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, 6 de febrero de 2021

Índice general

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 mathématiques formalisées de Patrick Massot.

El proyecto se crea con

```
leanproject new DAO_con_Lean
```

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) : by 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

example : P ∧ Q → Q ∧ P :=
begin
  assume h : P ∧ Q,
  show Q ∧ P, from ⟨h.2, h.1⟩,
end

-- 7ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
λ h, ⟨h.2, h.1⟩

-- 8ª demostración
-- ===============

example : P ∧ Q → Q ∧ P :=
by tauto

-- 9ª 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 hP hQ, h ⟨hP, hQ⟩)
          (λ h ⟨hP, hQ⟩, h hP hQ)

-- 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. En los naturales, mcd(x,y) = x syss x divide a y

- 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,
  { exact gcd_dvd a b, },
  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_refl : a ∣ a
--   + dvd_antisymm : a ∣ b → b ∣ a → a = b
--   + dvd_gcd_iff : c ∣ gcd a b ↔ c ∣ a ∧ c ∣ b
--   + gcd_dvd : gcd a b ∣ a ∧ gcd a b ∣ b

-- 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

Cuantificadores

4.1. Cuantificador universal

4.1.1. Eliminación del cuantificador universal

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

```
-- ---------------------------------------------------
-- Ejercicio 1. Demostrar que si todos los números
-- naturales tienen la propiedad P, entonces el cero
-- tiene la propiedad P.
-- ---------------------------------------------------

-- 1ª demostración
example
  (h : ∀ n, P n)
  : P 0 :=
-- by library_search
by exact h 0

-- 2ª demostración
example
  (h : ∀ n, P n)
  : P 0 :=
h 0

-- 3ª demostración
example
  (h : ∀ n, P n)
  : P 0 :=
begin
```

```
  specialize h 0,
  exact h,
end

-- 4ª demostración
example
  (h : ∀ n, P n)
  : P 0 :=
-- by hint
by tauto

-- 5ª demostración
example
  (h : ∀ n, P n)
  : P 0 :=
by finish
```

4.1.2. Introducción del cuantificador universal: La función cuadrado es par

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Demostrar que
--    ∀ x : ℝ, (-x)^2 = x^2
-- ----------------------------------------------------

-- 1ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
begin
  intro a,
  -- by library_search,
  exact neg_square a,
end

-- 2ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
neg_square

-- 3ª demostración
example :
```

```
  ∀ x : ℝ, (-x)^2 = x^2 :=
begin
  intro a,
  -- by hint,
  ring,
end

-- 4ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
begin
  intro a,
  norm_num,
end

-- 5ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
by norm_num

-- 6ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
begin
  intro a,
  finish,
end

-- 7ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
by finish

-- 8ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
begin
  intro a,
  simp,
end

-- 9ª demostración
example :
  ∀ x : ℝ, (-x)^2 = x^2 :=
by simp
```

4.1.3. Renombramiento de variables

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

```
-- ---------------------------------------------------
-- Ejercicio ?. Demostrar que si
--    ∀ n, P n
--    ∀ n, P (n-1) → Q n
-- entonces
--    ∀ n, Q n
-- ---------------------------------------------------


-- 1ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
begin
  intro n,
  apply hPQ,
  rename_var n m at hP,
  exact hP (n-1),
end

-- 2ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
begin
  intro a,
  apply hPQ,
  exact hP (a-1),
end

-- 3ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
begin
```

```
  intro a,
  exact hPQ a (hP (a-1)),
end

-- 4ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
λ a, hPQ a (hP (a-1))

-- 5ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
-- by hint
by tauto

-- 6ª demostración
example
  (hP : ∀ n, P n)
  (hPQ : ∀ n, P (n-1) → Q n)
  : ∀ n, Q n :=
by finish
```

4.2. Ejercicios sobre el cuantificador universal

4.2.1. La suma de dos funciones pares es una función par

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

```
variables (f g : ℝ → ℝ)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    par : (ℝ → ℝ) → Prop
-- tal que (par f) expresa que f es par.
-- ----------------------------------------------------
```

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

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    suma : (ℝ → ℝ) → (ℝ → ℝ) → (ℝ → ℝ)
-- tal que (suma f g) es la suma de las funciones f y g.
-- ----------------------------------------------------

@[simp]
def suma (f g: ℝ → ℝ) : ℝ → ℝ :=
λ x, f x + g x

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que la suma de funciones
-- pares es par.
-- ----------------------------------------------------

-- 1ª demostración
example :
  par f → par g →  par (suma f g) :=
begin
  intro hf,
  unfold par at hf,
  intro hg,
  unfold par at hg,
  unfold par,
  intro x,
  unfold suma,
  rw hf,
  rw hg,
end

-- 2ª demostración
example :
  par f → par g →  par (suma f g) :=
begin
  intros hf hg x,
  simp [suma],
  rw [hf, hg],
end

-- 3ª demostración
example :
  par f → par g →  par (suma f g) :=
```

```
begin
  intros hf hg x,
  unfold suma,
  rw [hf, hg],
end

-- 4ª demostración
example :
  par f → par g →  par (suma f g) :=
begin
  intros hf hg x,
  calc (f + g) (-x)
       = f (-x) + g (-x) : rfl
   ... = f x + g (-x)    : by rw hf
   ... = f x + g x       : by rw hg
   ... = (f + g) x       : rfl
end

-- 5ª demostración
example :
  par f → par g →  par (suma f g) :=
begin
  intros hf hg x,
  calc (f + g) (-x)
       = f (-x) + g (-x) : rfl
   ... = f x + g x       : by rw [hf, hg]
end
```

4.2.2. La composición con una función par es par

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    par : (ℝ → ℝ) → Prop
-- tal que (par f) expresa que f es par.
-- ----------------------------------------------------

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

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que si f es par, entonces
```

```
-- (g ∘ f) también lo es.
-- ----------------------------------------------------

-- 1ª demostración
example
 : par f → par (g ∘ f) :=
begin
  intros hf x,
  unfold function.comp,
  rw hf,
end

-- 2ª demostración
example
 : par f → par (g ∘ f) :=
begin
  intros hf x,
  simp,
  rw hf,
end

-- 3ª demostración
example
 : par f → par (g ∘ f) :=
begin
  intros hf x,
  calc (g ∘ f) (-x)
       = g (f (-x)) : rfl
   ... = g (f (x))  : by rw hf
   ... = (g ∘ f) x  : rfl
end

-- 4ª demostración
example
 : par f → par (g ∘ f) :=
begin
  intros hf x,
  calc (g ∘ f) (-x)
       = g (f (-x)) : rfl
   ... = g (f (x))  : by rw hf
end
```

4.2.3. La composición de funciones impares es impar

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    impar : (ℝ → ℝ) → Prop
-- tal que (impar f) expresa que f es impar.
-- ----------------------------------------------------

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

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que la composición de
-- funciones impares es impar.
-- ----------------------------------------------------

-- 1ª demostración
example :
  impar f → impar g →  impar (g ∘ f) :=
begin
  intros hf hg,
  unfold impar at *,
  intro a,
  unfold function.comp,
  specialize hf a,
  rw hf,
  specialize hg (f a),
  rw hg,
end

-- 2ª demostración
example :
  impar f → impar g →  impar (g ∘ f) :=
begin
  intros hf hg a,
  simp,
  rw hf,
  rw hg,
end

-- 3ª demostración
example :
  impar f → impar g →  impar (g ∘ f) :=
```

```
begin
  intros hf hg a,
  simp,
  rw [hf, hg],
end

-- 4ª demostración
example :
  impar f → impar g →  impar (g ∘ f) :=
begin
  intros hf hg x,
  calc (g ∘ f) (-x)
       = g (f (-x))   : rfl
   ... = g (- f x)    : by rw hf
   ... = -(g (f x))   : by rw hg
   ... = -(g ∘ f) x   : rfl,
end

-- 5ª demostración
example :
  impar f → impar g →  impar (g ∘ f) :=
begin
  intros hf hg x,
  calc (g ∘ f) (-x)
       = g (f (-x))   : rfl
   ... = -(g (f x))   : by rw [hf, hg]
end
```

4.2.4. La composición de funciones crecientes es creciente

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    creciente : (ℝ → ℝ) → Prop
-- tal que (creciente f) expresa que f es creciente.
-- ----------------------------------------------------

def creciente (f : ℝ → ℝ) : Prop :=
∀ {x₁ x₂}, x₁ ≤ x₂ → f x₁ ≤ f x₂

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

```
-- Ejercicio 2. Demostrar que la composición de
-- funciones crecientes es creciente.
-- ----------------------------------------------------

-- 1ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
begin
  unfold creciente at *,
  intros x y h,
  unfold function.comp,
  apply hg,
  apply hf,
  exact h,
end

-- 2ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
begin
  intros x y h,
  apply hg,
  apply hf,
  exact h,
end

-- 3ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
begin
  intros x xy h,
  apply hg,
  exact hf h,
end

-- 4ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
```

```
  : creciente (g ∘ f) :=
begin
  intros x y h,
  exact hg (hf h),
end

-- 4ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
λ x y h, hg (hf h)

-- 5ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
begin
  intros x y h,
  specialize hf h,
  exact hg hf,
end

-- 6ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
assume x y,
assume h1 : x ≤ y,
have h2 : f x ≤ f y,
  from hf h1,
show (g ∘ f) x ≤ (g ∘ f) y, from
  calc (g ∘ f) x
       = g (f x)   : rfl
   ... ≤ g (f y)   : hg h2
   ... = (g ∘ f) y : by refl

-- 7ª demostración
example
  (hf : creciente f)
  (hg : creciente g)
  : creciente (g ∘ f) :=
```

```
-- by hint
by tauto

-- Nota. La función predefinida monotone es equivalente
-- a creciente. El lema es
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
-- by library_search
monotone.comp hg hf
```

4.2.5. La composición de una función creciente y una decreciente es decreciente

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

```
-- ---------------------------------------------------
-- Ejercicio 1. Definir la función
--    creciente : (ℝ → ℝ) → Prop
-- tal que (creciente f) expresa que f es creciente.
-- ---------------------------------------------------

def creciente (f : ℝ → ℝ) : Prop :=
∀ {x₁ x₂}, x₁ ≤ x₂ → f x₁ ≤ f x₂

-- ---------------------------------------------------
-- Ejercicio 2. Definir la función
--    decreciente : (ℝ → ℝ) → Prop
-- tal que (decreciente f) expresa que f es decreciente.
-- ---------------------------------------------------

def decreciente (f : ℝ → ℝ) : Prop :=
∀ {x₁ x₂}, x₁ ≤ x₂ → f x₁ ≥ f x₂

-- ---------------------------------------------------
-- Ejercicio 3. Demostrar que si f es creciente y g es
-- decreciente, entonces (g ∘ f) es decreciente.
-- ---------------------------------------------------

-- 1ª demostración
example
  (hf : creciente f)
```

```
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
begin
  unfold creciente decreciente at *,
  intros x y h,
  unfold function.comp,
  apply hg,
  apply hf,
  exact h,
end

-- 2ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
begin
  intros x y h,
  apply hg,
  apply hf,
  exact h,
end

-- 3ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
begin
  intros x y h,
  apply hg,
  exact hf h,
end

-- 4ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
begin
  intros x y h,
  exact hg (hf h),
end

-- 5ª demostración
```

```
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
λ x y h, hg (hf h)

-- 6ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
assume x y,
assume h : x ≤ y,
have h1 : f x ≤ f y,
  from hf h,
show (g ∘ f) x ≥ (g ∘ f) y,
  from hg h1

-- 7ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
assume x y,
assume h : x ≤ y,
show (g ∘ f) x ≥ (g ∘ f) y,
  from hg (hf h)

-- 8ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
λ x y h, hg (hf h)

-- 9ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
-- by hint
by tauto
```

4.2.6. f es creciente syss ∀ x y, x <y → f x ≤ f y

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

```
-- ---------------------------------------------------
-- Ejercicio 1. Definir la función
--    creciente : (ℝ → ℝ) → Prop
-- tal que (creciente f) espresa que f es creciente.
-- ---------------------------------------------------

def creciente (f : ℝ → ℝ) : Prop :=
∀ {x₁ x₂}, x₁ ≤ x₂ → f x₁ ≤ f x₂

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar que f es creciente syss
-- ∀ x y, x < y → f x ≤ f y
-- ---------------------------------------------------

-- 1ª demostración
example :
  creciente f ↔ ∀ {x y}, x < y → f x ≤ f y :=
begin
  unfold creciente,
  split,
  { intros hf x y hxy,
    apply hf,
    -- by library_search
    exact le_of_lt hxy, },
  { intros h x y hxy,
    have h1: x = y ∨ x < y,
      apply eq_or_lt_of_le hxy,
    cases h1 with h2 h3,
    { rw h2, },
    { apply h,
      exact h3, }},
end

-- 2ª demostración
example :
  creciente f ↔ ∀ {x y}, x < y → f x ≤ f y :=
begin
  split,
  { intros hf x y hxy,
    apply hf,
    exact le_of_lt hxy, },
```

```
  { intros h x y hxy,
    have h1: x = y ∨ x < y,
      apply eq_or_lt_of_le hxy,
    cases h1 with h2 h3,
    { rw h2, },
    { apply h,
      exact h3, }},
end

-- 3ª demostración
example :
  creciente f ↔ ∀ {x y}, x < y → f x ≤ f y :=
begin
  split,
  { intros hf x y hxy,
    apply hf,
    linarith, },
  { intros h x y hxy,
    cases (eq_or_lt_of_le hxy) with h2 h3,
    { rw h2, },
    { exact h h3, }},
end
```

4.2.7. Una función creciente e involutiva es la identidad

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    creciente : (ℝ → ℝ) → Prop
-- tal que (creciente f) expresa que f es creciente.
-- ----------------------------------------------------

def creciente (f : ℝ → ℝ) : Prop :=
∀ {x₁ x₂}, x₁ ≤ x₂ → f x₁ ≤ f x₂

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    involutiva : (ℝ → ℝ) → Prop
-- tal que (involutiva f) expresa que f es involutiva.
-- ----------------------------------------------------

def involutiva (f : ℝ → ℝ) : Prop :=
```

```
∀ {x}, f (f x) = x

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que si f es creciente e
-- involutiva, entonces f es la identidad.
-- ----------------------------------------------------

-- 1ª demostración
example
  (hc : creciente f)
  (hi : involutiva f)
  : f = id :=
begin
  -- unfold creciente involutiva at *,
  funext,
  -- unfold id,
  cases (le_total (f x) x) with h1 h2,
  { apply antisymm h1,
    have h3 : f (f x) ≤ f x,
      { apply hc,
        exact h1, },
    rwa hi at h3, },
  { apply antisymm _ h2,
    have h4 : f x ≤ f (f x),
      { apply hc,
        exact h2, },
    rwa hi at h4, },
end

-- 2ª demostración
example
  (hc : creciente f)
  (hi : involutiva f)
  : f = id :=
begin
  funext,
  cases (le_total (f x) x) with h1 h2,
  { apply antisymm h1,
    calc x
         = f (f x) : hi.symm
     ... ≤ f x     : hc h1 },
  { apply antisymm _ h2,
    calc f x
         ≤ f (f x) : hc h2
     ... = x       : hi },
```

```
end
```

4.2.8. Propiedad: ∀ a b : ℝ, a = a * b → a = 0 ∨ b = 1

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Demostrar que para todo a y b, números
-- reales, se tiene
--    a = a * b → a = 0 ∨ b = 1
-- ----------------------------------------------------

-- 1ª demostración
example :
  a = a * b → a = 0 ∨ b = 1 :=
begin
  intro h1,
  have h2 : a * (1 - b) = 0,
    calc a * (1 - b)
         = a * 1 - a * b : mul_sub a 1 b
     ... = a - a * b     : by simp
     ... = 0             : by linarith,
  rw mul_eq_zero at h2,
  cases h2 with ha hb,
    { left,
      exact ha, },
    { right,
      linarith, },
end

-- 2ª demostración
example :
  a = a * b → a = 0 ∨ b = 1 :=
begin
  intro h1,
  have h2 : a * (1 - b) = 0,
    { calc a * (1 - b)
           = a - a * b     : by ring
       ... = 0             : by linarith, },
  rw mul_eq_zero at h2,
  cases h2 with ha hb,
    { left,
      exact ha, },
```

```
    { right,
      linarith, },
end

-- 3ª demostración
example :
  a = a * b → a = 0 ∨ b = 1 :=
begin
  intro h1,
  have h2 : a * (1 - b) = 0,
    { by linarith, },
  rw mul_eq_zero at h2,
  cases h2 with ha hb,
    { left,
      exact ha, },
    { right,
      linarith, },
end

-- 4ª demostración
example :
  a = a * b → a = 0 ∨ b = 1 :=
assume h1: a = a * b,
have h2 : a * (1 - b) = 0,
  by linarith,
have h3 : a = 0 ∨ 1 - b = 0,
  from mul_eq_zero.mp h2,
or.elim h3
  ( assume h3a : a = 0,
    show a = 0 ∨ b = 1,
      from or.inl h3a)
  ( assume h3b : 1 - b = 0,
    have h4 : b = 1,
      from by linarith,
    show a = 0 ∨ b = 1,
      from or.inr h4)
```

4.2.9. Propiedad: ∀ x : ℝ, x^2 = 1 → x = 1 ∨ x = -1

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Demostrar que si
--    x^2 = 1
-- entonces
--    x = 1 ∨ x = -1
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
begin
  have h1 : (x - 1) * (x + 1) = 0,
    calc (x - 1) * (x + 1)
         = x^2 - 1          : by ring
     ... = 1 - 1            : by rw h
     ... = 0                : by ring,
  have h2 : x - 1 = 0 ∨ x + 1 = 0,
    { -- by suggest,
      exact mul_eq_zero.mp h1 },
  cases h2 with h2a h2b,
    { left,
      -- by suggest,
      exact sub_eq_zero.mp h2a, },
    { right,
      -- by library_search,
      exact eq_neg_of_add_eq_zero h2b, },
end

-- 2ª demostración
example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
begin
  have h1 : (x - 1) * (x + 1) = 0,
    linarith,
  have h2 : x - 1 = 0 ∨ x + 1 = 0,
    finish,
  cases h2 with h2a h2b,
    { left,
      linarith, },
    { right,
      linarith, },
end
```

```
-- 3ª demostración
example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
have h1 : (x - 1) * (x + 1) = 0, from
  calc (x - 1) * (x + 1)
       = x^2 - 1          : by ring
   ... = 1 - 1            : by rw h
   ... = 0                : by ring,
have h2 : x - 1 = 0 ∨ x + 1 = 0,
  from mul_eq_zero.mp h1,
or.elim h2
  ( assume h2a : x - 1 = 0,
    have h3 : x = 1,
      from sub_eq_zero.mp h2a,
    show x = 1 ∨ x = -1,
      from or.inl h3)
  ( assume h2b : x + 1 = 0,
    have h4 : x = -1,
      from eq_neg_of_add_eq_zero h2b,
    show x = 1 ∨ x = -1,
      from or.inr h4)

