nonneg           :  $0 \leq a - b \leftrightarrow b \leq a$ )
--   mul_nonneg           :  $0 \leq a \rightarrow 0 \leq b \rightarrow 0 \leq a * b$ )
--   sub_mul a b c       :  $(a - b) * c = a * c - b * c$ )
--
-- Supongamos que
--    $a \leq b$                                      (1)
--    $0 \leq c$ 
-- De (1), por sub_nonneg, se tiene
--    $0 \leq b - a$ 
-- y con (2), por mul_nonneg, se tiene
--    $0 \leq (b - a) * c$ 
-- que, por sub_mul, da
--    $0 \leq b * c - a * c$ 
-- y, aplicándole sub_nonneg, se tiene
--    $a * c \leq b * c$ 

-- 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](#)

# Bibliografía

- [1] J. A. Alonso. [Lean para matemáticos](#) <sup>1</sup>, 2021.
- [2] J. A. Alonso. [Matemáticas en Lean](#) <sup>2</sup>, 2021.
- [3] J. A. Alonso. [DAO \(Demostración Asistida por Ordenador\) con Lean](#) <sup>3</sup>, 2021.
- [4] J. A. Alonso. [Calculus \(Vol. 1: Demostraciones con Isabelle/HOL y Lean3\)](#) <sup>4</sup>, 2021.
- [5] J. Avigad, L. de Moura, and S. Kong. [Theorem Proving in Lean4](#) <sup>5</sup>, 2021.
- [6] J. Avigad, G. Ebner, and S. Ullrich. [The Lean4 Manual](#) <sup>6</sup>, 2021.
- [7] J. Avigad, M. J. H. Heule, and W. Nawrocki. [Logic and mechanized reasoning](#) <sup>7</sup>, 2023.
- [8] J. Avigad, R. Y. Lewis, and F. van Doorn. [Logic and proof](#) <sup>8</sup>, 2021.
- [9] J. Avigad and P. Massot. [Mathematics in Lean](#) <sup>9</sup>, 2023.
- [10] A. Baanen, A. Bentkamp, J. Blanchette, J. Hölzl, and J. Limperg. [The Hitchhiker's Guide to Logical Verification](#) <sup>10</sup>, 2020.

---

<sup>1</sup>[https://github.com/jaalonso/Lean\\_para\\_matematicos](https://github.com/jaalonso/Lean_para_matematicos)

<sup>2</sup>[https://github.com/jaalonso/Matematicas\\_en\\_Lean](https://github.com/jaalonso/Matematicas_en_Lean)

<sup>3</sup>[https://raw.githubusercontent.com/jaalonso/DAO\\_con\\_Lean/master/DAO\\_con\\_Lean.pdf](https://raw.githubusercontent.com/jaalonso/DAO_con_Lean/master/DAO_con_Lean.pdf)

<sup>4</sup><https://raw.githubusercontent.com/jaalonso/Calculus/master/Calculus.pdf>

<sup>5</sup>[https://leanprover.github.io/theorem\\_proving\\_in\\_lean4/](https://leanprover.github.io/theorem_proving_in_lean4/)

<sup>6</sup><https://leanprover.github.io/lean4/doc/>

<sup>7</sup>[https://avigad.github.io/lamr/logic\\_and\\_mechanized\\_reasoning.pdf](https://avigad.github.io/lamr/logic_and_mechanized_reasoning.pdf)

<sup>8</sup>[https://leanprover.github.io/logic\\_and\\_proof/logic\\_and\\_proof.pdf](https://leanprover.github.io/logic_and_proof/logic_and_proof.pdf)

<sup>9</sup>[https://leanprover-community.github.io/mathematics\\_in\\_lean/](https://leanprover-community.github.io/mathematics_in_lean/)

<sup>10</sup>[https://raw.githubusercontent.com/blanchette/logical\\_verification\\_2020/master/hitchhikers\\_guide.pdf](https://raw.githubusercontent.com/blanchette/logical_verification_2020/master/hitchhikers_guide.pdf)

