Lógica con Lean

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 16 de diciembre de 2020

Índice general

Capítulo 1

Introducción

El objetivo de este trabajo es presentar una introducción a la Lógica 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.

El trabajo se basa fundamentalmente en

- El curso de ”Lógica matemática y fundamentos en que se estudia la deducción natural proposicional y de primer orden (basado en el libro Logic in computer science: Modelling and reasoning about systems de Michael Huth y Mark Ryan) y su formalización en Isabelle/HOL.

- Los apuntes de Lógica y demostración con Lean que son un resumen del libro Logic and Proof de Jeremy Avigad, Robert Y. Lewis y Floris van Doorn.

- Los apuntes Deducción natural en Lean en el que se presentan ejemplos de uso de las tácticas de Lean correspondientes a las reglas de la deducción natural.

- Los apuntes Matemáticas en Lean en el que se presentan la formalización en Lean de temas básicos de las matemáticas usando las librerías de mathlib. Está basado en el libro Mathematics in Lean de Jeremy Avigad, Kevin Buzzard, Robert Y. Lewis y Patrick Massot.

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 desarrolla como un proyecto en GitHub que contiene libro en PDF. Además, los vídeos correspondientes a cada uno de los ejercicios se encuentran en YouTube.

Capítulo 2

Lógica proposicional

2.1. Reglas del condicional

2.1.1. Regla de eliminación del condicional en P → Q, P ⊢ Q

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

```
-- Eliminación del condicional en Lean
-- ===================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--    (P → Q), P ⊢ Q.
-- ----------------------------------------------------

import tactic
variables (P Q : Prop)

-- 1ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
begin
  apply h1,
  exact h2,
end

-- 2ª demostración
```

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

-- 3ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
-- by library_search
by exact h1 h2

-- 4ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
h1 h2

-- 5ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
-- by hint
by tauto

-- 6ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by finish

-- 7ª demostración
example
  (h1 : P → Q)
  (h2 : P)
  : Q :=
by solve_by_elim
```

2.1.2. Pruebas de P, P → Q, P → (Q → R) ⊢ R

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

```
-- Pruebas de P, P → Q, P → (Q → R) ⊢ R
-- ====================================

-- ----------------------------------------------------
-- Ej 1. (p. 6) Demostrar que
--    P, P → Q, P → (Q → R) ⊢ R
-- ----------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
have h4 : Q,
  from h2 h1,
have h5 : Q → R,
  from h3 h1,
show R,
  from h5 h4

-- 2ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
have h4 : Q      := h2 h1,
have h5 : Q → R := h3 h1,
show R, from h5 h4

-- 3ª demostración
example
  (h1 : P)
```

```
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
show R, from (h3 h1) (h2 h1)

-- 4ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
(h3 h1) (h2 h1)

-- 5ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
-- by hint
by finish

-- 6ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
begin
  apply h3,
  { exact h1, },
  { apply h2,
    exact h1, },
end

-- 7ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
begin
  apply h3,
  { exact h1, },
  { exact h2 h1, },
```

```
end

-- 7ª demostración
example
  (h1 : P)
  (h2 : P → Q)
  (h3 : P → (Q → R))
  : R :=
begin
  { exact (h3 h1) (h2 h1), },
end
```

2.1.3. Regla de introducción del condicional en P → P

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

```
-- Introducción del condicional en Lean
-- ====================================

-- ----------------------------------------------------
-- Ej. 1. (p. 9) Demostrar que
--    P → P
-- ----------------------------------------------------

import tactic
variable (P : Prop)

-- 1ª demostración
example : P → P :=
assume h : P,
show P, from h

-- 2ª demostración
example : P → P :=
assume : P,
show P, from this

-- 3ª demostración
example : P → P :=
assume : P,
show P, from ‹P›

-- 4ª demostración
```

```
example : P → P :=
assume h : P, h

-- 5ª demostración
example : P → P :=
λ h, h

-- 6ª demostración
example : P → P :=
-- by library_search
id

-- 7ª demostración
example : P → P :=
begin
  intro h,
  exact h,
end

-- 8ª demostración
example : P → P :=
begin
  intro,
  exact ‹P›,
end

-- 9ª demostración
example : P → P :=
begin
  intro h,
  assumption,
end

-- 10ª demostración
example : P → P :=
begin
  intro,
  assumption,
end

-- 11ª demostración
example : P → P :=
-- by hint
by tauto
```

```
-- 12ª demostración
example : P → P :=
by finish

-- 13ª demostración
example : P → P :=
by simp
```

2.1.4. Pruebas de P → (Q → P)

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

```
-- Pruebas de P → (Q → P)
-- ======================

-- ----------------------------------------------------
-- Ej. 1. (p. 13) Demostrar
--    P → (Q → P)
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example : P → (Q → P) :=
assume (h1 : P),
show Q → P, from
  ( assume h2 : Q,
    show P, from h1)

-- 2ª demostración
example : P → (Q → P) :=
assume (h1 : P),
show Q → P, from
  ( assume h2 : Q, h1)

-- 3ª demostración
example : P → (Q → P) :=
assume (h1 : P),
show Q → P, from
  (λ h2, h1)
```

```
-- 4ª demostración
example : P → (Q → P) :=
assume (h1 : P), (λ h2, h1)

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

-- 6ª demostración
example : P → (Q → P) :=
λ h1 h2, h1

-- 7ª demostración
example : P → (Q → P) :=
λ h _, h

-- 8ª demostración
example : P → (Q → P) :=
-- by library_search
imp_intro

-- 9ª demostración
example : P → (Q → P) :=
begin
  intro h1,
  intro h2,
  exact h1,
end

-- 10ª demostración
example : P → (Q → P) :=
begin
  intros h1 h2,
  exact h1,
end

-- 11ª demostración
example : P → (Q → P) :=
λ h1 h2, h1

-- 12ª demostración
example : P → (Q → P) :=
-- by hint
by tauto
```

```
-- 13ª demostración
example : P → (Q → P) :=
by finish
```

2.1.5. Pruebas del silogismo hipotético: P → Q, Q → R ⊢ P → R

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

```
-- Pruebas del silogismo hipotético
-- ================================

import tactic

variables (P Q R : Prop)

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--    P → Q, Q → R ⊢ P → R
-- ----------------------------------------------------

-- 1º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
assume h : P,
have h3 : Q,
  from h1 h,
show R,
  from h2 h3

variable (h1 : P → Q)
variable (h2 : Q → R)
variable (h : P)
#check h1 h
#check h2 (h1 h)

-- 2º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
```

```
  : P → R :=
assume h : P,
have h3 : Q,
  from h1 h,
h2 h3

-- 3º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
assume h : P,
h2 (h1 h)

-- 4º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
λ h, h2 (h1 h)

-- 5º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
h2 ∘ h1

-- 6º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
begin
  intro h,
  apply h2,
  apply h1,
  exact h,
end

-- 7º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
```

```
begin
  intro h,
  apply h2,
  exact h1 h,
end

-- 8º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
begin
  intro h,
  exact h2 (h1 h),
end

-- 9º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
λ h, h2 (h1 h)

-- 10º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
-- by library_search
h2 ∘ h1

-- 11º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
-- by hint
by tauto

-- 12º demostración
example
  (h1 : P → Q)
  (h2 : Q → R)
  : P → R :=
by finish
```

2.2. Reglas de la conjunción

2.2.1. Reglas de la conjunción en P ∧ Q, R ⊢ Q ∧ R

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

```
-- Reglas de la conjunción
-- =======================

-- ----------------------------------------------------
-- Ej. 1 (p. 4). Demostrar que
--    P ∧ Q, R ⊢ Q ∧ R
-- ----------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
have hQ : Q,
  from and.right hPQ,
show Q ∧ R,
  from and.intro hQ hR

-- 2ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
have hQ : Q,
  from hPQ.right,
show Q ∧ R,
  from ⟨hQ, hR⟩

-- 3ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
have hQ : Q,
```

```
  from hPQ.2,
show Q ∧ R,
  from ⟨hQ, hR⟩

-- 4ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
have hQ : Q,
  from hPQ.2,
⟨hQ, hR⟩

-- 5ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
⟨hPQ.2, hR⟩

-- 6ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
match hPQ with ⟨hP, hQ⟩ :=
  and.intro hQ hR
end

-- 7ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
begin
  split,
  { cases hPQ with hP hQ,
    exact hQ, },
  { exact hR, },
end

-- 9ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
```

```
  : Q ∧ R :=
begin
  split,
  { cases hPQ,
    assumption, },
  { assumption, },
end

-- 10ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
begin
  constructor,
  { cases hPQ,
    assumption, },
  { assumption, },
end

-- 11ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
-- by hint
by tauto

-- 12ª demostración
example
  (hPQ : P ∧ Q)
  (hR : R)
  : Q ∧ R :=
by finish
```

2.2.2. Pruebas de P ∧ Q → Q ∧ P

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

```
-- Pruebas en Lean de P ∧ Q → Q ∧ P
-- ================================

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

```
-- Ej. 1. Demostrar que
--    P ∧ Q → Q ∧ P
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q,
have hP : P,
  from and.left h,
have hQ : Q,
  from and.right h,
show Q ∧ P,
  from and.intro hQ hP

-- 2ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q,
have hP : P,
  from h.left,
have hQ : Q,
  from h.right,
show Q ∧ P,
  from ⟨hQ, hP⟩

-- 3ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q,
have hP : P,
  from h.1,
have hQ : Q,
  from h.2,
show Q ∧ P,
  from ⟨hQ, hP⟩

-- 4ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q,
have hP : P := h.1,
have hQ : Q := h.2,
show Q ∧ P,
  from ⟨hQ, hP⟩
```

```
-- 5ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q,
show Q ∧ P,
  from ⟨h.2, h.1⟩

-- 6ª demostración
example : P ∧ Q → Q ∧ P :=
assume h : P ∧ Q, ⟨h.2, h.1⟩

-- 7ª demostración
example : P ∧ Q → Q ∧ P :=
λ h, ⟨h.2, h.1⟩

-- 8ª demostración
example : P ∧ Q → Q ∧ P :=
begin
  intro h,
  cases h with hP hQ,
  split,
  { exact hQ, },
  { exact hP, },
end

-- 9ª demostración
example : P ∧ Q → Q ∧ P :=
begin
  rintro ⟨hP, hQ⟩,
  exact ⟨hQ, hP⟩,
end

-- 10ª demostración
example : P ∧ Q → Q ∧ P :=
λ ⟨hP, hQ⟩, ⟨hQ, hP⟩

-- 11ª demostración
example : P ∧ Q → Q ∧ P :=
-- by library_search
and.comm.mp

-- 12ª demostración
example : P ∧ Q → Q ∧ P:=
-- by hint
by tauto
```

```
-- 13ª demostración
example : P ∧ Q → Q ∧ P :=
by finish
```