-- 4ª demostración
example
  (h : x^2 = 1)
  : x = 1 ∨ x = -1 :=
have h1 : (x - 1) * (x + 1) = 0,
  by linarith,
have h2 : x - 1 = 0 ∨ x + 1 = 0,
  by finish,
or.elim h2
  ( assume h2a : x - 1 = 0,
    have h3 : x = 1,
      by linarith,
    show x = 1 ∨ x = -1,
      from or.inl h3)
  ( assume h2b : x + 1 = 0,
    have h4 : x = -1,
      by linarith,
    show x = 1 ∨ x = -1,
      from or.inr h4)
```

4.2.10. Propiedad: ∀ x y : ℝ, x^2 = y^2 → x = y ∨ x = -y

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar si
--    x^2 = y^2
-- entonces
--    x = y ∨ x = -y
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
begin
  have h1 : (x - y) * (x + y) = 0,
    calc (x - y) * (x + y)
         = x^2 - y^2       : by ring
     ... = y^2 - y^2       : by rw h
     ... = 0               : by ring,
  have h2 : x - y = 0 ∨ x + y = 0,
    by exact mul_eq_zero.mp h1,
  cases h2 with h2a h2b,
    { left,
      exact sub_eq_zero.mp h2a, },
    { right,
      exact eq_neg_of_add_eq_zero h2b, },
end

-- 2ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
begin
  have h1 : (x - y) * (x + y) = 0,
    by linarith,
  have h2 : x - y = 0 ∨ x + y = 0,
    by finish,
  cases h2 with h2a h2b,
    { left,
      linarith, },
    { right,
      linarith, },
end
```

```
-- 3ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
have h1 : (x - y) * (x + y) = 0, from
  calc (x - y) * (x + y) = x^2 - y^2 : by ring
   ... = y^2 - y^2                    : by rw h
   ... = 0                            : by ring,
have h2 : x - y = 0 ∨ x + y = 0,
  from mul_eq_zero.mp h1,
or.elim h2
  ( assume h2a : x - y = 0,
    have h3 : x = y,
      from sub_eq_zero.mp h2a,
    show x = y ∨ x = -y,
      from or.inl h3)
  ( assume h2b : x + y = 0,
    have h4 : x = -y,
      from eq_neg_of_add_eq_zero h2b,
    show x = y ∨ x = -y,
      from or.inr h4)

-- 4ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
have h1 : (x - y) * (x + y) = 0, from
  by linarith,
have h2 : x - y = 0 ∨ x + y = 0,
  by finish,
or.elim h2
  ( assume h2a : x - y = 0,
    have h3 : x = y,
      by linarith,
    show x = y ∨ x = -y,
      from or.inl h3)
  ( assume h2b : x + y = 0,
    have h4 : x = -y,
      by linarith,
    show x = y ∨ x = -y,
      from or.inr h4)

-- 5ª demostración
example
```

```
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
-- by library_search
eq_or_eq_neg_of_pow_two_eq_pow_two x y h

-- 6ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
begin
 rw ← add_eq_zero_iff_eq_neg,
 rw ← sub_eq_zero,
 rw or_comm,
 rw ← mul_eq_zero,
 rw ← pow_two_sub_pow_two x y,
 rw sub_eq_zero,
 assumption,
end

-- 7ª demostración
example
  (h : x^2 = y^2)
  : x = y ∨ x = -y :=
by rwa [← add_eq_zero_iff_eq_neg,
        ← sub_eq_zero,
        or_comm,
        ← mul_eq_zero,
        ← pow_two_sub_pow_two x y,
        sub_eq_zero]
```

4.3. Cuantificador existencial

4.3.1. Eliminación del cuantificador existencial

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar
--      ∀ n, P n → Q ⊢ (∃ n, P n) → Q
-- ----------------------------------------------------
```

```
-- 1ª demostración
example
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
begin
  intro hP,
  cases hP with n₀ hn₀,
  specialize hPQ n₀,
  exact hPQ hn₀,
end

-- 2ª demostración
example
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
begin
  intro hP,
  cases hP with n₀ hn₀,
  apply hPQ n₀,
  exact hn₀
end

-- 3ª demostración
example
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
assume hP : ∃ n, P n,
exists.elim hP
  ( assume n₀,
    assume hn₀ : P n₀,
    show Q,
      from hPQ n₀ hn₀)

-- 4ª demostración
example
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
assume hP : ∃ n, P n,
exists.elim hP
  ( assume n₀,
    assume hn₀ : P n₀,
    hPQ n₀ hn₀)

-- 5ª demostración
example
```

```
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
assume hP : ∃ n, P n,
exists.elim hP
  ( λ n₀ hn₀, hPQ n₀ hn₀)

-- 6ª demostración
example
  (hPQ : ∀ n, P n → Q)
  : (∃ n, P n) → Q :=
λ hP, exists.elim hP (λ n₀ hn₀, hPQ n₀ hn₀)
```

4.3.2. Introducción del cuantificador existencial

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

```
-- Ejercicio. Demostrar que
--    ∃ k : ℕ, 8 = 2*k
-- ----------------------------------------------------

-- 1ª demostración
example : ∃ k : ℕ, 8 = 2*k :=
begin
  use 4,
  refl,
end

-- 2ª demostración
example : ∃ k : ℕ, 8 = 2*k :=
exists.intro 4
  ( show 8 = 2 * 4,
      from rfl)


-- 3ª demostración
example : ∃ k : ℕ, 8 = 2*k :=
exists.intro 4 rfl
```

4.4. Ejercicios con el cuantificador existencial

4.4.1. Propiedad: ∃ k, n = k + 1 ⊢ n >0

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar que si
--    ∃ k, n = k + 1
-- entonces
--    n > 0
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
begin
  cases h with k₀ hk₀,
  rw hk₀,
  exact nat.succ_pos k₀,
end

-- 2ª demostración
example
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
begin
  cases h with k₀ hk₀,
  rw hk₀,
  linarith,
end

-- 3ª demostración
example
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
begin
  cases h,
  linarith,
end

-- 4ª demostración
example
```

```
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
exists.elim h
 ( assume k₀,
   assume hk₀ : n = k₀ + 1,
   show n > 0, from
     calc n = k₀ + 1 : hk₀
      ...   > 0      : nat.succ_pos k₀)

-- 5ª demostración
example
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
exists.elim h
 ( assume k₀,
   assume hk₀ : n = k₀ + 1,
   show n > 0, from
     calc n = k₀ + 1 : hk₀
      ...   > 0      : by linarith)

-- 5ª demostración
example
  (h : ∃ k : ℕ, n = k + 1)
  : n > 0 :=
exists.elim h (λ _ _, by linarith)
```

4.4.2. Propiedad transitiva de la divisibilidad

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

```
-- #reduce a | b
-- #print notation (|)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si a divide a b y b divide
-- a c, entonces a divide a c.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
```

```
  : a ∣ c :=
begin
  -- unfold has_dvd.dvd at *,
  cases h₁ with k hk,
  cases h₂ with l hl,
  use k*l,
  rw hl,
  rw hk,
  rw mul_assoc,
end

-- 2ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
begin
  cases h₁ with k hk,
  cases h₂ with l hl,
  use k*l,
  calc c = b * l       : by rw hl
     ... = (a * k) * l : by rw hk
     ... = a * (k * l) : by rw mul_assoc,
end

-- 3ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
begin
  cases h₁ with k hk,
  cases h₂ with l hl,
  use k*l,
  rw [hl, hk, mul_assoc],
end

-- 4ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
begin
  cases h₁ with k hk,
```

```
  cases h₂ with l hl,
  use k*l,
  calc c = b * l         : by rw hl
     ... = (a * k) * l : by rw hk
     ... = a * (k * l) : by ring,
end

-- 5ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
begin
  rcases h₂ with ⟨l, rfl⟩,
  rcases h₁ with ⟨k, rfl⟩,
  use k*l,
  rw mul_assoc,
end

-- 6ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
exists.elim h₁
  ( assume k,
    assume h₃ : b = a * k,
    show a ∣ c, from
      exists.elim h₂
        ( assume l,
          assume h₄ : c = b * l,
          have h₅ : c = a * (k * l),
            calc c = b * l         : by rw h₄
               ... = (a * k) * l : by rw h₃
               ... = a * (k * l) : by rw mul_assoc,
          show a ∣ c,
            from exists.intro (k * l) h₅))

-- 7ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
exists.elim h₁
```

```
  (λ k h₃, exists.elim h₂
              (λ l h₄, exists.intro (k * l) (by rw [h₄, h₃, mul_assoc])))

-- 8ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
-- by library_search
dvd_trans h₁ h₂

-- 9ª demostración
example
  (h₁ : a ∣ b)
  (h₂ : b ∣ c)
  : a ∣ c :=
match h₁, h₂ with
| ⟨k, (h₃ : b = a * k)⟩, ⟨l, (h₄ : c = b * l)⟩ :=
  ⟨k * l, show c = a * (k * l), by simp [h₃, h₄, mul_assoc]⟩
end
```

4.4.3. Propiedad: Si divide a los sumandos divide a la suma

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

```
variables (a b c : ℤ)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si a divide a b y a c,
-- entonces a divide a b+c.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
begin
  -- unfold has_dvd.dvd at *,
  cases h1 with k hk,
  rw hk,
```

```
  cases h2 with l hl,
  rw hl,
  use k+l,
  rw left_distrib,
end

-- 2ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
begin
  cases h1 with k hk,
  cases h2 with l hl,
  use k+l,
  rw [hk, hl, left_distrib],
end

-- 3ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
begin
  rcases h1 with ⟨k, rfl⟩,
  rcases h2 with ⟨l, rfl⟩,
  use k+l,
  ring,
end

-- 4ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
dvd.elim h1
  ( assume k,
    assume hk : b = a * k,
    show a ∣ b + c, from
      dvd.elim h2
        ( assume l,
          assume hl : c = a * l,
          have h3 : a * (k + l) = b + c,
            by simp [hk, hl, left_distrib],
```

```
          show a ∣ b + c,
            from dvd.intro (k + l) h3))

-- 5ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
dvd.elim h1 (λ k hk,
  dvd.elim h2 (λ l hl,
     dvd.intro (k + l) (by simp [left_distrib, hk, hl])))

-- 6ª demostración
example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b + c :=
-- by library_search
dvd_add h1 h2
```

4.4.4. Propiedad: Si divide a los sumandos divide a la suma (con condicionales)

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar que
--    a | b → a | c → a | b+c
-- ----------------------------------------------------

#print notation (∣)

-- 1ª demostración
example :
  a ∣ b → a ∣ c → a ∣ b+c :=
begin
  intros hab hac,
  -- unfold has_dvd.dvd at hab,
  cases hab with k hk,
  rw hk,
  cases hac with l hl,
  rw hl,
```

```
  use k+l,
  ring,
end

-- 2ª demostración
example :
  a ∣ b → a ∣ c → a ∣ b+c :=
begin
  intros hab hac,
  -- unfold has_dvd.dvd at hab,
  rcases hab with ⟨k, rfl⟩,
  rcases hac with ⟨l, rfl⟩,
  use k+l,
  ring,
end

-- 3ª demostración
example :
  a ∣ b → a ∣ c → a ∣ b+c :=
begin
  rintros ⟨k, rfl⟩ ⟨l, rfl⟩,
  use k+l,
  ring,
end
```

4.4.5. CNS de divisible por cero

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

```
variable (a : ℤ)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que
--     0 | a ↔ a = 0
-- ----------------------------------------------------

-- 1ª demostración
example : 0 ∣ a ↔ a = 0 :=
begin
  -- unfold has_dvd.dvd,
  split,
  { intro h,
    cases h with k hk,
```

```
    rw hk,
    rw zero_mul, },
  { intro h,
    use 0,
    rw h,
    rw zero_mul, },
end

-- 2ª demostración
example : 0 ∣ a ↔ a = 0 :=
begin
  split,
  { intro h,
    rcases h with ⟨k, rfl⟩,
    rw zero_mul, },
  { intro h,
    rw h, },
end

-- 3ª demostración
example : 0 ∣ a ↔ a = 0 :=
begin
  split,
  { rintro ⟨k, rfl⟩,
    ring, },
  { rintro rfl,
    use 0,
    ring, },
end

-- 4ª demostración
example : 0 ∣ a ↔ a = 0 :=
iff.intro
  ( assume h: 0 ∣ a,
    show a = 0, from
      dvd.elim h
        ( assume k,
          assume hk : a = 0 * k,
          show a = 0, from
            calc a = 0 * k : hk
               ... = 0     : zero_mul k))
  ( assume h : a = 0,
    have h1 : 0 * 0 = a, from
      calc 0 * 0 = 0 : zero_mul 0
             ... = a : by rw ← h,
```

```
    show 0 ∣ a, from dvd.intro 0 h1)

-- 5ª demostración
example : 0 ∣ a ↔ a = 0 :=
iff.intro
  ( assume h: 0 ∣ a,
    show a = 0, from
      dvd.elim h
        ( assume k,
          assume hk : a = 0 * k,
          show a = 0, from
            calc a = 0 * k : hk
               ... = 0     : zero_mul k))
  ( assume h : a = 0,
    show 0 ∣ a,
      from by rw h)

-- 6ª demostración
example : 0 ∣ a ↔ a = 0 :=
iff.intro
  ( assume h: 0 ∣ a,
    show a = 0, from
      dvd.elim h
        ( assume k,
          assume hk : a = 0 * k,
          show a = 0,
            from by rw [hk, zero_mul]))
  ( assume h : a = 0,
    show 0 ∣ a,
      from by rw h)

-- 7ª demostración
example : 0 ∣ a ↔ a = 0 :=
iff.intro
  (λ h, dvd.elim h (λ k hk, by rw [hk, zero_mul]))
  (λ h, by rw h)

-- 8ª demostración
example : 0 ∣ a ↔ a = 0 :=
⟨λ h, dvd.elim h (λ k hk, by rw [hk, zero_mul]),
 λ h, by rw h⟩

-- 9ª demostración
example : 0 ∣ a ↔ a = 0 :=
-- by library_search
```

```
zero_dvd_iff

-- 8ª demostración
example : 0 ∣ a ↔ a = 0 :=
-- by hint
by norm_num

-- 10ª demostración
example : 0 ∣ a ↔ a = 0 :=
by finish

-- 11ª demostración
example : 0 ∣ a ↔ a = 0 :=
by simp
```

4.4.6. Propiedad: Si (g ∘ f) es suprayectiva, entonces g es suprayectiva

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

```
variables (f g : ℝ → ℝ)

-- #print surjective
-- #print notation (∘)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si (g ∘ f) es suprayectiva,
-- entonces g también lo es
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
begin
  -- unfold surjective at *,
  intro y,
  specialize h y,
  cases h with x hx,
  -- unfold comp at hx,
  use f x,
  exact hx,
```

```
end

-- 2ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
begin
  intro y,
  specialize h y,
  cases h with x hx,
  exact ⟨f x, hx⟩,
end

-- 3ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
begin
  intro y,
  cases h y with x hx,
  rw ← hx,
  use f x,
end

-- 4ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
begin
  intro y,
  rcases h y with ⟨x, rfl⟩,
  use f x,
end

-- 5ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
assume y,
have h1 : ∃ a, (g ∘ f) a = y,
  from h y,
exists.elim h1
  ( assume x,
    assume hx : (g ∘ f) x = y,
    show ∃ a, g a = y,
```

```
      from exists.intro (f x) hx)

-- 6ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
assume y,
have h1 : ∃ a, (g ∘ f) a = y,
  from h y,
exists.elim h1
  ( assume x,
    assume hx : (g ∘ f) x = y,
    exists.intro (f x) hx)

-- 7ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
assume y,
have h1 : ∃ a, (g ∘ f) a = y,
  from h y,
exists.elim h1
  (λ x hx, exists.intro (f x) hx)

-- 8ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
assume y,
exists.elim (h y)
  (λ x hx, exists.intro (f x) hx)

-- 9ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
λ y, exists.elim (h y) (λ x hx, exists.intro (f x) hx)

-- 10ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
λ y, exists.elim (h y) (λ x hx, ⟨f x, hx⟩)

-- 11ª demostración
```

```
example
  (h : surjective (g ∘ f))
  : surjective g :=
-- by library_search
surjective.of_comp h

-- 12ª demostración
example
  (h : surjective (g ∘ f))
  : surjective g :=
λ y, let ⟨x, hx⟩ := h y in ⟨f x, hx⟩
```

4.4.7. Propiedad: La composición de funciones suprayectivas es suprayectiva

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

```
variables (f g : ℝ → ℝ)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si f y g son suprayectivas,
-- entonces también los es su composición.
-- ----------------------------------------------------

-- 1ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
begin
  -- unfold surjective at *,
  intro z,
  cases hg z with y hy,
  rw ← hy,
  cases hf y with x hx,
  rw ← hx,
  -- unfold comp,
  use x,
end

-- 2ª demostración
example
```

```
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
begin
  intro z,
  rcases hg z with ⟨y, rfl⟩,
  rcases hf y with ⟨x, rfl⟩,
  use x,
end

-- 3ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
assume z,
exists.elim (hg z)
  ( assume y,
    assume hy : g y = z,
    exists.elim (hf y)
      ( assume x,
        assume hx : f x = y,
        show ∃ a, (g ∘ f) a = z,
          from exists.intro x
                ( calc (g ∘ f) x
                       = g (f x) : rfl
                   ... = g y     : by rw hx
                   ... = z       : hy)))

-- 4ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
assume z,
exists.elim (hg z)
  ( assume y,
    assume hy : g y = z,
    exists.elim (hf y)
      ( assume x,
        assume hx : f x = y,
        show ∃ a, (g ∘ f) a = z,
          from exists.intro x
                ( show g (f x) = z,
                    from by rw [hx, hy])))
```

```
-- 5ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
assume z,
exists.elim (hg z)
  (λ y hy, exists.elim (hf y)
             (λ x hx, exists.intro x (by simp [hx, hy])))

-- 5ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
assume z,
exists.elim (hg z)
  (λ y hy, exists.elim (hf y)
             (λ x hx, ⟨x, by simp [hx, hy]⟩))

-- 6ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
λ z, exists.elim (hg z) (λ y hy, exists.elim (hf y) (λ x hx, ⟨x, by simp [hx, hy]⟩))

-- 7ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
-- by library_search
surjective.comp hg hf
```

Capítulo 5

Límites de sucesiones

5.1. Límites de sucesiones

5.1.1. Límite de sucesiones constantes

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

```
variable (c : ℝ)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ----------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que si u es la sucesión
-- constante c, entonces el límite de u es c.
-- ----------------------------------------------------
```

```
-- 1ª demostración
example :
  limite (λ n, c) c :=
begin
  -- unfold limite,
  intros ε hε,
  use 0,
  intros n hn,
  -- dsimp,
  norm_num,
  linarith,
end

-- 2ª demostración
example :
  limite (λ n, c) c :=
begin
  intros ε hε,
  use 0,
  intros n hn,
  norm_num,
  linarith,
end

-- 3ª demostración
example :
  limite (λ n, c) c :=
begin
  intros ε hε,
  use 0,
  intros n hn,
  calc |(λ n, c) n - c|
       = |c - c|  : rfl
   ... = 0        : by norm_num
   ... ≤ ε        : by linarith,
end

-- 4ª demostración
example :
  limite (λ n, c) c :=
assume ε,
assume hε : ε > 0,
exists.intro 0
  ( assume n,
    assume hn : n ≥ 0,
```

```
    show |(λ n, c) n - c| ≤ ε, from
      calc |(λ n, c) n - c|
           = |c - c|   : rfl
       ... = 0         : by norm_num
       ... ≤ ε         : by linarith)
```

5.1.2. Si el límite de la sucesión u es c y c >0, entonces u(n) ≥ c/2 a partir de un N

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

```
variable (c : ℝ)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ----------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que si c > 0 y es  el límite
-- de la sucesión u, entonces u(n) ≥ c/2 a partir de un
-- N.
-- ----------------------------------------------------

-- 1ª demostración
example
  (hc : c > 0)
  (h : limite u c)
  : ∃ N, ∀ n ≥ N, u n ≥ c/2 :=
begin
```

```
  have h1 : c/2 > 0,
    by linarith,
  specialize h (c/2) h1,
  cases h with N hN,
  use N,
  intros n hn,
  specialize hN n hn,
  rw abs_le at hN,
  linarith,
end

-- 1ª demostración
example
  (hc : c > 0)
  (h : limite u c)
  : ∃ N, ∀ n ≥ N, u n ≥ c/2 :=
begin
  specialize h (c/2) (by linarith),
  cases h with N hN,
  use N,
  intros n hn,
  specialize hN n hn,
  rw abs_le at hN,
  linarith,
end

-- 3ª demostración
example
  (hc : c > 0)
  (h : limite u c)
  : ∃ N, ∀ n ≥ N, u n ≥ c/2 :=
begin
  cases h (c/2) (by linarith) with N hN,
  use N,
  intros n hn,
  specialize hN n hn,
  rw abs_le at hN,
  linarith,
end

-- 3ª demostración
example
  (hc : c > 0)
  (h : limite u c)
  : ∃ N, ∀ n ≥ N, u n ≥ c/2 :=
```

```
have h1 : c/2 > 0,
  by linarith,
have h2 : ∃ N, ∀ n, n ≥ N → |u n - c| ≤ c / 2,
  from h (c/2) h1,
exists.elim h2
  ( assume N,
    assume hN : ∀ n, n ≥ N → |u n - c| ≤ c / 2,
    have h3 : ∀ n ≥ N, u n ≥ c/2,
      { assume n,
        assume hn : n ≥ N,
        have h4 : -(c / 2) ≤ u n - c ∧ u n - c ≤ c / 2,
          from abs_le.mp (hN n hn),
        show u n ≥ c/2,
          from by linarith },
    show ∃ N, ∀ n ≥ N, u n ≥ c/2,
      from exists.intro N h3)
```

5.1.3. Limite de la suma de dos sucesiones convergentes

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

```
variables (a b c : ℝ)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ----------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

