Matemáticas en Lean4

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 29 de julio de 2023

Índice general

Capítulo 1

Introducción

1.1. Resumen

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

Los ejercicios se han ido publicando, desde el 10 de julio de 2022, en el blog Calculemus y su código en GitHub.

1.2. Presentación panorámica de Lean

1.2.1. Ejemplo de evaluación

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

#eval 2 + 3

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

1.2.2. Ejemplo de comprobación con check

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

```
#check 2 + 3

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

1.2.3. Ejemplo de definición de funciones

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

import Mathlib.Data.Nat.Basic

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

def f (x : ℕ) :=
  x + 3

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

#check f

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

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

#eval f 2

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

1.2.4. Ejemplo de proposiciones

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

import Mathlib.Data.Nat.Basic

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

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

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

#check ultimo_teorema_de_Fermat

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

1.2.5. Ejemplo de teoremas

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

import Mathlib.Data.Nat.Basic

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

theorem facil : 2 + 3 = 5 := rfl

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

```
-- 3. La táctica rfl (ver https://bit.ly/3OcOoZL) comprueba que 2+3 y 5
--    son iguales por definición.

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

#check facil

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

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

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

theorem dificil : ultimo_teorema_de_Fermat :=
sorry

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

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

#check dificil

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

1.2.6. Ejemplo de demostración

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

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

open Nat

-- 1ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  ring

-- 2ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  rw [mul_add]

-- 3ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk, mul_add]

-- 4ª demostración
example : ∀ m n : Nat, Even n → Even (m * n) := by
```

```
  rintro m n ⟨k, hk⟩; use m * k; rw [hk, mul_add]

-- 5ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  exact ⟨m * k, by rw [hk, mul_add]⟩

-- 6ª demostración
example : ∀ m n : Nat, Even n → Even (m * n) :=
fun m n ⟨k, hk⟩ ↦ ⟨m * k, by rw [hk, mul_add]⟩

-- 7ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  rintro m n ⟨k, hk⟩
  use m * k
  rw [hk]
  exact mul_add m k k

-- 8ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  intros m n hn
  unfold Even at *
  cases hn with
  | intro k hk =>
    use m * k
    rw [hk, mul_add]

-- 9ª demostración
example : ∀ m n : ℕ, Even n → Even (m * n) := by
  intros m n hn
  unfold Even at *
  cases hn with
  | intro k hk =>
    use m * k
    calc m * n
       = m * (k + k)   := by exact congrArg (HMul.hMul m) hk
     _ = m * k + m * k := by exact mul_add m k k

-- 10ª demostración
example : ∀ m n : Nat, Even n → Even (m * n) := by
  intros; simp [*, parity_simps]

-- Comentarios:
-- 1. Al poner el curso en la línea 1 sobre Mathlib.Data.Nat.Parity y pulsar M-.
--    se abre la teoría correspondiente.
```

```
-- 2. Al colocar el cursor sobre el nombre de un lema se ve su enunciado.
-- 3. Para completar el nombre de un lema basta escribir parte de su
--    nombre y completar con S-SPC (es decir, simultáneamente las teclas
--    de mayúscula y la de espacio).
-- 4. El lema que se ha usado es
--       mul_add a b c : a * (b + c) = a * b + a * c
-- 4. Se activa la ventana de objetivos (*Lean Goal*) pulsando C-c TAB
-- 5. Al mover el cursor sobre las pruebas se actualiza la ventana de
--    objetivos.
```

Capítulo 2

Aspectos básicos del razonamiento matemático en Lean

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

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

2.1. Cálculos

2.1.1. Asociativa conmutativa de los reales

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

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

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

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

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

-- 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]

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

-- El desarrollo de la prueba es:
--
--    inicio
--       a b c : ℝ
--       ⊢ (a * b) * c = b * (a * c)
--    rw [mul_comm a b]
--       a b c : ℝ
--       ⊢ (a * b) * c = b * (a * c)
--    rw [mul_assoc b a c]
--       goals accomplished
```

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

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

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

2.1.2. Ejercicio sobre aritmética real (1)

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

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

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

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

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

-- 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]