2.3. Reglas de la negación

2.3.1. Reglas de la negación con (⊥ ⊢ P), (P, ¬P ⊢ ⊥) y ¬(P ∧ ¬P)

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

```
-- Reglas de la negación
-- =====================

import tactic

variables (P Q : Prop)

-- Eliminación del falso
-- ---------------------

-- ----------------------------------------------------
-- Ej. 1. (p. 14) Demostrar que
--    ⊥ ⊢ P
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : false)
  : Q :=
false.elim h

-- 2ª demostración
example
  (h : false)
  : Q :=
-- by library_search
false.rec Q h

-- 3ª demostración
example
```

```
  (h : false)
  : P :=
-- by hint
by tauto

-- 4ª demostración
example
  (h : false)
  : P :=
by cases h

-- 5ª demostración
example
  (h : false)
  : P :=
by finish

-- 6ª demostración
example
  (h : false)
  : P :=
by solve_by_elim

-- Definición de la negación
-- -------------------------

-- #reduce ¬P
-- ¬P ≡ (P → false)

-- Eliminación de la negación
-- --------------------------

-- ----------------------------------------------------
-- Ej. 2. Demostrar que
--    P, ¬P ⊢ ⊥
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1: P)
  (h2: ¬P)
  : false :=
not.elim h2 h1

-- 2ª demostración
```

```
example
  (h1: P)
  (h2: ¬P)
  : false :=
-- by library_search
h2 h1

-- Introducción de la negación
-- ---------------------------

-- ----------------------------------------------------
-- Ej. 3. Demostrar
--    ¬(P ∧ ¬P)
-- ----------------------------------------------------

-- 1ª demostración
example : ¬(P ∧ ¬P) :=
not.intro
  ( assume h : P ∧ ¬P,
    have h1 : P  := h.1,
    have h2 : ¬P := h.2,
    show false, from h2 h1 )

-- 2ª demostración
example : ¬(P ∧ ¬P) :=
not.intro
  ( assume h : P ∧ ¬P,
    show false, from h.2 h.1 )

-- 3ª demostración
example : ¬(P ∧ ¬P) :=
not.intro
  ( assume h : P ∧ ¬P, h.2 h.1 )

-- 4ª demostración
example : ¬(P ∧ ¬P) :=
not.intro (λ h, h.2 h.1)

-- 5ª demostración
example : ¬(P ∧ ¬P) :=
begin
  intro h,
  cases h with h1 h2,
  apply h2,
  exact h1,
```

```
end

-- 6ª demostración
example : ¬(P ∧ ¬P) :=
begin
  rintro ⟨h1, h2⟩,
  exact h2 h1,
end

-- 7ª demostración
example : ¬(P ∧ ¬P) :=
λ ⟨h1, h2⟩, h2 h1

-- 8ª demostración
example : ¬(P ∧ ¬P) :=
-- by suggest
(and_not_self P).mp

-- 9ª demostración
example : ¬(P ∧ ¬P) :=
-- by hint
by tauto

-- 10ª demostración
example : ¬(P ∧ ¬P) :=
by finish

-- 11ª demostración
example : ¬(P ∧ ¬P) :=
by simp
```

2.3.2. Pruebas de P → Q, P → ¬Q ⊢ ¬P

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

```
-- Pruebas de P → Q, P → ¬Q ⊢ ¬P
-- =============================

import tactic

variables (P Q : Prop)

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

```
-- Ej. 1. (p. 16) Demostrar
--     P → Q, P → ¬Q ⊢ ¬P
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
assume h : P,
have h4 : Q,
  from h1 h,
have h5 : ¬Q,
  from h2 h,
show false,
  from h5 h4

-- 2ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
assume h : P,
have h4 : Q  := h1 h,
have h5 : ¬Q := h2 h,
show false,
  from h5 h4

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

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

-- 5ª demostración
```

```
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
λ h, (h2 h) (h1 h)

-- 6ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
begin
  intro h,
  have h3 : ¬Q := h2 h,
  apply h3,
  apply h1,
  exact h,
end

-- 7ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
begin
  intro h,
  have h3 : ¬Q := h2 h,
  apply h3,
  exact h1 h,
end

-- 8ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
begin
  intro h,
  have h3 : ¬Q := h2 h,
  exact h3 (h1 h),
end

-- 9ª demostración
example
  (h1 : P → Q)
```

```
  (h2 : P → ¬Q)
  : ¬P :=
begin
  intro h,
  exact (h2 h) (h1 h),
end

-- 10ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
λ h, (h2 h) (h1 h)

-- 11ª demostración
example
  (h1 : P → Q)
  (h2 : P → ¬Q)
  : ¬P :=
-- by hint
by finish
```

2.3.3. Pruebas del modus tollens: P → Q, ¬Q ⊢ ¬P

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

```
-- Pruebas del modus tollens
-- =========================

-- ----------------------------------------------------
-- Ej. 1. (p. 20) Demostrar
--    P → Q, ¬Q ⊢ ¬P
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
```

```
assume h3 : P,
have h4 : Q,
  from h1 h3,
show false,
  from h2 h4

-- 2ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
assume h3 : P,
have h4 : Q := h1 h3,
show false,
  from h2 h4

-- 3ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
assume h3 : P,
show false,
  from h2 (h1 h3)

-- 4ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
assume h3 : P, h2 (h1 h3)

-- 5ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
λ h3, h2 (h1 h3)

-- 6ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
h2 ∘ h1
```

```
-- 7ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
-- by library_search
mt h1 h2

-- 8ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
begin
  intro h,
  apply h2,
  apply h1,
  exact h,
end

-- 9ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
begin
  intro h,
  exact h2 (h1 h),
end

-- 10ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
λ h, h2 (h1 h)

-- 11ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
-- by hint
by tauto
```

```
-- 12ª demostración
example
  (h1 : P → Q)
  (h2 : ¬Q)
  : ¬P :=
by finish
```

2.3.4. Pruebas de P → (Q → R), P, ¬R ⊢ ¬Q

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

```
-- Pruebas de P → (Q → R), P, ¬R ⊢ ¬Q
-- ==================================

-- ----------------------------------------------------
-- Ej. 1 (p. 7). Demostrar
--    P → (Q → R), P, ¬R ⊢ ¬Q
-- ----------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
have h4 : Q → R,
  from h1 h2,
show ¬Q,
  from mt h4 h3

-- 2ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
have h4 : Q → R := h1 h2,
show ¬Q,
```

```
  from mt h4 h3

-- 3ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
show ¬Q,
  from mt (h1 h2) h3

-- 4ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
-- by library_search
mt (h1 h2) h3

-- 5ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
begin
  intro h4,
  apply h3,
  apply (h1 h2),
  exact h4,
end

-- 6ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
begin
  intro h4,
  apply h3,
  exact (h1 h2) h4,
end
```

```
-- 7ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
begin
  intro h4,
  exact h3 ((h1 h2) h4),
end

-- 8ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
λ h4, h3 ((h1 h2) h4)

-- 9ª demostración
example
  (h1 : P → (Q → R))
  (h2 : P)
  (h3 : ¬R)
  : ¬Q :=
-- by hint
by finish
```

2.3.5. Pruebas de $P \to Q \vdash \neg Q \to \neg P$

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

```
-- Pruebas de P → Q ⊢ ¬Q → ¬P
-- ==========================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    P → Q ⊢ ¬Q → ¬P
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)
```

```
-- 1ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
assume h2 : ¬Q,
show ¬P,
  from mt h1 h2

-- 2ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
assume h2 : ¬Q, mt h1 h2

-- 3ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
λ h2, mt h1 h2

-- 4ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
-- by library_search
mt h1

-- 5ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
begin
  intro h2,
  exact mt h1 h2,
end

-- 6ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
begin
  intro h2,
  intro h3,
  apply h2,
```

```
  apply h1,
  exact h3,
end

-- 7ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
begin
  intro h2,
  intro h3,
  apply h2,
  exact h1 h3,
end

-- 8ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
begin
  intro h2,
  intro h3,
  exact h2 (h1 h3),
end

-- 9ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
begin
  intros h2 h3,
  exact h2 (h1 h3),
end

-- 10ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
λ h2 h3, h2 (h1 h3)

-- 11ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
-- by hint
```

```
by tauto

-- 12ª demostración
example
  (h1 : P → Q)
  : ¬Q → ¬P :=
by finish
```

2.3.6. Regla de introducción de la doble negación: P ⊢ ¬¬P

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

```
-- Regla de introducción de la doble negación
-- ==========================================

-- ----------------------------------------------------
-- Ej. 1. (p. 21) Demostrar
--    P ⊢ ¬¬P
-- ----------------------------------------------------

import tactic
variable (P : Prop)

-- 1ª demostración
example
  (h1 : P)
  : ¬¬P :=
not.intro
  ( assume h2: ¬P,
    show false,
      from h2 h1)

-- 2ª demostración
example
  (h1 : P)
  : ¬¬P :=
assume h2: ¬P,
show false,
  from h2 h1

-- 3ª demostración
```

```
example
  (h1 : P)
  : ¬¬P :=
assume h2: ¬P, h2 h1

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

-- 5ª demostración
example
  (h1 : P)
  : ¬¬P :=
not_not.mpr h1

-- 6ª demostración
example
  (h1 : P)
  : ¬¬P :=
-- by library_search
not_not_intro h1

-- 7ª demostración
example
  (h1 : P)
  : ¬¬P :=
begin
  intro h2,
  exact h2 h1,
end

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

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

2.3.7. Pruebas de ¬Q → ¬P ⊢ P → ¬¬Q

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

```
-- Pruebas de ¬Q → ¬P ⊢ P → ¬¬Q
-- ============================

-- ----------------------------------------------------
-- Ej. 1. (p. 9) Demostrar
--    ¬Q → ¬P ⊢ P → ¬¬Q
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
assume h2 : P,
have h3 : ¬¬P,
  from not_not_intro h2,
show ¬¬Q,
  from mt h1 h3

-- 2ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
assume h2 : P,
have h3 : ¬¬P := not_not_intro h2,
show ¬¬Q,
  from mt h1 h3

-- 3ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
assume h2 : P,
show ¬¬Q,
  from mt h1 (not_not_intro h2)

-- 4ª demostración
example
```

```
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
assume h2 : P, mt h1 (not_not_intro h2)

-- 5ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
λ h2, mt h1 (not_not_intro h2)

-- 6ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
-- by library_search
imp_not_comm.mp h1

-- 7ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
begin
  intro h2,
  apply mt h1,
  apply not_not_intro,
  exact h2,
end

-- 8ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
begin
  intro h2,
  apply mt h1,
  exact not_not_intro h2,
end

-- 9ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
begin
  intro h2,
  exact mt h1 (not_not_intro h2),
```

```
end

-- 10ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
λ h2, mt h1 (not_not_intro h2)

-- 11ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
begin
  intro h2,
  intro h3,
  have h4 : ¬P := h1 h3,
  exact h4 h2,
end

-- 12ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
begin
  intros h2 h3,
  exact (h1 h3) h2,
end

-- 13ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
λ h2 h3, (h1 h3) h2

-- 14ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
-- by hint
by tauto

-- 15ª demostración
example
  (h1 : ¬Q → ¬P)
  : P → ¬¬Q :=
```

```
by finish
```

