DAO (Demostración Asistida por Ordenador) con Lean

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 30 de agosto de 2020

Índice general

1 Introducción **5**

2 Igualdad **7**
2.1 Prueba mediante reescritura . . . 7
2.2 Prueba con lemas y mediante encadenamiento de ecuaciones . . . 11
2.3 Teorema aritmético con hipótesis y uso de lemas . . . 13
2.4 Ejercicios sobre aritmética real . . . 16

3 Conectivas: implicación, equivalencia, conjunción y disyunción) **23**
3.1 Reglas de la implicación . . . 23
3.1.1 Eliminación de la implicación . . . 23
3.1.2 Introducción de la implicación . . . 26
3.2 Reglas de la equivalencia . . . 28
3.2.1 Eliminación de la equivalencia . . . 28
3.3 Reglas de la conjunción . . . 33
3.3.1 Eliminación de la conjunción . . . 33
3.3.2 Introducción de la conjunción . . . 36
3.4 Reglas de la disyunción . . . 39
3.4.1 Eliminación de la disyunción . . . 39
3.5 Ejercicios . . . 41
3.5.1 Monotonía de la suma por la izquierda . . . 41
3.5.2 Monotonía de la suma por la derecha . . . 45
3.5.3 La suma de no negativos es expansiva . . . 48
3.5.4 Suma de no negativos . . . 51
3.5.5 Suma de desigualdades . . . 52
3.5.6 Monotonía de la multiplicación por no negativo . . . 54
3.5.7 Monotonía de la multiplicación por no positivo . . . 56
3.5.8 Conectivas y desigualdades . . . 58
3.5.9 Conmutatividad de la conjunción . . . 63
3.5.10 Formulación equivalente de lemas con dos hipótesis . . . 65

3.5.11 Caracterización de máximo común divisor igual al primer número 66

4 Apéndices **69**

4.1 Resumen de tácticas usadas 69

4.1.1 Demostraciones estructuradas 71

4.1.2 Composiciones y descomposiciones 71

4.2 Resumen de teoremas usados 72

4.3 Estilos de demostración 73

Capítulo 1

Introducción

El objetivo de este trabajo es presentar una introducción a la DAO (Demostración Asistida con Ordenador) usando Lean para usarla en las clases de la asignatura de Razonamiento automático del Máster Universitario en Lógica, Computación e Inteligencia Artificial de la Universidad de Sevilla. Por tanto, el único prerrequisito es, como en el Máster, cierta madurez matemática como la que deben tener los alumnos de los Grados de Matemática y de Informática.

La exposición se hará mediante una colección de ejercicios. En cada ejercicios se mostrarán distintas pruebas del mismo resultado y se comentan las tácticas conforme se van usando y los lemas utilizados en las demostraciones.

Además, en cada ejercicio hay tres enlaces: uno al código, otro que al pulsarlo abre el ejercicio en Lean Web (en una sesión del navegador) de forma que se puede navegar por las pruebas y editar otras alternativas, y el tercero es un enlace a un vídeo explicando las soluciones del ejercicio.

El trabajo se presenta en 2 formas:

- Como un libro en PDF
- Como un proyecto en GitHub.

Además, los vídeos correspondientes a cada uno de los ejercicios se encuentran en YouTube.

El trabajo se basa fundamentalmente en el proyecto lean-tutorials de la Comunidad Lean que, a su vez, se basa en el curso Introduction aux mathématiquesformalisées de Patrick Massot.

Capítulo 2

Igualdad

En este capítulos se presenta el razonamiento con igualdades mediante reescritura.

2.1. Prueba mediante reescritura

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En esta relación se muestra distintas pruebas de la transitividad de
-- la igualdad de los números reales: por reescritura, sustitución, con
-- lemas y automáticas.

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

import data.real.basic   -- 1
variables (x y z : ℝ)    -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que la igualdad en los reales es transitiva; es
-- decir, si x, y y z son números reales tales que x = y e y = z,
-- entonces x = z.
-- ----------------------------------------------------------------------

-- 1ª demostración (con reescritura)
-- =================================
```

```
example
  (h : x = y)
  (h' : y = z)
  : x = z :=
begin
  rw h,
  exact h',
end

-- Prueba:
/-
  x y z : ℝ,
  h : x = y,
  h' : y = z
  ⊢ x = z
rw h,
  ⊢ y = z
exact h',
  no goals
-/

-- Comentarios:
-- + La táctica (rw h) cuando h es una igualdad sustituye en la
--   conclusión el término izquierdo de h por el derecho.
-- + La táctica (exact h) concluye la demostración si h es del tipo de
--   la conclusión.

-- 2ª demostración (con reescritura inversa)
-- =========================================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
begin
  rw ← h',
  exact h,
end

-- Prueba:
/-
  x y z : ℝ,
  h : x = y,
  h' : y = z
  ⊢ x = z
```

```
rw ← h',
  ⊢ x = y
exact h,
  no goals
-/

-- Comentarios:
-- + La táctica (rw ← h) cuando h es una igualdad sustituye en la
--   conclusión el término derecho de h por el izquierdo

-- 3ª demostración (con reescritura en hipótesis)
-- ===============================================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
begin
  rw h' at h,
  exact h,
end

-- Prueba:
/-
  x y z : ℝ,
  h : x = y,
  h' : y = z
  ⊢ x = z
rw h' at h,
  h : x = z
  ⊢ x = z
exact h,
  no goals
-/

-- Comentarios:
-- + La táctica (rw h1 at h2) cuando h1 es una igualdad sustituye en la
--   hipótesis h2 el término izquierdo de h1 por el derecho.

-- 4ª demostración (con reescritura inversa en hipótesis)
-- =======================================================

example
  (h : x = y)
  (h' : y = z)
```

```
  : x = z :=
begin
  rw ← h at h',
  exact h',
end

-- Prueba:
/-
  x y z : ℝ,
  h : x = y,
  h' : y = z
  ⊢ x = z
rw ← h at h',
  h' : x = z
  ⊢ x = z
exact h',
  no goals
-/

-- Comentarios:
-- + La táctica (rw ← h1 at h2) cuando h1 es una igualdad sustituye en la
--   hipótesis h2 el término derecho de h1 por el izquierdo

-- 5ª demostración (con un lema)
-- =============================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
eq.trans h h'

-- Comentarios:
-- + Se ha usado el lema
--   + eq.trans : a = b → b = c → a = c
-- + El lema se puede encontrar con
--      by suggest

-- 6ª demostración (por sustitución)
-- =================================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
```

```
h' ▸ h

-- Comentario:
-- + Si h es una igualdad entonces h ▸ h' es la expresión obtenida sustituyendo
--   en h' el término izquierdo de h por el derecho.

-- 7ª demostración (automática con linarith)
-- =========================================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
by linarith

-- Comentarios:
-- + La táctica linarith demuestra la conclusión mediante aritmética
--   lineal.
-- + La sugerencia de usar linarith se puede obtener escrbiendo
--      by hint