-- Desarrollo de la prueba:
-- -----------------------

--    a b c : ℝ
--    ⊢ (c * b) * a = b * (a * c)
-- rw [mul_comm c b]
--    a b c : ℝ
--    ⊢ (b * c) * a = b * (a * c)
-- rw [mul_assoc]
--    a b c : ℝ
--    ⊢ b * (c * a) = b * (a * c)
-- rw [mul_comm c a]
--    goals accomplished

-- 3ª demostración
-- ===============

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

2.1.3. Ejercicio sobre aritmética real (2)

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

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

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

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

import Mathlib.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]

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

-- Desarrollo de la prueba
-- -----------------------

--    a b c : ℝ
```

```
--    ⊢ a * (b * c) = b * (a * c)
-- rw [←mul_assoc]
--    a b c : ℝ
--    ⊢ (a * b) * c = b * (a * c)
-- rw [mul_comm a b]
--    a b c : ℝ
--    ⊢ (b * a) * c = b * (a * c)
-- rw [mul_assoc]
--    goals accomplished

-- 3ª demostración
-- ===============

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

2.1.4. Ejemplo de rw con hipótesis

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

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

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

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

import Mathlib.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]

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

-- Desarrollo de la prueba
-- -----------------------

-- inicio
--    a b c d e f : ℝ,
--    h1 : a * b = c * d,
--    h2 : e = f
--    ⊢ a * (b * e) = c * (d * f)
-- rw [h2]
--    S
--    ⊢ a * (b * f) = c * (d * f)
```

```
-- rw [←mul_assoc]
--    S
--    ⊢ (a * b) * f = c * (d * f)
-- rw [h1]
--    S
--    ⊢ (c * d) * f = c * (d * f)
-- rw [mul_assoc]
--    goals accomplished
--
-- En el desarrollo anterior, S es el conjunto de las hipótesis; es
-- decir,
--    S = {a b c d e f : ℝ,
--         h1 : a * b = c * d,
--         h2 : e = f}

-- 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]
```

2.1.5. Ejercicio de rw con hipótesis (1)

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

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

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

```
--    = ((ae)f)d    [por la asociativa]

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

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

-- 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]

-- El desarrollo de la prueba es
--
-- inicio
--    a b c d e f : ℝ,
--    h : b * c = e * f
--    ⊢ (a * (b * c)) * d = ((a * e) * f) * d
-- rw [mul_assoc a]
--    S
--    ⊢ a * (b * c) * d = a * e * f * d
-- rw [h]
--    S
--    ⊢ a * (e * f) * d = a * e * f * d
```

```
-- rw [←mul_assoc a]
--    goals accomplished
--
-- En el desarrollo anterior, S es el conjunto de hipótesis; es decir,
--    S = {a b c d e f : ℝ,
--         h : b * c = e * f}
```

2.1.6. Ejercicio de rw con hipótesis (2)

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

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

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

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

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

-- 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]

-- Comentario: El último lema se puede encontrar escribiendo previamente
--    exact?
-- y afirma que
--    ∀ (a : G), a - a = 0

-- Desarrollo de la prueba:
--
-- inicio
--    a b c d : ℝ,
--    h1 : c = b * a - d,
--    h2 : d = a * b
--    ⊢ c = 0
-- rw [h1]
--    S
--    ⊢ b * a - d = 0
-- rw [mul_comm]
--    S
--    ⊢ a * b - d = 0
-- rw [h2]
--    S
--    ⊢ a * b - a * b = 0
-- rw sub_self]
--    goals accomplished
--
-- En el desarrollo anterior, S es el conjunto de hipótesis; es decir,
--    S = {a b c d : ℝ,
--         h1 : c = b * a - d,
```

```
--          h2 : d = a * b}
```

2.1.7. Declaración de variables en secciones

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

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

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

section

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

variable (a b c : ℝ)

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

#check a

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

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

#check a + b

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

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

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

#check (a : ℝ)

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

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

#check mul_comm a b

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

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

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

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

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

#check mul_assoc c a b

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

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

```
#check mul_comm a

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

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

#check mul_comm

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

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

#check @mul_comm

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

end
```