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

```
-- Ejercicio 3. Demostrar que si a es el límite de la
-- sucesión u y b el de la v, entonces el límite de
-- (u + v) es (a + b).
-- ----------------------------------------------------

-- 1ª demostración
example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε hε,
  have hε2 : 0 < ε / 2,
    { linarith },
  cases hu (ε / 2) hε2 with Nu hNu,
  cases hv (ε / 2) hε2 with Nv hNv,
  clear hu hv hε2 hε,
  use max Nu Nv,
  intros n hn,
  have hn₁ : n ≥ Nu,
    { exact le_of_max_le_left hn },
  specialize hNu n hn₁,
  have hn₂ : n ≥ Nv,
    { exact le_of_max_le_right hn },
  specialize hNv n hn₂,
  clear hn hn₁ hn₂ Nu Nv,
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|     : rfl
   ... = |(u n - a) + (v n -  b)| : by {congr, ring}
   ... ≤ |u n - a| + |v n -  b|   : by apply abs_add
   ... ≤ ε / 2 + ε / 2            : by linarith
   ... = ε                        : by apply add_halves,
end

-- 2ª demostración
-- ===============

lemma max_ge_iff
  {α : Type*}
  [linear_order α]
  {p q r : α}
  : r ≥ max p q  ↔ r ≥ p ∧ r ≥ q :=
max_le_iff

example
```

```
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε hε,
  cases hu (ε/2) (by linarith) with Nu hNu,
  cases hv (ε/2) (by linarith) with Nv hNv,
  clear hu hv hε,
  use max Nu Nv,
  intros n hn,
  cases max_ge_iff.mp hn with hn₁ hn₂,
  have cota₁ : |u n - a| ≤ ε/2,
    from hNu n hn₁,
  have cota₂ : |v n - b| ≤ ε/2,
    from hNv n hn₂,
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|    : rfl
   ... = |(u n - a) + (v n - b)| : by { congr, ring }
   ... ≤ |u n - a| + |v n - b|    : by apply abs_add
   ... ≤ ε                        : by linarith,
end

-- 3ª demostración
example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε hε,
  cases hu (ε/2) (by linarith) with Nu hNu,
  cases hv (ε/2) (by linarith) with Nv hNv,
  clear hu hv hε,
  use max Nu Nv,
  intros n hn,
  cases max_ge_iff.mp hn with hn₁ hn₂,
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|    : rfl
   ... = |(u n - a) + (v n - b)| : by { congr, ring }
   ... ≤ |u n - a| + |v n - b|    : by apply abs_add
   ... ≤ ε/2 + ε/2                : add_le_add (hNu n hn₁) (hNv n hn₂)
   ... = ε                        : by simp
end

-- 4ª demostración
example
```

```
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε hε,
  cases hu (ε/2) (by linarith) with Nu hNu,
  cases hv (ε/2) (by linarith) with Nv hNv,
  use max Nu Nv,
  intros n hn,
  rw max_ge_iff at hn,
  calc |(u + v) n - (a + b)|
       = |u n + v n - (a + b)|   : rfl
   ... = |(u n - a) + (v n - b)| : by { congr, ring }
   ... ≤ |u n - a| + |v n - b|   : by apply abs_add
   ... ≤ ε                       : by linarith [hNu n (by linarith), hNv n (by linarith)
end
```

5.1.4. Teorema del emparedado

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

```
variable  (a : ℝ)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ----------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que si dos sucesiones tienen
-- el mismo límite, entonces las sucesiones que están
```

```
-- comprendidas entre éstas también tienen el mismo
-- límite.
-- ----------------------------------------------------

-- Nota. En la demostración se usará el siguiente lema:
lemma max_ge_iff
  {p q r : ℕ}
  : r ≥ max p q ↔ r ≥ p ∧ r ≥ q :=
max_le_iff

-- 1ª demostración
-- ===============

example
  (hu : limite u a)
  (hw : limite w a)
  (h : ∀ n, u n ≤ v n)
  (h' : ∀ n, v n ≤ w n) :
  limite v a :=
begin
  intros ε hε,
  cases hu ε hε with N hN, clear hu,
  cases hw ε hε with N' hN', clear hw hε,
  use max N N',
  intros n hn,
  rw max_ge_iff at hn,
  specialize hN n hn.1,
  specialize hN' n hn.2,
  specialize h n,
  specialize h' n,
  clear hn,
  rw abs_le at *,
  split,
  { calc -ε
         ≤ u n - a : hN.1
     ... ≤ v n - a : by linarith, },
  { calc v n - a
         ≤ w n - a : by linarith
     ... ≤ ε       : hN'.2, },
end

-- 2ª demostración
example
  (hu : limite u a)
  (hw : limite w a)
```

```
  (h : ∀ n, u n ≤ v n)
  (h' : ∀ n, v n ≤ w n) :
  limite v a :=
begin
  intros ε hε,
  cases hu ε hε with N hN, clear hu,
  cases hw ε hε with N' hN', clear hw hε,
  use max N N',
  intros n hn,
  rw max_ge_iff at hn,
  specialize hN n (by linarith),
  specialize hN' n (by linarith),
  specialize h n,
  specialize h' n,
  rw abs_le at *,
  split,
  { linarith, },
  { linarith, },
end

-- 3ª demostración
example
  (hu : limite u a)
  (hw : limite w a)
  (h : ∀ n, u n ≤ v n)
  (h' : ∀ n, v n ≤ w n) :
  limite v a :=
begin
  intros ε hε,
  cases hu ε hε with N hN, clear hu,
  cases hw ε hε with N' hN', clear hw hε,
  use max N N',
  intros n hn,
  rw max_ge_iff at hn,
  specialize hN n (by linarith),
  specialize hN' n (by linarith),
  specialize h n,
  specialize h' n,
  rw abs_le at *,
  split ; linarith,
end

-- 4ª demostración
example
  (hu : limite u a)
```

```
  (hw : limite w a)
  (h : ∀ n, u n ≤ v n)
  (h' : ∀ n, v n ≤ w n) :
  limite v a :=
assume ε,
assume hε : ε > 0,
exists.elim (hu ε hε)
  ( assume N,
    assume hN : ∀ (n : ℕ), n ≥ N → |u n - a| ≤ ε,
    exists.elim (hw ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |w n - a| ≤ ε,
        show ∃ N, ∀ n, n ≥ N → |v n - a| ≤ ε, from
          exists.intro (max N N')
            ( assume n,
              assume hn : n ≥ max N N',
              have h1 : n ≥ N ∧ n ≥ N',
                from max_ge_iff.mp hn,
              have h2 : -ε ≤ v n - a,
                { have h2a : |u n - a| ≤ ε,
                    from hN n h1.1,
                  calc -ε
                       ≤ u n - a : and.left (abs_le.mp h2a)
                   ... ≤ v n - a : by linarith [h n], },
              have h3 : v n - a ≤ ε,
                { have h3a : |w n - a| ≤ ε,
                    from hN' n h1.2,
                  calc v n - a
                       ≤ w n - a : by linarith [h' n]
                   ... ≤ ε       : and.right (abs_le.mp h3a), },
              show |v n - a| ≤ ε,
                from abs_le.mpr (and.intro h2 h3))))
```

5.1.5. Si |x| <ε, para todo ε >0, entonces x = 0

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x
```

```
-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que si |x| < ε, para todo
-- ε > 0, entonces x = 0
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { intros ε hε,
    apply le_of_lt,
    exact h ε hε, },
end

-- 2ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { intros ε hε,
    exact le_of_lt (h ε hε), },
end

-- 3ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { exact λ ε hε, le_of_lt (h ε hε), },
end

-- 4ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
```

```
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense
        (abs_nonneg x)
        (λ ε hε, le_of_lt (h ε hε)),
end

-- 5ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense
    (abs_nonneg x)
    (λ ε hε, le_of_lt (h ε hε)))

-- 6ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : ∀ ε, ε > 0 → |x| ≤ ε,
  { assume ε,
    assume hε : ε > 0,
    have h2a : |x| < ε,
      from h ε hε,
    show |x| ≤ ε,
      from le_of_lt h2a },
have h3 : |x| = 0,
  from eq_of_le_of_forall_le_of_dense h1 h2,
show x = 0,
  from abs_eq_zero.mp h3

-- 7ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : ∀ ε, ε > 0 → |x| ≤ ε,
  { assume ε,
    assume hε : ε > 0,
    have h2a : |x| < ε,
```

```
      from h ε hε,
    show |x| ≤ ε,
      from le_of_lt h2a },
have h3 : |x| = 0,
  from eq_of_le_of_forall_le_of_dense h1 h2,
abs_eq_zero.mp h3

-- 8ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : ∀ ε, ε > 0 → |x| ≤ ε,
  { assume ε,
    assume hε : ε > 0,
    have h2a : |x| < ε,
      from h ε hε,
    show |x| ≤ ε,
      from le_of_lt h2a },
abs_eq_zero.mp (eq_of_le_of_forall_le_of_dense h1 h2)

-- 9ª demostración
example
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : ∀ ε, ε > 0 → |x| ≤ ε,
  { assume ε,
    assume hε : ε > 0,
    show |x| ≤ ε,
      from le_of_lt (h ε hε) },
abs_eq_zero.mp (eq_of_le_of_forall_le_of_dense h1 h2)

-- 10ª demostración
lemma cero_de_abs_mn_todos
  (h : ∀ ε > 0, |x| < ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense
    (abs_nonneg x)
    (λ ε hε, le_of_lt (h ε hε)))
```

5.1.6. Si |x| ≤ ε, para todo ε >0, entonces x = 0

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ----------------------------------------------------

notation `|`x`|` := abs x

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que si |x| ≤ ε, para todo
-- ε > 0, entonces x = 0
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { intros ε hε,
    exact h ε hε, },
end

-- 2ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { intros ε hε,
    exact h ε hε, },
end

-- 3ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
begin
```

```
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense,
  { exact abs_nonneg x, },
  { exact λ ε hε, h ε hε, },
end

-- 4ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
begin
  rw ← abs_eq_zero,
  apply eq_of_le_of_forall_le_of_dense
        (abs_nonneg x)
        h,
end

-- 5ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense (abs_nonneg x) h)

-- 6ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : |x| = 0,
  from eq_of_le_of_forall_le_of_dense h1 h,
show x = 0,
  from abs_eq_zero.mp h2

-- 7ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
have h2 : |x| = 0,
  from eq_of_le_of_forall_le_of_dense h1 h,
abs_eq_zero.mp h2
```

```
-- 8ª demostración
example
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
have h1 : 0 ≤ |x|,
  from abs_nonneg x,
abs_eq_zero.mp (eq_of_le_of_forall_le_of_dense h1 h)

-- 9ª demostración
lemma cero_de_abs_mn_todos
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense (abs_nonneg x) h)
```

5.1.7. Si |x - y| ≤ ε, para todo ε >0, entonces x = y

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

```
-- ---------------------------------------------------
-- Ejercicio 1. Definir la notación |x| para el valor
-- absoluto de x.
-- ---------------------------------------------------

notation `|`x`|` := abs x

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar que si |x - y| ≤ ε, para todo
-- ε > 0, entonces x = y.
-- ---------------------------------------------------

-- Se usará el lema demostrado anteriormente
lemma cero_de_abs_mne_todos
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense (abs_nonneg x) h)

-- 1ª demostración
example
  (h : ∀ ε > 0, |x - y| ≤ ε)
  : x = y :=
begin
```

```
  rw ← sub_eq_zero,
  exact cero_de_abs_mne_todos (x - y) h,
end

-- 2ª demostración
lemma ig_de_abs_sub_mne_todos
  (h : ∀ ε > 0, |x - y| ≤ ε)
  : x = y :=
sub_eq_zero.mp (cero_de_abs_mne_todos (x - y) h)
```

5.1.8. Unicidad del límite de las sucesiones

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

```
variables (a b x y : ℝ)

-- ----------------------------------------------------
-- Nota. Se usarán las siguientes notaciones,
-- definiciones y lemas estudiados anteriormente:
-- + |x| = abs x
-- + limite u c :
--      ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε
-- + cero_de_abs_mn_todos:
--      (∀ ε > 0, |x| ≤ ε) → x = 0
-- + ig_de_abs_sub_mne_todos:
--      (∀ ε > 0, |x - y| ≤ ε) → x = y
-- ----------------------------------------------------

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

lemma cero_de_abs_mn_todos
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense (abs_nonneg x) h)

lemma ig_de_abs_sub_mne_todos
  (h : ∀ ε > 0, |x - y| ≤ ε)
  : x = y :=
```

```
sub_eq_zero.mp (cero_de_abs_mn_todos (x - y) h)

-- ---------------------------------------------------
-- Ejercicio. Demostrar que cada sucesión tiene como
-- máximo un límite.
-- ---------------------------------------------------

-- 1ª demostración
example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  apply ig_de_abs_sub_mne_todos,
  intros ε hε,
  cases ha (ε/2) (half_pos hε) with Na hNa,
  cases hb (ε/2) (half_pos hε) with Nb hNb,
  let N := max Na Nb,
  clear ha hb,
  specialize hNa N (le_max_left  _ _),
  specialize hNb N (le_max_right  _ _),
  calc |a - b|
       = |(a - u N) + (u N - b)| : by ring
   ... ≤ |a - u N| + |u N - b|   : by apply abs_add
   ... = |u N - a| + |u N - b|   : by rw abs_sub
   ... ≤ ε/2 + ε/2               : add_le_add hNa hNb
   ... = ε                       : add_halves ε
end

-- 2ª demostración
lemma unicidad_limite
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  apply ig_de_abs_sub_mne_todos,
  intros ε hε,
  cases ha (ε/2) (by linarith) with Na hNa,
  cases hb (ε/2) (by linarith) with Nb hNb,
  let N := max Na Nb,
  specialize hNa N (by finish),
  specialize hNb N (by finish),
  calc |a - b|
       = |(a - u N) + (u N - b)| : by ring
   ... ≤ |a - u N| + |u N - b|   : by apply abs_add
```

```
   ... = |u N - a| + |u N - b|    : by rw abs_sub
   ... ≤ ε                        : by linarith [hNa, hNb]
end

-- 3ª demostración
example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contradiction H,
  change a ≠ b at H,
  have h1 : |a - b| > 0,
    exact abs_pos.mpr (sub_ne_zero_of_ne H),
  cases ha (|a - b|/4) (by linarith) with N hN,
  cases hb (|a - b|/4) (by linarith) with N' hN',
  let N₀ := max N N',
  specialize hN N₀ (le_max_left _ _),
  specialize hN' N₀ (le_max_right _ _),
  have h2 : |a - b| < |a - b |,
    calc  |a - b| = |(a - u N₀) + (u N₀ - b)| : by ring
    ... ≤ |a - u N₀| + |u N₀ - b|             : by apply abs_add
    ... = |u N₀ - a| + |u N₀ - b|             : by rw abs_sub
    ... < |a - b|                             : by linarith,
  linarith,
end
```

5.1.9. Los supremos de las sucesiones no decrecientes son sus límites

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

```
variable (M : ℝ)

-- ----------------------------------------------------
-- Nota. Se usarán las siguientes notaciones y
-- definiciones estudiadas anteriormente:
-- + |x| = abs x
-- + limite u c :
--       ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε
-- ----------------------------------------------------
```

```
notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    no_decreciente : (ℕ → ℝ) → Prop
-- tal que (no_decreciente) expresa que la sucesión u
-- es no decreciente.
-- ----------------------------------------------------

def no_decreciente : (ℕ → ℝ) → Prop
| u := ∀ n m, n ≤ m → u n ≤ u m

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    es_sup_suc : ℝ → (ℕ → ℝ) → Prop
-- tal que (es_sup_suc M u) expresa que M es el supremo
-- de la sucesión u.
-- ----------------------------------------------------

def es_sup_suc : ℝ → (ℕ → ℝ) → Prop
| M u := (∀ n, u n ≤ M) ∧ ∀ ε > 0, ∃ n₀, u n₀ ≥ M - ε

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que si M es un supremo de la
-- sucesión no decreciente u, entonces el límite de u
-- es M.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : es_sup_suc M u)
  (h' : no_decreciente u)
  : limite u M :=
begin
  -- unfold limite,
  intros ε hε,
  -- unfold es_sup_suc at h,
  cases h with hM₁ hM₂,
  cases hM₂ ε hε with n₀ hn₀,
  use n₀,
  intros n hn,
  rw abs_le,
```

```
  split,
  { -- unfold no_decreciente at h',
    specialize h' n₀ n hn,
    calc -ε
         = (M - ε) - M : by ring
     ... ≤ u n₀ - M    : sub_le_sub_right hn₀ M
     ... ≤ u n - M     : sub_le_sub_right h' M },
  { calc u n - M
         ≤ M - M       : sub_le_sub_right (hM₁ n) M
     ... = 0           : sub_self M
     ... ≤ ε           : le_of_lt hε, },
end

-- 2ª demostración
example
  (h : es_sup_suc M u)
  (h' : no_decreciente u)
  : limite u M :=
begin
  intros ε hε,
  cases h with hM₁ hM₂,
  cases hM₂ ε hε with n₀ hn₀,
  use n₀,
  intros n hn,
  rw abs_le,
  split,
  { linarith [h' n₀ n hn] },
  { linarith [hM₁ n] },
end

-- 3ª demostración
example
  (h : es_sup_suc M u)
  (h' : no_decreciente u)
  : limite u M :=
begin
  intros ε hε,
  cases h with hM₁ hM₂,
  cases hM₂ ε hε with n₀ hn₀,
  use n₀,
  intros n hn,
  rw abs_le,
  split ; linarith [h' n₀ n hn, hM₁ n],
end
```

```
-- 4ª demostración
example
  (h : es_sup_suc M u)
  (h' : no_decreciente u)
  : limite u M :=
assume ε,
assume hε : ε > 0,
have hM₁ : ∀ (n : ℕ), u n ≤ M,
  from h.left,
have hM₂ : ∀ (ε : ℝ), ε > 0 → (∃ (n₀ : ℕ), u n₀ ≥ M - ε),
  from h.right,
exists.elim (hM₂ ε hε)
  ( assume n₀,
    assume hn₀ : u n₀ ≥ M - ε,
    have h1 : ∀ n, n ≥ n₀ → |u n - M| ≤ ε,
      { assume n,
        assume hn : n ≥ n₀,
        have h2 : -ε ≤ u n - M,
          { have h'a : u n₀ ≤ u n,
              from h' n₀ n hn,
            calc -ε
                 = (M - ε) - M : by ring
             ... ≤ u n₀ - M    : sub_le_sub_right hn₀ M
             ... ≤ u n - M     : sub_le_sub_right h'a M },
        have h3 : u n - M ≤ ε,
          { calc u n - M
                 ≤ M - M       : sub_le_sub_right (hM₁ n) M
             ... = 0           : sub_self M
             ... ≤ ε           : le_of_lt hε },
        show |u n - M| ≤ ε,
          from abs_le.mpr (and.intro h2 h3) },
    show ∃ N, ∀ n, n ≥ N → |u n - M| ≤ ε,
      from exists.intro n₀ h1)
```

5.2. Subsucesiones

5.2.1. La función identidad es menor o igual que la función de extracción

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

```
variable {φ : ℕ → ℕ}

set_option pp.structure_projections false

-- ----------------------------------------------------
-- Ejercicio 1. Para extraer una sucesión se aplica una
-- función de extracción que conserva el orden; por
-- ejemplo, la subsucesión
--    uₒ, u₂, u₄, u₆, ...
-- se ha obtenido con la función de extracción φ tal
-- que φ(n) = 2*n.
--
-- Definir la función
--    extraccion : (ℕ → ℕ) → Prop
-- tal que (extraccion φ) expresa que φ es una función
-- de extracción
-- ----------------------------------------------------

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que si φ es una función de
-- extracción, entonces
--    ∀ n, n ≤ φ n
-- ----------------------------------------------------

-- 1ª demostración
example :
  extraccion φ → ∀ n, n ≤ φ n :=
begin
  intros h n,
  induction n with m HI,
  { exact nat.zero_le (φ 0), },
  { apply nat.succ_le_of_lt,
    have h1 : m < succ m,
      from lt_add_one m,
    calc m ≤ φ m        : HI
       ... < φ (succ m) : h m (succ m) h1, },
end

-- 2ª demostración
example :
  extraccion φ → ∀ n, n ≤ φ n :=
begin
```

```
  intros h n,
  induction n with m HI,
  { exact nat.zero_le _ },
  { apply nat.succ_le_of_lt,
    calc m ≤ φ m        : HI
       ... < φ (succ m) : by linarith [h m (m+1) (by linarith)] },
end

-- 3ª demostración
example :
  extraccion φ → ∀ n, n ≤ φ n :=
begin
  intros h n,
  induction n with m HI,
  { linarith },
  { apply nat.succ_le_of_lt,
    linarith [h m (m+1) (by linarith)] },
end

-- 4ª demostración
example :
  extraccion φ → ∀ n, n ≤ φ n :=
begin
  intros h n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

-- 5ª demostración
example :
  extraccion φ → ∀ n, n ≤ φ n :=
assume h : extraccion φ,
assume n,
nat.rec_on n
  ( show 0 ≤ φ 0,
      from nat.zero_le (φ 0) )
  ( assume m,
    assume HI : m ≤ φ m,
    have h1 : m < succ m,
      from lt_add_one m,
    have h2 : m < φ (succ m), from
      calc m ≤ φ m        : HI
         ... < φ (succ m) : h m (succ m) h1,
    show succ m ≤ φ (succ m),
```

```
      from nat.succ_le_of_lt h2)
```