-- 8ª demostración (automática con finish)
-- =======================================

example
  (h : x = y)
  (h' : y = z)
  : x = z :=
by finish

-- Comentario:
-- + La táctica finish demuestra la conclusión de forma automática.
-- + La sugerencia de usar finish se puede obtener escrbiendo
--      by hint
```

2.2. Prueba con lemas y mediante encadenamiento de ecuaciones

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En esta relación se presentan distintas pruebas con Lean de una
-- igualdad con productos de números reales. La primera es por
-- reescritura usando las propiedades asociativa y conmutativa, La
-- segunda es con encadenamiento de ecuaciones. Las restantes son
-- automáticas.

-- ----------------------------------------------------------------------
-- Ejercicio. Sean a, b y c números reales. Demostrar que
--    (a * b) * c = b * (a * c)
-- ----------------------------------------------------------------------

import data.real.basic

variables (a b c : ℝ)

-- 1ª demostración (hacia atrás con rw)
-- ===================================

example : (a * b) * c = b * (a * c) :=
begin
  rw mul_comm a b,
  rw mul_assoc,
end

-- Prueba:
/-
  a b c : ℝ
  ⊢ (a * b) * c = b * (a * c)
rw mul_comm a b,
  ⊢ (b * a) * c = b * (a * c)
rw mul_assoc,
  no goals
-/

-- Comentarios:
-- + Se han usado los lemas
--   + mul_comm : ∀ (a b : ℝ), a * b = b * a
--   + mul_assoc : ∀ (a b c : ℝ), a * b * c = a * (b * c)

-- 2ª demostración (encadenamiento de igualdades)
-- ==============================================

example : (a * b) * c = b * (a * c) :=
begin
  calc (a * b) * c = (b * a) * c : by rw mul_comm a b
```

```
              ...  = b * (a * c) : by rw mul_assoc,
end

-- 3ª demostración (automática con linarith)
-- =========================================

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

-- 4ª demostración (automática con finish)
-- =======================================

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

-- 5ª demostración (automática con ring)
-- =====================================

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

-- Comentarios:
-- + La táctica ring demuestra la conclusión normalizando las
--   expresiones con las reglas de los anillos.
```

2.3. Teorema aritmético con hipótesis y uso de lemas

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En esta relación comentan distintas pruebas con Lean de una igualdad
-- con productos de números reales. La primera es por reescritura usando
-- las propiedades asociativa y conmutativa, La segunda es con
-- encadenamiento de ecuaciones. Las restantes son automáticas.

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

```
import data.real.basic

variables (a b c d : ℝ)

-- 1ª demostración (reescribiendo las hipótesis)
-- ============================================

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
begin
  rw h2 at h1,
  rw mul_comm at h1,
  rw ← two_mul (a * d) at h1,
  rw ← mul_assoc at h1,
  exact h1,
end

-- Prueba:
/-
  a b c d : ℝ,
  h1 : c = d * a + b,
  h2 : b = a * d
  ⊢ c = 2 * a * d
rw h2 at h1,
  h1 : c = d * a + a * d
  ⊢ c = 2 * a * d
rw mul_comm at h1,
  h1 : c = a * d + a * d
  ⊢ c = 2 * a * d
rw ← two_mul (a * d) at h1,
  h1 : c = 2 * (a * d)
  ⊢ c = 2 * a * d
rw ← mul_assoc at h1,
  h1 : c = 2 * a * d
  ⊢ c = 2 * a * d
exact h1,
  no goals
-/

-- Comentarios:
-- + Se han usado los siguientes lemas
--   + mul_comm : ∀ (a b : ℝ), a * b = b * a
```

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

-- 2ª demostración (encadenamiento de ecuaciones)
-- ==============================================

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
begin
  calc
    c   = d * a + b     : by exact h1
    ... = d * a + a * d : by rw h2
    ... = a * d + a * d : by rw mul_comm
    ... = 2 * (a * d)   : by rw two_mul (a * d)
    ... = 2 * a * d     : by rw mul_assoc,
end

-- 3ª demostración (encadenamiento de ecuaciones)
-- ==============================================

example
  (h1 : c = d * a + b)
  (h2 : b = a * d)
  : c = 2 * a * d :=
begin
  calc
    c   = d * a + b     : by exact h1
    ... = d * a + a * d : by rw h2
    ... = 2 * a * d     : by ring,
end

-- 4ª demostración (automática con linarith)
-- =========================================

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

2.4. Ejercicios sobre aritmética real

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En esta relación se comentan distintas pruebas con Lean de ejercicios
-- sobre la aritmética de los números reales. La primera es por
-- reescritura, la segunda es con encadenamiento de ecuaciones y las
-- restantes son automáticas.

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

import data.real.basic     -- 1
variables (a b c d : ℝ)    -- 2

-- ----------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    (c * b) * a = b * (a * c)
--
-- Indicación: Para alguna pueba pueden ser útiles los lemas
-- + mul_assoc : (a * b) * c = a * (b * c)
-- + mul_comm  : a * b = b * a
-- -----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : (c * b) * a = b * (a * c) :=
begin
  rw mul_comm c b,
  rw mul_assoc,
  rw mul_comm c a,
end

-- Prueba:
/-
  a b c : ℝ
  ⊢ (c * b) * a = b * (a * c)
rw mul_comm c b,
  ⊢ (b * c) * a = b * (a * c)
rw mul_assoc,
```

```
  ⊢ b * (c * a) = b * (a * c)
rw mul_comm c a,
  no goals
-/

-- 2ª demostración
-- ===============

example : (c * b) * a = b * (a * c) :=
begin
  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,
end

-- 3ª demostración
-- ===============

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

-- 4ª demostración
-- ===============

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

-- 5ª demostración
-- ===============

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

-- ---------------------------------------------------------------------
-- Ejercicio 3. Demostrar que si
--    c = b * a - d
--    d = a * b
-- entonces c = 0.
--
-- Indicación: Para alguna pueba pueden ser útiles los lemas
-- + mul_comm : a * b = b * a
-- + sub_self : a - a = 0
-- ----------------------------------------------------------------------

example
```

```
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
begin
  rw h2 at h1,
  rw mul_comm b a at h1,
  rw sub_self (a * b) at h1,
  exact h1,
end

-- Prueba:
/-
  a b c d : ℝ,
  h1 : c = b * a - d,
  h2 : d = a * b
  ⊢ c = 0
rw h2 at h1,
  h1 : c = b * a - a * b
  ⊢ c = 0
rw mul_comm b a at h1,
  h1 : c = a * b - a * b
  ⊢ c = 0
rw sub_self (a * b) at h1,
  h1 : c = 0
  ⊢ c = 0
exact h1,
  no goals
-/

-- 2ª demostración
-- ===============

example
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
begin
  calc c = b * a - d     : by rw h1
     ... = b * a - a * b : by rw h2
     ... = a * b - a * b : by rw mul_comm a b
     ... = 0             : by rw sub_self (a*b),
end