2.1.8. Demostración con calc

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

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

-- Por la siguiente cadena de igualdades
--    (a + b)(a + b)
--    = (a + b)a + (a + b)b    [por la distributiva]
--    = aa + ba + (a + b)b     [por la distributiva]
```

```
--     = aa + ba + (ab + bb)     [por la distributiva]
--     = aa + ba + ab + bb       [por la asociativa]
--     = aa + (ba + ab) + bb     [por la asociativa]
--     = aa + (ab + ab) + bb     [por la conmutativa]
--     = aa + 2(ab) + bb         [por def. de doble]

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

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

variable (a b c : ℝ)

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

example :
  (a + b) * (a + b) = a * a + 2 * (a * b) + b * b :=
```

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

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


-- El desarrollo de la prueba es
--
--    a b : ℝ
--    ⊢ (a + b) * (a + b) = a * a + 2 * (a * b) + b * b
-- rw [mul_add]
--    ⊢ (a + b) * a + (a + b) * b = a * a + 2 * (a * b) + b * b
-- rw [add_mul]
--    ⊢ a * a + b * a + (a + b) * b = a * a + 2 * (a * b) + b * b
-- rw [add_mul]
--    ⊢ a * a + b * a + (a * b + b * b) = a * a + 2 * (a * b) + b * b
-- rw [←add_assoc]
--    ⊢ a * a + b * a + a * b + b * b = a * a + 2 * (a * b) + b * b
-- rw add_assoc (a * a)]
--    ⊢ a * a + (b * a + a * b) + b * b = a * a + 2 * (a * b) + b * b
-- rw [mul_comm b a]
--    ⊢ a * a + (a * b + a * b) + b * b = a * a + 2 * (a * b) + b * b
-- rw [←two_mul]
```

```
--    goals accomplished

-- 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]

-- El desarrollo de la prueba es
--
--    a b : ℝ
--    ⊢ a * a + (a * b + a * b) + b * b = a * a + 2 * (a * b) + b * b
-- rw [mul_add, add_mul, add_mul]
--    ⊢ a * a + b * a + (a * b + b * b) = a * a + 2 * (a * b) + b * b
-- rw [←add_assoc, add_assoc (a * a)]
--    ⊢ a * a + (b * a + a * b) + b * b = a * a + 2 * (a * b) + b * b
-- rw [mul_comm b a, ←two_mul]
--    goals accomplished

-- Comentario:
-- Los lemas usados son:
-- + add_assoc a b c : a + b + c = a + (b + c)
-- + add_mul a b c   : (a + b) * c = a * c + b * c
-- + mul_add a b c   : a * (b + c) = a * b + a * c
-- + mul_comm a b    : a * b = b * a
-- + two_mul a       : 2 * a = a + a
```

2.1.9. Ejercicio con calc

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

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

-- Por la siguiente cadena de igualdades
--    (a + b)(c + d)
```

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

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

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

variable (a b c d : ℝ)

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

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

```
example
  : (a + b) * (c + d) = a * c + a * d + b * c + b * d :=
by
   rw [add_mul]
   rw [mul_add]
   rw [mul_add]
   rw [← add_assoc]

-- El desarrollo de la prueba es
--
--    a b c d : ℝ
--    ⊢ (a + b) * (c + d) = a * c + a * d + b * c + b * d
-- rw [add_mul]
--    ⊢ a * (c + d) + b * (c + d) = a * c + a * d + b * c + b * d
-- rw [mul_add]
--    ⊢ a * c + a * d + b * (c + d) = a * c + a * d + b * c + b * d
-- rw [mul_add]
--    ⊢ a * c + a * d + (b * c + b * d) = a * c + a * d + b * c + b * d
-- rw [← add_assoc]
--    goals accomplished

-- 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]
```

2.1.10. Ejercicio: Suma por diferencia

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

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

-- Por la siguiente cadena de igualdades:
--    (a + b)(a - b)
--    = a(a - b) + b(a - b)           [por la distributiva]
--    = (aa - ab) + b(a - b)          [por la distributiva]
--    = (a^2 - ab) + b(a - b)         [por def. de cuadrado]
```