2.4. Reglas de la disyunción

2.4.1. Reglas de introducción de la disyunción

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

```
-- Reglas de introducción de la disyunción
-- ========================================

import tactic

variables (P Q R : Prop)

-- ------------------------------------------------------
-- Ej. 1. (p. 11) Demostrar
--    P ⊢ P ∨ Q
-- ------------------------------------------------------

-- 1ª demostración
example
  (h : P)
  : P ∨ Q :=
or.intro_left Q h

-- 2ª demostración
example
  (h : P)
  : P ∨ Q :=
-- by library_search
or.inl h

-- 3ª demostración
example
  (h : P)
  : P ∨ Q :=
-- by hint
by tauto

-- 4ª demostración
example
```

```
  (h : P)
  : P ∨ Q :=
by finish

-- ----------------------------------------------------
-- Ej. 2. Demostrar
--    P ∧ Q ⊢ P ∨ R
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
have h2 : P,
  from and.elim_left h1,
show P ∨ R,
  from or.inl h2

-- 2ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
have h2 : P,
  from h1.1,
show P ∨ R,
  from or.inl h2

-- 3ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
have h2 : P := h1.1,
show P ∨ R,
  from or.inl h2

-- 4ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
show P ∨ R,
  from or.inl h1.1

-- 5ª demostración
example
  (h1 : P ∧ Q)
```

```
  : P ∨ R :=
-- by suggest
or.inl h1.1

-- 6ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
-- by hint
by tauto

-- 7ª demostración
example
  (h1 : P ∧ Q)
  : P ∨ R :=
by finish

-- ----------------------------------------------------
-- Ej. 3. Demostrar
--    Q ⊢ P ∨ Q
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : Q)
  : P ∨ Q :=
or.intro_right P h

-- 2ª demostración
example
  (h : Q)
  : P ∨ Q :=
-- by suggest
or.inr h

-- 3ª demostración
example
  (h : Q)
  : P ∨ Q :=
-- by hint
by tauto

-- 4ª demostración
example
  (h : Q)
```

```
  : P ∨ Q :=
by finish

-- ----------------------------------------------------
-- Ej. 4. Demostrar
--    P ∧ Q ⊢ R ∨ Q
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
have h2 : Q,
  from and.elim_right h1,
show R ∨ Q,
  from or.inr h2

-- 2ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
have h2 : Q,
  from h1.2,
show R ∨ Q,
  from or.inr h2

-- 3ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
have h2 : Q := h1.2,
show R ∨ Q,
  from or.inr h2

-- 4ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
show R ∨ Q,
  from or.inr h1.2

-- 5ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
```

```
-- by suggest
or.inr h1.2

-- 6ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
-- by hint
by tauto

-- 7ª demostración
example
  (h1 : P ∧ Q)
  : R ∨ Q :=
by finish
```

2.4.2. Regla de eliminación de la disyunción

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

```
-- Regla de eliminación de la disyunción
-- =====================================

import tactic

variables (P Q R : Prop)

-- ----------------------------------------------------
-- Ej. 1. (p. 11) Demostrar
--    P ∨ Q, P → R, Q → R ⊢ R
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : P ∨ Q)
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
or.elim h1 h2 h3

-- 2ª demostración
example
  (h1 : P ∨ Q)
```

```
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
h1.elim h2 h3

-- 3ª demostración
example
  (h1 : P ∨ Q)
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
-- by library_search
or.rec h2 h3 h1

-- 4ª demostración
example
  (h1 : P ∨ Q)
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
begin
  cases h1 with hP hQ,
  { exact h2 hP, },
  { exact h3 hQ, },
end

-- 5ª demostración
example
  (h1 : P ∨ Q)
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
-- by hint
by tauto

-- 6ª demostración
example
  (h1 : P ∨ Q)
  (h2 : P → R)
  (h3 : Q → R)
  : R :=
by finish
```

2.4.3. Pruebas de P ∨ Q ⊢ Q ∨ P

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

```
-- Pruebas de P ∨ Q ⊢ Q ∨ P
-- ========================

import tactic

variables (P Q R : Prop)

-- ----------------------------------------------------
-- Ej. 1. (p. 11) Demostrar
--    P ∨ Q ⊢ Q ∨ P
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
or.elim h1
  ( assume h2 : P,
    show Q ∨ P,
      from or.inr h2 )
  ( assume h3 : Q,
    show Q ∨ P,
      from or.inl h3 )

-- 2ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
or.elim h1
  ( λ h, or.inr h )
  ( λ h, or.inl h )

-- 3ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
or.elim h1 or.inr or.inl

-- 4ª demostración
example
  (h1 : P ∨ Q)
```

```
  : Q ∨ P :=
h1.elim or.inr or.inl

-- 5ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
or.rec or.inr or.inl h1

-- 6ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
-- by library_search
or.swap h1

-- 7ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
begin
  cases h1 with h2 h3,
  { exact or.inr h2, },
  { exact or.inl h3, },
end

-- 7ª demostración
example
  (P ∨ Q)
  : Q ∨ P :=
begin
  cases ‹P ∨ Q›,
  { exact or.inr ‹P›, },
  { exact or.inl ‹Q›, },
end

-- 8ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
begin
  cases h1 with h2 h3,
  { right,
    exact h2, },
```

```
  { left,
    exact h3, },
end

-- 9ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
-- by hint
by tauto

-- 10ª demostración
example
  (h1 : P ∨ Q)
  : Q ∨ P :=
by finish

-- ----------------------------------------------------
-- Ej. 2 (p. 12). Demostrar
--    Q → R ⊢ P ∨ Q → P ∨ R
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P,
    show P ∨ R,
      from or.inl h3 )
  ( assume h4 : Q,
    have h5 : R := h1 h4,
    show P ∨ R,
      from or.inr h5 )

-- 2ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P, or.inl h3 )
  ( assume h4 : Q,
    show P ∨ R,
```

```
      from or.inr (h1 h4) )

-- 3ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P, or.inl h3 )
  ( assume h4 : Q, or.inr (h1 h4) )

-- 4ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( λ h3, or.inl h3 )
  ( λ h4, or.inr (h1 h4) )

-- 5ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  or.inl
  (λ h, or.inr (h1 h) )

-- 6ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
λ h2, or.elim h2 or.inl (λ h, or.inr (h1 h))

-- 7ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
λ h2, or.elim h2 or.inl (or.inr ∘ h1)

-- 8ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
```

```
λ h2, h2.elim or.inl (or.inr ∘ h1)

-- 9ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
-- by library_search
or.imp_right h1

-- 10ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  intro h2,
  cases h2 with h3 h4,
  { exact or.inl h3, },
  { exact or.inr (h1 h4), },
end

-- 11ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  intro h2,
  cases h2 with h3 h4,
  { left,
    exact h3, },
  { right,
    exact (h1 h4), },
end

-- 12ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  rintro (h3 | h4),
  { left,
    exact h3, },
  { right,
    exact (h1 h4), },
end
```

```
-- 13ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
-- by hint
by tauto

-- 14ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
by finish

-- ----------------------------------------------------
-- Ej. 4 (p. 15). Demostrar
--     ¬P ∨ Q ⊢ P → Q
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( assume h5 : Q,
    show Q, from h5)

-- 2ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( assume h5 : Q, h5)
```

```
-- 3ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( λ h5, h5)

-- 4ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  id

-- 5ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    show Q,
      from false.elim (h3 h2))
  id

-- 6ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P, false.elim (h3 h2))
  id
```

```
-- 7ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( λ h3, false.elim (h3 h2))
  id

-- 8ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, or.elim h1 (λ h3, false.elim (h3 h2)) id

example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, h1.elim (λ h3, false.elim (h3 h2)) id

example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, h1.elim (λ h3, (h3 h2).elim) id

-- 9ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
-- by library_search
imp_iff_not_or.mpr h1

-- 10ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { apply false.rec,
    exact h3 h2, },
  { exact h4, },
end
```

```
-- 11ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { exact false.elim (h3 h2), },
  { exact h4, },
end

-- 12ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { exfalso,
    exact h3 h2, },
  { exact h4, },
end

-- 13ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
-- by hint
by tauto

-- 14ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
by finish
```

2.4.4. Pruebas de Q → R ⊢ P ∨ Q → P ∨ R

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

```
-- Pruebas de Q → R ⊢ P ∨ Q → P ∨ R
-- ================================
```

```
-- ----------------------------------------------------
-- Ej. 1. (p. 12) Demostrar
--     Q → R ⊢ P ∨ Q → P ∨ R
-- ----------------------------------------------------

import tactic

variables (P Q R : Prop)

-- 1ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P,
    show P ∨ R,
      from or.inl h3 )
  ( assume h4 : Q,
    have h5 : R := h1 h4,
    show P ∨ R,
      from or.inr h5 )

-- 2ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P, or.inl h3 )
  ( assume h4 : Q,
    show P ∨ R,
      from or.inr (h1 h4) )

-- 3ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( assume h3 : P, or.inl h3 )
  ( assume h4 : Q, or.inr (h1 h4) )

-- 4ª demostración
example
```

```
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  ( λ h3, or.inl h3 )
  ( λ h4, or.inr (h1 h4) )

-- 5ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
assume h2 : P ∨ Q,
or.elim h2
  or.inl
  ( λ h, or.inr (h1 h) )

-- 6ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
λ h2, or.elim h2 or.inl (λ h, or.inr (h1 h))

example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
λ h2, h2.elim or.inl (λ h, or.inr (h1 h))

example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
λ h2, h2.elim or.inl (or.inr ∘ h1)

-- 7ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
-- by library_search
or.imp_right h1

-- 8ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  intro h2,
```

```
  cases h2 with h3 h4,
  { exact or.inl h3, },
  { exact or.inr (h1 h4), },
end

-- 9ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  intro h2,
  cases h2 with h3 h4,
  { left,
    exact h3, },
  { right,
    exact (h1 h4), },
end

-- 10ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
begin
  rintro (h3 | h4),
  { left,
    exact h3, },
  { right,
    exact (h1 h4), },
end

-- 11ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
-- by hint
by tauto

-- 12ª demostración
example
  (h1 : Q → R)
  : P ∨ Q → P ∨ R :=
by finish
```