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

```
example
  (h1 : c = b * a - d)
  (h2 : d = a * b)
  : c = 0 :=
begin
  calc c = b * a - d     : by rw h1
     ... = b * a - a * b : by rw h2
     ... = 0             : by ring,
end

-- ----------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    (a + b) + a = 2 * a + b
--
-- Indicación: Para alguna pueba pueden ser útiles los lemas
-- + add_assoc : (a + b) + c = a + (b + c)
-- + add_comm  : a + b = b + a
-- + two_mul   : 2 * a = a + a
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : (a + b) + a = 2 * a + b :=
begin
  calc (a + b) + a = a + (b + a) : by rw add_assoc
               ... = a + (a + b) : by rw add_comm b a
               ... = (a + a) + b : by rw ← add_assoc
               ... = 2 * a + b   : by rw two_mul,
end

-- 2ª demostración
-- ===============

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

-- ----------------------------------------------------------------------
-- Ejercicio 5. Demostrar que
--    (a + b) * (a - b) = a^2 - b^2
--
-- Indicación: Para alguna pueba pueden ser útiles los lemas
-- + add_mul  : (a + b) * c = a * c + b * c
-- + add_sub  : a + (b - c) = (a + b) - c
```

```
-- + add_zero : a + 0 = a
-- + mul_comm : a * b = b * a
-- + mul_sub  : a * (b - c) = a * b - a * c
-- + pow_two  : a^2 = a * a
-- + sub_self : a - a = 0
-- + sub_sub  : (a - b) - c = a - (b + c)
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : (a + b) * (a - b) = a^2 - b^2 :=
begin
  rw pow_two a,
  rw pow_two b,
  rw mul_sub (a + b) a b,
  rw add_mul a b a,
  rw add_mul a b b,
  rw mul_comm b a,
  rw ← sub_sub,
  rw ← add_sub,
  rw sub_self,
  rw add_zero,
end

-- Prueba:
/-
  a b : ℝ
  ⊢ (a + b) * (a - b) = a ^ 2 - b ^ 2
rw pow_two a,
  ⊢ (a + b) * (a - b) = a * a - b ^ 2
rw pow_two b,
  ⊢ (a + b) * (a - b) = a * a - b * b
rw mul_sub (a + b) a b,
  ⊢ (a + b) * a - (a + b) * b = a * a - b * b
rw add_mul a b a,
  ⊢ a * a + b * a - (a + b) * b = a * a - b * b
rw add_mul a b b,
  ⊢ a * a + b * a - (a * b + b * b) = a * a - b * b
rw mul_comm b a,
  ⊢ a * a + a * b - (a * b + b * b) = a * a - b * b
rw ← sub_sub,
  ⊢ a * a + a * b - a * b - b * b = a * a - b * b
rw ← add_sub,
  ⊢ a * a + (a * b - a * b) - b * b = a * a - b * b
```

```
rw sub_self,
  ⊢ a * a + 0 - b * b = a * a - b * b
rw add_zero,
  no goals
-/

-- 2ª demostración
-- ===============

example : (a + b) * (a - b) = a^2 - b^2 :=
begin
  calc (a + b) * (a - b)
           = (a + b) * a - (a + b) * b       : by rw mul_sub (a + b) a b
       ... = a * a + b * a - (a + b) * b     : by rw add_mul a b a
       ... = a * a + b * a - (a * b + b * b) : by rw add_mul a b b
       ... = a * a + a * b - (a * b + b * b) : by rw mul_comm b a
       ... = a * a + a * b - a * b - b * b   : by rw sub_sub
       ... = a * a + (a * b - a * b) - b * b : by rw add_sub
       ... = a * a + 0 - b * b               : by rw sub_self
       ... = a * a - b * b                   : by rw add_zero
       ... = a^2 - b * b                     : by rw pow_two a
       ... = a^2 - b^2                       : by rw pow_two b,
end

-- 3ª demostración
-- ===============

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

Capítulo 3

Conectivas: implicación, equivalencia, conjunción y disyunción)

3.1. Reglas de la implicación

3.1.1. Eliminación de la implicación

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con eliminación de la implicación.

-- ----------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar P y Q como variables sobre proposiciones.
-- ----------------------------------------------------------------------

import tactic            -- 1
variables (P Q : Prop)   -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si se tiene (P → Q) y P, entonces se tiene
-- Q.
-- ----------------------------------------------------------------------

-- 1ª demostración (hacia atrás)
-- =============================
```

```
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
begin
  apply h1,
  exact h2,
end

-- Prueba:
/-
  P Q : Prop,
  h1 : P → Q,
  h2 : P
  ⊢ Q
apply h1,
  ⊢ P
exact h2,
  no goals
-/

-- Comentarios:
-- + La táctica (apply h), cuando h es una implicación, aplica la regla
--   de eliminación de la implicación; es decir, si h es (P → Q) y la
--   conclusión coincide con Q, entonces sustituye la conclusión por P.

-- 2ª demostración (hacia adelante)
-- ================================

example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
begin
  exact h1 h2,
end

-- Comentarios:
-- + Si h1 es una demostración de (P → Q) y h2 es una demostración de P,
--   entonces (h1 h2) es una demostración de Q.

-- 3ª demostración (simplificació de la 2ª)
-- ========================================
```

```
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by exact h1 h2

-- 4ª demostración (mediante un término)
-- =====================================

example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
h1 h2

-- 5ª demostración (automática con tauto)
-- ======================================

example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by tauto

-- Comentarios:
-- + La táctica tauto demuestra automáticamente las tautologías.

-- 6ª demostración (automática con finish)
-- =======================================

example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by finish

-- 6ª demostración (automática con solve_by_elim)
-- ==============================================

example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by solve_by_elim
```

```
-- Comentarios:
-- + La táctica solve_by_elim intnta demostrar el objetivo aplicándole
--   reglas de eliminación.
```

3.1.2. Introducción de la implicación

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con eliminación de la implicación.

-- ----------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar P como variable sobre proposiciones.
-- ----------------------------------------------------------------------

import tactic          -- 1
variables (P : Prop)   -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    P → P
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : P → P :=
begin
  intro h,
  exact h,
end

-- Prueba:
/-
  P : Prop
  ⊢ P → P
intro h,
  h : P
  ⊢ P
exact h,
  no goals
```

```
-/

-- Comentarios:
-- + La táctica (intro h), cuando la conclusión es una implicación,
--   aplica la regla de introducción de la implicación; es decir, si la
--   conclusión es (P → Q) entonces añade la hipótesis (h : P) y cambia
--   la conclusión a Q.

-- 3ª demostración (por un término)
-- ================================

example : P → P :=
λ h, h

-- 4ª demostración (mediante id)
-- =============================

example : P → P :=
id

-- Comentario: Se usa el lema
-- + id : P → P

-- 5ª demostración (estructurada)
-- ==============================

example : P → P :=
begin
  assume h : P,
  show P, from h,
end

-- 6ª demostración (estructurada)
-- ==============================

example : P → P :=
assume h, h

-- 7ª demostración (automática con tauto)
-- ======================================

example : P → P :=
by tauto

-- 8ª demostración (automática con finish)
```

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

example : P → P :=
by finish

-- 9ª demostración (por simplificación)
-- ====================================

example : P → P :=
by simp

-- Comentarios:
-- + La táctica simp aplica reglas de simplificación a la conclusión.
```

3.2. Reglas de la equivalencia

3.2.1. Eliminación de la equivalencia

- Enlaces al código y a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con eliminación de la equivalencia

-- ----------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar P, Q y R como variables sobre proposiciones.
-- ----------------------------------------------------------------------

import tactic              -- 1
variables (P Q R : Prop)   -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    P ↔ Q
--    Q → R
-- entonces
--    P → R
-- ----------------------------------------------------------------------

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

```
example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
begin
  intro hP,
  apply hQR,
  cases h with hPQ hQP,
  apply hPQ,
  exact hP,
end

-- Prueba:
/-
  P Q R : Prop,
  h : P ↔ Q,
  hQR : Q → R
  ⊢ P → R
intro hP,
  hP : P
  ⊢ R
apply hQR,
  ⊢ Q
cases h with hPQ hQP,
  hPQ : P → Q,
  hQP : Q → P
  ⊢ Q
apply hPQ,
  ⊢ P
exact hP,
  no goals
-/

-- Comentarios:
-- + La táctica (cases h with h1 h2), cuando la hipótesis h es una
--   equivalencia aplica la regla de eliminación de la equivalencia; es
--   decir, si h es (P ↔ Q), entonces elimina h y añade las hipótesis
--   (h1 : P → Q) y (h2 : Q → P).

-- 2ª demostración (simplificando los últimos pasos de la anterior)
-- ================================================================

example
  (h : P ↔ Q)
```

```
  (hQR : Q → R)
  : P → R :=
begin
  intro hP,
  apply hQR,
  cases h with hPQ hQP,
  exact hPQ hP,
end

-- Prueba:
/-
  P Q R : Prop,
  h : P ↔ Q,
  hQR : Q → R
  ⊢ P → R
intro hP,
  hP : P
  ⊢ R
apply hQR,
  ⊢ Q
cases h with hPQ hQP,
  hPQ : P → Q,
  hQP : Q → P
  ⊢ Q
exact hPQ hP,
  no goals
-/

-- 3ª demostración (simplificando los últimos pasos de la anterior)
-- ===============================================================

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
begin
  intro hP,
  exact hQR (h.1 hP),
end

-- Comentarios:
-- + Si h es la equivalencia (P ↔ Q), entonces h.1 es (P → Q) y h.2 es
--   (Q → P).

-- 4ª demostración (por un término)
```

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

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
λ hP, hQR (h.1 hP)

-- 5ª demostración (por reescritura)
-- ==================================

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
begin
  rw h,
  exact hQR,
end

-- Prueba:
/-
  P Q R : Prop,
  h : P ↔ Q,
  hQR : Q → R
  ⊢ P → R
rw h,
  ⊢ Q → R
exact hQR,
  no goals
-/

-- Comentarios:
-- + La táctica (rw h), cuando h es una equivalencia como (P ↔ Q),
--   sustituye en la conclusión P por Q.