5.2.2. Las funciones de extracción no están acotadas

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

```
variable {φ : ℕ → ℕ}

-- ----------------------------------------------------
-- Nota. Se usará la siguiente definición y lema
-- estudiado anteriormente.
-- ----------------------------------------------------

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

lemma id_mne_extraccion
  (h : extraccion φ)
  : ∀ n, n ≤ φ n :=
begin
  intros n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

-- ----------------------------------------------------
-- Ejercicio. Demostrar que las funciones de extracción
-- no está acotadas; es decir, que si φ es una función
-- de extracción, entonces
--     ∀ N N', ∃ n ≥ N', φ n ≥ N
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
begin
  intros N N',
  let n := max N N',
  use n,
  split,
  { exact le_max_right N N', },
```

```
  { calc N ≤ n   : le_max_left N N'
       ... ≤ φ n : id_mne_extraccion h n, },
end

-- 2ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
begin
  intros N N',
  let n := max N N',
  use n,
  split,
  { exact le_max_right N N', },
  { exact le_trans (le_max_left N N')
                   (id_mne_extraccion h n), },
end

-- 3ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
begin
  intros N N',
  use max N N',
  split,
  { exact le_max_right N N', },
  { exact le_trans (le_max_left N N')
                   (id_mne_extraccion h (max N N')), },
end

-- 4ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
begin
  intros N N',
  use max N N',
  exact ⟨le_max_right N N',
         le_trans (le_max_left N N')
                  (id_mne_extraccion h (max N N'))⟩,
end

-- 5ª demostración
example
```

```
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
λ N N',
  ⟨max N N', ⟨le_max_right N N',
              le_trans (le_max_left N N')
                       (id_mne_extraccion h (max N N'))⟩⟩

-- 6ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
assume N N',
let n := max N N' in
have h1 : n ≥ N',
  from le_max_right N N',
show ∃ n ≥ N', φ n ≥ N, from
exists.intro n
  (exists.intro h1
    (show φ n ≥ N, from
       calc N ≤ n   : le_max_left N N'
          ... ≤ φ n : id_mne_extraccion h n))

-- 7ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
assume N N',
let n := max N N' in
have h1 : n ≥ N',
  from le_max_right N N',
show ∃ n ≥ N', φ n ≥ N, from
⟨n, h1, calc N ≤ n   : le_max_left N N'
          ...  ≤ φ n : id_mne_extraccion h n⟩

-- 8ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
assume N N',
let n := max N N' in
have h1 : n ≥ N',
  from le_max_right N N',
show ∃ n ≥ N', φ n ≥ N, from
⟨n, h1, le_trans (le_max_left N N')
                 (id_mne_extraccion h (max N N'))⟩
```

```
-- 9ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
assume N N',
let n := max N N' in
have h1 : n ≥ N',
  from le_max_right N N',
⟨n, h1, le_trans (le_max_left N N')
                 (id_mne_extraccion h n)⟩

-- 10ª demostración
example
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
assume N N',
⟨max N N', le_max_right N N',
           le_trans (le_max_left N N')
                    (id_mne_extraccion h (max N N'))⟩

-- 11ª demostración
lemma extraccion_mye
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
λ N N',
  ⟨max N N', le_max_right N N',
             le_trans (le_max_left N N')
             (id_mne_extraccion h (max N N'))⟩
```

5.2.3. Si a es un punto de acumulación de u, entonces ∀ ε >0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε

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

```
variables {a : ℝ}
variable  {φ : ℕ → ℕ}

-- ----------------------------------------------------
-- Nota. Usaremos los siguientes conceptos estudiados
-- anteriormente.
-- ----------------------------------------------------
```

```
notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

lemma id_mne_extraccion
  (h : extraccion φ)
  : ∀ n, n ≤ φ n :=
begin
  intros n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

lemma extraccion_mye
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
λ N N',
  ⟨max N N', le_max_right N N',
             le_trans (le_max_left N N')
             (id_mne_extraccion h (max N N'))⟩

-- ---------------------------------------------------
-- Ejercicio 1. Definir la función
--    punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
-- tal que (punto_acumulacion u a) expresa que a es un
-- punto de acumulación de u; es decir, que es el
-- límite de alguna subsucesión de u.
-- ---------------------------------------------------

def punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
| u a := ∃ φ, extraccion φ ∧ limite (u ∘ φ) a

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar que si a es un punto de
-- acumulación de u, entonces
--    ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε
-- ---------------------------------------------------

-- 1ª demostración
example
```

```
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
begin
  intros ε hε N,
  -- unfold punto_acumulacion at h,
  rcases h with ⟨φ, hφ1, hφ2⟩,
  -- unfold limite at hφ2,
  cases hφ2 ε hε with N' hN',
  rcases extraccion_mye hφ1 N N' with ⟨m, hm, hm'⟩,
  -- clear hφ1 hφ2,
  use φ m,
  split,
  { exact hm', },
  { exact hN' m hm, },
end

-- 2ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
begin
  intros ε hε N,
  rcases h with ⟨φ, hφ1, hφ2⟩,
  cases hφ2 ε hε with N' hN',
  rcases extraccion_mye hφ1 N N' with ⟨m, hm, hm'⟩,
  use φ m,
  exact ⟨hm', hN' m hm⟩,
end

-- 3ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
begin
  intros ε hε N,
  rcases h with ⟨φ, hφ1, hφ2⟩,
  cases hφ2 ε hε with N' hN',
  rcases extraccion_mye hφ1 N N' with ⟨m, hm, hm'⟩,
  exact ⟨φ m, hm', hN' _ hm⟩,
end

-- 4ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
```

```
begin
  intros ε hε N,
  rcases h with ⟨φ, hφ1, hφ2⟩,
  cases hφ2 ε hε with N' hN',
  rcases extraccion_mye hφ1 N N' with ⟨m, hm, hm'⟩,
  use φ m ; finish
end

-- 5ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          ( assume m,
            assume hm : ∃ (H : m ≥ N'), φ m ≥ N,
            exists.elim hm
              ( assume hm1 : m ≥ N',
                assume hm2 : φ m ≥ N,
                have h2 : |u (φ m) - a| ≤ ε,
                  from hN' m hm1,
                show ∃ n ≥ N, |u n - a| ≤ ε,
                  from exists.intro (φ m) (exists.intro hm2 h2)))))

-- 6ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
```

```
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          ( assume m,
            assume hm : ∃ (H : m ≥ N'), φ m ≥ N,
            exists.elim hm
              ( assume hm1 : m ≥ N',
                assume hm2 : φ m ≥ N,
                have h2 : |u (φ m) - a| ≤ ε,
                  from hN' m hm1,
                show ∃ n ≥ N, |u n - a| ≤ ε,
                  from ⟨φ m, hm2, h2⟩))))

-- 7ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          ( assume m,
            assume hm : ∃ (H : m ≥ N'), φ m ≥ N,
            exists.elim hm
              ( assume hm1 : m ≥ N',
                assume hm2 : φ m ≥ N,
                have h2 : |u (φ m) - a| ≤ ε,
                  from hN' m hm1,
                ⟨φ m, hm2, h2⟩))))

-- 8ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
```

```
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          ( assume m,
            assume hm : ∃ (H : m ≥ N'), φ m ≥ N,
            exists.elim hm
              ( assume hm1 : m ≥ N',
                assume hm2 : φ m ≥ N,
                ⟨φ m, hm2, hN' m hm1⟩))))

-- 9ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          ( assume m,
            assume hm : ∃ (H : m ≥ N'), φ m ≥ N,
            exists.elim hm
              (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))

-- 10ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
```

```
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        have h1 : ∃ n ≥ N', φ n ≥ N,
          from extraccion_mye hφ.1 N N',
        exists.elim h1
          (λ m hm, exists.elim hm (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))

-- 11ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      ( assume N',
        assume hN' : ∀ (n : ℕ), n ≥ N' → |(u ∘ φ) n - a| ≤ ε,
        exists.elim (extraccion_mye hφ.1 N N')
          (λ m hm, exists.elim hm (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))

-- 12ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    exists.elim (hφ.2 ε hε)
      (λ N' hN', exists.elim (extraccion_mye hφ.1 N N')
        (λ m hm, exists.elim hm
          (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))

-- 13ª demostración
example
```

```
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
assume ε,
assume hε : ε > 0,
assume N,
exists.elim h
  (λ φ hφ, exists.elim (hφ.2 ε hε)
    (λ N' hN', exists.elim (extraccion_mye hφ.1 N N')
      (λ m hm, exists.elim hm
        (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))

-- 14ª demostración
example
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
λ ε hε N, exists.elim h
  (λ φ hφ, exists.elim (hφ.2 ε hε)
    (λ N' hN', exists.elim (extraccion_mye hφ.1 N N')
      (λ m hm, exists.elim hm
        (λ hm1 hm2, ⟨φ m, hm2, hN' m hm1⟩))))
```

5.2.4. Las subsucesiones tienen el mismo límite que la sucesión

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

```
variables {a : ℝ}
variable  {φ : ℕ → ℕ}

-- ----------------------------------------------------
-- Nota. Usaremos los siguientes conceptos estudiados
-- anteriormente.
-- ----------------------------------------------------

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m
```

```
lemma id_mne_extraccion
  (h : extraccion φ)
  : ∀ n, n ≤ φ n :=
begin
  intros n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

lemma extraccion_mye
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
λ N N',
  ⟨max N N', le_max_right N N',
             le_trans (le_max_left N N')
             (id_mne_extraccion h (max N N'))⟩

-- ---------------------------------------------------
-- Ejercicio. Demostrar que las subsucesiones de las
-- sucesiones convergentes tienen el mismo límite que
-- la sucesión.
-- ---------------------------------------------------

-- 1ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  -- unfold limite,
  intros ε hε,
  -- unfold limite at h,
  cases h ε hε with N hN,
  use N,
  intros n hn,
  apply hN,
  apply le_trans hn,
  exact id_mne_extraccion hφ n,
end

-- 2ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
```

```
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  cases h ε hε with N hN,
  use N,
  intros n hn,
  apply hN,
  exact le_trans hn (id_mne_extraccion hφ n),
end

-- 3ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  cases h ε hε with N hN,
  use N,
  intros n hn,
  exact hN (φ n) (le_trans hn (id_mne_extraccion hφ n)),
end

-- 4ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  cases h ε hε with N hN,
  use N,
  exact λ n hn, hN (φ n) (le_trans hn (id_mne_extraccion hφ n)),
end

-- 5ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  -- unfold limite at h,
  cases h ε hε with N hN,
  exact ⟨N, λ n hn, hN (φ n) (le_trans hn (id_mne_extraccion hφ n))⟩,
```

```
end

-- 6ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  exact (exists.elim (h ε hε)
          (λ N hN,
            ⟨N, λ n hn,
                  hN (φ n) (le_trans hn (id_mne_extraccion hφ n))⟩)),
end

-- 7ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
λ ε hε,
  (exists.elim (h ε hε)
    (λ N hN,
      ⟨N, λ n hn,
            hN (φ n) (le_trans hn (id_mne_extraccion hφ n))⟩))

-- 8ª demostración
example
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
begin
  intros ε hε,
  cases h ε hε with N hN,
  use N,
  intros n hn,
  apply hN,
  calc N ≤ n   : hn
     ... ≤ φ n : id_mne_extraccion hφ n,
end

-- 9ª demostración
lemma limite_subsucesion
  (h : limite u a)
  (hφ : extraccion φ)
```

```
  : limite (u ∘ φ) a :=
assume ε,
assume hε : ε > 0,
exists.elim (h ε hε)
 ( assume N,
   assume hN : ∀ n, n ≥ N → |u n - a| ≤ ε,
   have h1 : ∀n, n ≥ N → |(u ∘ φ) n - a| ≤ ε,
     { assume n,
       assume hn : n ≥ N,
       have h2 : N ≤ φ n, from
         calc N ≤ n   : hn
           ... ≤ φ n : id_mne_extraccion hφ n,
       show |(u ∘ φ) n - a| ≤ ε,
         from hN (φ n) h2,
     },
   show ∃ N, ∀n, n ≥ N → |(u ∘ φ) n - a| ≤ ε,
     from exists.intro N h1)
```

5.2.5. El punto de acumulación de las convergentes es su límite

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

```
variables {a b: ℝ}
variables (x y : ℝ)
variable  {φ : ℕ → ℕ}

-- ----------------------------------------------------
-- Nota. Usaremos los siguientes conceptos estudiados
-- anteriormente.
-- ----------------------------------------------------

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

lemma cero_de_abs_mn_todos
  (h : ∀ ε > 0, |x| ≤ ε)
  : x = 0 :=
abs_eq_zero.mp
  (eq_of_le_of_forall_le_of_dense (abs_nonneg x) h)
```

```
lemma ig_de_abs_sub_mne_todos
  (h : ∀ ε > 0, |x - y| ≤ ε)
  : x = y :=
sub_eq_zero.mp (cero_de_abs_mn_todos (x - y) h)

lemma unicidad_limite
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  apply ig_de_abs_sub_mne_todos,
  intros ε hε,
  cases ha (ε/2) (by linarith) with Na hNa,
  cases hb (ε/2) (by linarith) with Nb hNb,
  let N := max Na Nb,
  specialize hNa N (by finish),
  specialize hNb N (by finish),
  calc |a - b|
       = |(a - u N) + (u N - b)| : by ring
   ... ≤ |a - u N| + |u N - b|   : by apply abs_add
   ... = |u N - a| + |u N - b|   : by rw abs_sub
   ... ≤ ε                       : by linarith [hNa, hNb]
end

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

lemma id_mne_extraccion
  (h : extraccion φ)
  : ∀ n, n ≤ φ n :=
begin
  intros n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

def punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
| u a := ∃ φ, extraccion φ ∧ limite (u ∘ φ) a

lemma limite_subsucesion
  (h : limite u a)
  (hφ : extraccion φ)
  : limite (u ∘ φ) a :=
```

```
assume ε,
assume hε : ε > 0,
exists.elim (h ε hε)
 ( assume N,
   assume hN : ∀ n, n ≥ N → |u n - a| ≤ ε,
   have h1 : ∀n, n ≥ N → |(u ∘ φ) n - a| ≤ ε,
     { assume n,
       assume hn : n ≥ N,
       have h2 : N ≤ φ n, from
         calc N ≤ n   : hn
           ... ≤ φ n : id_mne_extraccion hφ n,
       show |(u ∘ φ) n - a| ≤ ε,
         from hN (φ n) h2,
     },
   show ∃ N, ∀n, n ≥ N → |(u ∘ φ) n - a| ≤ ε,
     from exists.intro N h1)

-- ---------------------------------------------------
-- Ejercicio. Demostrar que si a es un punto de
-- acumulación de una sucesión de límite b, entonces a
-- y b son iguales.
-- ---------------------------------------------------

-- 1ª demostración
example
  (ha : punto_acumulacion u a)
  (hb : limite u b)
  : a = b :=
begin
  -- unfold punto_acumulacion at ha,
  rcases ha with ⟨φ, hφ₁, hφ₂⟩,
  have hφ₃ : limite (u ∘ φ) b,
    from limite_subsucesion hb hφ₁,
  exact unicidad_limite hφ₂ hφ₃,
end

-- 2ª demostración
example
  (ha : punto_acumulacion u a)
  (hb : limite u b)
  : a = b :=
begin
  rcases ha with ⟨φ, hφ₁, hφ₂⟩,
  exact unicidad_limite hφ₂ (limite_subsucesion hb hφ₁),
end
```

```
-- 3ª demostración
example
  (ha : punto_acumulacion u a)
  (hb : limite u b)
  : a = b :=
exists.elim ha
  (λ φ hφ, unicidad_limite hφ.2 (limite_subsucesion hb hφ.1))

-- 4ª demostración
example
  (ha : punto_acumulacion u a)
  (hb : limite u b)
  : a = b :=
exists.elim ha
  ( assume φ,
    assume hφ : extraccion φ ∧ limite (u ∘ φ) a,
    have hφ' : limite (u ∘ φ) b,
      from limite_subsucesion hb hφ.1,
    show a = b,
      from unicidad_limite hφ.2 hφ')
```

5.2.6. Toda sucesión convergente es una sucesión de Cauchy

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

```
-- ---------------------------------------------------
-- Nota. Usaremos los siguientes conceptos estudiados
-- anteriormente.
-- ---------------------------------------------------

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ---------------------------------------------------
-- Ejercicio 1. Definir la función
--    sucesion_convergente : (ℕ → ℝ) → Prop
-- tal que (sucesion_convergente u) expresa que la
-- sucesión u es convergente.
-- ---------------------------------------------------
```

```
def sucesion_convergente : (ℕ → ℝ) → Prop
| u := ∃ a, limite u a

-- ----------------------------------------------------
-- Ejercicio 2. Definir la función
--    sucesion_de_Cauchy : (ℕ → ℝ) → Prop
-- tal que (sucesion_de_Cauchy u) expresa que la
-- sucesión u es una sucesión de Cauchy.
-- ----------------------------------------------------

def sucesion_de_Cauchy : (ℕ → ℝ) → Prop
| u := ∀ ε > 0, ∃ N, ∀ p q, p ≥ N → q ≥ N → |u p - u q| ≤ ε

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar que toda sucesión convergente
-- es una sucesión de Cauchy.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : sucesion_convergente u)
  : sucesion_de_Cauchy u :=
begin
  -- unfold sucesion_convergente at h,
  cases h with a ha,
  -- unfold sucesion_de_Cauchy,
  intros ε hε,
  -- unfold limite at ha,
  cases ha (ε/2) (half_pos hε) with N hN,
  use N,
  intros p q hp hq,
  calc  |u p - u q|
      = |(u p - a) + (a - u q)| : by ring
  ... ≤ |u p - a| + |a - u q|   : by apply abs_add
  ... = |u p - a| + |u q - a|   : by rw abs_sub (u q) a
  ... ≤ ε/2 + |u q - a|         : add_le_add_right (hN p hp) _
  ... ≤ ε/2 + ε/2               : add_le_add_left (hN q hq) (ε/2)
  ... = ε                       : add_halves ε
end

-- 2ª demostración
example
  (h : sucesion_convergente u)
  : sucesion_de_Cauchy u :=
begin
```

```
  cases h with a ha,
  intros ε hε,
  cases ha (ε/2) (by linarith) with N hN,
  use N,
  intros p q hp hq,
  calc  |u p - u q|
      = |(u p - a) + (a - u q)| : by ring
  ... ≤ |u p - a| + |a - u q|   : by simp only [abs_add]
  ... = |u p - a| + |u q - a|   : by simp only [abs_add, abs_sub]
  ... ≤ ε                       : by linarith [hN p hp, hN q hq],
end

-- 3ª demostración
example
  (h : sucesion_convergente u)
  : sucesion_de_Cauchy u :=
exists.elim h
  (assume a,
   assume ha : limite u a,
   show sucesion_de_Cauchy u, from
     (assume ε,
      assume hε : ε > 0,
      exists.elim (ha (ε/2) (by linarith))
        (assume N,
         assume hN : ∀ n, n ≥ N → |u n - a| ≤ ε/2,
         show ∃ N, ∀ p q, p ≥ N → q ≥ N → |u p - u q| ≤ ε,
           from exists.intro N
             (assume p q,
              assume hp: p ≥ N,
              assume hq: q ≥ N,
              show |u p - u q| ≤ ε, from
                calc  |u p - u q|
                    = |(u p - a) + (a - u q)| : by ring
                ... ≤ |u p - a| + |a - u q|   : by simp only [abs_add]
                ... = |u p - a| + |u q - a|   : by simp only [abs_add, abs_sub]
                ... ≤ ε                       : by linarith [hN p hp, hN q hq]))))
```