2.4.5. Pruebas de ¬P ∨ Q ⊢ P → Q

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

```
-- Prueba de ¬P ∨ Q ⊢ P → Q
-- ========================

-- ----------------------------------------------------
-- Ej. 1. (p. 15) Demostrar
--    ¬P ∨ Q ⊢ P → Q
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( assume h5 : Q,
    show Q, from h5)

-- 2ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( assume h5 : Q, h5)

-- 3ª demostración
example
```

```
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  ( λ h5, h5)

-- 4ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    have h4 : false,
      from h3 h2,
    show Q,
      from false.elim h4)
  id

-- 5ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P,
    show Q,
      from false.elim (h3 h2))
  id

-- 6ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( assume h3 : ¬P, false.elim (h3 h2))
  id

-- 7ª demostración
```

```
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
assume h2 : P,
or.elim h1
  ( λ h3, false.elim (h3 h2))
  id

-- 8ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, or.elim h1 (λ h3, false.elim (h3 h2)) id

example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, h1.elim (λ h3, false.elim (h3 h2)) id

example
  (h1 : ¬P ∨ Q)
  : P → Q :=
λ h2, h1.elim (λ h3, (h3 h2).elim) id

-- 9ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
-- by library_search
imp_iff_not_or.mpr h1

-- 10ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { apply false.rec,
    exact h3 h2, },
  { exact h4, },
end

-- 11ª demostración
example
```

```
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { exact false.elim (h3 h2), },
  { exact h4, },
end

-- 12ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
begin
  intro h2,
  cases h1 with h3 h4,
  { exfalso,
    exact h3 h2, },
  { exact h4, },
end

-- 13ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
-- by hint
by tauto

-- 14ª demostración
example
  (h1 : ¬P ∨ Q)
  : P → Q :=
by finish
```

2.5. Reglas del bicondicional

2.5.1. Regla de introducción del bicondicional en P ∧ Q ↔ Q ∧ P

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

```
-- Regla de introducción del bicondicional
-- =======================================

-- ----------------------------------------------------
-- Ej. 1. (p. 17) Demostrar
--    P ∧ Q ↔ Q ∧ P
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

-- 1ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q,
    have h2 : P,
      from and.elim_left h1,
    have h3 : Q,
      from and.elim_right h1,
    show Q ∧ P,
      from and.intro h3 h2)
  ( assume h4 : Q ∧ P,
    have h5 : Q,
      from and.elim_left h4,
    have h6 : P,
      from and.elim_right h4,
    show P ∧ Q,
      from and.intro h6 h5)

-- 2ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q,
    have h2 : P,
      from h1.1,
    have h3 : Q,
      from h1.2,
    show Q ∧ P,
      from and.intro h3 h2)
  ( assume h4 : Q ∧ P,
    have h5 : Q,
      from h4.1,
    have h6 : P,
      from h4.2,
```

```
    show P ∧ Q,
      from and.intro h6 h5)

-- 3ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q,
    have h2 : P := h1.1,
    have h3 : Q := h1.2,
    show Q ∧ P,
      from and.intro h3 h2)
  ( assume h4 : Q ∧ P,
    have h5 : Q := h4.1,
    have h6 : P := h4.2,
    show P ∧ Q,
      from and.intro h6 h5)

-- 4ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q,
    show Q ∧ P,
      from and.intro h1.2 h1.1)
  ( assume h4 : Q ∧ P,
    show P ∧ Q,
      from and.intro h4.2 h4.1)

-- 5ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q, and.intro h1.2 h1.1)
  ( assume h4 : Q ∧ P, and.intro h4.2 h4.1)

-- 6ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( assume h1 : P ∧ Q, ⟨h1.2, h1.1⟩)
  ( assume h4 : Q ∧ P, ⟨h4.2, h4.1⟩)

-- 7ª demostración
example : P ∧ Q ↔ Q ∧ P :=
iff.intro
  ( λ h, ⟨h.2, h.1⟩)
  ( λ h, ⟨h.2, h.1⟩)
```

```
-- 8ª demostración
lemma aux :
  P ∧ Q → Q ∧ P :=
λ h, ⟨h.2, h.1⟩

example : P ∧ Q ↔ Q ∧ P :=
iff.intro (aux P Q) (aux Q P)

-- 9ª demostración
example : P ∧ Q ↔ Q ∧ P :=
-- by library_search
and.comm

-- 10ª demostración
example : P ∧ Q ↔ Q ∧ P :=
begin
  split,
  { intro h1,
    cases h1 with h2 h3,
    split,
    { exact h3, },
    { exact h2, }},
  { intro h4,
    cases h4 with h5 h6,
    split,
    { exact h6, },
    { exact h5, }},
end

-- 11ª demostración
example : P ∧ Q ↔ Q ∧ P :=
begin
  split,
  { rintro ⟨h2, h3⟩,
    split,
    { exact h3, },
    { exact h2, }},
  { rintro ⟨h5, h6⟩,
    split,
    { exact h6, },
    { exact h5, }},
end

-- 12ª demostración
example : P ∧ Q ↔ Q ∧ P :=
```

```
begin
  constructor,
  { rintro ⟨h2, h3⟩,
    constructor,
    { exact h3, },
    { exact h2, }},
  { rintro ⟨h5, h6⟩,
    constructor,
    { exact h6, },
    { exact h5, }},
end

-- 13ª demostración
example : P ∧ Q ↔ Q ∧ P :=
-- by hint
by tauto

-- 14ª demostración
example : P ∧ Q ↔ Q ∧ P :=
by finish
```

2.5.2. Reglas de eliminación del bicondicional en $P \leftrightarrow Q$, $P \lor Q \vdash P \land Q$

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

```
-- Reglas de eliminacion del condicional
-- =====================================

-- ----------------------------------------------------
-- Ej. 1. (p. 18) Demostrar
--     P ↔ Q, P ∨ Q ⊢ P ∧ Q
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

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

```
  : P ∧ Q :=
or.elim h2
  ( assume h3 : P,
    have h4 : P → Q,
      from iff.elim_left h1,
    have h5 : Q,
      from h4 h3,
    show P ∧ Q,
      from and.intro h3 h5 )
  ( assume h6 : Q,
    have h7 : Q → P,
      from iff.elim_right h1,
    have h8 : P,
      from h7 h6,
    show P ∧ Q,
      from and.intro h8 h6 )

-- 2ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
or.elim h2
  ( assume h3 : P,
    have h4 : P → Q := h1.1,
    have h5 : Q := h4 h3,
    show P ∧ Q, from ⟨h3, h5⟩ )
  ( assume h6 : Q,
    have h7 : Q → P := h1.2,
    have h8 : P := h7 h6,
    show P ∧ Q, from ⟨h8, h6⟩ )

-- 3ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
or.elim h2
  ( assume h3 : P,
    show P ∧ Q, from ⟨h3, (h1.1 h3)⟩ )
  ( assume h6 : Q,
    show P ∧ Q, from ⟨h1.2 h6, h6⟩ )

-- 4ª demostración
example
```

```
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
or.elim h2
  (λh, ⟨h, (h1.1 h)⟩)
  (λh, ⟨h1.2 h, h⟩)

example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
h2.elim
  (λh, ⟨h, (h1.1 h)⟩)
  (λh, ⟨h1.2 h, h⟩)

-- 5ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
begin
  cases h2 with h3 h4,
  { split,
    { exact h3, },
    { apply h1.mp,
      exact h3, }},
  { split,
    { apply h1.mpr,
      exact h4, },
    { exact h4, }},
end

-- 6ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
begin
  cases h2 with h3 h4,
  { split,
    { exact h3, },
    { rw ← h1,
      exact h3, }},
  { split,
    { rw h1,
```

```
      exact h4, },
    { exact h4, }},
end

-- 7ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
-- by hint
by tauto

-- 8ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
by finish

-- 9ª demostración
example
  (h1 : P ↔ Q)
  (h2 : P ∨ Q)
  : P ∧ Q :=
begin
  simp [h1] at h2 |-,
  assumption,
end
```

2.6. Reglas de la lógica clásica

2.6.1. Pruebas de la regla de reducción al absurdo

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

```
-- Prueba de la regla de reducción al absurdo
-- ==========================================

import tactic

variable (P : Prop)
```

```
-- ----------------------------------------------------
-- Ej. 1 (p. 22) Demostrar que
--    ¬P → false ⊢ P
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : ¬P → false)
  : P :=
have h2 : ¬¬P, from
  assume h3 : ¬P,
  show false, from h1 h3,
show P, from not_not.mp h2

-- 2ª demostración
example
  (h1 : ¬P → false)
  : P :=
begin
  apply not_not.mp,
  intro h2,
  exact h1 h2,
end

-- 3ª demostración
example
  (h1 : ¬P → false)
  : P :=
begin
  apply not_not.mp,
  exact λ h2, h1 h2,
end

-- 4ª demostración
example
  (h1 : ¬P → false)
  : P :=
not_not.mp (λ h2, h1 h2)

-- #print axioms not_not

-- 5ª demostración
example
  (h1 : ¬P → false)
  : P :=
```

```
-- by library_search
by_contra h1

-- #print axioms by_contra

-- 6ª demostración
lemma RAA
  (h1 : ¬P → false)
  : P :=
-- by hint
by finish

-- #print axioms RAA
```

2.6.2. Pruebas de la eliminación de la doble negación

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

```
-- Pruebas de la eliminación de la doble negación
-- ==============================================

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

import tactic

variable (P : Prop)

open_locale classical

-- 1ª demostración
example
  (h1 : ¬¬P)
  : P :=
by_contra
  ( assume h2 : ¬P,
    show false,
      from h1 h2 )

-- 2ª demostración
example
```

```
  (h1 : ¬¬P)
  : P :=
by_contra
  ( assume h2 : ¬P,
    h1 h2 )

-- 3ª demostración
example
  (h1 : ¬¬P)
  : P :=
by_contra (λ h2, h1 h2)

-- 4ª demostración
example
  (h1 : ¬¬P)
  : P :=
-- by library_search
not_not.mp h1

-- 5ª demostración
example
  (h1 : ¬¬P)
  : P :=
begin
  by_contradiction h2,
  exact h1 h2,
end

-- 6ª demostración
example
  (h1 : ¬¬P)
  : P :=
-- by hint
by tauto

-- 7ª demostración
lemma aux
  (h1 : ¬¬P)
  : P :=
by finish

-- #print axioms aux
```

2.6.3. Pruebas del principio del tercio excluso

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

```
-- Pruebas del principio del tercio excluso
-- ========================================

-- ----------------------------------------------------
-- Ej. 1. (p. 23) Demostrar que
--    F ∨ ¬F
-- ----------------------------------------------------

import tactic

variable (F : Prop)

open_locale classical

-- 1ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
      assume h3 : F,
      have h4 : F ∨ ¬F, from or.inl h3,
      show false, from h1 h4,
    have h5 : F ∨ ¬F, from or.inr h2,
    show false, from h1 h5 )