-- 6ª demostración (por reescritura en hipótesis)
-- ==============================================

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
begin
  rw ← h at hQR,
```

```
  exact hQR,
end

-- Prueba:
/-
  P Q R : Prop,
  h : P ↔ Q,
  hQR : Q → R
  ⊢ P → R
rw ← h at hQR,
  hQR : P → R
  ⊢ P → R
exact hQR,
  no goals
-/

-- Comentarios:
-- + La táctica (rw ← h at h'), cuando h es una equivalencia como (P ↔- Q),
--   sustituye en la  hipótesis h' la fórmula Q por P.

-- 7ª demostración (estructurada)
-- ===============================

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
begin
  assume hP : P,
  have hQ : Q, from h.1 hP,
  show R, from hQR hQ,
end

-- Comentarios:
-- + La táctica (assume h : P), cuando la conclusión es de la forma
--   (P → Q), añade la hipótesis P y cambia la conclusión a Q.
-- + La táctica (have h : e) genera dos subojetivos: el primero tiene
--   como conclusión e y el segundo tiene la conclusión actual pero se le
--   añade la hipótesis (h : e).
-- + la táctica (show P, from h) demuestra la conclusión con la prueba h.

-- 8ª demostración (estructurada)
-- ===============================

example
```

```
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
assume hP, hQR (h.1 hP)

-- 9ª demostración (automática)
-- ============================

example
  (h : P ↔ Q)
  (hQR : Q → R)
  : P → R :=
by tauto
```

3.3. Reglas de la conjunción

3.3.1. Eliminación de la conjunción

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con eliminación de la conjunción.

import tactic
variables (P Q : Prop)

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    P ∧ Q → P
-- -----------------------------------------------------------------------

-- 1ª demostración (con intro, cases y exact)
-- ==========================================

example : P ∧ Q → P :=
begin
  intro h,
  cases h with hP hQ,
  exact hP,
end

-- Prueba:
```

```
/-
  P Q : Prop
  ⊢ P ∧ Q → P
intro h,
  h : P ∧ Q
  ⊢ P
cases h with hP hQ,
  hP : P,
  hQ : Q
  ⊢ P
exact hP,
  no goals
-/

-- Comentarios:
-- + La táctica (cases h with h1 h2), cuando la hipótesis h es una
--   conjunción aplica la regla de eliminación de la conjunción; es
--   decir, si h es (P ∧ Q), entonces elimina h y añade las hipótesis
--   (h1 : P) y (h2 : Q).

-- 2ª demostración (con rintro y exact)
-- ====================================

example : P ∧ Q → P :=
begin
  rintro ⟨hP, hQ⟩,
  exact hP,
end

-- Prueba:
/-
  P Q : Prop
  ⊢ P ∧ Q → P
rintro ⟨hP, hQ⟩,
  hP : P,
  hQ : Q
  ⊢ P
exact hP,
  no goals
-/

-- Comentarios:
-- + La táctica (rintro ⟨h1, h2⟩), cuando la conclusión es una
--   implicación cuyo antecedente es una conjunción, aplica las reglsa
--   de introducción de la implicación y de eliminación de la conjunción;
```

```
--   es decir, si la conclusión es (P ∧ Q → R) entonces añade las
--   hipótesis (h1 : P) y (h2 : Q) y cambia la conclusión a R.

-- 3ª demostración (con rintro y assumption)
-- =========================================

example : P ∧ Q → P :=
begin
  rintro ⟨hP, hQ⟩,
  assumption,
end

-- Comentarios:
-- + la táctica assumption concluye la demostración si la conclusión
--   coincide con alguna de las hipótesis.

-- 4ª demostración (estructurada)
-- ==============================

example : P ∧ Q → P :=
begin
  assume h : P ∧ Q,
  show P, from h.1,
end

-- 5ª demostración (estructurada)
-- ==============================

example : P ∧ Q → P :=
assume h, h.1

-- 6ª demostración (con término de prueba)
-- =======================================

example : P ∧ Q → P :=
λ ⟨hP,_⟩, hP

-- 7ª demostración (con lema)
-- ==========================

example : P ∧ Q → P :=
and.left

-- Comentarios:
-- + Se usa el lema
```

```
--      and.left : P ∧ Q → P
-- + El lema se encuentra con
--      by library_search

-- 8ª demostración (automática con auto)
-- =====================================

example : P ∧ Q → P :=
by tauto

-- 9ª demostración (automática con finish)
-- =======================================

example : P ∧ Q → P :=
by finish
```

3.3.2. Introducción de la conjunción

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con introducción de la conjunción.

-- ----------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar P y Q como variables sobre proposiciones.
-- ----------------------------------------------------------------------

import tactic            -- 1
variables (P Q : Prop)   -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que de P y (P → Q) se deduce P ∧ Q
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
```

```
begin
  split,
  { exact hP },
  { apply hPQ,
    exact hP },
end

-- Comentario
-- ----------

-- La táctica split, cuando la conclusión es una conjunción, aplica la
-- regla de eliminación de la conjunción; es decir, si la conclusión es
-- (P ∧ Q), entonces crea dos subojetivos: el primero en el que la
-- conclusión es P y el segundo donde es Q.

-- 2ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
begin
  split,
  { exact hP },
  { exact hPQ hP },
end

-- 3ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
begin
  have hQ : Q := hPQ hP,
  show P ∧ Q, by exact ⟨hP, hQ⟩,
end

-- 4ª demostración
-- ===============

example
  (hP : P)
```

```
  (hPQ : P → Q)
  : P ∧ Q :=
begin
  show P ∧ Q, by exact ⟨hP, hPQ hP⟩,
end

-- 4ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
begin
  exact ⟨hP, hPQ hP⟩,
end

-- 5ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
by exact ⟨hP, hPQ hP⟩

-- 6ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
⟨hP, hPQ hP⟩

-- 7ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
and.intro hP (hPQ hP)

-- Comentario: Se ha usado el lema
```

```
-- + and.intro : P → Q → P ∧ Q

-- 8ª demostración
-- ===============

example
  (hP : P)
  (hPQ : P → Q)
  : P ∧ Q :=
by tauto
```

3.4. Reglas de la disyunción

3.4.1. Eliminación de la disyunción

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En este relación se muestra distintas formas de demostrar un teorema
-- con eliminación de la disyunción

-- ----------------------------------------------------------------------
-- Ejercicio. Realizar las siguientes acciones:
-- 1. Importar la librería de tácticas.
-- 2. Declarar P, Q y R como variables sobre proposiciones.
-- ----------------------------------------------------------------------

import tactic              -- 1
variables (P Q R : Prop)   -- 2

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si
--    P → R y
--    Q → R
-- entonces
--    P ∨ Q → R
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example
  (hPR : P → R)
```

```
  (hQR : Q → R)
  : P ∨ Q → R :=
begin
  intro h,
  cases h with hP hQ,
  { exact hPR hP },
  { exact hQR hQ },
end

-- Comentario
-- ----------

-- La táctica (cases h with h1 h2), cuando la hipótesis h es una
-- disyunción aplica la regla de eliminación de la disyunción; es decir,
-- si h es (P ∨ Q), entonces elimina h y crea dos casos: uno añadiendo
-- la hipótesis (h1 : P)   y otro añadiendo la hipótesis (h2 : Q).

-- 2ª demostración
-- ===============

example
  (hPR : P → R)
  (hQR : Q → R)
  : P ∨ Q → R :=
begin
  rintro (hP | hQ),
  { exact hPR hP },
  { exact hQR hQ },
end

-- Comentario
-- ----------

-- La táctica (rintro (h1 | h2)), cuando la conclusión es una
-- implicación cuyo antecedente es una disyunción, aplica las regla des
-- introducción de la implicación y de eliminación de la disyunción; es
-- decir, si la conclusión es (P ∨ Q → R) entonces crea dos casos: en el
-- primero añade la hipótesis (h1 : P) y cambia a conclusión a R; en el
-- segundo añade la hipótesis (h2 : Q) y cambia la conclusión a R.

-- 3ª demostración
-- ===============

example
  (hPR : P → R)
```

```
  (hQR : Q → R)
  : P ∨ Q → R :=
λ h, or.elim h hPR hQR

-- Comentario: Se ha usado el lema
-- + or.elim : P ∨ Q → (P → R) → (Q → R) → R

-- 3ª demostración
-- ===============

example
  (hPR : P → R)
  (hQR : Q → R)
  : P ∨ Q → R :=
or.rec hPR hQR

-- Comentario: Se ha usado el lema
-- + or.rec    : (P → R) → (Q → R) → P ∨ Q → R

-- 4ª demostración
-- ===============

example
  (hPR : P → R)
  (hQR : Q → R)
  : P ∨ Q → R :=
by tauto
```