5.2.7. Si a es un punto de acumulación de la sucesión de Cauchy u, entonces a es el límite de u

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

```
variables {a : ℝ}
variable  {φ : ℕ → ℕ}

-- ----------------------------------------------------
-- Nota. Usaremos los siguientes conceptos estudiados
-- anteriormente.
-- ----------------------------------------------------

notation `|`x`|` := abs x

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

lemma id_mne_extraccion
  (h : extraccion φ)
  : ∀ n, n ≤ φ n :=
begin
  intros n,
  induction n with m HI,
  { linarith },
  { exact nat.succ_le_of_lt (by linarith [h m (m+1) (by linarith)]) },
end

lemma extraccion_mye
  (h : extraccion φ)
  : ∀ N N', ∃ n ≥ N', φ n ≥ N :=
λ N N',
  ⟨max N N', le_max_right N N',
             le_trans (le_max_left N N')
             (id_mne_extraccion h (max N N'))⟩

def punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
| u a := ∃ φ, extraccion φ ∧ limite (u ∘ φ) a

lemma cerca_acumulacion
  (h : punto_acumulacion u a)
  : ∀ ε > 0, ∀ N, ∃ n ≥ N, |u n - a| ≤ ε :=
begin
  intros ε hε N,
  rcases h with ⟨φ, hφ1, hφ2⟩,
  cases hφ2 ε hε with N' hN',
  rcases extraccion_mye hφ1 N N' with ⟨m, hm, hm'⟩,
```

```
  exact ⟨φ m, hm', hN' _ hm⟩,
end

def sucesion_de_Cauchy : (ℕ → ℝ) → Prop
| u := ∀ ε > 0, ∃ N, ∀ p q, p ≥ N → q ≥ N → |u p - u q| ≤ ε

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si u es una sucesión de
-- Cauchy y a es un punto de acumulación de u, entonces
-- a es el límite de u.
-- ----------------------------------------------------

-- 1ª demostración
example
  (hu : sucesion_de_Cauchy u)
  (ha : punto_acumulacion u a)
  : limite u a :=
begin
  -- unfold limite,
  intros ε hε,
  -- unfold sucesion_de_Cauchy at hu,
  cases hu (ε/2) (half_pos hε) with N hN,
  use N,
  have ha' : ∃ N' ≥ N, |u N' - a| ≤ ε/2,
    apply cerca_acumulacion ha (ε/2) (half_pos hε),
  cases ha' with N' h,
  cases h with hNN' hN',
  intros n hn,
  calc   |u n - a|
       = |(u n - u N') + (u N' - a)| : by ring
   ... ≤ |u n - u N'| + |u N' - a|   : abs_add (u n - u N') (u N' - a)
   ... ≤ ε/2 + |u N' - a|            : add_le_add_right (hN n N' hn hNN') _
   ... ≤ ε/2 + ε/2                   : add_le_add_left hN' (ε / 2)
   ... = ε                           : add_halves ε
end

-- 2ª demostración
example
  (hu : sucesion_de_Cauchy u)
  (ha : punto_acumulacion u a)
  : limite u a :=
begin
  intros ε hε,
  cases hu (ε/2) (by linarith) with N hN,
  use N,
```

```
  have ha' : ∃ N' ≥ N, |u N' - a| ≤ ε/2,
    apply cerca_acumulacion ha (ε/2) (by linarith),
  rcases ha' with ⟨N', hNN', hN'⟩,
  intros n hn,
  calc  |u n - a|
      = |(u n - u N') + (u N' - a)| : by ring
  ... ≤ |u n - u N'| + |u N' - a|   : by simp [abs_add]
  ... ≤ ε                           : by linarith [hN n N' hn hNN'],
end
```

Capítulo 6

Negación

6.1. Falso y negación

6.1.1. Principio de no contradicción

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Demostrar el principio de no contradicción
--    P ∧ ¬ P → false
-- ----------------------------------------------------

-- 1ª demostración
example : P ∧ ¬ P → false :=
begin
  intro h1,
  cases h1 with hP hnP,
  apply hnP,
  exact hP,
end

-- 2ª demostración
example : P ∧ ¬ P → false :=
begin
  rintro ⟨hP, hnP⟩,
  exact hnP hP,
end

-- 3ª demostración
example : P ∧ ¬ P → false :=
λ ⟨hP, hnP⟩, hnP hP
```

```
-- 4ª demostración
example : P ∧ ¬ P → false :=
-- by library_search
(and_not_self P).mp

-- 5ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
have hP : P,
  from and.elim_left h,
have hnP : ¬P,
  from and.elim_right h,
show false,
  from absurd hP hnP

-- 6ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
have hP : P,
  from and.left h,
have hnP : ¬P,
  from and.right h,
show false,
  from absurd hP hnP

-- 7ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
have hP : P,
  from h.left,
have hnP : ¬P,
  from h.right,
show false,
  from absurd hP hnP

-- 8ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
have hP : P,
  from h.1,
have hnP : ¬P,
  from h.2,
show false,
  from absurd hP hnP
```

```
-- 9ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
show false,
  from absurd h.1 h.2

-- 10ª demostración
example : P ∧ ¬ P → false :=
assume h : P ∧ ¬ P,
absurd h.1 h.2

-- 11ª demostración
example : P ∧ ¬ P → false :=
λ h, absurd h.1 h.2

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar el principio de no contradicción
--    P ∧ ¬ P → 0 = 1
-- ----------------------------------------------------

-- 1ª demostración
example : P ∧ ¬ P → 0 = 1 :=
begin
  rintro ⟨hP, hnP⟩,
  exfalso,
  exact hnP hP,
end

-- 2ª demostración
example : P ∧ ¬ P → 0 = 1 :=
λ h, absurd h.1 h.2
```

6.1.2. Introducción de la doble negación

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

```
open_locale classical

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    P ⊢ ¬¬P
-- ----------------------------------------------------
```

```
-- 1ª demostración
example
  (h1 : P)
  : ¬¬P :=
begin
  intro h2,
  exact h2 h1,
end

-- 2ª demostración
example
  (h1 : P)
  : ¬¬P :=
not.intro
  ( assume h2: ¬P,
    show false,
      from h2 h1)

-- 3ª demostración
example
  (h1 : P)
  : ¬¬P :=
assume h2: ¬P,
show false,
  from h2 h1

-- 4ª demostración
example
  (h1 : P)
  : ¬¬P :=
assume h2: ¬P, h2 h1

-- 5ª demostración
example
  (h1 : P)
  : ¬¬P :=
λ h2, h2 h1

-- 6ª demostración
example
  (h1 : P)
  : ¬¬P :=
not_not.mpr h1
```

```
-- 7ª demostración
example
  (h1 : P)
  : ¬¬P :=
-- by library_search
not_not_intro h1

-- 8ª demostración
example
  (h1 : P)
  : ¬¬P :=
-- by hint
by tauto

-- 9ª demostración
example
  (h1 : P)
  : ¬¬P :=
by finish
```

6.1.3. La relación menor es irreflexiva en los reales

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

```
variable {x : ℝ}

-- ------------------------------------------------------
-- Ejercicio. Demostrar que la relación menor es
-- irreflexiva en los reales.
-- ------------------------------------------------------

-- 1ª demostración
example : ¬ x < x :=
begin
  intro h1,
  rw lt_iff_le_and_ne at h1,
  cases h1 with h2 h3,
  -- clear h2,
  -- change x = x → false at h3,
  apply h3,
  refl,
end
```

```
-- 2ª demostración
example : ¬ x < x :=
begin
  intro h1,
  rw lt_iff_le_and_ne at h1,
  cases h1 with h2 h3,
  apply h3,
  refl,
end

-- 3ª demostración
example : ¬ x < x :=
begin
  intro h1,
  cases (lt_iff_le_and_ne.mp h1) with h2 h3,
  apply h3,
  refl,
end

-- 4ª demostración
example : ¬ x < x :=
begin
  intro h1,
  apply (lt_iff_le_and_ne.mp h1).2,
  refl,
end

-- 5ª demostración
example : ¬ x < x :=
begin
  intro h1,
  exact absurd rfl (lt_iff_le_and_ne.mp h1).2,
end

-- 6ª demostración
example : ¬ x < x :=
λ h, absurd rfl (lt_iff_le_and_ne.mp h).2

-- 7ª demostración
example : ¬ x < x :=
assume h1 : x < x,
have h2 : x ≤ x ∧ x ≠ x,
  from lt_iff_le_and_ne.mp h1,
have h3 : x ≠ x,
  from and.right h2,
```

```
have h4 : x = x,
  from rfl,
show false,
  from absurd h4 h3

-- 8ª demostración
example : ¬ x < x :=
assume h1 : x < x,
have h2 : x ≤ x ∧ x ≠ x,
  from lt_iff_le_and_ne.mp h1,
absurd rfl (and.right h2)

-- 9ª demostración
example : ¬ x < x :=
assume h1 : x < x,
absurd rfl (and.right (lt_iff_le_and_ne.mp h1))

-- 10ª demostración
example : ¬ x < x :=
assume h1 : x < x,
absurd rfl (lt_iff_le_and_ne.mp h1).2

-- 11ª demostración
example : ¬ x < x :=
λ h, absurd rfl (lt_iff_le_and_ne.mp h).2

-- 12ª demostración
example : ¬ x < x :=
-- by library_search
irrefl x

-- 12ª demostración
example : ¬ x < x :=
-- by hint
by simp

-- 13ª demostración
example : ¬ x < x :=
by finish

-- 14ª demostración
example : ¬ x < x :=
by norm_num

-- 15ª demostración
```

```
example : ¬ x < x :=
by linarith

-- 16ª demostración
example : ¬ x < x :=
by nlinarith
```

6.1.4. Demostración con hipótesis inconsistentes

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

```
variable (n : ℤ)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si un número es par y no es
-- par, entonces 0 = 1.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
begin
  exfalso,
  exact h2 h1,
end

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
begin
  exact absurd h1 h2,
end

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
absurd h1 h2
```

```
-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
-- by library_search
by exact false.rec (0 = 1) (h2 h1)

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
-- by hint
by tauto

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
by finish

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
by finish

-- ?ª demostración
example
  (h1 : even n)
  (h2 : ¬ even n)
  : 0 = 1 :=
by solve_by_elim
```

6.2. Principio del tercio excluso y reducción al absurdo

6.2.1. Eliminación de la doble negación

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

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

import tactic

variable (P : Prop)

-- 1ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
begin
  intro hnnP,
  cases h with hP hnP,
  { exact hP, },
  { exfalso,
    apply hnnP,
    exact hnP, },
end

-- 2ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
begin
  intro hnnP,
  cases h with hP hnP,
  { exact hP, },
  { exfalso,
    exact hnnP hnP, },
end

-- 3ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
assume hnnP : ¬¬P ,
or.elim h
```

```
  ( assume hP : P,
    show P,
      from hP)
  ( assume hnP : ¬P,
    show P,
      from absurd hnP hnnP)

-- 4ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
assume hnnP : ¬¬P ,
or.elim h
  (λ hP, hP)
  (λ hnP, absurd hnP hnnP)

-- 5ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
λ hnnP, or.elim h id (λ hnP, absurd hnP hnnP)

-- 6ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
-- by library_search
or.resolve_right h

-- 7ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
-- by hint
by tauto

-- 8ª demostración
example
  (h : P ∨ ¬ P)
  : ¬¬P → P :=
by finish

-- 9ª demostración
example
  (h : P ∨ ¬ P)
```

```
  : ¬¬P → P :=
by simp
```

6.2.2. Demostración por casos: P → R, ¬P → Q, Q → R ⊢ R

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

```
open_locale classical

-- ----------------------------------------------------
-- Ejercicio. Demostrar que
--    P → R, ¬P → Q, Q → R ⊢ R
-- ----------------------------------------------------

-- 1ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
begin
  by_cases hP : P,
  { apply hPR,
    exact hP, },
  { apply hQR,
    apply hPQ,
    exact hP, },
end

-- 2ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
begin
  by_cases hP : P,
  { exact hPR hP, },
  { exact hQR (hPQ hP), },
end

-- 3ª demostración
example
```

```
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
dite P (λ h, hPR h) (λ h, hQR (hPQ h))

-- 4ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
  from em P,
or.elim h
  ( assume hP : P,
    show R,
      from hPR hP)
  ( assume hnP : ¬P,
    have hQ : Q,
      from hPQ hnP,
    show R,
      from hQR hQ)

-- 5ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
  from em P,
or.elim h
  ( assume hP : P,
    show R,
      from hPR hP)
  ( assume hnP : ¬P,
    have hQ : Q,
      from hPQ hnP,
    hQR hQ)

-- 6ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
```

```
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
  from em P,
or.elim h
  ( assume hP : P,
    show R,
      from hPR hP)
  ( assume hnP : ¬P,
    hQR (hPQ hnP))

-- 7ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
  from em P,
or.elim h
  ( assume hP : P,
    show R,
      from hPR hP)
  (λ hnP, hQR (hPQ hnP))

-- 8ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
  from em P,
or.elim h
  ( assume hP : P,
    hPR hP)
  (λ hnP, hQR (hPQ hnP))

-- 9ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
have h : P ∨ ¬P,
```

```
  from em P,
or.elim h
  (λ hP, hPR hP)
  (λ hnP, hQR (hPQ hnP))

-- 10ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
or.elim (em P) (λ h, hPR h) (λ h, hQR (hPQ h))

-- 11 demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
-- by hint
by tauto

-- 12ª demostración
example
  (hPR : P → R)
  (hPQ : ¬ P → Q)
  (hQR : Q → R)
  : R :=
by finish
```

6.2.3. Demostración de P ∨ Q, ¬(P ∧ Q) ⊢ ¬P ↔ Q

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar
--    P ∨ Q, ¬(P ∧ Q) ⊢ ¬P ↔ Q
-- ----------------------------------------------------

-- 1ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
```

```
  : ¬P ↔ Q :=
begin
  split,
  { intro hnP,
    cases h₁ with hP hQ,
    { exfalso,
      exact hnP hP, },
    { exact hQ }, },
  { intros hQ hP,
    exact h₂ ⟨hP, hQ⟩ },
end

-- 2ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
           from absurd hP hnP)
       (assume hQ,
        show Q,
          from hQ))
  (assume hQ : Q,
   assume hP : P,
   have hPQ : P ∧ Q,
     from and.intro hP hQ,
   show false,
     from h₂ hPQ)

-- 3ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
```

```
            from absurd hP hnP)
        (assume hQ,
         show Q,
           from hQ))
  (assume hQ : Q,
   assume hP : P,
   have hPQ : P ∧ Q,
     from and.intro hP hQ,
   h₂ hPQ)

-- 4ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
          from absurd hP hnP)
       (assume hQ,
        show Q,
          from hQ))
  (assume hQ : Q,
   assume hP : P,
   h₂ (and.intro hP hQ))

-- 5ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
          from absurd hP hnP)
       (assume hQ,
        show Q,
          from hQ))
  (λ hQ hP, h₂ (and.intro hP hQ))
```

```
-- 6ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
           from absurd hP hnP)
       (assume hQ,
        hQ))
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 7ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
           from absurd hP hnP)
       (λ hQ, hQ))
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 8ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        show Q,
           from absurd hP hnP)
       id)
```

```
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 9ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (assume hP: P,
        absurd hP hnP)
       id)
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 10ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   show Q, from
     or.elim h₁
       (λ hP, absurd hP hnP)
       id)
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 11ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
  (assume hnP: ¬P,
   or.elim h₁ (λ hP, absurd hP hnP) id)
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 12ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
iff.intro
```

```
  (λ hnP, or.elim h₁ (λ hP, absurd hP hnP) id)
  (λ hQ hP, h₂ (and.intro hP hQ))

-- 13ª demostración
example
  (h₁ : P ∨ Q)
  (h₂ : ¬(P ∧ Q))
  : ¬P ↔ Q :=
-- by hint
by finish
```

6.2.4. Principio de contraposición

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

```
open_locale classical

-- ----------------------------------------------------
-- Ejercicio. Demostrar el principio de contraposición
--    (P → Q) ↔ (¬Q → ¬P)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
begin
  split,
  { intros hPQ hnQ hP,
    apply hnQ,
    apply hPQ,
    exact hP,},
  { intros hQP hP,
    by_contradiction hnQ,
    apply absurd hP,
    apply hQP,
    exact hnQ, },
end

-- 2ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
begin
  split,
```

```
  { intros hPQ hnQ hP,
    exact hnQ (hPQ hP),},
  { intros hQP hP,
    by_contradiction hnQ,
    exact absurd hP (hQP hnQ), },
end

-- 3ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
begin
  split,
  { exact λ hPQ hnQ hP, hnQ (hPQ hP), },
  { exact λ hQP hP, by_contradiction (λ hnQ , absurd hP (hQP hnQ)), },
end

-- 4ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
⟨λ hPQ hnQ hP, hnQ (hPQ hP),
 λ hQP hP, by_contradiction (λ hnQ , absurd hP (hQP hnQ))⟩

-- 5ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
begin
  split,
  { intros h1 h2 h3,
    apply h2,
    exact h1 h3, },
  { intro h4,
    contrapose,
    exact h4, },
end

-- 6ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
iff.intro
  ( assume hPQ : P → Q,
    assume hnQ : ¬Q,
    assume hP : P,
    have hQ : Q,
      from hPQ hP,
    show false,
```

```
      from hnQ hQ )
  ( assume hQP : (¬Q → ¬P),
    assume hP : P,
    show Q, from
      by_contradiction
        ( assume hnQ : ¬Q,
          have hnP : ¬P,
            from hQP hnQ,
          show false,
            from hnP hP))

-- 7ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
iff.intro
  (λ hPQ hnQ hP, hnQ (hPQ hP))
  (λ hQP hP, by_contradiction (λ hnQ, (hQP hnQ) hP))

-- 8ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
-- by library_search
not_imp_not.symm

-- 9ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
-- by hint
by tauto

-- 10ª demostración
example :
  (P → Q) ↔ (¬Q → ¬P) :=
by finish
```

6.2.5. Definición del condicional mediante la negación y la disyunción

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

```
open_locale classical
```

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar que
--    (P → Q) ↔ (¬ P ∨ Q)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
begin
  split,
  { intro hPQ,
    by_cases hP : P,
    { right,
      apply hPQ,
      exact hP, },
    { left,
      exact hP, }},
  { intros h hP,
    cases h with hnP hQ,
    { exact absurd hP hnP, },
    { exact hQ, }},
end

-- 2ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
begin
  split,
  { intro hPQ,
    by_cases hP : P,
    { exact or.inr (hPQ hP), },
    { exact or.inl hP, }},
  { rintros (hnP | hQ) hP,
    { exact absurd hP hnP, },
    { exact hQ, }},
end

-- 3ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
iff.intro
  ( assume hPQ : P → Q,
    show ¬P ∨ Q, from
      or.elim (em P)
      ( assume hP : P,
```

```
        have hQ : Q,
          from hPQ hP,
        show ¬P ∨ Q,
          from or.inr hQ)
      ( assume hnP : ¬P,
        show ¬P ∨ Q,
          from or.inl hnP))
  ( assume hnPQ : ¬P ∨ Q,
    assume hP: P,
    show Q, from
      or.elim hnPQ
        ( assume hnP : ¬P,
          show Q,
            from absurd hP hnP)
        ( assume hQ : Q,
          show Q,
            from hQ))

-- 4ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
iff.intro
  (λ hPQ, or.elim (em P)
    (λ hP,  or.inr (hPQ hP))
    (λ hnP, or.inl hnP))
  (λ hnPQ hP, or.elim hnPQ
    (λ hnP, absurd hP hnP)
    id)

-- 5ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
-- by library_search
imp_iff_not_or

-- 6ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
⟨λ hPQ, if hP : P then or.inr (hPQ hP) else or.inl hP,
 λ hnPQ hP, or.elim hnPQ (λ hnP, absurd hP hnP) id⟩

-- 7ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
-- by hint
```

```
by tauto

-- 8ª demostración
example :
  (P → Q) ↔ (¬ P ∨ Q) :=
by finish
```