- [11] M. Ballard. [Transition to advanced mathematics \(Thinking and communicating like a mathematician\)](#) <sup>11</sup>.
- [12] K. Buzzard. [Sets and logic \(in Lean\)](#) <sup>12</sup>.
- [13] K. Buzzard. [Functions and relations \(in Lean\)](#) <sup>13</sup>.
- [14] K. Buzzard. [Course on formalising mathematics](#) <sup>14</sup>, 2021.
- [15] K. Buzzard. [Course on formalising mathematics](#) <sup>15</sup>, 2023.
- [16] K. Buzzard and M. Pedramfar. [The Natural Number Game, version 1.3.3](#) <sup>16</sup>.
- [17] D. T. Christiansen. [Functional programming in Lean](#) <sup>17</sup>, 2023.
- [18] M. Dvořák. [Lean 4 Cheatsheet](#) <sup>18</sup>.
- [19] S. Hazratpour. [Introduction to proofs](#) <sup>19</sup>, 2022.
- [20] S. Hazratpour. [Introduction to proofs with Lean proof assistant](#) <sup>20</sup>, 2022.
- [21] R. Lewis. [Formal proof and verification, 2022](#) <sup>21</sup>, 2022.
- [22] R. Lewis. [Discrete structures and probability](#) <sup>22</sup>, 2023.
- [23] C. Löb. [Exploring formalisation \(A primer in human-readable mathematics in Lean 3 with examples from simplicial topology\)](#) <sup>23</sup>, 2022.
- [24] H. Macbeth. [The mechanics of proof](#) <sup>24</sup>, 2023.
- [25] P. Massot. [Introduction aux mathématiques formalisées](#) <sup>25</sup>.

---

<sup>11</sup><https://300.f22.matthewrobertballard.com/>

<sup>12</sup>[https://www.ma.imperial.ac.uk/~buzzard/M4000x\\_html/M40001/M40001\\_C1.html](https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C1.html)

<sup>13</sup>[https://www.ma.imperial.ac.uk/~buzzard/M4000x\\_html/M40001/M40001\\_C2.html](https://www.ma.imperial.ac.uk/~buzzard/M4000x_html/M40001/M40001_C2.html)

<sup>14</sup><https://github.com/ImperialCollegeLondon/formalising-mathematics>

<sup>15</sup><https://github.com/ImperialCollegeLondon/formalising-mathematics-2023>

<sup>16</sup>[https://www.ma.imperial.ac.uk/~buzzard/xena/natural\\_number\\_game/](https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/)

<sup>17</sup>[https://leanprover.github.io/functional\\_programming\\_in\\_lean/](https://leanprover.github.io/functional_programming_in_lean/)

<sup>18</sup><https://raw.githubusercontent.com/madvorak/lean4-cheatsheet/main/lean-tactics.pdf>

<sup>19</sup><https://introproofs.github.io/s22/>

<sup>20</sup><https://sinhp.github.io/teaching/2022-introduction-to-proofs-with-Lean>

<sup>21</sup><https://github.com/BrownCS1951x/fpv2022>

<sup>22</sup><https://github.com/brown-cs22/CS22-Lean-2023>

<sup>23</sup><https://loeh.app.uni-regensburg.de/mapa/main.pdf>

<sup>24</sup><https://hrmacbeth.github.io/math2001/index.html>

<sup>25</sup><https://www.imo.universite-paris-saclay.fr/~pmassot/enseignement/math114/>

- [26] F. L. Roux. [Code Lean contenant les preuves d'un cours standard sur les espaces métriques](#) <sup>26</sup>, 2020.
- [27] W. Schulze. [Learning LeanProver](#) <sup>27</sup>.
- [28] Varios. [LFTCM 2020: Lean for the Curious Mathematician 2020](#) <sup>28</sup>.
- [29] D. J. Velleman. [How to prove it with Lean](#) <sup>29</sup>.

---

<sup>26</sup>[https://github.com/FredericLeRoux/LEAN\\_ESPACES\\_METRIQUES](https://github.com/FredericLeRoux/LEAN_ESPACES_METRIQUES)

<sup>27</sup>[https://youtube.com/playlist?list=PLYwF9EIrl42RFQgbmcR\\_LSCwRIx2WKbXs](https://youtube.com/playlist?list=PLYwF9EIrl42RFQgbmcR_LSCwRIx2WKbXs)