3.5. Ejercicios

3.5.1. Monotonía de la suma por la izquierda

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si a, b y c son números reles tales que
-- a ≤ b, entonces c + a ≤ c + b.
--
-- Indicación: Se puede usar el lema
--    sub_nonneg : 0 ≤ a - b ↔ b ≤ a
-- ----------------------------------------------------------------------
```

```
import data.real.basic
variables {a b c : ℝ}

-- 1ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  have h : (c + b) - (c + a) = b - a,
  { ring, },
  { rw h,
    rw sub_nonneg,
    exact hab, },
end

-- Comentario: Se ha usado el lema
-- + sub_nonneg : 0 ≤ a - b ↔ b ≤ a

-- 2ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a             : by exact sub_nonneg.mpr hab
       ... = c + b - (c + a) : by exact (add_sub_add_left_eq_sub b a c).symm,
end

-- Comentario: Se usa el lema
-- + add_sub_add_left_eq_sub : c + a - (c + b) = a - b

-- 3ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a             : sub_nonneg.mpr hab
```

```
       ... = c + b - (c + a) : (add_sub_add_left_eq_sub b a c).symm,
end

-- 4ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a           : sub_nonneg.mpr hab
       ... = c + b - (c + a) : ring
end

-- 5ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  simp,
  exact hab,
end

-- 6ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  rw ← sub_nonneg,
  simp [hab],
end

-- Comentario:
-- + La táctica (simp [h]) aplica reglas de simplificación, ampliadas con
--   h, a la conclusión.

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

```
example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
begin
  simp [hab],
end

-- 8ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
by simp [hab]

-- 9ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
add_le_add_left hab c

-- Comentario: Se ha usado el lema
-- + add_le_add_left : a ≤ b → ∀ (c : ℝ), c + a ≤ c + b

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

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
by linarith

-- 11ª demostración
-- ===============

example
  (hab : a ≤ b)
  : c + a ≤ c + b :=
by finish
```

3.5.2. Monotonía de la suma por la derecha

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si a, b y c son números reles tales que
-- a ≤ b, entonces a + c ≤ b + c.
-- ----------------------------------------------------------------------

import data.real.basic
variables {a b c : ℝ}

-- 1ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  have h : (b + c) - (a + c) = b - a,
  { ring, },
  { rw h,
    rw sub_nonneg,
    exact hab, },
end

-- 2ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a           : by exact sub_nonneg.mpr hab
       ... = b + c - (a + c) : by exact (add_sub_add_right_eq_sub b a c).symm,
end

-- Comentario: Se usa el lema
-- + add_sub_add_right_eq_sub : a + c - (b + c) = a - b

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

```
example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a             : sub_nonneg.mpr hab
       ... = b + c - (a + c) : (add_sub_add_right_eq_sub b a c).symm,
end

-- 4ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  calc 0   ≤ b - a             : sub_nonneg.mpr hab
       ... = b + c - (a + c) : by ring,
end

-- 5ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  simp,
  exact hab,
end

-- 6ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  rw ← sub_nonneg,
  simp [hab],
end

-- 7ª demostración
```

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

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
begin
  simp [hab],
end

-- 8ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
by simp [hab]

-- 9ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
add_le_add_right hab c

-- Comentario: Se ha usado el lema
-- + add_le_add_right : a ≤ b → ∀ (c : ℝ), a + c ≤ b + c

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

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
by linarith

-- 11ª demostración
-- ===============

example
  (hab : a ≤ b)
  : a + c ≤ b + c :=
by finish
```

3.5.3. La suma de no negativos es expansiva

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio 1.. Demostrar si a y b son números reales y a es no
-- negativo, entonces b ≤ a + b
-- ----------------------------------------------------------------------

import data.real.basic
variables {a b : ℝ}

-- 1ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
begin
  calc b = 0 + b : by rw zero_add
     ... ≤ a + b : by exact add_le_add_right ha b,
end

-- Comentario: Se ha usado el lema
-- + zero_add : 0 + a = a

-- 2ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
begin
  calc b = 0 + b : (zero_add b).symm
     ... ≤ a + b : add_le_add_right ha b,
end

-- 3ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
begin
  calc b = 0 + b : by ring
```

```
     ... ≤ a + b : by exact add_le_add_right ha b,
end

-- 4ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
by simp [ha]

-- 5ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
by linarith

-- 6ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
by finish

-- 7ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  : b ≤ a + b :=
le_add_of_nonneg_left ha

-- Comentario: Se ha usado el lema
-- + le_add_of_nonneg_left : 0 ≤ b → a ≤ b + a

-- ----------------------------------------------------------------------
-- Ejercicio 2. Demostrar si a y b son números reales y b es no
-- negativo, entonces a ≤ a + b
-- -----------------------------------------------------------------------

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

```
example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
begin
  calc a = a + 0 : by rw add_zero
     ... ≤ a + b : by exact add_le_add_left hb a,
end

-- Comentario: Se ha usado el lema
-- + add_le_add_left : a ≤ b → ∀ (c : ℝ), c + a ≤ c + b

-- 2ª demostración
-- ===============

example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
begin
  calc a = a + 0 : (add_zero a).symm
     ... ≤ a + b : add_le_add_left hb a,
end

-- 3ª demostración
-- ===============

example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
begin
  calc a = a + 0 : by ring
     ... ≤ a + b : add_le_add_left hb a,
end

-- 4ª demostración
-- ===============

example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
by simp [hb]

-- 5ª demostración
-- ===============
```

```
example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
by linarith

-- 6ª demostración
-- ===============

example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
by finish

-- 7ª demostración
-- ===============

example
  (hb : 0 ≤ b)
  : a ≤ a + b :=
le_add_of_nonneg_right hb

-- Comentario: Se usa el lema
-- + le_add_of_nonneg_right : 0 ≤ b → a ≤ a + b
```

3.5.4. Suma de no negativos

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio 1. Demostrar si a y b son números reales no negativos,
-- entonces a + b es no negativo.
-- ----------------------------------------------------------------------

import data.real.basic

variables (a b : ℝ)

-- 1ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  (hb : 0 ≤ b)
```

```
  : 0 ≤ a + b :=
begin
  calc 0 ≤ a     : ha
     ... ≤ a + b : le_add_of_nonneg_right hb,
end

-- 2ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  (hb : 0 ≤ b)
  : 0 ≤ a + b :=
add_nonneg ha hb

-- Comentario: Se usa el lema
-- + add_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a + b

-- 3ª demostración
-- ===============

example
  (ha : 0 ≤ a)
  (hb : 0 ≤ b)
  : 0 ≤ a + b :=
by linarith
```

3.5.5. Suma de desigualdades

- Enlaces al código, a la sesión en Lean Web y al vídeo.