6.2.6. Un número es par syss lo es su cuadrado

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

```
variable (n : ℤ)

-- ----------------------------------------------------
-- Ejercicio. Demostrar que un número es par syss lo es
-- su cuadrado.
-- ----------------------------------------------------

-- 1ª demostración
example :
  even (n^2) ↔ even n :=
begin
  split,
  { contrapose,
    rw ← odd_iff_not_even,
    rw ← odd_iff_not_even,
    unfold odd,
    intro h,
    cases h with k hk,
    use 2*k*(k+1),
    rw hk,
    ring, },
  { unfold even,
    intro h,
    cases h with k hk,
    use 2*k^2,
    rw hk,
    ring, },
end

-- 2ª demostración
example :
```

```
  even (n^2) ↔ even n :=
begin
  split,
  { contrapose,
    rw ← odd_iff_not_even,
    rw ← odd_iff_not_even,
    rintro ⟨k, rfl⟩,
    use 2*k*(k+1),
    ring, },
  { rintro ⟨k, rfl⟩,
    use 2*k^2,
    ring, },
end

-- 3ª demostración
example :
  even (n^2) ↔ even n :=
iff.intro
  ( have h : ¬even n → ¬even (n^2),
      { assume h1 : ¬even n,
        have h2 : odd n,
          from odd_iff_not_even.mpr h1,
        have h3: odd (n^2), from
          exists.elim h2
            ( assume k,
              assume hk : n = 2*k+1,
              have h4 : n^2 = 2*(2*k*(k+1))+1, from
                calc  n^2
                    = (2*k+1)^2         : by rw hk
                ... = 4*k^2+4*k+1       : by ring
                ... = 2*(2*k*(k+1))+1 : by ring,
              show odd (n^2),
                from exists.intro (2*k*(k+1)) h4),
        show ¬even (n^2),
          from odd_iff_not_even.mp h3 },
    show even (n^2) → even n,
      from not_imp_not.mp h )
  ( assume h1 : even n,
    show even (n^2), from
      exists.elim h1
        ( assume k,
          assume hk : n = 2*k ,
          have h2 : n^2 = 2*(2*k^2), from
            calc  n^2
```

```
                = (2*k)^2    : by rw hk
            ... = 2*(2*k^2) : by ring,
        show even (n^2),
          from exists.intro (2*k^2) h2 ))
```

6.2.7. Pruebas de la ley de De Morgan: ¬(P ∧ Q) ↔ ¬P ∨ ¬Q

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar que
--    ¬(P ∧ Q) ↔ ¬P ∨ ¬Q
-- ----------------------------------------------------

-- 1ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
begin
  split,
  { intro h,
    by_cases hP : P,
    { right,
      intro hQ,
      apply h,
      split,
      { exact hP, },
      { exact hQ, }},
    { left,
      exact hP, }},
  { intros h h',
    cases h' with hP hQ,
    cases h with hnP hnQ,
    { exact hnP hP, },
    { exact hnQ hQ, }},
end

-- 2ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
begin
  split,
  { intro h,
    by_cases hP : P,
```

```
    { right,
      intro hQ,
      exact h ⟨hP, hQ⟩, },
    { left,
      exact hP, }},
  { rintros (hnP | hnQ) ⟨hP, hQ⟩,
    { exact hnP hP, },
    { exact hnQ hQ, }},
end

-- 3ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
      or.elim (classical.em P)
        ( assume hP : P,
          have hnQ : ¬Q,
            { assume hQ : Q,
              have hPQ: P ∧ Q,
                from and.intro hP hQ,
              show false,
                from h hPQ },
          show ¬P ∨ ¬Q,
            from or.inr hnQ)
        ( assume hnP : ¬P,
          show ¬P ∨ ¬Q,
            from or.inl hnP))
  ( assume h: ¬P ∨ ¬Q,
    show ¬(P ∧ Q), from
      assume hPQ : P ∧ Q,
      show false, from
        or.elim h
          ( assume hnP: ¬P,
            show false,
              from hnP (and.left hPQ))
          ( assume hnQ: ¬Q,
            show false,
              from hnQ (and.right hPQ)))

-- 4ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
```

```
      or.elim (classical.em P)
        ( assume hP : P,
          have hnQ : ¬Q,
            { assume hQ : Q,
              have hPQ: P ∧ Q,
                from and.intro hP hQ,
              show false,
                from h hPQ },
          show ¬P ∨ ¬Q,
            from or.inr hnQ)
        or.inl)
  ( assume h: ¬P ∨ ¬Q,
    show ¬(P ∧ Q), from
      assume hPQ : P ∧ Q,
      show false, from
        or.elim h
          ( assume hnP: ¬P,
            show false,
              from hnP (and.left hPQ))
          (λ hnQ, hnQ (and.right hPQ)))

-- 5ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
      or.elim (classical.em P)
        ( assume hP : P,
          have hnQ : ¬Q,
            { assume hQ : Q,
              have hPQ: P ∧ Q,
                from and.intro hP hQ,
              show false,
                from h hPQ },
          or.inr hnQ)
        or.inl)
  ( assume h: ¬P ∨ ¬Q,
    show ¬(P ∧ Q), from
      assume hPQ : P ∧ Q,
      show false, from
        or.elim h
          (λ hnP, hnP (and.left hPQ))
          (λ hnQ, hnQ (and.right hPQ)))

-- 6ª demostración
```

```
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
      or.elim (classical.em P)
        ( assume hP : P,
          have hnQ : ¬Q,
            { assume hQ : Q,
              show false,
                from h (and.intro hP hQ) },
          or.inr hnQ)
        or.inl)
  ( assume h: ¬P ∨ ¬Q,
    show ¬(P ∧ Q), from
      λ hPQ, or.elim h (λ hnP, hnP (and.left hPQ))
                       (λ hnQ, hnQ (and.right hPQ)))

-- 7ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
      or.elim (classical.em P)
        ( assume hP : P,
          or.inr (λ hQ, h (and.intro hP hQ)))
        or.inl)
  ( λ h hPQ, or.elim h (λ hnP, hnP (and.left hPQ))
                       (λ hnQ, hnQ (and.right hPQ)))

-- 8ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  ( assume h : ¬(P ∧ Q),
    show ¬P ∨ ¬Q, from
      or.elim (classical.em P)
        (λ hP, or.inr (λ hQ, h (and.intro hP hQ)))
        or.inl)
  ( λ h hPQ, or.elim h (λ hnP, hnP (and.left hPQ))
                       (λ hnQ, hnQ (and.right hPQ)))

-- 9ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
iff.intro
  (λ h, or.elim (classical.em P)
          (λ hP, or.inr (λ hQ, h (and.intro hP hQ)))
```

```
          or.inl)
  (λ h hPQ, or.elim h
              (λ hnP, hnP (and.left hPQ))
              (λ hnQ, hnQ (and.right hPQ)))

-- 10ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
⟨λ h, or.elim (classical.em P) (λ hP, or.inr (λ hQ, h ⟨hP, hQ⟩)) or.inl,
 λ h hPQ, or.elim h (λ hnP, hnP hPQ.1) (λ hnQ, hnQ hPQ.2)⟩

-- 11ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
-- by library_search
not_and_distrib

-- 12ª demostración
example : ¬(P ∧ Q) ↔ ¬P ∨ ¬Q :=
-- by hint
by finish
```

6.2.8. Negación del existencial: Caracterización de números no pares.

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

```
-- ----------------------------------------------------
-- Ejercicio. Demostrar que si n es un número entero,
-- entonces
--    ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k
-- ----------------------------------------------------

-- 1ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
begin
  split,
  { intro h,
    intro k,
    intro hk,
    apply h,
    use k,
    exact hk, },
  { intro h1,
    intro h2,
```

```
    cases h2 with k hk,
    apply h1 k,
    exact hk, },
end

-- 2ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
begin
  split,
  { intros h k hk,
    exact h ⟨k, hk⟩ },
  { rintros h ⟨k, rfl⟩,
    apply h k,
    refl, },
end

-- 3ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
begin
  split,
  { intros h k hk,
    exact h ⟨k, hk⟩ },
  { rintros h ⟨k, rfl⟩,
    exact h k rfl },
end

-- 4ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
begin
  push_neg,
end

-- 5ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
by push_neg

-- 6ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  ( assume h1 : ¬(∃ k, n = 2*k),
    show ∀ k, n ≠ 2*k, from
      assume k,
      show n ≠ 2*k, from
        assume h2 : n = 2*k,
        have h3 : ∃ k, n = 2*k,
```

```
          from exists.intro k h2,
        show false,
          from h1 h3)
  ( assume h1 : ∀ k, n ≠ 2*k,
    show ¬(∃ k, n = 2*k), from
      assume h2 : ∃ k, n = 2*k,
      show false, from
        exists.elim h2
          ( assume k,
            assume hk : n = 2*k,
            have h3 : n ≠ 2*k,
              from h1 k,
            show false,
              from h3 hk))

-- 7ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  ( assume h1 : ¬(∃ k, n = 2*k),
    show ∀ k, n ≠ 2*k, from
      assume k,
      show n ≠ 2*k, from
        assume h2 : n = 2*k,
        have h3 : ∃ k, n = 2*k,
          from exists.intro k h2,
        h1 h3)
  ( assume h1 : ∀ k, n ≠ 2*k,
    show ¬(∃ k, n = 2*k), from
      assume h2 : ∃ k, n = 2*k,
      show false, from
        exists.elim h2
          ( assume k,
            assume hk : n = 2*k,
            have h3 : n ≠ 2*k,
              from h1 k,
            h3 hk))

-- 8ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  ( assume h1 : ¬(∃ k, n = 2*k),
    show ∀ k, n ≠ 2*k, from
      assume k,
      show n ≠ 2*k, from
        assume h2 : n = 2*k,
```

```
        h1 (exists.intro k h2))
  ( assume h1 : ∀ k, n ≠ 2*k,
    show ¬(∃ k, n = 2*k), from
      assume h2 : ∃ k, n = 2*k,
      show false, from
        exists.elim h2
          ( assume k,
            assume hk : n = 2*k,
            (h1 k) hk))

-- 9ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  ( assume h1 : ¬(∃ k, n = 2*k),
    show ∀ k, n ≠ 2*k, from
      assume k,
      λ h2, h1 (exists.intro k h2))
  ( assume h1 : ∀ k, n ≠ 2*k,
    show ¬(∃ k, n = 2*k), from
      assume h2 : ∃ k, n = 2*k,
      show false, from
        exists.elim h2
          (λ k hk, (h1 k) hk))

-- 10ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  ( assume h1 : ¬(∃ k, n = 2*k),
    λ k, λ h2, h1 (exists.intro k h2))
  ( assume h1 : ∀ k, n ≠ 2*k,
    show ¬(∃ k, n = 2*k), from
      assume h2 : ∃ k, n = 2*k,
      exists.elim h2 (λ k hk, (h1 k) hk))

-- 11ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
iff.intro
  (λ h1 k h2, h1 (exists.intro k h2))
  (λ h1 h2, exists.elim h2 (λ k hk, (h1 k) hk))

-- 12ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
⟨λ h1 k h2, h1 ⟨k, h2⟩,
 λ h1 h2, exists.elim h2 (λ k hk, (h1 k) hk)⟩
```

```
-- 13ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
-- by hint
by simp

-- 14ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
by finish

-- 15ª demostración
example : ¬(∃ k, n = 2*k) ↔ ∀ k, n ≠ 2*k :=
by norm_num
```

6.2.9. Negación del universal: Caracterización de funciones no pares

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    par : (ℝ → ℝ) → Prop
-- tal que (par f) expresa que f es par.
-- ----------------------------------------------------

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

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que
--    ¬par f ↔ ∃ x, f (-x) ≠ f x
-- ----------------------------------------------------

-- 1ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
begin
  split,
  { contrapose,
    intro h1,
    rw not_not,
    unfold par,
    intro x,
    by_contradiction h2,
```

```
    apply h1,
    use x, },
  { intro h1,
    intro h2,
    unfold par at h2,
    cases h1 with x hx,
    apply hx,
    exact h2 x, },
end

-- 2ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
begin
  split,
  { contrapose,
    intro h1,
    rw not_not,
    intro x,
    by_contradiction h2,
    apply h1,
    use x, },
  { rintros ⟨x, hx⟩ h',
    exact hx (h' x) },
end

-- 3ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
begin
  unfold par,
  push_neg,
end

-- 4ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
iff.intro
  ( have h1 : ¬(∃ x, f (-x) ≠ f x) → ¬¬par f,
      { assume h2 : ¬(∃ x, f (-x) ≠ f x),
        have h3 : par f,
          { assume x,
            have h4 : ¬(f (-x) ≠ f x),
              { assume h5 : f (-x) ≠ f x,
                have h6 : ∃ x, f (-x) ≠ f x,
                  from exists.intro x h5,
                show false,
                  from h2 h6, },
```

```
            show f (-x) = f x,
              from not_not.mp h4},
        show ¬¬par f,
          from not_not.mpr h3 },
    show ¬par f → (∃ x, f (-x) ≠ f x),
      from not_imp_not.mp h1)
  ( assume h1 : ∃ x, f (-x) ≠ f x,
    show ¬par f,
      { assume h2 : par f,
        show false, from
          exists.elim h1
            ( assume x,
              assume hx : f (-x) ≠ f x,
              have h3 : f (-x) = f x,
                from h2 x,
              show false,
                from hx h3)})

-- 5ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
-- by suggest
not_forall

-- 6ª demostración
example : ¬par f ↔ ∃ x, f (-x) ≠ f x :=
-- by hint
by simp [par]
```

6.2.10. La función duplicadora no es par

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

```
-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    par : (ℝ → ℝ) → Prop
-- tal que (par f) expresa que f es par.
-- ----------------------------------------------------

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

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que la función que a cada x
```

```
-- le asigna 2*x no es par.
-- ----------------------------------------------------

example :
  ¬par (λ x, 2*x) :=
begin
  unfold par,
  push_neg,
  use 42,
  linarith,
end
```

6.2.11. La función identidad no está acotada superiormente

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

```
-- Ejercicio 1. Definir la función
--    acotada_superiormente : (ℝ → ℝ) → Prop
-- tal que (acotada_superiormente f) expresa que la
-- función f está acotada superiormente.
-- ----------------------------------------------------

def acotada_superiormente : (ℝ → ℝ) → Prop
| f := ∃ M, ∀ x, f x ≤ M

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar que la función identidad no
-- está acotada superiormente.
-- ----------------------------------------------------

-- 1ª demostración
example : ¬acotada_superiormente id :=
begin
  unfold acotada_superiormente,
  unfold id,
  by_contradiction h,
  cases h with M hM,
  specialize hM (M+1),
  contrapose hM,
  simp only [not_le],
  exact lt_add_one M,
```

```
end

-- 2ª demostración
example : ¬acotada_superiormente id :=
begin
  unfold acotada_superiormente id,
  push_neg,
  intro M,
  use M + 1,
  linarith,
end

-- 3ª demostración
example : ¬acotada_superiormente id :=
begin
  unfold acotada_superiormente id,
  push_neg,
  exact no_top,
end

-- 4ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
show false, from
  exists.elim h2
    ( assume M,
      assume hM : ∀ x, id x ≤ M,
      have h3 : M + 1 ≤ M,
        from hM (M+1),
      have h4 : ¬(M < M + 1),
        from not_lt.mpr h3,
      have h5 : M < M + 1,
        from lt_add_one M,
      show false,
        from h4 h5)

-- 5ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
show false, from
  exists.elim h2
```

```
    ( assume M,
      assume hM : ∀ x, id x ≤ M,
      have h3 : M + 1 ≤ M,
        from hM (M+1),
      have h4 : ¬(M < M + 1),
        from not_lt.mpr h3,
      have h5 : M < M + 1,
        from lt_add_one M,
      h4 h5)

-- 6ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
show false, from
  exists.elim h2
    ( assume M,
      assume hM : ∀ x, id x ≤ M,
      have h3 : M + 1 ≤ M,
        from hM (M+1),
      (not_lt.mpr h3) (lt_add_one M))

-- 7ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
show false, from
  exists.elim h2
    ( assume M,
      assume hM : ∀ x, id x ≤ M,
      (not_lt.mpr (hM (M+1))) (lt_add_one M))

-- 8ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
show false, from
  exists.elim h2
    (λ M hM, (not_lt.mpr (hM (M+1))) (lt_add_one M))

-- 9ª demostración
example : ¬acotada_superiormente id :=
```

```
assume h1 : acotada_superiormente id,
have h2 : ∃ M, ∀ x, id x ≤ M,
  from h1,
exists.elim h2
  (λ M hM, (not_lt.mpr (hM (M+1))) (lt_add_one M))

-- 10ª demostración
example : ¬acotada_superiormente id :=
assume h1 : acotada_superiormente id,
exists.elim h1
  (λ M hM, (not_lt.mpr (hM (M+1))) (lt_add_one M))

-- 11ª demostración
example : ¬acotada_superiormente id :=
λ h1, exists.elim h1 (λ M hM, (not_lt.mpr (hM (M+1))) (lt_add_one M))
```

6.2.12. CS menor o igual que cero

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

```
-- 1ª demostración
example :
  (∀ ε > 0, x ≤ ε) → x ≤ 0 :=
begin
  contrapose,
  push_neg,
  intro h,
  use x/2,
  split ; linarith,
end

-- 2ª demostración
example :
  (∀ ε > 0, x ≤ ε) → x ≤ 0 :=
begin
  contrapose!,
  intro h,
  use x/2,
  split ; linarith,
end
```

6.2.13. Equivalencia de definiciones de creciente

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

```
example :
  (∀ x y, x < y → f x < f y) ↔ (∀ x y, (x ≤ y ↔ f x ≤ f y)) :=
begin
  split,
  { intros hf x y,
    split,
    { intros hxy,
      cases eq_or_lt_of_le hxy with hxy hxy,
      { rw hxy },
      { linarith [hf x y hxy]} },
    { contrapose!,
      apply hf } },
  { intros hf x y,
    contrapose!,
    intro h,
    rwa hf, }
end
```