<sup>28</sup><https://leanprover-community.github.io/lftcm2020/schedule.html>

<sup>29</sup><https://djvelleman.github.io/HTPIwL/>





# Lemas usados

```
import Mathlib.Algebra.Group.Basic
import Mathlib.Algebra.Order.Ring.Defs      -- 1
import Mathlib.Algebra.Ring.Defs
import Mathlib.Analysis.SpecialFunctions.Log.Basic
import Mathlib.Data.Real.Basic
import Mathlib.Order.Lattice
import Mathlib.Topology.MetricSpace.Basic

-- Números naturales
-- =====

section naturales
variable (x y z k m n : ℕ)
#check (_root_.dvd_antisymm : m | n → n | m → m = n)
#check (dvd_add : x | y → x | z → x | y + z)
#check (dvd_gcd : k | m → k | n → k | gcd m n)
#check (dvd_mul_left x y : x | y * x)
#check (dvd_mul_of_dvd_left : x | y → ∀ (c : ℕ), x | y * c)
#check (dvd_mul_of_dvd_right : x | y → ∀ (c : ℕ), x | c * y)
#check (dvd_mul_right x y : x | x * y)
#check (dvd_trans : x | y → y | z → x | z)
#check (gcd_comm m n : gcd m n = gcd n m)
#check (gcd_dvd_left m n : gcd m n | m)
#check (gcd_dvd_right m n : gcd m n | n)
end naturales

-- Números reales
-- =====

section reales
open Real
variable (a b c d : ℝ)
#check (abs_add a b : |a + b| ≤ |a| + |b|)
#check (abs_le' : |a| ≤ b ↔ a ≤ b ∧ -a ≤ b)
```

```

#check (abs_mul a b : |a * b| = |a| * |b|)
#check (abs_nonneg a : 0 ≤ |a|)
#check (abs_sub_abs_le_abs_sub a b : |a| - |b| ≤ |a - b|)
#check (add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d)
#check (add_le_add_left : b ≤ c → ∀ (a : ℝ), a + b ≤ a + c)
#check (add_le_add_right : b ≤ c → ∀ (a : ℝ), b + a ≤ c + a)
#check (add_lt_add_of_le_of_lt : a ≤ b → c < d → a + c < b + d)
#check (add_lt_add_of_lt_of_le : a < b → c ≤ d → a + c < b + d)
#check (add_lt_add_right : b < c → ∀ (a : ℝ), b + a < c + a)
#check (add_neg_le_iff_le_add : a - b ≤ c ↔ a ≤ c + b)
#check (add_pos : 0 < a → 0 < b → 0 < a + b)
#check (add_sub_cancel a b : a + b - b = a)
#check (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 (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_refl a : a ≤ a)
#check (log_le_log' : 0 < a → a ≤ b → log a ≤ log 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_trans : a < b → b < c → a < c)
#check (max_comm a b : max a b = max b a)
#check (max_le : a ≤ c → b ≤ c → max a b ≤ c)
#check (min_add_add_right a b c : min (a + c) (b + c) = min a b + c)
#check (min_assoc a b c : min (min a b) c = min a (min b c))
#check (min_comm a b : min a b = min b a)
#check (min_eq_left : a ≤ b → min a b = a)
#check (min_eq_right : b ≤ a → min a b = b)
#check (min_le_left a b : min a b ≤ a)
#check (min_le_right a b : min a b ≤ b)
#check (mul_comm a b : a * b = b * a)
#check (mul_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_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_sub a b c : a * (b - c) = a * b - a * c)
#check (mul_two a : a * 2 = a + a)
#check (ne_comm : a ≠ b ↔ b ≠ a)
#check (neg_add_self a : -a + a = 0)

```

```

#check (neg_mul_neg a b : -a * -b = a * b)
#check (nonneg_of_mul_nonneg_left : 0 ≤ a * b → 0 < b → 0 ≤ a)
#check (pow_two a : a ^ 2 = a * a)
#check (pow_two_nonneg a : 0 ≤ a ^ 2)
#check (sq_nonneg a : 0 ≤ a ^ 2)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_le_sub_left : a ≤ b → ∀ (c : ℝ), c - b ≤ c - a)
#check (sub_le_sub_right : a ≤ b → ∀ (c : ℝ), a - c ≤ b - c)
#check (sub_sq a b : (a - b) ^ 2 = a ^ 2 - 2 * a * b + b ^ 2)
#check (two_mul a : 2 * a = a + a)
#check (two_mul_le_add_sq a b : 2 * a * b ≤ a ^ 2 + b ^ 2)
#check (zero_lt_one : 0 < 1)
#check (zero_lt_two : 0 < 2)
end reales

-- Anillos
-- =====

section anillos
variable {R : Type _} [Ring R]
variable (a b c : R)
#check (add_assoc a b c : (a + b) + c = a + (b + c))
#check (add_comm a b : a + b = b + a)
#check (add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b)
#check (add_left_cancel : a + b = a + c → b = c)
#check (add_left_neg a : -a + a = 0)
#check (add_mul a b c : (a + b) * c = a * c + b * c)
#check (add_neg_cancel_right a b : (a + b) + -b = a)
#check (add_neg_self a : a + -a = 0)
#check (add_right_cancel : a + b = c + b → a = c)
#check (add_right_neg a : a + -a = 0)
#check (add_zero a : a + 0 = a)
#check (mul_add a b c : a * (b + c) = a * b + a * c)
#check (mul_zero a : a * 0 = 0)
#check (neg_add_cancel_left a b : -a + (a + b) = b)
#check (neg_eq_iff_add_eq_zero : -a = b ↔ a + b = 0)
#check (neg_eq_of_add_eq_zero_left : a + b = 0 → -b = a)
#check (neg_eq_of_add_eq_zero_right : a + b = 0 → -a = b)
#check (neg_neg a : -(-a) = a)
#check (neg_zero : -0 = 0)
#check (one_add_one_eq_two : (1 : R) + 1 = 2)
#check (sub_add_cancel a b : a - b + b = a)
#check (sub_eq_add_neg a b : a - b = a + -b)
#check (sub_mul a b c : (a - b) * c = a * c - b * c)
#check (sub_self a : a - a = 0)