-- 2ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
      assume h3 : F,
      have h4 : F ∨ ¬F, from or.inl h3,
      show false, from h1 h4,
    have h5 : F ∨ ¬F, from or.inr h2,
    h1 h5 )

-- 3ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
```

```
      assume h3 : F,
      have h4 : F ∨ ¬F, from or.inl h3,
      show false, from h1 h4,
    h1 (or.inr h2) )

-- 4ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
      assume h3 : F,
      have h4 : F ∨ ¬F, from or.inl h3,
      h1 h4,
    h1 (or.inr h2) )

-- 5ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
      assume h3 : F,
      h1 (or.inl h3),
    h1 (or.inr h2) )

-- 6ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    have h2 : ¬F, from
      λ h3, h1 (or.inl h3),
    h1 (or.inr h2) )

-- 7ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( assume h1 : ¬(F ∨ ¬F),
    h1 (or.inr (λ h3, h1 (or.inl h3))) )

-- 8ª demostración
example : F ∨ ¬F :=
by_contradiction
  ( λ h1, h1 (or.inr (λ h3, h1 (or.inl h3))) )

-- 9ª demostración
example : F ∨ ¬F :=
```

```
-- by library_search
em F

-- #print axioms em

-- 10ª demostración
example : F ∨ ¬F :=
begin
  by_contra h1,
  apply h1,
  apply or.inr,
  intro h2,
  apply h1,
  exact or.inl h2,
end

-- 11ª demostración
example : F ∨ ¬F :=
begin
  by_contra h1,
  apply h1,
  apply or.inr,
  intro h2,
  exact h1 (or.inl h2),
end

-- 12ª demostración
example : F ∨ ¬F :=
begin
  by_contra h1,
  apply h1,
  apply or.inr,
  exact λ h2, h1 (or.inl h2),
end

-- 13ª demostración
example : F ∨ ¬F :=
begin
  by_contra h1,
  apply h1,
  exact or.inr (λ h2, h1 (or.inl h2)),
end

-- 14ª demostración
example : F ∨ ¬F :=
```

```
begin
  by_contra h1,
  exact h1 (or.inr (λ h2, h1 (or.inl h2))),
end

-- 15ª demostración
example : F ∨ ¬F :=
by_contra (λ h1, h1 (or.inr (λh2, h1 (or.inl h2))))

-- 16ª demostración
example : F ∨ ¬F :=
begin
  by_contra h1,
  apply h1,
  right,
  intro h2,
  apply h1,
  left,
  exact h2,
end

-- 17ª demostración
example : F ∨ ¬F :=
-- by hint
by tauto

-- 18ª demostración
example : F ∨ ¬F :=
by finish
```

2.6.4. Pruebas de $P \to Q \vdash \neg P \lor Q$

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

```
-- Pruebas de P → Q ⊢ ¬P ∨ Q
-- =========================

-- ------------------------------------------------------
-- Ej. 1. (p. 24) Demostrar
--    P → Q ⊢ ¬P ∨ Q
-- ------------------------------------------------------

import tactic
```

```
variables (P Q : Prop)

open_locale classical

-- 1ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( assume h3 : P,
    have h4 : Q,
      from h1 h3,
    show ¬P ∨ Q,
      from or.inr h4)
  ( assume h5 : ¬P,
    show ¬P ∨ Q,
      from or.inl h5)

-- 2ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( assume h3 : P,
    have h4 : Q,
      from h1 h3,
    show ¬P ∨ Q,
      from or.inr h4)
  ( assume h5 : ¬P,
    or.inl h5)

-- 3ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( assume h3 : P,
    have h4 : Q,
```

```
      from h1 h3,
    show ¬P ∨ Q,
      from or.inr h4)
  ( λ h5, or.inl h5)

-- 4ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( assume h3 : P,
    have h4 : Q,
      from h1 h3,
    or.inr h4)
  ( λ h5, or.inl h5)

-- 5ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( assume h3 : P,
    or.inr (h1 h3))
  ( λ h5, or.inl h5)

-- 6ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
have h2 : P ∨ ¬P,
  from em P,
or.elim h2
  ( λ h3, or.inr (h1 h3))
  ( λ h5, or.inl h5)

-- 7ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
or.elim (em P)
  ( λ h3, or.inr (h1 h3))
```

```
  ( λ h5, or.inl h5)

example
  (h1 : P → Q)
  : ¬P ∨ Q :=
(em P).elim
  ( λ h3, or.inr (h1 h3))
  ( λ h5, or.inl h5)

-- 8ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
-- by library_search
not_or_of_imp h1

-- 9ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
if h3 : P then or.inr (h1 h3) else or.inl h3

-- 10ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
begin
  by_cases h2 : P,
  { apply or.inr,
    exact h1 h2, },
  { exact or.inl h2, },
end

-- 11ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
begin
  by_cases h2 : P,
  { exact or.inr (h1 h2), },
  { exact or.inl h2, },
end

-- 12ª demostración
example
```

```
  (h1 : P → Q)
  : ¬P ∨ Q :=
begin
  by_cases h2 : P,
  { right,
    exact h1 h2, },
  { left,
    exact h2, },
end

-- 13ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
-- by hint
by tauto

-- 14ª demostración
example
  (h1 : P → Q)
  : ¬P ∨ Q :=
-- by hint
by finish
```

2.6.5. Pruebas de P, ¬¬(Q ∧ R) ⊢ ¬¬P ∧ R

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

```
-- Pruebas de P, ¬¬(Q ∧ R) ⊢ ¬¬P ∧ R
-- =================================

-- ----------------------------------------------------
-- Ej. 1. (p. 5) Demostrar
--    P, ¬¬(Q ∧ R) ⊢ ¬¬P ∧ R
-- ----------------------------------------------------

import tactic

variables (P Q R : Prop)

open_locale classical

-- 1ª demostración
```

```
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
have h3 : ¬¬P,   from not_not_intro h1,
have h4 : Q ∧ R, from not_not.mp h2,
have h5 : R,     from and.elim_right h4,
show ¬¬P ∧ R,    from and.intro h3 h5

-- 2ª demostración
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
have h3 : ¬¬P,   from not_not_intro h1,
have h4 : Q ∧ R, from not_not.mp h2,
have h5 : R,     from and.elim_right h4,
and.intro h3 h5

-- 3ª demostración
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
have h3 : ¬¬P,   from not_not_intro h1,
have h4 : Q ∧ R, from not_not.mp h2,
have h5 : R,     from h4.2,
and.intro h3 h5

-- 5ª demostración
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
and.intro (not_not_intro h1) (not_not.mp h2).2

-- 6ª demostración
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
begin
  split,
  { exact not_not_intro h1, },
  { push_neg at h2,
```

```
    exact h2.2, },
end

-- 7ª demostración
example
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
-- by hint
by tauto

-- 8ª demostración
lemma aux
  (h1 : P)
  (h2 : ¬¬(Q ∧ R))
  : ¬¬P ∧ R :=
by finish

-- #print axioms aux
```

2.6.6. Pruebas de ¬P → Q, ¬Q ⊢ P

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

```
-- Pruebas de ¬P → Q, ¬Q ⊢ P
-- =========================

-- ----------------------------------------------------
-- Ej. 1. (p. 7) Demostrar
--    ¬P → Q, ¬Q ⊢ P
-- ----------------------------------------------------

import tactic

variables (P Q : Prop)

open_locale classical

-- 1ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
```

```
have h3 : ¬¬P, from mt h1 h2,
show P,        from not_not.mp h3

-- 2ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
not_not.mp (mt h1 h2)

-- 3ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
begin
  by_contra h3,
  apply h2,
  exact h1 h3,
end

-- 4ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
begin
  by_contra h3,
  exact h2 (h1 h3),
end

-- 5ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
by_contra (λ h3, h2 (h1 h3))

-- 6ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
by_contra (λ h3, (h2 ∘ h1) h3)
```

```
-- 7ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
by_contra (h2 ∘ h1)

-- 8ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
-- by library_search
not_not.mp (mt h1 h2)

-- 9ª demostración
example
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
-- by hint
by tauto

-- 10ª demostración
lemma aux
  (h1 : ¬P → Q)
  (h2 : ¬Q)
  : P :=
-- by hint
by finish

-- #print axioms aux
```

2.6.7. Pruebas de (Q → R) → ((¬Q → ¬P) → (P → R))

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

```
-- Pruebas de (Q → R) → ((¬Q → ¬P) → (P → R))
-- ==========================================

-- ------------------------------------------------------
-- Ej. 1. (p. 10) Demostrar
--    (Q → R) → ((¬Q → ¬P) → (P → R))
```

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

import tactic

variables (P Q R : Prop)

-- 1ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      ( assume h3 : P,
        have h4 : ¬¬P, from not_not_intro h3,
        have h5 : ¬¬Q, from mt h2 h4,
        have h6 : Q,   from not_not.mp h5,
        show R, from h1 h6))

-- 2ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      ( assume h3 : P,
        have h4 : ¬¬P, from not_not_intro h3,
        have h5 : ¬¬Q, from mt h2 h4,
        have h6 : Q,   from not_not.mp h5,
        h1 h6))

-- 3ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      ( assume h3 : P,
        have h4 : ¬¬P, from not_not_intro h3,
        have h5 : ¬¬Q, from mt h2 h4,
        h1 (not_not.mp h5)))

-- 4ª demostración
```

```
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      ( assume h3 : P,
        have h4 : ¬¬P, from not_not_intro h3,
        h1 (not_not.mp (mt h2 h4))))

-- 5ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      ( assume h3 : P,
        h1 (not_not.mp (mt h2 (not_not_intro h3)))))

-- 6ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    show P → R, from
      (λh3, h1 (not_not.mp (mt h2 (not_not_intro h3)))))

-- 7ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( assume h2 : ¬Q → ¬P,
    (λh3, h1 (not_not.mp (mt h2 (not_not_intro h3)))))

-- 8ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
show (¬Q → ¬P) → (P → R), from
  ( λh2,
    (λh3, h1 (not_not.mp (mt h2 (not_not_intro h3)))))
```

```
-- 9ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
assume h1 : Q → R,
(λ h2 h3, h1 (not_not.mp (mt h2 (not_not_intro h3))))

-- 10ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
λ h1 h2 h3, h1 (not_not.mp (mt h2 (not_not_intro h3)))

-- 11ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
begin
  intro h1,
  intro h2,
  intro h3,
  apply h1,
  apply not_not.mp,
  apply mt h2,
  exact not_not_intro h3,
end

-- 12ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
begin
  intros h1 h2 h3,
  apply h1,
  apply not_not.mp,
  apply mt h2,
  exact not_not_intro h3,
end

-- 13ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
begin
  intros h1 h2 h3,
  apply h1,
  apply not_not.mp,
  exact mt h2 (not_not_intro h3),
end
```

```
-- 14ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
begin
  intros h1 h2 h3,
  exact h1 (not_not.mp (mt h2 (not_not_intro h3))),
end

-- 15ª demostración
example :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
λ h1 h2 h3, h1 (not_not.mp (mt h2 (not_not_intro h3)))

-- 16ª demostración
lemma aux :
  (Q → R) → ((¬Q → ¬P) → (P → R)) :=
-- by hint
by finish

-- #print axioms aux
```