Capítulo 7

Límites y negaciones

- Límites y negaciones

```
variables  (f : ℝ → ℝ)
variables  (x₀ l : ℝ)

-- ------------------------------------------------------
-- Ejercicio. Definir la notación |x| para el valor
-- absoluto de x.
-- ------------------------------------------------------

notation `|`x`|` := abs x

-- ------------------------------------------------------
-- Ejercicio. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ------------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ¬ limite u l ↔ ∃ ε > 0, ∀ N, ∃ n ≥ N, |u n - l| > ε
-- -----------------------------------------------------------------------

example :
  ¬ limite u l ↔ ∃ ε > 0, ∀ N, ∃ n ≥ N, |u n - l| > ε :=
begin
  unfold limite,
```

```
  push_neg,
  simp,
end

-- ----------------------------------------------------
-- Ejercicio. Definir la función
--    continua : (ℝ → ℝ) → ℝ → Prop
-- tal que (continua f x₀) se verifica si f es continua
-- en x₀.
-- ----------------------------------------------------

def continua : (ℝ → ℝ) → ℝ → Prop :=
λ f x₀, ∀ ε > 0, ∃ δ > 0, ∀ x, |x - x₀| ≤ δ →  |f x - f x₀| ≤ ε

-- ------------------------------------------------------------------
-- Ejercicio. Demostrar que
--   ¬ (continua f x₀) ↔
--   ∃ ε > 0, ∀ δ > 0, ∃ x, |x - x₀| ≤ δ ∧ |f x - f x₀| > ε
-- -------------------------------------------------------------------

example :
  ¬ (continua f x₀) ↔
  ∃ ε > 0, ∀ δ > 0, ∃ x, |x - x₀| ≤ δ ∧ |f x - f x₀| > ε :=
begin
  unfold continua,
  push_neg,
  simp,
end

-- ----------------------------------------------------
-- Ejercicio. Definir la función
--    uniformemente_continua : (ℝ → ℝ) → Prop
-- tal que (uniformemente_continua f) se verifica si f
-- es uniformemente continua.
-- ----------------------------------------------------

def uniformemente_continua : (ℝ → ℝ) → Prop :=
λ f, ∀ ε > 0, ∃ δ > 0, ∀ x x', |x' - x| ≤ δ →  |f x' - f x| ≤ ε

-- ------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ¬ (uniformemente_continua f) ↔
--    ∃ ε > 0, ∀ δ > 0, ∃ x x', |x' - x| ≤ δ ∧ |f x' - f x| > ε :=
-- -------------------------------------------------------------------
```

```
example :
  ¬ (uniformemente_continua f) ↔
  ∃ ε > 0, ∀ δ > 0, ∃ x x', |x' - x| ≤ δ ∧ |f x' - f x| > ε :=
begin
  unfold uniformemente_continua,
  push_neg,
  simp,
end

-- ----------------------------------------------------
-- Ejercicio. Definir la función
--    secuencialmente_continua : (ℝ → ℝ) → Prop
-- tal que (secuencialmente_continua f x₀) se verifica
-- si f es secuencialmente continua en x₀.
-- ----------------------------------------------------

def secuencialmente_continua : (ℝ → ℝ) → ℝ → Prop :=
λ f x₀, ∀ u : ℕ → ℝ, (∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - x₀| ≤ ε) →
                     (∀ ε > 0, ∃ N, ∀ n ≥ N, |(f ∘ u) n - f x₀| ≤ ε)

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ¬ secuencialmente_continua f x₀  ↔
--    ∃ u : ℕ → ℝ,
--      (∀ δ > 0, ∃ N, ∀ n ≥ N, |u n - x₀| ≤ δ) ∧
--      (∃ ε > 0,  ∀ N, ∃ n ≥ N, |f (u n) - f x₀| > ε)
-- ----------------------------------------------------------------------

example :
  ¬ secuencialmente_continua f x₀  ↔
  ∃ u : ℕ → ℝ,
    (∀ δ > 0, ∃ N, ∀ n ≥ N, |u n - x₀| ≤ δ) ∧
    (∃ ε > 0,  ∀ N, ∃ n ≥ N, |f (u n) - f x₀| > ε) :=
begin
  unfold secuencialmente_continua,
  push_neg,
  simp,
end
end

-- ----------------------------------------------------
-- Ejercicio. Definir la función
-- ----------------------------------------------------

def tiende_a_infinito : (ℕ → ℝ) → Prop :=
```

```
λ u, ∀ A, ∃ N, ∀ n ≥ N, u n ≥ A

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    tiende_a_infinito u → ∀ l, ¬ limite u l
-- ----------------------------------------------------------------------

example
  {u : ℕ → ℝ}
  : tiende_a_infinito u → ∀ l, ¬ limite u l :=
begin
  intros h l hl,
  cases hl 1 (by linarith) with N hN,
  cases h (l+2) with N' hN',
  let N₀ := max N N',
  specialize hN N₀ (le_max_left _ _),
  specialize hN' N₀ (le_max_right _ _),
  rw abs_le at hN,
  linarith,
end

-- ---------------------------------------------------
-- Ejercicio. Definir la función
--    no_decreciente : (ℕ → ℝ) → Prop
-- tal que (no_dreciente u) expresa que la sucesión u
-- es no decreciente.
-- ---------------------------------------------------

def no_decreciente : (ℕ → ℝ) → Prop :=
λ u, ∀ n m, n ≤ m → u n ≤ u m

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si u es una sucesión no decreciente y su
-- límite es l, entonces
--    ∀ n, u n ≤ l
-- ----------------------------------------------------------------------

example
  (u : ℕ → ℝ)
  (l : ℝ)
  (h : limite u l)
  (h' : no_decreciente u)
  : ∀ n, u n ≤ l :=
begin
  intro n,
```

```
  by_contradiction H,
  push_neg at H,
  cases h ((u n - l)/2) (by linarith) with N hN,
  specialize hN (max n N) (le_max_right _ _),
  specialize h' n (max n N) (le_max_left _ _),
  rw abs_le at hN,
  linarith,
end

-- ---------------------------------------------------
-- Ejercicio. Definir la función
--    cota_superior : set ℝ → ℝ → Prop
-- tal que (cota_superior A x) expresa que x es una
-- cota superior de A.
-- ---------------------------------------------------

def cota_superior : set ℝ → ℝ → Prop :=
λ A x, ∀ a ∈ A, a ≤ x

-- ---------------------------------------------------
-- Ejercicio. Definir la función
--    es_supremo : set ℝ → ℝ → Prop
-- tal que (es_supremos A x) expresa que x es el
-- supremo de A.
-- ---------------------------------------------------

def es_supremo : set ℝ → ℝ → Prop :=
λ A x, cota_superior A x ∧ ∀ y, cota_superior A y → x ≤ y

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si x es el supremo de A, entonces
--    ∀ y, y < x → ∃ a ∈ A, y < a
-- ----------------------------------------------------------------------

example
  {A : set ℝ}
  {x : ℝ}
  (hx : es_supremo A x)
  : ∀ y, y < x → ∃ a ∈ A, y < a :=
begin
  intro y,
  contrapose!,
  exact hx.right y,
end
```

```
-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (∀ ε > 0, y ≤ x + ε) →  y ≤ x
-- ----------------------------------------------------------------------

lemma le_of_le_add_all'
  {x y : ℝ} :
  (∀ ε > 0, y ≤ x + ε) →  y ≤ x :=
begin
  contrapose!,
  intro h,
  use (y-x)/2,
  split ; linarith,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si x es el límite de u e y es una cota
-- superior de u, entonces x ≤ y.
-- ----------------------------------------------------------------------

example
  {x y : ℝ}
  {u : ℕ → ℝ}
  (hu : limite u x)
  (h : ∀ n, u n ≤ y)
  : x ≤ y :=
begin
  apply le_of_le_add_all',
  intros ε hε,
  cases hu ε hε with N hN,
  specialize hN N (by linarith),
  rw abs_le at hN,
  linarith [h N],
end
```

Capítulo 8

Ampliación de límites

- Ampliación de límites

```
variable {f : ℝ → ℝ}
variable {x₀ : ℝ}
variable {u : ℕ → ℝ}
variable {φ : ℕ → ℕ}

-- ---------------------------------------------------
-- Ejercicio. Definir la notación |x| para el valor
-- absoluto de x.
-- ---------------------------------------------------

notation `|`x`|` := abs x

-- ---------------------------------------------------
-- Ejercicio. Definir la función
--    limite : (ℕ → ℝ) → ℝ → Prop
-- tal que (limite u c) expresa que c es el límite de
-- la sucesión u.
-- ---------------------------------------------------

def limite : (ℕ → ℝ) → ℝ → Prop :=
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| ≤ ε

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    (∀ ε > 0, y ≤ x + ε) →  y ≤ x
-- -----------------------------------------------------------------------

lemma le_of_le_add_all
  {x y : ℝ}
```

```
  : (∀ ε > 0, y ≤ x + ε) →  y ≤ x :=
begin
  contrapose!,
  intro h,
  use (y-x)/2,
  split ; linarith,
end

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si x es el límite de u y
--    ∃ N, ∀ n ≥ N, y ≤ u n
-- entonce y ≤ x.
-- ----------------------------------------------------------------------

lemma le_lim
  {x y : ℝ}
  {u : ℕ → ℝ}
  (hu : limite u x)
  (h : ∃ N, ∀ n ≥ N, y ≤ u n)
  : y ≤ x :=
begin
  apply le_of_le_add_all,
  intros ε hε,
  cases hu ε hε with N hN,
  cases h with N' hN',
  let N₀ := max N N',
  specialize hN N₀ (le_max_left N N'),
  specialize hN' N₀ (le_max_right N N'),
  rw abs_le at hN,
  linarith,
end


-- ----------------------------------------------------
-- Ejercicio. Definir la función
--    cota_superior : set ℝ → ℝ → Prop
-- tal que (cota_superior A x) expresa que x es una
-- cota superior de A.
-- ----------------------------------------------------

def cota_superior : set ℝ → ℝ → Prop :=
λ A x, ∀ a ∈ A, a ≤ x

-- ----------------------------------------------------
-- Ejercicio. Definir la función
```

```
--    es_supremo : set ℝ → ℝ → Prop
-- tal que (es_supremos A x) expresa que x es el
-- supremo de A.
-- ----------------------------------------------------

def es_supremo : set ℝ → ℝ → Prop :=
λ A x, cota_superior A x ∧ ∀ y, cota_superior A y → x ≤ y

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si x es el supremo de A,
-- entonces
--    ∀ y, y < x → ∃ a ∈ A, y < a
-- ---------------------------------------------------------------------

lemma lt_sup
  {A : set ℝ}
  {x : ℝ} (
  hx : es_supremo A x)
  : ∀ y, y < x → ∃ a ∈ A, y < a :=
begin
  intro y,
  contrapose!,
  exact hx.right y,
end

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    limite (λ n, x) x
-- ---------------------------------------------------------------------

lemma limit_const
  (x : ℝ)
  : limite (λ n, x) x :=
λ ε ε_pos, ⟨0, λ _ _, by simp [le_of_lt ε_pos]⟩

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si x es el límite de u e y es una cota
-- superior de u, entonces x ≤ y.
-- ---------------------------------------------------------------------

lemma lim_le
  {x y : ℝ}
  {u : ℕ → ℝ}
  (hu : limite u x)
  (ineg : ∀ n, u n ≤ y)
```

```
  : x ≤ y :=
begin
  apply le_of_le_add_all,
  intros ε ε_pos,
  cases hu ε ε_pos with N hN,
  specialize hN N (by linarith),
  specialize ineg N,
  rw abs_le at hN,
  linarith,
end

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si dos sucesiones tienen
-- el mismo límite, entonces las sucesiones que están
-- comprendidas entre éstas también tienen el mismo
-- límite.
-- ----------------------------------------------------

-- Nota. En la demostración se usará el siguiente lema:
lemma max_ge_iff
  {p q r : ℕ}
  : r ≥ max p q ↔ r ≥ p ∧ r ≥ q :=
max_le_iff

lemma emparedado
  {u v w : ℕ → ℝ}
  {a : ℝ}
  (hu : limite u a)
  (hw : limite w a)
  (h : ∀ n, u n ≤ v n)
  (h' : ∀ n, v n ≤ w n)
  : limite v a :=
begin
  intros ε hε,
  cases hu ε hε with N hN, clear hu,
  cases hw ε hε with N' hN', clear hw hε,
  use max N N',
  intros n hn,
  rw max_ge_iff at hn,
  specialize hN n (by linarith),
  specialize hN' n (by linarith),
  specialize h n,
  specialize h' n,
  rw abs_le at *,
  split ; linarith,
```

```
end

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ∀ ε > 0, ∃ N : ℕ, ∀ n ≥ N, 1/(n + 1 : ℝ) ≤ ε
-- ----------------------------------------------------------------------

lemma inv_succ_le_all :
  ∀ ε > 0, ∃ N : ℕ, ∀ n ≥ N, 1/(n + 1 : ℝ) ≤ ε :=
begin
  convert metric.tendsto_at_top.mp (tendsto_one_div_add_at_top_nhds_0_nat),
  apply propext,
  simp only [real.dist_eq, sub_zero],
  split,
    { intros h ε ε_pos,
      cases h (ε/2) (by linarith) with N hN,
      use N,
      intros n hn,
      rw abs_of_pos (nat.one_div_pos_of_nat : 1/(n+1 : ℝ) > 0),
      specialize hN n hn,
      linarith, },
  intros h ε ε_pos,
  cases h ε (by linarith) with N hN,
  use N,
  intros n hn,
  specialize hN n hn,
  rw abs_of_pos (@nat.one_div_pos_of_nat ℝ _ n) at hN,
  linarith,
end

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    ∀ n, |u n - x| ≤ 1/(n+1) ⊢ limite u x
-- ----------------------------------------------------------------------

lemma limit_of_sub_le_inv_succ
  {u : ℕ → ℝ}
  {x : ℝ}
  (h : ∀ n, |u n - x| ≤ 1/(n+1))
  : limite u x :=
begin
  intros ε ε_pos,
  rcases inv_succ_le_all ε ε_pos with ⟨N, hN⟩,
  use N,
  intros n hn,
```

```
  specialize h n,
  specialize hN n hn,
  linarith,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que x es el límite de
--    x - 1/(n+1)) x
-- -----------------------------------------------------------------------

lemma limit_const_sub_inv_succ
  (x : ℝ)
  : limite (λ n, x - 1/(n+1)) x :=
begin
  refine limit_of_sub_le_inv_succ (λ n, _),
  rw [show x - 1 / (n + 1) - x = -(1/(n+1)), by ring, abs_neg,  abs_of_pos],
  linarith [@nat.one_div_pos_of_nat ℝ _ n]
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que son equivalentes
-- + es_supremo A x
-- + cota_superior A x ∧ ∃ u : ℕ → ℝ, limite u x ∧ ∀ n, u n ∈ A
-- -----------------------------------------------------------------------

lemma es_supremo_iff
  (A : set ℝ)
  (x : ℝ)
  : (es_supremo A x) ↔
    (cota_superior A x ∧ ∃ u : ℕ → ℝ, limite u x ∧ ∀ n, u n ∈ A ) :=
begin
  split,
  { intro h,
    split,
    { exact h.left, },
    { have : ∀ n : ℕ, ∃ a ∈ A, x - 1/(n+1) < a,
      { intros n,
        have : 1/(n+1 : ℝ) > 0,
          exact nat.one_div_pos_of_nat,
        exact lt_sup h _ (by linarith), },
      choose u hu using this,
      use u,
      split,
      { apply emparedado (limit_const_sub_inv_succ x) (limit_const x),
        { intros n,
```

```
          exact le_of_lt (hu n).2, },
        { intro n,
          exact h.1 _ (hu n).left, } },
      { intro n,
        exact (hu n).left }}},
  { rintro ⟨maj, u, limu, u_in⟩,
    split,
    { exact maj },
    { intros y ymaj,
      apply lim_le limu,
      intro n,
      apply ymaj,
      apply u_in }},
end

-- ----------------------------------------------------
-- Ejercicio. Definir la función
--    continua_en_punto : (ℝ → ℝ) → ℝ → Prop
-- tal que (continua_en_punto f x₀) expresa que f es
-- continua en x₀.
-- ----------------------------------------------------

def continua_en_punto : (ℝ → ℝ) → ℝ → Prop :=
λ f x₀, ∀ ε > 0, ∃ δ > 0, ∀ x, |x - x₀| ≤ δ → |f x - f x₀| ≤ ε

-- -------------------------------------------------------------------
-- Ejercicio. Demostrar que si el límite de f es x₀ y f es continua en
-- x₀, entonces el límite de (f ∘ u) es f(x₀).
-- --------------------------------------------------------------------

lemma seq_continuous_of_continuous
  (hf : continua_en_punto f x₀)
  (hu : limite u x₀)
  : limite (f ∘ u) (f x₀) :=
begin
  intros ε ε_pos,
  rcases hf ε ε_pos with ⟨δ, δ_pos, hδ⟩,
  cases hu δ δ_pos with N hN,
  use N,
  intros n hn,
  apply hδ,
  exact hN n hn,
end

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

```
-- Ejercicio. Demostrar que si
--    ∀ u : ℕ → ℝ, limite u x₀ → limite (f ∘ u) (f x₀)
-- entonces f es continua en x₀.
-- ----------------------------------------------------------------------

example :
  (∀ u : ℕ → ℝ, limite u x₀ → limite (f ∘ u) (f x₀)) →
  continua_en_punto f x₀ :=
begin
  contrapose!,
  intro hf,
  unfold continua_en_punto at hf,
  push_neg at hf,
  cases hf with ε h,
  cases h with ε_pos hf,
  have H : ∀ n : ℕ, ∃ x, |x - x₀| ≤ 1/(n+1) ∧ ε < |f x - f x₀|,
    intro n,
    apply hf,
    exact nat.one_div_pos_of_nat,
  clear hf,
  choose u hu using H,
  use u,
  split,
    intros η η_pos,
    have fait : ∃ (N : ℕ), ∀ (n : ℕ), n ≥ N → 1 / (↑n + 1) ≤ η,
      exact inv_succ_le_all η η_pos,
    cases fait with N hN,
    use N,
    intros n hn,
    calc |u n - x₀| ≤ 1/(n+1) : (hu n).left
                ... ≤ η       : hN n hn,
  unfold limite,
  push_neg,
  use [ε, ε_pos],
  intro N,
  use N,
  split,
  linarith,
  exact (hu N).right,
end

-- -------------------------------------------------------
-- Ejercicio. Definir la función
-- -------------------------------------------------------
```

```
def tiende_a_infinito : (ℕ → ℝ) → Prop :=
λ u, ∀ A, ∃ N, ∀ n ≥ N, u n ≥ A

-- ----------------------------------------------------
-- Ejercicio. Para extraer una sucesión se aplica una
-- función de extracción que conserva el orden; por
-- ejemplo, la subsucesión
--    u₀, u₂, u₄, u₆, ...
-- se ha obtenido con la función de extracción φ tal
-- que φ(n) = 2*n.
--
-- Definir la función
--    extraccion : (ℕ → ℕ) → Prop
-- tal que (extraccion φ) expresa que φ es una función
-- de extracción
-- ----------------------------------------------------

def extraccion : (ℕ → ℕ) → Prop
| φ := ∀ n m, n < m → φ n < φ m

-- ----------------------------------------------------
-- Ejercicio. Demostrar que si φ es una función de
-- extracción, entonces
--    ∀ n, n ≤ φ n
-- ----------------------------------------------------

lemma id_mne_extraccion :
  extraccion φ → ∀ n, n ≤ φ n :=
begin
  intros h n,
  induction n with m HI,
  { linarith },
  { apply nat.succ_le_of_lt,
    linarith [h m (m+1) (by linarith)] },
end

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si u tiende a infinito y φ es una función de
-- extracción, entonces (u ∘ φ) tiende a infinito.
-- ----------------------------------------------------------------------

lemma subseq_tenstoinfinity
  (h : tiende_a_infinito u)
  (hφ : extraccion φ)
  : tiende_a_infinito (u ∘ φ) :=
```

```
begin
  intros A,
  cases h A with N hN,
  use N,
  intros n hn,
  apply hN,
  calc N ≤ n   : hn
     ... ≤ φ n : id_mne_extraccion hφ n,
end

-- ------------------------------------------------------------------
-- Ejercicio. Demostrar que si u tiende al infinito y
--    ∀ n, u n ≤ v n
-- entonces v tiende al infinito.
-- -------------------------------------------------------------------

lemma squeeze_infinity
  {u v : ℕ → ℝ}
  (hu : tiende_a_infinito u)
  (huv : ∀ n, u n ≤ v n)
  : tiende_a_infinito v :=
begin
  intros A,
  cases hu A with N hN,
  use N,
  intros n hn,
  specialize hN n hn,
  specialize huv n,
  linarith,
end

open set

-- ------------------------------------------------------------------
-- Ejercicio. Demostrar que la sucesión identidad tiende a infinito.
-- -------------------------------------------------------------------

lemma limite_id : tiende_a_infinito (λ n, n) :=
begin
  intros A,
  cases exists_nat_gt A with N hN,
  use N,
  intros n hn,
  have : (n : ℝ) ≥ N,
    exact_mod_cast hn,
```

```
  linarith,
end

-- ---------------------------------------------------
-- Ejercicio. Definir la función
--    punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
-- tal que (punto_acumulacion u a) expresa que a es un
-- punto de acumulación de u; es decir, que es el
-- límite de alguna subsucesión de u.
-- ---------------------------------------------------

def punto_acumulacion : (ℕ → ℝ) → ℝ → Prop
| u a := ∃ φ, extraccion φ ∧ limite (u ∘ φ) a

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si es una sucesión en [a,b], entonces
-- existe algún c en [a,b] que es el punto de acumulación de u.
-- ----------------------------------------------------------------------

lemma bolzano_weierstrass
  {a b : ℝ}
  {u : ℕ → ℝ}
  (h : ∀ n, u n ∈ Icc a b)
  : ∃ c ∈ Icc a b, punto_acumulacion u c :=
begin
  rcases (compact_Icc : is_compact (Icc a b)).tendsto_subseq h with ⟨c, c_in, φ, hφ, lim
  use [c, c_in, φ, hφ],
  simp_rw [metric.tendsto_nhds, filter.eventually_at_top, real.dist_eq] at lim,
  intros ε ε_pos,
  rcases lim ε ε_pos with ⟨N, hN⟩,
  use N,
  intros n hn,
  exact le_of_lt (hN n hn)
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que
--    tiende_a_infinito u → ∀ x, ¬ limite u x
-- ----------------------------------------------------------------------

lemma not_seq_limit_of_tendstoinfinity
  {u : ℕ → ℝ} :
  tiende_a_infinito u → ∀ x, ¬ limite u x :=
begin
  intros lim_infinie x lim_x,
```