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

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

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

variable (a b c d : ℝ)

-- 1ª demostración
-- ===============

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

-- Comentario. Se han usado los siguientes lemas:
-- + pow_two a : a ^ 2 = a * a
-- + mul_sub a b c : a * (b - c) = a * b - a * c
```

```
-- + add_mul a b c : (a + b) * c = a * c + b * c
-- + add_sub a b c : a + (b - c) = a + b - c
-- + sub_sub a b c : a - b - c = a - (b + c)
-- + add_zero a : a + 0 = a

-- 2ª demostración
-- ===============

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

-- 3ª demostración
-- ===============

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

-- 4ª demostración (por reescritura usando el lema anterior)
-- ==========================================================

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

-- La demostración es
example : (a + b) * (a - b) = a^2 - b^2 :=
by
  rw [sub_eq_add_neg]
  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]


-- El desarrollo de la demostración es
--    ⊢ (a + b) * (a - b) = a ^ 2 - b ^ 2
-- rw [sub_eq_add_neg]
--    ⊢ (a + b) * (a + -b) = a ^ 2 - b ^ 2
-- rw aux]
--    ⊢ a * a + a * -b + b * a + b * -b = a ^ 2 - b ^ 2
-- rw [mul_neg_eq_neg_mul_symm]
--    ⊢ a * a + -(a * b) + b * a + b * -b = a ^ 2 - b ^ 2
-- rw [add_assoc (a * a)]
--    ⊢ a * a + (-(a * b) + b * a) + b * -b = a ^ 2 - b ^ 2
-- rw [mul_comm b a]
--    ⊢ a * a + (-(a * b) + a * b) + b * -b = a ^ 2 - b ^ 2
-- rw [neg_add_self]
--    ⊢ a * a + 0 + b * -b = a ^ 2 - b ^ 2
-- rw [add_zero]
--    ⊢ a * a + b * -b = a ^ 2 - b ^ 2
-- rw [← pow_two]
--    ⊢ a ^ 2 + b * -b = a ^ 2 - b ^ 2
-- rw [mul_neg_eq_neg_mul_symm]
--    ⊢ a ^ 2 + -(b * b) = a ^ 2 - b ^ 2
-- rw [← pow_two]
--    ⊢ a ^ 2 + -b ^ 2 = a ^ 2 - b ^ 2
-- rw [← sub_eq_add_neg]
--    goals accomplished
```

2.1.11. Reescritura en hipótesis y táctica exact

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

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

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

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

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

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

variable (a b c d : ℝ)

-- 1ª demostración
-- ===============

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
calc
  c = d * a + b     := by rw [h1]
  _ = d * a + a * d := by rw [h2]
  _ = a * d + a * d := by rw [mul_comm d a]
  _ = 2 * (a * d)   := by rw [← two_mul (a * d)]
  _ = 2 * a * d     := by rw [mul_assoc]

-- 2ª demostración
-- ===============

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
by
  rw [h2] at h1
  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

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

-- El desarrollo de la prueba es
--
--    a b c d : ℝ,
--    h1 : c = d * a + b,
--    h2 : b = a * d
--    ⊢ c = 2 * a * d
-- rw [h2] at h1
--    a b c d : ℝ,
--    h2 : b = a * d,
--    h1 : c = d * a + a * d
--    ⊢ c = 2 * a * d
-- clear h2
--    a b c d : ℝ,
--    h1 : c = d * a + a * d
--    ⊢ c = 2 * a * d
-- rw [mul_comm d a] at h1
--    a b c d : ℝ,
--    h1 : c = a * d + a * d
--    ⊢ c = 2 * a * d
-- rw [← two_mul (a*d)] at h1
--    a b c d : ℝ,
--    h1 : c = 2 * (a * d)
--    ⊢ c = 2 * a * d
-- rewrite [← mul_assoc 2 a d] at h1
--    a b c d : ℝ,
--    h1 : c = 2 * a * d
--    ⊢ c = 2 * a * d
-- exact h1
--    goals accomplished

-- 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

-- El desarrollo de la prueba es
--
--    a b c d : ℝ,
--    h1 : c = d * a + b,
--    h2 : b = a * d
--    ⊢ c = 2 * a * d
-- rw [h1]
--    ⊢ d * a + b = 2 * a * d
-- rw [h2]
--    ⊢ d * a + a * d = 2 * a * d
-- ring,
--    goals accomplished

-- 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
```