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

import data.real.basic

variables (a b c d : ℝ)

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

```
example
  (hab : a ≤ b)
  (hcd : c ≤ d)
  : a + c ≤ b + d :=
begin
  calc
    a + c ≤ b + c : add_le_add_right hab c
    ...   ≤ b + d : add_le_add_left hcd b,
end

-- 2ª demostración
example
  (hab : a ≤ b)
  (hcd : c ≤ d)
  : a + c ≤ b + d :=
begin
  have h1 : a + c ≤ b + c :=
    add_le_add_right hab c,
  have h2 : b + c ≤ b + d :=
    add_le_add_left hcd b,
  show a + c ≤ b + d,
    from le_trans h1 h2,
end

-- Comentario: Se ha usado el lema
-- + le_trans: a ≤ b → b ≤ c → a ≤ c

-- 3ª demostración
-- ===============

example
  (hab : a ≤ b)
  (hcd : c ≤ d)
  : a + c ≤ b + d :=
add_le_add hab hcd

-- Comentario: Se ha usado el lema
-- + add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d

-- 4ª demostración
-- ===============

example
  (hab : a ≤ b)
  (hcd : c ≤ d)
```

```
  : a + c ≤ b + d :=
by linarith
```

3.5.6. Monotonía de la multiplicación por no negativo

- Enlaces al código y a la sesión en Lean Web y al vídeo.

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

import data.real.basic

variables {a b c : ℝ}

-- 1ª demostración
-- ===============

example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
  : a * c ≤ b * c :=
begin
  rw ← sub_nonneg,
  have h : b * c - a * c = (b - a) * c,
  { ring },
  { rw h,
    apply mul_nonneg,
    { rw sub_nonneg,
      exact hab },
    { exact hc }},
end

-- Comentario: Se ha usado el lema
-- + mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b

-- 2ª demostración
-- ===============

example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
```

```
  : a * c ≤ b * c :=
begin
  have hab' : 0 ≤ b - a,
  { rw ← sub_nonneg at hab,
    exact hab, },
  have h1 : 0 ≤ (b - a) * c,
  { exact mul_nonneg hab' hc, },
  have h2 : (b - a) * c = b * c - a * c,
  { ring, },
  have h3 : 0 ≤ b * c - a * c,
  { rw h2 at h1,
    exact h1, },
  rw sub_nonneg at h3,
  exact h3,
end

-- 3ª demostración
-- ===============

example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
  : a * c ≤ b * c :=
begin
  have hab' : 0 ≤ b - a,
  { rwa ← sub_nonneg at hab, },
  have h1 : 0 ≤ (b - a) * c,
  { exact mul_nonneg hab' hc },
  have h2 : (b - a) * c = b * c - a * c,
  { ring, },
  have h3 : 0 ≤ b * c - a * c,
  { rwa h2 at h1, },
  rwa sub_nonneg at h3,
end

-- Comentario:
-- + La táctica (rwa h at h'), cuando h es una igualdad. sustituye en la
--   hipótesis h' el término izquierdo de h por el derecho y, a
--   continuación, aplica assumption.
-- + La táctica (rwa ← h at h'), cuando h es una igualdad, sustituye en
--   la hipótesis h' el término derecho de h por el izquierdo y, a
--   continuación, aplica assumption.

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

```
example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
  : a * c ≤ b * c :=
begin
  rw ← sub_nonneg,
  calc 0 ≤ (b - a)*c  : mul_nonneg (by rwa sub_nonneg) hc
     ... =  b*c - a*c : by ring,
end

-- 5ª demostración
-- ===============

example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
  : a * c ≤ b * c :=
mul_mono_nonneg hc hab

-- Comentario: Se usa el lema
-- + mul_mono_nonneg : 0 ≤ c → a ≤ b → a * c ≤ b * c

-- 6ª demostración
-- ===============

example
  (hc : 0 ≤ c)
  (hab : a ≤ b)
  : a * c ≤ b * c :=
by nlinarith

-- Comentario:
-- + La táctica nlinarith es una extensión de linarith con un
--   preprocesamiento que permite resolver problemas aritméticos no
--   lineales.
```

3.5.7. Monotonía de la multiplicación por no positivo

- Enlaces al código y a la sesión en Lean Web y al vídeo.

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

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

import data.real.basic

variables {a b c : ℝ}

-- 1ª demostración
-- ===============

example
  (hc : c ≤ 0)
  (hab : a ≤ b)
  : b * c ≤ a * c :=
begin
  rw ← sub_nonneg,
  have h : a * c - b * c = (a - b) * c,
  { ring },
  { rw h,
    apply mul_nonneg_of_nonpos_of_nonpos,
    { rwa sub_nonpos, },
    { exact hc, }},
end

-- Comentario: Se ha usado los lemas
-- + mul_nonneg_of_nonpos_of_nonpos : a ≤ 0 → b ≤ 0 → 0 ≤ a * b
-- + sub_nonpos : a - b ≤ 0 ↔ a ≤ b

-- 2ª demostración
-- ===============

example
  (hc : c ≤ 0)
  (hab : a ≤ b)
  : b * c ≤ a * c :=
begin
  have hab' : a - b ≤ 0,
  { rwa ← sub_nonpos at hab, },
  have h1 : 0 ≤ (a - b) * c,
  { exact mul_nonneg_of_nonpos_of_nonpos hab' hc, },
  have h2 : (a - b) * c = a * c - b * c,
  { ring, },
  have h3 : 0 ≤ a * c - b * c,
  { rwa h2 at h1, },
  rwa sub_nonneg at h3,
end
```

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

example
  (hc : c ≤ 0)
  (hab : a ≤ b)
  : b * c ≤ a * c :=
begin
  rw ← sub_nonneg,
  have hab' : a - b ≤ 0,
  { rwa sub_nonpos, },
  calc 0 ≤ (a - b)*c  : mul_nonneg_of_nonpos_of_nonpos hab' hc
     ... =  a*c - b*c : by ring,
end

-- 4ª demostración
-- ===============

example
  (hc : c ≤ 0)
  (hab : a ≤ b)
  : b * c ≤ a * c :=
mul_mono_nonpos hc hab

-- Comentario: Se usa el lema
-- + mul_mono_nonpos : 0 ≥ c → b ≤ a → a * c ≤ b * c

-- 5ª demostración
-- ===============

example
  (hc : c ≤ 0)
  (hab : a ≤ b)
  : b * c ≤ a * c :=
by nlinarith
```

3.5.8. Conectivas y desigualdades

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- En esta relación se formulan algunas de las anteriores propiedades de
-- las desigualdades de los números reales usando conectivas.
```

```
import data.real.basic

variables (a b c : ℝ)

-- ----------------------------------------------------------------------
-- Ejercicio 1. Demostrar que
--    0 ≤ a → b ≤ a + b
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : 0 ≤ a → b ≤ a + b :=
begin
  intro ha,
  exact le_add_of_nonneg_left ha,
end

-- 2ª demostración
-- ===============

example : 0 ≤ a → b ≤ a + b :=
le_add_of_nonneg_left

-- 3ª demostración
-- ===============

example : 0 ≤ a → b ≤ a + b :=
by finish

-- ----------------------------------------------------------------------
-- Ejercicio 2. Demostrar que
--    0 ≤ b → a ≤ a + b
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example: 0 ≤ b → a ≤ a + b :=
begin
  intro hb,
  exact le_add_of_nonneg_right hb,
end
```

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

example: 0 ≤ b → a ≤ a + b :=
le_add_of_nonneg_right

-- 3ª demostración
-- ===============

example: 0 ≤ b → a ≤ a + b :=
by finish

-- ----------------------------------------------------------------------
-- Ejercicio 3. Demostrar que
--    (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b
-- ----------------------------------------------------------------------

-- 1ª demostración
-- ===============

example : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b :=
begin
  intros hab,
  cases hab with ha hb,
  exact add_nonneg ha hb,
end

-- 2ª demostración
-- ===============

example : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b :=
begin
  rintros ⟨ha, hb⟩,
  exact add_nonneg ha hb,
end

-- 3ª demostración
-- ===============

example : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b :=
λ ⟨ha, hb⟩, add_nonneg ha hb

-- ----------------------------------------------------------------------
-- Ejercicio 4. Demostrar que
--    0 ≤ a → (0 ≤ b → 0 ≤ a + b)
```