Capítulo 3

Lógica de primer orden

3.1. Reglas del cuantificador universal

3.1.1. Regla de eliminación del cuantificador universal

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

```
-- Regla de eliminación del cuantificador universal
-- ===================================================

-- Ej. 1. Demostrar
--    P(c), ∀x (P(x) → ¬Q(x)) ⊢ ¬Q(c)

import tactic

variable  U : Type
variable  c : U
variables P Q : U → Prop

-- 1ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
have h3 : P c → ¬Q c, from h2 c,
show ¬Q c,            from h3 h1

-- 2ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
```

```
  : ¬Q c :=
have h3 : P c → ¬Q c, from h2 c,
h3 h1

-- 3ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
(h2 c) h1

-- 4ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
-- by library_search
h2 c h1

-- 5ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
-- by hint
by tauto

-- 6ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
by finish

-- 7ª demostración
example
  (h1 : P c)
  (h2 : ∀x, P x → ¬Q x)
  : ¬Q c :=
begin
  apply h2,
  exact h1,
end
```

3.1.2. Regla de introducción del cuantificador universal

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

```
-- Regla de introducción del cuantificador universal
-- ==================================================

-- Ej. 1. Demostrar
--    ∀x [P(x) → ¬Q(x)], ∀x P(x) ⊢ ∀x ¬Q(x)

import tactic

variable  U : Type
variables P Q : U → Prop

-- 1ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
assume x₀,
have h4 : P x₀ → ¬Q x₀, from h1 x₀,
have h5 : P x₀,         from h2 x₀,
show ¬Q x₀,             from h4 h5

-- 2ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
assume x₀, (h1 x₀) (h2 x₀)

-- 3ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
λ x₀, (h1 x₀) (h2 x₀)

-- 4ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
begin
```

```
  intro x₀,
  apply h1,
  apply h2,
end

-- 5ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
begin
  intro x₀,
  specialize h1 x₀,
  specialize h2 x₀,
  apply h1,
  exact h2,
end

-- 6ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
begin
  intro x₀,
  specialize h1 x₀,
  specialize h2 x₀,
  exact h1 h2,
end

-- 7ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
-- by hint
by tauto

-- 8ª demostración
example
  (h1 : ∀x, P x → ¬Q x)
  (h2 : ∀x, P x)
  : ∀x, ¬Q x :=
by finish
```

3.2. Reglas del cuantificador existencial

3.2.1. Regla de introducción del cuantificador existencial

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

```
-- Regla de introducción del cuantificador existencial
-- ====================================================

-- Ej. 1. Demostrar
--    ∀x P(x) ⊢ ∃x P(x)

import tactic

variable U : Type
variable c : U
variable P : U -> Prop

-- 1ª demostración
example
  (h1 : ∀x, P x)
  : ∃x, P x :=
have h2 : P c, from h1 c,
show ∃x, P x,  from exists.intro c h2

-- 2ª demostración
example
  (h1 : ∀x, P x)
  : ∃x, P x :=
show ∃x, P x,  from exists.intro c (h1 c)

-- 3ª demostración
example
  (h1 : ∀x, P x)
  : ∃x, P x :=
exists.intro c (h1 c)

-- 4ª demostración
example
  (h1 : ∀x, P x)
  : ∃x, P x :=
⟨c, h1 c⟩
```

```
-- 5ª demostración
example
  (a : U)
  (h1 : ∀x, P x)
  : ∃x, P x :=
begin
  use a,
  apply h1,
end

-- 6ª demostración
example
  (a : U)
  (h1 : ∀x, P x)
  : ∃x, P x :=
begin
  constructor,
  apply h1 a,
end

-- 7ª demostración
example
  [inhabited U]
  (h1 : ∀x, P x)
  : ∃x, P x :=
begin
  constructor,
  apply h1 (default U),
end

-- 8ª demostración
example
  (h : nonempty U)
  (h1 : ∀x, P x)
  : ∃x, P x :=
begin
  use (classical.choice h),
  apply h1,
end
```

3.2.2. Regla de eliminación del cuantificador existencial

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

```
-- Regla de eliminación del cuantificador existencial
-- ====================================================

-- Ej. 1. Demostrar
--    ∀x [P(x) → Q(x)], ∃x P(x) ⊢ ∃x Q(x)

import tactic

variable  U : Type
variables P Q : U -> Prop

-- 1ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2
  ( assume x₀ (h3 : P x₀),
    have h4 : P x₀ → Q x₀, from h1 x₀,
    have h5 : Q x₀,        from h4 h3,
    show ∃x, Q x,          from exists.intro x₀ h5 )

-- 2ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2
  ( assume x₀ (h3 : P x₀),
    have h4 : P x₀ → Q x₀, from h1 x₀,
    have h5 : Q x₀,        from h4 h3,
    show ∃x, Q x,          from ⟨x₀, h5⟩ )

-- 3ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2
```

```
  ( assume x₀ (h3 : P x₀),
    have h4 : P x₀ → Q x₀, from h1 x₀,
    have h5 : Q x₀,        from h4 h3,
    ⟨x₀, h5⟩ )

-- 4ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2
  ( assume x₀ (h3 : P x₀),
    have h4 : P x₀ → Q x₀, from h1 x₀,
    ⟨x₀, h4 h3⟩ )

-- 5ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2
  ( assume x₀ (h3 : P x₀),
    ⟨x₀, h1 x₀ h3⟩ )

-- 6ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
exists.elim h2 (λ x₀ h3, ⟨x₀, h1 x₀ h3⟩)

-- 7ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
-- by library_search
```

```
Exists.imp h1 h2

-- 8ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
begin
  cases h2 with x₀ h3,
  use x₀,
  apply h1,
  exact h3,
end

-- 9ª demostración
-- ===============

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
begin
  cases h2 with x₀ h3,
  use x₀,
  specialize h1 x₀,
  apply h1,
  exact h3,
end

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

example
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
-- by hint
by tauto

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

example
```

```
  (h1 : ∀x, P x → Q x)
  (h2 : ∃x, P x)
  : ∃x, Q x :=
by finish
```

3.3. Ejercicios sobre cuantificadores

3.3.1. Pruebas de ¬∀x P(x) ↔ ∃x ¬P(x)

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

```
-- Pruebas de ¬∀x P(x) ↔ ∃x ¬P(x)
-- ==============================

import tactic

variable {U : Type}
variable {P : U -> Prop}

-- ------------------------------------------------------
-- Ej. 1. Demostrar que
--    ¬∀x P(x) ⊢ ∃x ¬P(x)
-- ------------------------------------------------------

-- 1ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra
            ( assume h4 : ¬P x₀,
              have h5 : ∃x, ¬P x, from exists.intro x₀ h4,
              show false, from h2 h5 )),
    show false, from h1 h8)

-- 2ª demostración
example
  (h1 : ¬∀x, P x)
```

```
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra
            ( assume h4 : ¬P x₀,
              have h5 : ∃x, ¬P x, from exists.intro x₀ h4,
              show false, from h2 h5 )),
    h1 h8)

-- 3ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra
            ( assume h4 : ¬P x₀,
              have h5 : ∃x, ¬P x, from exists.intro x₀ h4,
              h2 h5 )),
    h1 h8)

-- 4ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra
            ( assume h4 : ¬P x₀,
              have h5 : ∃x, ¬P x, from ⟨x₀, h4⟩,
              h2 h5 )),
    h1 h8)

-- 5ª demostración
example
  (h1 : ¬∀x, P x)
```

```
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra
            ( assume h4 : ¬P x₀,
              h2 ⟨x₀, h4⟩ )),
    h1 h8)

-- 6ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        show P x₀, from
          by_contra (λ h4, h2 ⟨x₀, h4⟩)),
    h1 h8)

-- 7ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      ( assume x₀,
        by_contra (λ h4, h2 ⟨x₀, h4⟩)),
    h1 h8)

-- 8ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    have h8 : ∀x, P x, from
      (λ x₀, by_contra (λ h4, h2 ⟨x₀, h4⟩)),
    h1 h8)

-- 9ª demostración
```

```
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra
  ( assume h2 : ¬∃x, ¬P x,
    h1 (λ x₀, by_contra (λ h4, h2 ⟨x₀, h4⟩)))

-- 10ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
by_contra (λ h2, h1 (λ x₀, by_contra (λ h4, h2 ⟨x₀, h4⟩)))

-- 11ª demostración
example
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
-- by library_search
not_forall.mp h1

-- 12ª demostración
lemma aux1
  (h1 : ¬∀x, P x)
  : ∃x, ¬P x :=
-- by hint
by finish

-- ----------------------------------------------------
-- Ej. 2. Demostrar que
--    ∃x ¬P(x) ⊢ ¬∀x P(x)
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
assume h2 : ∀x, P x,
exists.elim h1
  ( assume x₀ (h3 : ¬P x₀),
    have h4 : P x₀, from h2 x₀,
    show false, from h3 h4)

-- 2ª demostración
example
  (h1 : ∃x, ¬P x)
```

```
  : ¬∀x, P x :=
assume h2 : ∀x, P x,
exists.elim h1
  ( assume x₀ (h3 : ¬P x₀),
    have h4 : P x₀, from h2 x₀,
    h3 h4)

-- 3ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
assume h2 : ∀x, P x,
exists.elim h1
  ( assume x₀ (h3 : ¬P x₀),
    h3 (h2 x₀))

-- 4ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
assume h2 : ∀x, P x,
exists.elim h1
  (λ x₀ h3, h3 (h2 x₀))

-- 5ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
λ h2, exists.elim h1 (λ x₀ h3, h3 (h2 x₀))

-- 6ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
-- by library_search
not_forall.mpr h1

-- 7ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
assume h2 : ∀x, P x,
match h1 with ⟨x₀, (h3 : ¬P x₀)⟩ :=
  ( have h4 : P x₀, from h2 x₀,
    show false,     from h3 h4)
```

```
end

-- 8ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
begin
  intro h2,
  cases h1 with x₀ h3,
  apply h3,
  apply h2,
end

example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
begin
  intro h2,
  obtain ⟨x₀, h3⟩ := h1,
  apply h3,
  apply h2,
end

-- 9ª demostración
example
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
-- by hint
by tauto

-- 10ª demostración
lemma aux2
  (h1 : ∃x, ¬P x)
  : ¬∀x, P x :=
by finish

#print axioms aux2

-- -------------------------------------------------------
-- Ej. 3. Demostrar que
--    ¬∀x P(x) ↔ ∃x ¬P(x)
-- -------------------------------------------------------

-- 1ª demostración
example :
```

```
  (¬∀x, P x) ↔ (∃x, ¬P x) :=
iff.intro
  ( assume h1 : ¬∀x, P x,
    show ∃x, ¬P x, from aux1 h1)
  ( assume h2 : ∃x, ¬P x,
    show ¬∀x, P x, from aux2 h2)

-- 2ª demostración
example :
  (¬∀x, P x) ↔ (∃x, ¬P x) :=
iff.intro aux1 aux2

-- 3ª demostración
example :
  (¬∀x, P x) ↔ (∃x, ¬P x) :=
-- by library_search
not_forall

-- 4ª demostración
example :
  (¬∀x, P x) ↔ (∃x, ¬P x) :=
begin
  split,
  { exact aux1, },
  { exact aux2, },
end

-- 5ª demostración
example :
  (¬∀x, P x) ↔ (∃x, ¬P x) :=
-- by hint
by finish
```