```

```

#check (two_mul a : 2 * a = a + a)
#check (zero_add a : 0 + a = a)
#check (zero_mul a : 0 * a = 0)
end anillos

-- Grupos
-- =====

section grupos
variable {G : Type _} [Group G]
variable (a b c : G)
#check (inv_eq_of_mul_eq_one_right : a * b = 1 → a-1 = b)
#check (mul_assoc a b c : (a * b) * c = a * (b * c))
#check (mul_inv_self a : a * a-1 = 1)
#check (mul_inv_rev a b : (a * b)-1 = b-1 * a-1)
#check (mul_left_inv a : a-1 * a = 1)
#check (mul_one a : a * 1 = a)
#check (mul_right_inv a : a * a-1 = 1)
#check (one_mul a : 1 * a = a)
end grupos

-- Retículos
-- =====

section reticulos
variable {α : Type _} [Lattice α]
variable (x y z : α)
#check (inf_assoc : (x ⊓ y) ⊓ z = x ⊓ (y ⊓ z))
#check (inf_comm : x ⊓ y = y ⊓ x)
#check (inf_le_left : x ⊓ y ≤ x)
#check (inf_le_of_left_le : x ≤ z → x ⊓ y ≤ z)
#check (inf_le_of_right_le : y ≤ z → x ⊓ y ≤ z)
#check (inf_le_right : x ⊓ y ≤ y)
#check (inf_sup_self : x ⊓ (x ⊔ y) = x)
#check (le_antisymm : x ≤ y → y ≤ x → x = y)
#check (le_inf : z ≤ x → z ≤ y → z ≤ x ⊓ y)
#check (le_rfl : x ≤ x)
#check (le_sup_left : x ≤ x ⊔ y)
#check (le_sup_of_le_left : z ≤ x → z ≤ x ⊔ y)
#check (le_sup_of_le_right : z ≤ y → z ≤ x ⊔ y)
#check (le_sup_right : y ≤ x ⊔ y)
#check (le_trans : x ≤ y → y ≤ z → x ≤ z)
#check (sup_assoc : (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z))
#check (sup_comm : x ⊔ y = y ⊔ x)
#check (sup_inf_self : x ⊔ (x ⊓ y) = x)

```

```

#check (sup_le :  $x \leq z \rightarrow y \leq z \rightarrow x \sqcup y \leq z$ )
end reticulos

-- AnillosOrdenados
-- =====

section AnillosOrdenados
variable {R : Type _} [StrictOrderedRing R]
variable (a b c : R)
#check (add_le_add_right :  $b \leq c \rightarrow \forall (a : R), b + a \leq c + a$ )
#check (mul_le_mul_of_nonneg_left :  $b \leq c \rightarrow 0 \leq a \rightarrow a * b \leq a * c$ )
#check (mul_le_mul_of_nonneg_right :  $a \leq b \rightarrow 0 \leq c \rightarrow a * c \leq b * c$ )
#check (mul_nonneg :  $0 \leq a \rightarrow 0 \leq b \rightarrow 0 \leq a * b$ )
#check (sub_le_sub_right :  $a \leq b \rightarrow \forall (c : R), a - c \leq b - c$ )
#check (sub_nonneg :  $0 \leq a - b \leftrightarrow b \leq a$ )
end AnillosOrdenados

-- Espacios métricos
-- =====

section EspacioMetrico
variable {X : Type _} [MetricSpace X]
variable (x y z : X)
#check (dist_comm x y :  $\text{dist } x \ y = \text{dist } y \ x$ )
#check (dist_nonneg :  $0 \leq \text{dist } x \ y$ )
#check (dist_self x :  $\text{dist } x \ x = 0$ )
#check (dist_triangle x y z :  $\text{dist } x \ z \leq \text{dist } x \ y + \text{dist } y \ z$ )
end EspacioMetrico

```