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

-- 1ª demostración
-- ===============

example : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
begin
  intro ha,
  intro hb,
  exact add_nonneg ha hb,
end

-- 2ª demostración
-- ===============

example : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
begin
  intros ha hb,
  exact add_nonneg ha hb,
end

-- 3ª demostración
-- ===============

example : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
λ ha hb, add_nonneg ha hb

-- 4ª demostración
-- ===============

example : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
add_nonneg

-- 5ª demostración
-- ===============

example : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
by intros ; linarith

-- --------------------------------------------------------------------
-- Ejercicio 5. Demostrar que si
--    (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b
-- entonces
--    0 ≤ a → (0 ≤ b → 0 ≤ a + b)
-- ---------------------------------------------------------------------
```

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

example
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
begin
  intro ha,
  intro hb,
  apply H,
  split,
  { exact ha, },
  { exact hb, },
end

-- 2ª demostración
-- ===============

example
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
begin
  intros ha hb,
  apply H,
  split,
  { exact ha, },
  { exact hb, },
end

-- 3ª demostración
-- ===============

example
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
begin
  intros ha hb,
  exact H ⟨ha, hb⟩,
end

-- 4ª demostración
-- ===============

example
```

```
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
λ ha hb, H ⟨ha, hb⟩

-- 5ª demostración
-- ===============

example
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
by tauto

-- 6ª demostración
-- ===============

example
  (H : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b)
  : 0 ≤ a → (0 ≤ b → 0 ≤ a + b) :=
by finish
```

3.5.9. Conmutatividad de la conjunción

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    P ∧ Q → Q ∧ P
-- ----------------------------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
begin
  intro h,
  cases h with hP hQ,
  split,
  { exact hQ },
  { exact hP },
```

```
end

-- 2ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
begin
  rintro ⟨hP, hQ⟩,
  exact ⟨hQ, hP⟩,
end

-- 3ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
λ ⟨hP, hQ⟩, ⟨hQ, hP⟩

-- 4ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
and.comm.mp

-- Comentarios:
-- 1. Se usa el lema
--    + and.comm : P ∧ Q ↔ Q ∧ P
-- 2. Si h es una equivalencia (P ↔ Q, entonces h.mp es (P → Q).
-- 3. Si h es una equivalencia (P ↔ Q, entonces h.mpr es (Q → P).

-- 5ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
begin
  assume h : P ∧ Q,
  have hP : P := h.left,
  have hQ : Q := h.right,
  show Q ∧ P, from ⟨hQ, hP⟩,
end

-- 6ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
begin
```

```
  assume h : P ∧ Q,
  show Q ∧ P, from ⟨h.right, h.left⟩,
end

-- 7ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
by tauto

-- 8ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
by finish
```

3.5.10. Formulación equivalente de lemas con dos hipótesis

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (P ∧ Q → R) ↔ (P → (Q → R))
-- ----------------------------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
-- ===============

example : (P ∧ Q → R) ↔ (P → (Q → R)) :=
begin
  split,
  { intros h hP hQ,
    exact h ⟨hP, hQ⟩, },
  { rintro h ⟨hP, hQ⟩,
    exact h hP hQ, },
end

-- 2ª demostración
```

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

example : (P ∧ Q → R) ↔ (P → (Q → R)) :=
iff.intro (λ h ha hb, h ⟨ha, hb⟩) (λ h ⟨ha, hb⟩, h ha hb)

-- Comentario: Se ha usado el lema
-- + iff.intro : (P → Q) → (Q → P) → (P ↔ Q)

-- 3ª demostración
-- ===============

example : (P ∧ Q → R) ↔ (P → (Q → R)) :=
and_imp

-- Comentario: Se usa el lema
-- + and_imp : (P ∧ Q → R) ↔ (P → (Q → R))

-- 4ª demostración
-- ===============

example : (P ∧ Q → R) ↔ (P → (Q → R)) :=
by simp

-- 5ª demostración
-- ===============

example : (P ∧ Q → R) ↔ (P → (Q → R)) :=
by finish
```

3.5.11. Caracterización de máximo común divisor igual al primer número

- Enlaces al código, a la sesión en Lean Web y al vídeo.

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si a y b son números naturales, entonces
--    a ∣ b ↔ gcd a b = a
-- ----------------------------------------------------------------------

import data.nat.gcd

open nat
```

```
variables (a b : ℕ)

-- 1ª demostración
-- ===============

example : a ∣ b ↔ gcd a b = a :=
begin
  have h1 : gcd a b ∣ a ∧ gcd a b ∣ b,
  { rw ← dvd_gcd_iff, },
  split,
  { intro h2,
    apply dvd_antisymm h1.left,
    rw dvd_gcd_iff,
    exact ⟨dvd_refl a, h2⟩, },
  { intro h3,
    rw ← h3,
    exact h1.right, },
end

-- Comentarios:
-- + La orden (open nat) abre el es espacio de nombre de los naturales.
-- + La relación (a ∣ b) se verifica si a divide a b.
-- + (gcd a b) es el máximo común divisor de a y b.
-- + Si h es la conjunción (P ∧ Q), entonces h.letf es P y h.right es
--   Q.
-- + Se han usado los lemas
--   + dvd_gcd_iff : c ∣ gcd a b ↔ c ∣ a ∧ c ∣ b
--   + dvd_antisymm : a ∣ b → b ∣ a → a = b
--   + dvd_refl a : a ∣ a

-- 2ª demostración
-- ===============

example : a ∣ b ↔ gcd a b = a :=
gcd_eq_left_iff_dvd

-- Comentario: Se ha usado el lema
-- + gcd_eq_left_iff_dvd : a ∣ b ↔ gcd a b = a
```

Capítulo 4

Apéndices

4.1. Resumen de tácticas usadas

- `(apply h)`, cuando `h` es una implicación, aplica la regla de eliminación de la implicación; es decir, si `h` es `(P → Q)` y la conclusión coincide con `Q`, entonces sustituye la conclusión por `P`.
- `assumption` concluye la demostración si la conclusión coincide con alguna de las hipótesis.
- `exact h` concluye la demostración si `h` es del tipo de la conclusión.
- `(cases h with h1 h2)`, cuando la hipótesis `h` es una equivalencia aplica la regla de eliminación de la equivalencia; es decir, si `h` es `(P ↔ Q)`, entonces elimina `h` y añade las hipótesis `(h1 : P → Q)` y `(h2 : Q → P)`.
- `(cases h with h1 h2)`, cuando la hipótesis `h` es una conjunción aplica la regla de eliminación de la conjunción; es decir, si `h` es `(P ∧ Q)`, entonces elimina `h` y añade las hipótesis `(h1 : P)` y `(h2 : Q)`.
- `(cases h with h1 h2)`, cuando la hipótesis `h` es una disyunción aplica la regla de eliminación de la disyunción; es decir, si `h` es `(P ∨ Q)`, entonces elimina `h` y crea dos casos: uno añadiendo la hipótesis `(h1 : P)` y otro añadiendo la hipótesis `(h2 : Q)`.
- `intro h`, cuando la conclusión es una implicación, aplica la regla de introducción de la implicación; es decir, si la conclusión es `(P → Q)` entonces añade la hipótesis `(h : P)` y cambia la conclusión a `Q`.
- `finish` demuestra la conclusión de forma automática.
- `linarith` demuestra la conclusión mediante aritmética lineal.