3.3.2. Pruebas de ∀x (P(x) ∧ Q(x)) ↔ ∀x P(x) ∧ ∀x Q(x)

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

```
-- Pruebas de ∀x (P(x) ∧ Q(x)) ↔ ∀x P(x) ∧ ∀x Q(x)
-- ==============================================

import tactic

section
```

```
variable  {U : Type}
variables {P Q : U -> Prop}

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    ∀x (P(x) ∧ Q(x)) ⊢ ∀x P(x) ∧ ∀x Q(x)
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    assume x₀,
    have h3 : P x₀ ∧ Q x₀,  from h1 x₀,
    show P x₀,              from and.elim_left h3,
have h9 : ∀x, Q x, from
    assume x₁,
    have h7 : P x₁ ∧ Q x₁,  from h1 x₁,
    show Q x₁,              from and.elim_right h7,
show (∀x, P x) ∧ (∀x, Q x), from and.intro h5 h9

-- 2ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    assume x₀,
    have h3 : P x₀ ∧ Q x₀,  from h1 x₀,
    show P x₀,              from h3.left,
have h9 : ∀x, Q x, from
    assume x₁,
    have h7 : P x₁ ∧ Q x₁,  from h1 x₁,
    show Q x₁,              from h7.right,
show (∀x, P x) ∧ (∀x, Q x), from ⟨h5, h9⟩

-- 3ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    assume x₀,
    have h3 : P x₀ ∧ Q x₀,  from h1 x₀,
    h3.left,
```

```
have h9 : ∀x, Q x, from
    assume x₁,
    have h7 : P x₁ ∧ Q x₁,  from h1 x₁,
    h7.right,
show (∀x, P x) ∧ (∀x, Q x), from ⟨h5, h9⟩

-- 4ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    assume x₀,
    (h1 x₀).left,
have h9 : ∀x, Q x, from
    assume x₁,
    (h1 x₁).right,
show (∀x, P x) ∧ (∀x, Q x), from ⟨h5, h9⟩

-- 5ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    λ x₀, (h1 x₀).left,
have h9 : ∀x, Q x, from
    λ x₁, (h1 x₁).right,
show (∀x, P x) ∧ (∀x, Q x), from ⟨h5, h9⟩

-- 6ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
have h5 : ∀x, P x, from
    λ x₀, (h1 x₀).left,
have h9 : ∀x, Q x, from
    λ x₁, (h1 x₁).right,
⟨h5, h9⟩

-- 7ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
⟨λ x₀, (h1 x₀).left, λ x₁, (h1 x₁).right⟩

-- 8ª demostración
```

```
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
-- by library_search
forall_and_distrib.mp h1

-- 9ª demostración
example
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
begin
  split,
  { intro x₀,
    specialize h1 x₀,
    exact h1.left, },
  { intro x₁,
    specialize h1 x₁,
    exact h1.right, },
end

-- 9ª demostración
lemma aux1
  (h1 : ∀x, P x ∧ Q x)
  : (∀x, P x) ∧ (∀x, Q x) :=
-- by hint
by finish

-- ------------------------------------------------------
-- Ej. 2. Demostrar
--    ∀x P(x) ∧ ∀x Q(x) ⊢ ∀x (P(x) ∧ Q(x))
-- ------------------------------------------------------

-- 1ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from and.elim_left h1,
have h4 : P x₀,    from h3 x₀,
have h5 : ∀x, Q x, from and.elim_right h1,
have h6 : Q x₀,    from h5 x₀,
show P x₀ ∧ Q x₀,  from and.intro h4 h6

-- 2ª demostración
example
```

```
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from h1.left,
have h4 : P x₀,    from h3 x₀,
have h5 : ∀x, Q x, from h1.right,
have h6 : Q x₀,    from h5 x₀,
show P x₀ ∧ Q x₀,  from ⟨h4, h6⟩

-- 3ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from h1.left,
have h4 : P x₀,    from h3 x₀,
have h5 : ∀x, Q x, from h1.right,
have h6 : Q x₀,    from h5 x₀,
⟨h4, h6⟩

-- 4ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from h1.left,
have h4 : P x₀,    from h3 x₀,
have h5 : ∀x, Q x, from h1.right,
⟨h4, h5 x₀⟩

-- 5ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from h1.left,
have h4 : P x₀,    from h3 x₀,
⟨h4, h1.right x₀⟩

-- 6ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
have h3 : ∀x, P x, from h1.left,
```

```
⟨h3 x₀, h1.right x₀⟩

-- 7ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
assume x₀,
⟨h1.left x₀, h1.right x₀⟩

-- 8ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
λ x₀, ⟨h1.left x₀, h1.right x₀⟩

-- 9ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
-- by library_search
forall_and_distrib.mpr h1

-- 10ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
begin
  cases h1 with h2 h3,
  intro x₀,
  split,
  { apply h2, },
  { apply h3, },
end

-- 11ª demostración
example
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
-- by hint
by tauto

-- 12ª demostración
lemma aux2
  (h1 : (∀x, P x) ∧ (∀x, Q x))
  : ∀x, P x ∧ Q x :=
```

```
by finish

-- ---------------------------------------------------------
-- Ej. 3. Demostrar
--    ∀x (P(x) ∧ Q(x)) ↔ ∀x P(x) ∧ ∀x Q(x)
-- ---------------------------------------------------------

-- 1ª demostración
example :
  (∀x, P x ∧ Q x) ↔ (∀x, P x) ∧ (∀x, Q x) :=
iff.intro aux1 aux2

-- 2ª demostración
example :
  (∀x, P x ∧ Q x) ↔ (∀x, P x) ∧ (∀x, Q x) :=
-- by library_search
forall_and_distrib

-- 3ª demostración
example :
  (∀x, P x ∧ Q x) ↔ (∀x, P x) ∧ (∀x, Q x) :=
-- by hint
by finish

end
```

3.3.3. Pruebas de ∃x (P(x) ∨ Q(x)) ↔ ∃x P(x) ∨ ∃x Q(x)

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

```
-- Pruebas de ∃x (P(x) ∨ Q(x)) ↔ ∃x P(x) ∨ ∃x Q(x)
-- ================================================

import tactic

section

variable  {U : Type}
variables {P Q : U -> Prop}

-- --------------------------------------------------------
-- Ej. 1. Demostrar
--    ∃x (P(x) ∨ Q(x)) ⊢ ∃x P(x) ∨ ∃x Q(x)
```

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

-- 1ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  ( assume x₀ (h2 : P x₀ ∨ Q x₀),
    or.elim h2
    ( assume h3 : P x₀,
      have h4 : ∃x, P x,           from exists.intro x₀ h3,
      show (∃x, P x) ∨ (∃x, Q x), from or.inl h4 )
    ( assume h6 : Q x₀,
      have h7 : ∃x, Q x,           from exists.intro x₀ h6,
      show (∃x, P x) ∨ (∃x, Q x), from or.inr h7 ))

-- 2ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  ( assume x₀ (h2 : P x₀ ∨ Q x₀),
    or.elim h2
    ( assume h3 : P x₀,
      have h4 : ∃x, P x,           from ⟨x₀, h3⟩,
      show (∃x, P x) ∨ (∃x, Q x), from or.inl h4 )
    ( assume h6 : Q x₀,
      have h7 : ∃x, Q x,           from ⟨x₀, h6⟩,
      show (∃x, P x) ∨ (∃x, Q x), from or.inr h7 ))

-- 3ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  ( assume x₀ (h2 : P x₀ ∨ Q x₀),
    or.elim h2
    ( assume h3 : P x₀,
      have h4 : ∃x, P x,           from ⟨x₀, h3⟩,
      or.inl h4 )
    ( assume h6 : Q x₀,
      have h7 : ∃x, Q x,           from ⟨x₀, h6⟩,
      or.inr h7 ))

-- 4ª demostración
```

```
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  ( assume x₀ (h2 : P x₀ ∨ Q x₀),
    or.elim h2
    ( assume h3 : P x₀,
      or.inl ⟨x₀, h3⟩ )
    ( assume h6 : Q x₀,
      or.inr ⟨x₀, h6⟩ ))

-- 5ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  ( assume x₀ (h2 : P x₀ ∨ Q x₀),
    or.elim h2
    ( λ h3, or.inl ⟨x₀, h3⟩ )
    ( λ h6, or.inr ⟨x₀, h6⟩ ))

-- 6ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
exists.elim h1
  (λ x₀ h2, h2.elim (λ h3, or.inl ⟨x₀, h3⟩)
                    (λ h6, or.inr ⟨x₀, h6⟩))

-- 7ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
-- by library_search
exists_or_distrib.mp h1

-- 8ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
match h1 with ⟨x₀, (h2 : P x₀ ∨ Q x₀)⟩ :=
  ( or.elim h2
    ( assume h3 : P x₀,
      have h4 : ∃x, P x,           from exists.intro x₀ h3,
      show (∃x, P x) ∨ (∃x, Q x), from or.inl h4 )
```

```
    ( assume h6 : Q x₀,
      have h7 : ∃x, Q x,            from exists.intro x₀ h6,
      show (∃x, P x) ∨ (∃x, Q x), from or.inr h7 ))
end

-- 9ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
begin
  cases h1 with x₀ h3,
  cases h3 with hp hq,
  { left,
    use x₀,
    exact hp, },
  { right,
    use x₀,
    exact hq, },
end

-- 10ª demostración
example
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
begin
  rcases h1 with ⟨x₀, hp | hq⟩,
  { left,
    use x₀,
    exact hp, },
  { right,
    use x₀,
    exact hq, },
end

-- 11ª demostración
lemma aux1
  (h1 : ∃x, P x  ∨ Q x)
  : (∃x, P x) ∨ (∃x, Q x) :=
-- by hint
by finish

-- ----------------------------------------------------
-- Ej. 2. Demostrar
--    ∃x P(x) ∨ ∃x Q(x) ⊢ ∃x (P(x) ∨ Q(x))
-- ----------------------------------------------------
```