```
  cases lim_x 1 (by linarith) with N hN,
  cases lim_infinie (x+2) with N' hN',
  let N₀ := max N N',
  specialize hN N₀ (le_max_left _ _),
  specialize hN' N₀ (le_max_right _ _),
  rw abs_le at hN,
  linarith,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es continua en [a,b], entonces existe
-- un M tal que, para todo x ∈ [a,b], f(x) ≤ M.
-- ----------------------------------------------------------------------

lemma bdd_above_segment
  {f : ℝ → ℝ}
  {a b : ℝ}
  (hf : ∀ x ∈ Icc a b, continua_en_punto f x)
  : ∃ M, ∀ x ∈ Icc a b, f x ≤ M :=
begin
  by_contradiction H,
  push_neg at H,
  have clef : ∀ n : ℕ, ∃ x, x ∈ Icc a b ∧ f x > n,
    intro n,
    apply H,
    clear H,
  choose u hu using clef,
  have lim_infinie : tiende_a_infinito (f ∘ u),
    apply squeeze_infinity (limite_id),
    intros n,
    specialize hu n,
    linarith,
  have bornes : ∀ n, u n ∈ Icc a b,
    intro n,
    exact (hu n).left,
  rcases bolzano_weierstrass bornes with ⟨c, c_dans, φ, φ_extr, lim⟩,
  have lim_infinie_extr : tiende_a_infinito (f ∘ (u ∘ φ)),
    exact subseq_tenstoinfinity lim_infinie φ_extr,
  have lim_extr : limite (f ∘ (u ∘ φ)) (f c),
    exact seq_continuous_of_continuous (hf c c_dans) lim,
  exact not_seq_limit_of_tendstoinfinity lim_infinie_extr (f c) lim_extr,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es continua en x₀, también lo es -f.
```

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

lemma continuous_opposite
  {f : ℝ → ℝ}
  {x₀ : ℝ}
  (h : continua_en_punto f x₀)
  : continua_en_punto (λ x, -f x) x₀ :=
begin
  intros ε ε_pos,
  cases h ε ε_pos with δ h,
  cases h with δ_pos h,
  use [δ, δ_pos],
  intros y hy,
  have :  -f y - -f x₀ = -(f y - f x₀), ring,
  rw [this, abs_neg],
  exact h y hy,
end

-- ------------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es continua en [a,b], entonces existe
-- un m en [a,b] tal que, para todo x de [a,b], m ≤ f(x).
-- ------------------------------------------------------------------------

lemma bdd_below_segment
  {f : ℝ → ℝ}
  {a b : ℝ}
  (hf : ∀ x ∈ Icc a b, continua_en_punto f x)
  : ∃ m, ∀ x ∈ Icc a b, m ≤ f x :=
begin
  have : ∃ M, ∀ x ∈ Icc a b, -f x ≤ M,
  { apply bdd_above_segment,
    intros x x_dans,
    exact continuous_opposite (hf x x_dans), },
  cases this with M hM,
  use -M,
  intros x x_dans,
  specialize hM x x_dans,
  linarith,
end

open real

-- ------------------------------------------------------------------------
-- Ejercicio. Demostrar que si A es un subconjunto no vacío de [a,b],
-- entonces existe un x en [a,b] que es el supremo de A.
```

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

lemma sup_segment
  {a b : ℝ}
  {A : set ℝ}
  (hnonvide : ∃ x, x ∈ A)
  (h : A ⊆ Icc a b)
  : ∃ x ∈ Icc a b, es_supremo A x :=
begin
  have b_maj : ∀ (y : ℝ), y ∈ A → y ≤ b,
    from λ y y_in, (h y_in).2,
  have Sup_maj : cota_superior A (Sup A),
  { intro x,
    apply real.le_Sup,
    use [b, b_maj] } ,
  refine ⟨Sup A, _, _⟩,
  { split,
    { cases hnonvide with x x_in,
      exact le_trans (h x_in).1 (Sup_maj _ x_in) },
    { apply Sup_le_ub A hnonvide b_maj } },
  { use Sup_maj,
    intros y y_in,
    rwa real.Sup_le _ hnonvide ⟨b, b_maj⟩ },
end

-- ----------------------------------------------------
-- Ejercicio. Demostrar que cada sucesión tiene como
-- máximo un límite.
-- ----------------------------------------------------

lemma unicidad_limite
  {a b : ℝ}
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contradiction H,
  change a ≠ b at H,
  have h1 : |a - b| > 0,
    exact abs_pos.mpr (sub_ne_zero_of_ne H),
  cases ha (|a - b|/4) (by linarith) with N hN,
  cases hb (|a - b|/4) (by linarith) with N' hN',
  let N₀ := max N N',
  specialize hN N₀ (le_max_left _ _),
  specialize hN' N₀ (le_max_right _ _),
```

```
  have h2 : |a - b| < |a - b |,
    calc  |a - b| = |(a - u N₀) + (u N₀ - b)| : by ring
    ... ≤ |a - u N₀| + |u N₀ - b|             : by apply abs_add
    ... = |u N₀ - a| + |u N₀ - b|             : by rw abs_sub
    ... < |a - b|                             : by linarith,
  linarith,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si l es el límite de u y φ es una extracción,
-- entonces l es el límite de (u ∘ φ).
-- ----------------------------------------------------------------------

lemma subseq_tendsto_of_tendsto
  {l : ℝ}
  (h : limite u l)
  (hφ : extraccion φ)
  : limite (u ∘ φ) l :=
begin
  intros ε ε_pos,
  cases h ε ε_pos with N hN,
  use N,
  intros n hn,
  apply hN,
  calc N ≤ n   : hn
     ... ≤ φ n : id_mne_extraccion hφ n,
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es continua en [a,b], existe un x₀ ∈ [a,b],
-- tal que, para todo x ∈ [a,b], f(x) ≤ f(x₀).
-- ----------------------------------------------------------------------

example
  {a b : ℝ}
  (hab : a ≤ b)
  (hf : ∀ x ∈ Icc a b, continua_en_punto f x)
  : ∃ x₀ ∈ Icc a b, ∀ x ∈ Icc a b, f x ≤ f x₀ :=
begin
  cases bdd_below_segment hf with m hm,
  cases bdd_above_segment hf with M hM,
  let A := {y | ∃ x ∈ Icc a b, y = f x},
  obtain ⟨y₀, y_dans, y_sup⟩ : ∃ y₀ ∈ Icc m M, es_supremo A y₀,
  { apply sup_segment,
```

```
    { use [f a, a, by linarith, hab, by ring], },
    { rintros y ⟨x, x_in, rfl⟩,
      exact ⟨hm x x_in, hM x x_in⟩ } },
  rw es_supremo_iff at y_sup,
  rcases y_sup with ⟨y_maj, u, lim_u, u_dans⟩,
  choose v hv using u_dans,
  cases forall_and_distrib.mp hv with v_dans hufv,
  replace hufv : u = f ∘ v := funext hufv,
  rcases bolzano_weierstrass v_dans with ⟨x₀, x₀_in, φ, φ_extr, lim_vφ⟩,
  use [x₀, x₀_in],
  intros x x_dans,
  have lim : limite (f ∘ v ∘ φ) (f x₀),
  { apply seq_continuous_of_continuous,
    exact hf x₀ x₀_in,
    exact lim_vφ },
  have unique : f x₀ = y₀,
  { apply unicidad_limite lim,
    rw hufv at lim_u,
    exact subseq_tendsto_of_tendsto lim_u φ_extr },
  rw unique,
  apply y_maj,
  use [x, x_dans],
end

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que si x ∈ [a,b] y x ≠ b, entonces x < b.
-- -----------------------------------------------------------------------

lemma stupid
  {a b x : ℝ}
  (h : x ∈ Icc a b)
  (h' : x ≠ b)
  : x < b :=
lt_of_le_of_ne h.right h'

-- ----------------------------------------------------
-- Ejercicio. Definir I como el intervalo [0,1].
-- ----------------------------------------------------

def I := (Icc 0 1 : set ℝ)

-- ----------------------------------------------------------------------
-- Ejercicio. Demostrar que x es el límite de
--    x + 1/(n+1)
-- -----------------------------------------------------------------------
```

```
lemma limit_const_add_inv_succ
  (x : ℝ)
  : limite (λ n, x + 1/(n+1)) x :=
limit_of_sub_le_inv_succ (λ n, by rw abs_of_pos ;
linarith [@nat.one_div_pos_of_nat ℝ _ n])

-- ---------------------------------------------------------------------
-- Ejercicio. Demostrar que si f es continua, f(0) < 0 y f(1) > 0,
-- entonces existe un x₀ ∈ I tal que f(x₀) = 0.
-- ----------------------------------------------------------------------

example
  (f : ℝ → ℝ)
  (hf : ∀ x, continua_en_punto f x)
  (h₀ : f 0 < 0)
  (h₁ : f 1 > 0)
  : ∃ x₀ ∈ I, f x₀ = 0 :=
begin
  let A := { x | x ∈ I ∧ f x < 0},
  have ex_x₀ : ∃ x₀ ∈ I, es_supremo A x₀,
  { apply sup_segment,
      use 0,
      split,
        split, linarith, linarith,
      exact h₀,
    intros x hx,
    exact hx.left },
  rcases ex_x₀ with ⟨x₀, x₀_in, x₀_sup⟩,
  use [x₀, x₀_in],
  have : f x₀ ≤ 0,
  { rw es_supremo_iff at x₀_sup,
    rcases x₀_sup with ⟨maj_x₀, u, lim_u, u_dans⟩,
    have : limite (f ∘ u) (f x₀),
      exact seq_continuous_of_continuous (hf x₀) lim_u,
    apply lim_le this,
    intros n,
    have : f (u n) < 0,
      exact (u_dans n).right,
    linarith  },
  have x₀_1: x₀ < 1,
  { apply stupid x₀_in,
    intro h,
    rw ← h at h₁,
    linarith  },
```

```
  have : f x₀ ≥ 0,
  { have in_I : ∃ N : ℕ, ∀ n ≥ N, x₀ + 1/(n+1) ∈ I,
    { have : ∃ N : ℕ, ∀ n≥ N, 1/(n+1 : ℝ) ≤ 1-x₀,
      { apply inv_succ_le_all,
        linarith, },
      cases this with N hN,
      use N,
      intros n hn,
      specialize hN n hn,
      have : 1/(n+1 : ℝ) > 0,
        exact nat.one_div_pos_of_nat,
      change 0 ≤ x₀ ∧ x₀ ≤ 1 at x₀_in,
      split ; linarith, },
    have not_in : ∀ n : ℕ, x₀ + 1/(n+1) ∉ A,
    -- By definition, x ∉ A means ¬ (x ∈ A).
    { intros n hn,
      cases x₀_sup with x₀_maj _,
      specialize x₀_maj _ hn,
      have : 1/(n+1 : ℝ) > 0,
        from nat.one_div_pos_of_nat,
      linarith, },
    dsimp [A] at not_in, -- This is useful to unfold a let
    push_neg at not_in,
    have lim : limite (λ n, f(x₀ + 1/(n+1))) (f x₀),
    { apply seq_continuous_of_continuous (hf x₀),
      apply limit_const_add_inv_succ },
    apply le_lim lim,
    cases in_I with N hN,
    use N,
    intros n hn,
    exact not_in n (hN n hn), },
  linarith,
end
```

Capítulo 9

Apéndices

9.1. 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`.
- `apply h`, con las hipótesis `h : ∀ (x : U), P x → Q x` y `a : U` y la conclusión `Q a`, cambia la conclusión a `P a`.
- `assumption` concluye la demostración si la conclusión coincide con alguna de las hipótesis.
- `(by_cases hP: P)` divide el objetivo en dos añadiendo la hipótesis `hP : P` al primero y `hP : ¬P` al segundo.
- `by_contradiction h`, si la conclusión es `P`, entonces añade la hipótesis `h : ¬P` y cambia la conclusión a `false`.
- `(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)`.

- `(cases h with a ha)`, cuando la hipótesis `h` es un existencial aplica la regla de eliminación del existencia; es decir, si `h` es `(∃ x : U, P x)`, entonces elimina `h` y añade las hipótesis `(a : U)` y `(ha : P a)`.
- `(cases h e (by t) with a ha)`, cuando la hipótesis `h` es de la forma `∀ x, P x → ∃ y : U, Q x y` y la táctica `t` prueba `P e` entonces añade las hipótesis `a : U` y `ha : Q e a`. (Ver ejemplo).
- `contrapose` transforma la conclusión en su contrapositiva. (Ver ejemplo).
- `exact h` concluye la demostración si `h` es del tipo de la conclusión.
- `funext`, cuando la conclusión es una igualdad de funciones `f = g` de dominio `U` introduce la hipótesis `x : U` y cambia la conclusión a `f x = g x`; es decir, aplica el principio de extensionalidad de funciones.
- `cong`, si la conclusión es una igualdad `A = B` intenta identificar ambos lados y deja como nuevos objetivos los subtérminos de A y B que no son iguales. Por ejemplo, supongamos que el objetivo es `x * f y = g w * f z`. Entonces `congr` produce dos objetivos: `x = g w` e `y = z`.
- `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`.
- `intros h1 ... hn` introduce las hipótesis h1, ..., hn correspondiente a la introducciones de condicionales y universales de la conclusión.
- `intro a`, cuando la conclusión es `∀ x : U, P x`, aplica la regla de introducción del cuantificador universal; es decir, añade la hipótesis `a : U` y cambia la conclusión a `P a`
- `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.
- `norm_num` normaliza expresiones aritméticas.
- `push_neg` interioriza las negaciones en la conclusión.
- `rcases h with ⟨a, rfl⟩`, cuando `h` es una fórmula existencial cuyo cuerpo es una ecuación, sustituye la variable del existencial por `a` y reescribe con la ecuación obtenida.

- `refl` reduce ambos términos de una ecuación y comprueba que son iguales.
- `rename_var x y at h`, cuando se tiene la hipótesis `h : ∀ x, P x` la cambia a `h : ∀ y, P y`.
- `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`.
- `rintro ⟨a, rfl⟩`, cuando la conclusión es una fórmula existencial cuyo cuerpo es una ecuación, sustituye la variable del existencial por `a` y reescribe con la ecuación obtenida.
- `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 [h1, ... hn]` reescribe usando sucesivamente las ecuaciones `h1`, …, `hn`.
- `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.
- Si `specialize h a`, cuando `h : ∀ x : U, P x` y `a : U`, cambia `h` en `h : P a`.
- `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`.
- `split`, cuando la conclusión es un bicondicional, aplica la regla de eliminación del bicondicional; es decir, si la conclusión es `(P ↔ Q)`, entonces crea dos subojetivos: el primero en el que la conclusión es `P → Q` y el segundo donde es `Q → P`.
- `tauto` demuestra automáticamente las tautologías.
- `tidy` demuestra la conclusión usando una variedad de tácticas conservativas.
- `unfold f` expande la definición de la función `f` en la conclusión.
- `unfold f at h` expande la definición de la función `f` en la hipótesis `h`.
- `use a`, cuando la conclusión `∃ x : U, P x` y se tiene la hipótesis `x : U`, aplica la regla de introducción el cuantificador existencial; es decir, cambia la conclusión a `P a`.

9.1.1. 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`.

9.1.2. 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 : ∀ x : U, P x` y `a : U`, entonces `h a` es `P a`.
- Si `h` es una igualdad entonces `h ▸ h’` es la expresión obtenida sustituyendo en h’ el término izquierdo de `h` por el derecho.
- `intro h, cases h with a ha` se reduce a `rintro \<a, ha\>`
- `intros h1 h2, cases h2 with a ha` se reduce a `rintro h \<a, ha\>`.

9.2. Resumen de teoremas usados

Los teoremas utilizados son los siguientes:

```
+ abs_add : abs (a + b) ≤ abs a + abs b
+ abs_eq_zero : abs a = 0 ↔ a = 0
+ abs_le : abs a ≤ b ↔ - b ≤ a ∧ a ≤ b
+ abs_nonneg : 0 ≤ abs a
+ add_assoc : (a + b) + c = a + (b + c)
+ add_comm : a + b = b + a
+ add_eq_zero_iff_eq_neg : a + b = 0 ↔ a = -b
```

```
+ add_halves : a / 2 + a / 2 = 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))
+ antisymm : a ≤ b → b ≤ a → a = b
+ classical.em : p ∨ ¬p
+ congr_arg : a₁ = a₂ → f a₁ = f a₂
+ dvd_add : a ∣ b → a ∣ c → a ∣ b + c
+ dvd_antisymm : a ∣ b → b ∣ a → a = b
+ dvd_gcd_iff : c ∣ gcd a b ↔ c ∣ a ∧ c ∣ b
+ dvd_refl: a ∣ a
+ dvd_trans : a ∣ b → b ∣ c → a ∣ c
+ eq.trans : a = b → b = c → a = c
+ eq_neg_of_add_eq_zero : a + b = 0 → a = -b
+ eq_of_le_of_forall_le_of_dense : a₂ ≤ a₁ → (∀ a₃ > a₂, a₁ ≤ a₃) →  a₁ = a₂
+ eq_or_eq_neg_of_pow_two_eq_pow_two : a ^ 2 = b ^ 2 → a = b ∨ a = -b
+ eq_or_lt_of_le : a ≤ b → a = b ∨ a < b
+ eq_zero_or_eq_zero_of_mul_eq_zero : a * b = 0 → a = 0 ∨ b = 0
+ exists.elim : ∃ x, p x → ∀ (a : α), p a → b → b
+ gcd_dvd : gcd a b ∣ a ∧ gcd a b ∣ b
+ gcd_eq_left_iff_dvd : a ∣ b ↔ gcd a b = a
+ ge_max_iff : r ≥ max p q  ↔ r ≥ p ∧ r ≥ q
+ 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_max_left : p ≤ max p q
+ le_max_right : q ≤ max p q
+ le_of_lt : a < b → a ≤ b
+ le_of_max_le_left : max a b ≤ c → a ≤ c
+ le_of_max_le_right : max a b ≤ c → b ≤ c
+ le_total : a ≤ b ∨ b ≤ a
+ le_trans: a ≤ b → b ≤ c → a ≤ c
+ max_le_iff : max a b ≤ c ↔ a ≤ c ∧ b ≤ c
```

```
+ monotone.comp : monotone g → monotone f → monotone (g ∘ f)
+ mul_assoc : (a * b) * c = a * (b * c)
+ mul_comm : a * b = b * a
+ mul_eq_zero : a * b = 0 ↔ a = 0 ∨ b = 0
+ 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
+ nat.succ_pos : 0 < succ n
+ neg_square : (-z)^2 = z^2
+ not_imp_not : (¬a → ¬b) ↔ (b → a)
+ not_not : ¬¬a ↔ a
+ odd_iff_not_even : odd n ↔ ¬ even n
+ or_comm : a ∨ b ↔ b ∨ a
+ or.elim : P ∨ Q → (P → R) → (Q → R) → R
+ or.rec : (P → R) → (Q → R) → P ∨ Q → R
+ pow_two : a^2 = a * a
+ pow_two_sub_pow_two : a ^ 2 - b ^ 2 = (a + b) * (a - b)
+ rfl : a = a
+ sub_eq_zero : a - b = 0 ↔ a = b
+ sub_le_sub_right : a ≤ b → a - c ≤ b - c
+ 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)
+ surjective f : ∀ b, ∃ a, f a = b
+ surjective.comp : surjective g → surjective f → surjective (g ∘ f)
+ surjective.of_comp : surjective (f ∘ g) → surjective f
+ two_mul : 2 * a = a + a
+ zero_add : 0 + a = a
+ zero_dvd_iff : 0 ∣ a ↔ a = 0
+ zero_eq_mul : 0 = a * b ↔ a = 0 ∨ b = 0
+ zero_mul : 0 * a = 0
```

9.3. 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`

9.4. Nomenclatura

La nomenclatura de los teoremas en inglés sigue las reglas de Mathlib naming conventions. En su adaptación al castellano su usará:

- `de` (en lugar de `of`) para separar las hipótesis de la conclusión. Por ejemplo, `mn_no_mye` (en lugar de `lt_of_not_ge`)

Castellano	Inglés	Significado
abs	abs	valor absoluto
adi	add	adición (o suma)
antisim	antisymm	antisimétrica
asim	asymm	asimética
asoc	assoc	asociativa
cancel	cancel	cancelativa
cero	zero	cero
congr	congr	congruencia
conj	and	conjunción
conm	comm	conmutativa
dcha	right	derecha
def	def	definición
disy	or	disyunción
elim	elim	eliminación
existe	exists	existe
falso	false	falso
ig	eq	igual
intro	intro	introducción
iny	inj	inyectiva
irefl	irrefl	irreflexiva
izq	left	izquierda
mismo	self	mismo
mn	lt	menor
mne	le	menor o igual
mp	mp	implicación de izquierda a derecha
mpr	mpr	implicación de derecha a izquierda
mul	mul	multiplicación (o producto)
my	gt	mayor
mye	ge	mayor o igual
neg	neg	negativo
nig	ne	no igual
no	not	no
noneg	nonneg	no negativo
nopos	nonpos	no positivo
o	or	o
pos	pos	positivo
pred	pred	predecesor
rec	rec	recursor
refl	refl	reflexiva
sim	symm	simétrica
sub	sub	substracción (o resta)
suc	succ	sucesor
sust	subst	sustitución
syss	iff	si y sólo si
todos	all	todos
trans	trans	transitiva
uno	one	uno