- `nlinarith` es una extensión de `linarith` con un preprocesamiento que permite resolver problemas aritméticos no lineales.
- `ring` demuestra la conclusión normalizando las expresiones con las regñlas de los anillos.
- `rintro ⟨h1, h2⟩`, cuando la conclusión es una implicación cuyo antecedente es una conjunción, aplica las regla de introducción de la implicación y de eliminación de la conjunción; es decir, si la conclusión es `(P ∧ Q → R)` entonces añade las hipótesis `(h1 : P)` y `(h2 : Q)` y cambia la conclusión a `R`.
- `rintro (h1 | h2)`, cuando la conclusión es una implicación cuyo antecedente es una disyunción, aplica las regla des introducción de la implicación y de eliminación de la disyunción; es decir, si la conclusión es `(P ∨ Q → R)` entonces crea dos casos: en el primero añade la hipótesis `(h1 : P)` y cambia a conclusión a R; en el segundo añade la hipótesis `(h2 : Q)` y cambia la conclusión a `R`.
- `rw h` cuando `h` es una igualdad sustituye en la conclusión el término izquierdo de `h` por el derecho.
- `rw h`, cuando `h` es una equivalencia como `(P ↔ Q)`, sustituye en la conclusión `P` por `Q`.
- `rw ← h` cuando `h` es una igualdad sustituye en la conclusión el término derecho de `h` por el izquierdo
- `rw h at h'` cuando `h` es una igualdad sustituye en la hipótesis `h'` el término izquierdo de `h` por el derecho.
- `rw h at h'` cuando `h` es una equivalencia como `(P ↔ Q)` sustituye en la hipótesis `h'` la fórmula `P` por `Q`.
- `rw ← h at h'` cuando `h` es una igualdad sustituye en la hipótesis `h'` el término derecho de `h` por el izquierdo
- `rw ← h at h'` cuando `h` es una equivalencia como `(P ↔ Q)` sustituye en la hipótesis `h'` la fórmula `Q` por `P`.
- `rwa h` cuando `h` es una igualdad sustituye en la conclusión el término izquierdo de `h` por el derecho y, a continuación, aplica `assumption`.
- `rwa h at h'` cuando `h` es una igualdad sustituye en la hipótesis `h'` el término izquierdo de `h` por el derecho y, a continuación, aplica `assumption`..

- `rwa ← h at h'` cuando `h` es una igualdad sustituye en la hipótesis `h'` el término derecho de `h` por el izquierdo y, a continuación, aplica `assumption`.
- `simp` aplica reglas de simplificación a la conclusión.
- `simp [h]` aplica reglas de simplificación, ampliadas con `h`, a la conclusión.
- `solve_by_elim` intenta demostrar el objetivo aplicándole reglas de eliminación.
- `split`, cuando la conclusión es una conjunción, aplica la regla de eliminación de la conjunción; es decir, si la conclusión es `(P ∧ Q)`, entonces crea dos subojetivos: el primero en el que la conclusión es `P` y el segundo donde es `Q`.
- `tauto` demuestra automáticamente las tautologías.

4.1.1. Demostraciones estructuradas

- `(assume h : P)`, cuando la conclusión es de la forma `(P → Q)`, añade la hipótesis `P` y cambia la conclusión a `Q`.
- `(have h : e)` genera dos subojetivos: el primero tiene como conclusión `e` y el segundo tiene la conclusión actual pero se le añade la hipótesis `(h : e)`.
- `show P, from h` demuestra la conclusión con la prueba `h`.

4.1.2. Composiciones y descomposiciones

- Si `h1` es una demostración de `(P → Q)` y `h2` es una demostración de `P`, entonces `(h1 h2)` es una demostración de `Q`.
- Si `h` es la conjunción `(P ∧ Q)`, entonces `h.letf` es `P` y `h.right` es `Q`.
- Si `h` es la conjunción `(P ∧ Q)`, entonces `h.1` es `P` y `h.2` es `Q`.
- Si `h` es la equivalencia `(P ↔ Q)`, entonces `h.mp` es `(P → Q)` y `h.mpr` es `(Q → P)`.
- Si `h` es la equivalencia `(P ↔ Q)`, entonces `h.1` es `(P → Q)` y `h.2` es `(Q → P)`.
- Si `h` es una igualdad entonces `h ▸ h'` es la expresión obtenida sustituyendo en h' el término izquierdo de `h` por el derecho.

4.2. Resumen de teoremas usados

Los teoremas utilizados son los siguientes:

```
+ add_assoc : (a + b) + c = a + (b + c)
+ add_comm : a + b = b + a
+ add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d
+ add_le_add_left : a ≤ b → ∀ (c : ℝ), c + a ≤ c + b
+ add_le_add_right : a ≤ b → ∀ (c : ℝ), a + c ≤ b + c
+ add_mul : (a + b) * c = a * c + b * c
+ add_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a + b
+ add_sub : a + (b - c) = (a + b) - c
+ add_sub_add_left_eq_sub : c + a - (c + b) = a - b
+ add_sub_add_right_eq_sub : a + c - (b + c) = a - b
+ add_zero : a + 0 = a
+ and.comm : P ∧ Q ↔ Q ∧ P
+ and.intro : P → Q → P ∧ Q
+ and.left : P ∧ Q → P
+ and_imp : (P ∧ Q → R) ↔ (P → (Q → R))
+ dvd_antisymm : a ∣ b → b ∣ a → a = b
+ dvd_gcd_iff : c ∣ gcd a b ↔ c ∣ a ∧ c ∣ b
+ dvd_refl a : a ∣ a
+ eq.trans : a = b → b = c → a = c
+ gcd_eq_left_iff_dvd : a ∣ b ↔ gcd a b = a
+ id : P → P
+ iff.intro : (P → Q) → (Q → P) → (P ↔ Q)
+ le_add_of_nonneg_left : 0 ≤ b → a ≤ b + a
+ le_add_of_nonneg_right : 0 ≤ b → a ≤ a + b
+ le_trans: a ≤ b → b ≤ c → a ≤ c
+ mul_assoc : (a * b) * c = a * (b * c)
+ mul_comm : a * b = b * a
+ mul_mono_nonneg : 0 ≤ c → a ≤ b → a * c ≤ b * c
+ mul_mono_nonpos : 0 ≥ c → b ≤ a → a * c ≤ b * c
+ mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b
+ mul_nonneg_of_nonpos_of_nonpos : a ≤ 0 → b ≤ 0 → 0 ≤ a * b
+ mul_sub : a * (b - c) = a * b - a * c
+ or.elim : P ∨ Q → (P → R) → (Q → R) → R
+ or.rec : (P → R) → (Q → R) → P ∨ Q → R
+ pow_two : a^2 = a * a
+ sub_nonneg : 0 ≤ a - b ↔ b ≤ a
+ sub_nonpos : a - b ≤ 0 ↔ a ≤ b
+ sub_self : a - a = 0
+ sub_sub : (a - b) - c = a - (b + c)
+ two_mul : 2 * a = a + a
+ zero_add : 0 + a = a
```

4.3. Estilos de demostración

Demostración con tácticas	Demostración estructurada	Término de prueba en bruto
intro x,	fix x,	λ x,
intro h,	assume h,	λ h,
have k := _,	have k := _,	have k := _,
let x := _,	let x := _ in	let x := _ in
exact (_ : P)	show P, from _	_:P

- `intro x` equivale a `λ x, _`
- `apply f` equivale a `f _ _ _`
- `refine e1 (e2 _) (e3 _)` equivale a `e1 (e2 _) (e3 _)`
- `exact e` equivale a `e`
- `change e` equivale a `(_ : e)`
- `rw h` equivale a `eq.rec_on h` _
- `induction e` equivale a `T.rec_on foo _ _` donde `T` es el tipo de `e`
- `cases e` equivale a `T.cases_on e _ _` donde `T` es el tipo de `e`
- `split` equivale a `and.intro _ _`
- `have x :` t= equivale a `(λ x, _) t`
- `let x :` t= equivale a `let x :` t in _=
- `revert x` equivale a `_ x`