2.1.12. Demostraciones con ring

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

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

variable (a b c d : ℝ)

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

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

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

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

- Ejemplo con nth_rewrite

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

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

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

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

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

variable (a b c : ℝ)

-- 1ª demostración
example
  (h : a + b = c)
  : (a + b) * (a + b) = a * c + b * c :=
calc
  (a + b) * (a + b)
    = (a + b) * c   := 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]
```

2.2. Demostraciones en estructuras algebraicas

2.2.1. Demostraciones en anillos

2.2.2. Axiomas de anillos

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

import Mathlib.Algebra.Ring.Defs

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

variable (R : Type _) [Ring R]

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

#check (add_assoc : ∀ a b c : R, a + b + c = a + (b + c))
#check (add_comm : ∀ a b : R, a + b = b + a)
#check (zero_add : ∀ a : R, 0 + a = a)
#check (add_left_neg : ∀ a : R, -a + a = 0)
#check (mul_assoc : ∀ a b c : R, a * b * c = a * (b * c))
#check (mul_one : ∀ a : R, a * 1 = a)
#check (one_mul : ∀ a : R, 1 * a = a)
#check (mul_add : ∀ a b c : R, a * (b + c) = a * b + a * c)
#check (add_mul : ∀ a b c : R, (a + b) * c = a * c + b * c)
```

2.2.3. Propiedades de anillos conmutativos

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

import Mathlib.Tactic

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

variable (R : Type _) [CommRing R]

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

variable (a b c d : R)

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

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

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

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

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

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

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

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

2.2.4. Propiedades básicas de anillos

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

import Mathlib.Algebra.Ring.Defs

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

namespace myRing

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

variable {R : Type _} [Ring R]

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

variable (a : R)
```

```
-- ---------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    a + 0 = a
-- ----------------------------------------------------------------------

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

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- El desarrollo de la prueba es
--
--    R : Type ?u.599
--    inst : Ring R
--    a : R
--    ⊢ a + 0 = a
-- rw [add_comm]
--    ⊢ 0 + a = a
-- rw [zero_add]
--    goals accomplished

-- 3ª demostración
-- ===============

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

```
-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--    a + -a = 0
-- ----------------------------------------------------------------------

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

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

-- 1ª demostración
-- ===============

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

-- 2ª demostración
-- ===============

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

-- El desarrollo de la prueba es
--
--    R : Type ?u.1925
--    inst : Ring R
--    a : R
--    ⊢ a + -a = 0
-- rw [add_comm]
--    ⊢ -a + a = 0
-- rw [add_left_neg]
--    no goals

-- 3ª demostración
-- ===============

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

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

end myRing

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

#check @myRing.add_zero

-- Comentario: Al colocar el cursor sobre check se obtiene
--    myRing.add_zero : ∀ {R : Type u_1} [_inst_1 : Ring R] (a : R),
--                      a + 0 = a

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

#check @add_zero

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

2.2.5. Lema neg_add_cancel_left

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

import Mathlib.Algebra.Ring.Defs

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

namespace MyRing

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

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

-- ------------------------------------------------------------------
-- Ejercicio 5. Demostrar que para todo a, b ∈ R,
--    -a + (a + b) = b
-- -------------------------------------------------------------------

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

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

-- 1ª demostración
-- ===============

example
  (a b : R)
  : -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
-- ===============

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

-- El desarrollo de la prueba es
--
--    R : Type u_1,
--    _inst_1 : ring R,
--    a b : R
--    ⊢ -a + (a + b) = b
-- rw ← add_assoc,
--    ⊢ -a + a + b = b
-- rw add_left_neg,
--    ⊢ 0 + b = b
-- rw zero_add,
```

```
--    no goals

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

end MyRing
```

2.2.6. Ejercicio neg_add_cancel_right

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

import Mathlib.Algebra.Ring.Defs

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

namespace MyRing

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

variable {R : Type _} [Ring R]

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

variable (a b : R)

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

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

-- Por la siguiente cadena de igualdades
```

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

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

-- 1ª demostración
-- ===============

theorem neg_add_cancel_right : (a + b) + -b = a :=
calc
  (a + b) + -b = a + (b + -b) := by 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]

-- El desarrollo de la prueba es
--
--    R : Type ?u.930
--    inst : Ring R
--    a b : R
--    ⊢ a + b + -b = a
-- rw [add_assoc]
--    ⊢ a + (b + -b) = a
-- rw [add_right_neg]
--    ⊢ a + 0 = a
-- rw [add_zero]
--    goals accomplished

-- 3ª demostración
-- ===============

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

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

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

end MyRing
```

2.2.7. Ejercicio: Cancelativas de la suma

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

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

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

namespace MyRing

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

variable {R : Type _} [Ring R]

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

variable {a b c : R}

-- ---------------------------------------------------------------------
-- Ejercicio 5. Demostrar que si
--    a + b = a + c
-- entonces
--    b = c
-- ----------------------------------------------------------------------

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

-- 1ª demostración en LN
```

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

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

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

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

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

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

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

-- 1ª demostración
-- ===============

theorem add_left_cancel
  (h : a + b = a + c)
  : b = c :=
calc
  b = 0 + b         := by rw [zero_add]
  _ = (-a + a) + b := by rw [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

-- El desarrollo de la prueba es
--
--    R : Type ?u.3961
--    inst : Ring R
--    a b c : R,
--    h : a + b = a + c
--    ⊢ b = c
-- have h1 : -a + (a + b) = -a + (a + c)
--    |    ⊢ -a + (a + b) = -a + (a + c) :=
--    |          congrArg (HAdd.hAdd (-a)) h
--    h : a + b = a + c,
--    h1 : -a + (a + b) = -a + (a + c)
--    ⊢ b = c
-- clear h
--    h1 : -a + (a + b) = -a + (a + c)
--    ⊢ b = c
-- rw [← add_assoc at h1]
--    h1 : -a + a + b = -a + (a + c)
--    ⊢ b = c
-- rw [add_left_neg at h1]
--    h1 : 0 + b = -a + (a + c)
--    ⊢ b = c
-- rw [zero_add at h1]
--    h1 : b = -a + (a + c)
--    ⊢ b = c
```

```
-- rw [← add_assoc at h1]
--    h1 : b = -a + a + c
--    ⊢ b = c
-- rw [add_left_neg at h1]
--    h1 : b = 0 + c
--    ⊢ b = c
-- rw [zero_add at h1]
--    h1 : b = c
--    ⊢ b = c
-- exact h1
--    goals accomplished

-- 3ª demostración
-- ===============

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

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

-- 4ª demostración
-- ===============

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

```
-- ---------------------------------------------------------------------
-- Ejercicio 6. Demostrar que si
--    a + b = c + b
-- entonces
--    a = c
-- ----------------------------------------------------------------------

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end MyRing
```

2.2.8. Lema mul_zero con have

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que en los anillos
--    a * 0 = 0
-- ----------------------------------------------------------------------

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

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

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

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

namespace MyRing

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

-- 1ª demostración
-- ===============

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

end MyRing
```

2.2.9. Ejercicio zero_mul

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que en los anillos,
--    0 * a = 0
-- -----------------------------------------------------------------------

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

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

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

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

namespace MyRing

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

-- 1ª demostración
-- ===============

example : a = a + 0 := (add_zero a).symm
example : a + 0 = a := add_zero a

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

end MyRing
```

Capítulo 3

Bibliografía

- Lean 4 cheatsheet. ~ Martin Dvořák.
- Lean 4 manual.
- Mathematics in Lean. ~ Jeremy Avigad y Patrick Massot.
- Reference sheet for people who know Lean 3 and want to write tactic-based proofs in Lean 4. ~ Martin Dvořák.
- Theorem proving in Lean 4. ~ Jeremy Avigad, Leonardo de Moura, Soonho Kong y Sebastian Ullrich.