```
-- 1ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
or.elim h1
  ( assume h2 : ∃x, P x,
    exists.elim h2
      ( assume x₀ (h3 : P x₀),
        have h4 : P x₀ ∨ Q x₀, from or.inl h3,
        show ∃x, P x  ∨ Q x,   from exists.intro x₀ h4 ))
  ( assume h2 : ∃x, Q x,
    exists.elim h2
      ( assume x₀ (h3 : Q x₀),
        have h4 : P x₀ ∨ Q x₀, from or.inr h3,
        show ∃x, P x  ∨ Q x,   from exists.intro x₀ h4 ))

-- 2ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
h1.elim
  ( assume ⟨x₀, (h3 : P x₀)⟩,
    have h4 : P x₀ ∨ Q x₀, from or.inl h3,
    show ∃x, P x  ∨ Q x,   from ⟨x₀, h4⟩ )
  ( assume ⟨x₀, (h3 : Q x₀)⟩,
    have h4 : P x₀ ∨ Q x₀, from or.inr h3,
    show ∃x, P x  ∨ Q x,   from ⟨x₀, h4⟩ )

-- 3ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
h1.elim
  ( assume ⟨x₀, (h3 : P x₀)⟩,
    have h4 : P x₀ ∨ Q x₀, from or.inl h3,
    ⟨x₀, h4⟩ )
  ( assume ⟨x₀, (h3 : Q x₀)⟩,
    have h4 : P x₀ ∨ Q x₀, from or.inr h3,
    ⟨x₀, h4⟩ )

-- 4ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
```

```
h1.elim
  ( assume ⟨x₀, (h3 : P x₀)⟩,
    ⟨x₀, or.inl h3⟩ )
  ( assume ⟨x₀, (h3 : Q x₀)⟩,
    ⟨x₀, or.inr h3⟩ )

-- 5ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
h1.elim
  (λ ⟨x₀, h3⟩, ⟨x₀, or.inl h3⟩)
  (λ ⟨x₀, h3⟩, ⟨x₀, or.inr h3⟩)

-- 6ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
-- by library_search
exists_or_distrib.mpr h1

-- 7ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
begin
  cases h1 with hp hq,
  { cases hp with x₀ hx₀,
    use x₀,
    left,
    exact hx₀, },
  { cases hq with x₁ hx₁,
    use x₁,
    right,
    exact hx₁, },
end

-- 8ª demostración
example
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
begin
  rcases h1 with ⟨x₀, hx₀⟩ | ⟨x₁, hx₁⟩,
  { use x₀,
    left,
```

```
    exact hx₀, },
  { use x₁,
    right,
    exact hx₁, },
end

-- 9ª demostración
lemma aux2
  (h1 : (∃x, P x) ∨ (∃x, Q x))
  : ∃x, P x  ∨ Q x :=
-- by hint
by finish

-- ----------------------------------------------------
-- Ej. 3. Demostrar
--    ∃x (P(x) ∨ Q(x)) ↔ ∃x P(x) ∨ ∃x Q(x)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (∃x, P x  ∨ Q x) ↔ (∃x, P x) ∨ (∃x, Q x) :=
iff.intro aux1 aux2

-- 2ª demostración
example :
  (∃x, P x  ∨ Q x) ↔ (∃x, P x) ∨ (∃x, Q x) :=
⟨aux1, aux2⟩

-- 3ª demostración
example :
  (∃x, P x  ∨ Q x) ↔ (∃x, P x) ∨ (∃x, Q x) :=
-- by library_search
exists_or_distrib

end
```

3.3.4. Pruebas de ∃x∃y P(x,y) ↔ ∃y∃x P(x,y)

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

```
-- Pruebas de ∃x∃yP(x,y) ↔ ∃y∃xP(x,y)
-- ===================================
```

```
import tactic

section

variable {U : Type}
variable {P : U -> U -> Prop}

-- ------------------------------------------------------
-- Ej. 1. Demostrar que
--    ∃x∃yP(x,y) → ∃y∃xP(x,y)
-- ------------------------------------------------------

-- 1ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
assume h1 : ∃ x, ∃ y, P x y,
exists.elim h1
  ( assume x₀ (h2 : ∃ y, P x₀ y),
    exists.elim h2
      ( assume y₀ (h3 : P x₀ y₀),
        have h4 : ∃ x, P x y₀, from exists.intro x₀ h3,
        show ∃ y, ∃ x, P x y,  from exists.intro y₀ h4))

-- 2ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
assume ⟨x₀, y₀, (h1 : P x₀ y₀)⟩,
have h2 : ∃ x, P x y₀, from ⟨x₀, h1⟩,
show (∃ y, ∃ x, P x y), from ⟨y₀, h2⟩

-- 3ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
assume ⟨x₀, y₀, (h1 : P x₀ y₀)⟩,
show (∃ y, ∃ x, P x y), from ⟨y₀, ⟨x₀, h1⟩⟩

-- 4ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
assume ⟨x₀, y₀, (h1 : P x₀ y₀)⟩,
show (∃ y, ∃ x, P x y), from ⟨y₀, x₀, h1⟩

-- 5ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
```

```
assume ⟨x₀, y₀, (h1 : P x₀ y₀)⟩,
⟨y₀, x₀, h1⟩

-- 6ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
λ ⟨x₀, y₀, h1⟩, ⟨y₀, x₀, h1⟩

-- 7ª demostración
example :
  (∃ x y, P x y) → (∃ y x, P x y) :=
-- by library_search
exists_comm.mp

-- 8ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
begin
  intro h1,
  cases h1 with x₀ h2,
  cases h2 with y₀ h3,
  use y₀,
  use x₀,
  exact h3,
end

-- 9ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
begin
  intro h1,
  cases h1 with x₀ h2,
  cases h2 with y₀ h3,
  use [y₀, x₀],
  exact h3,
end

-- 10ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
begin
  intro h1,
  rcases h1 with ⟨x₀, y₀, h2⟩,
  use [y₀, x₀],
  exact h2,
```

```
end

-- 11ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
begin
  intro h1,
  rcases h1 with ⟨x₀, y₀, h2⟩,
  exact ⟨y₀, x₀, h2⟩,
end

-- 12ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
begin
  rintro ⟨x₀, y₀, h2⟩,
  exact ⟨y₀, x₀, h2⟩,
end

-- 13ª demostración
example :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
-- by hint
by tauto

-- 14ª demostración
lemma aux :
  (∃ x, ∃ y, P x y) → (∃ y, ∃ x, P x y) :=
by finish

-- ----------------------------------------------------
-- Ej. 2. Demostrar que
--    ∃x∃yP(x,y) ↔ ∃y∃xP(x,y)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (∃ x y, P x y) ↔ (∃ y x, P x y) :=
iff.intro aux aux

-- 2ª demostración
example :
  (∃ x y, P x y) ↔ (∃ y x, P x y) :=
⟨aux, aux⟩
```

```
-- 3ª demostración
example :
  (∃ x y, P x y) ↔ (∃ y x, P x y) :=
-- by library_search
exists_comm

-- 4ª demostración
example :
  (∃ x y, P x y) ↔ (∃ y x, P x y) :=
-- by hint
by tauto

end
```

3.4. Reglas de la igualdad

3.4.1. Regla de eliminación de la igualdad

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

```
-- Regla de eliminación de la igualdad
-- ===================================

import tactic

#check @eq.subst

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--       x + 1 = 1 + x
--       x + 1 > 1 → x + 1 > 0
--       ⊢ 1 + x > 1 → 1 + x > 0
-- ----------------------------------------------------

variable (U : Type)
variable (x : U)
variable [has_add U]
variable [has_one U]
variable [has_lt U]
variable [has_zero U]

-- 1ª demostración
```

```
example
  (h1 : x + 1 = 1 + x)
  (h2 : x + 1 > 1 → x + 1 > 0)
  : 1 + x > 1 → 1 + x > 0 :=
eq.subst h1 h2


-- 2ª demostración
example
  (h1 : x + 1 = 1 + x)
  (h2 : x + 1 > 1 → x + 1 > 0)
  : 1 + x > 1 → 1 + x > 0 :=
h1 ▶ h2

-- 3ª demostración
example
  (h1 : x + 1 = 1 + x)
  (h2 : x + 1 > 1 → x + 1 > 0)
  : 1 + x > 1 → 1 + x > 0 :=
begin
  rw h1 at h2,
  exact h2,
end

-- 4ª demostración
example
  (h1 : x + 1 = 1 + x)
  (h2 : x + 1 > 1 → x + 1 > 0)
  : 1 + x > 1 → 1 + x > 0 :=
begin
  rw ←h1,
  exact h2,
end

-- 5ª demostración
example
  (h1 : x + 1 = 1 + x)
  (h2 : x + 1 > 1 → x + 1 > 0)
  : 1 + x > 1 → 1 + x > 0 :=
-- by hint
by finish
```

3.4.2. Pruebas de la transitividad de la igualdad

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

```
-- Pruebas de la transitividad de la igualdad
-- ==========================================

import tactic

variable  (U : Type)
variables (x y z : U)

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--    x = y, y = z ⊢ x = z
-- ----------------------------------------------------

-- 1ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
eq.subst h2 h1

-- 2ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
h2 ▸ h1

-- 3ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
eq.substr h1 h2

-- 4ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
eq.trans h1 h2
```

```
-- 5ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
begin
  rw h1,
  exact h2,
end

-- 6ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
begin
  rw h1,
  assumption,
end

-- 7ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
begin
  rwa h1,
end

-- 8ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
by rwa h1

-- 9ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
begin
  rwa ←h2,
end
```

```
-- 10ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
begin
  rwa h2 at h1,
end

-- 11ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
by simp *

-- 12ª demostración
example
  (h1 : x = y)
  (h2 : y = z)
  : x = z :=
-- by hint
by finish
```

3.4.3. Regla de introducción de la igualdad

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

```
-- Regla de introducción de la igualdad
-- ====================================

import tactic

#check @eq.refl

variable  (U : Type)
variables (x y : U)

-- ----------------------------------------------------
-- Ej. Demostrar
--    x = y ⊢ y = x
-- ----------------------------------------------------
```

```
-- 1ª demostración
example
  (h1 : x = y)
  : y = x :=
have h2 : x = x, from eq.refl x,
show y = x,      from eq.subst h1 h2

-- 2ª demostración
example
  (h1 : x = y)
  : y = x :=
have h2 : x = x, from eq.refl x,
show y = x,      from h1 ▸ h2

-- 3ª demostración
example
  (h1 : x = y)
  : y = x :=
have h2 : x = x, from eq.refl x,
h1 ▸ h2

-- 4ª demostración
example
  (h1 : x = y)
  : y = x :=
h1 ▸ eq.refl x

-- 5ª demostración
example
  (h1 : x = y)
  : y = x :=
h1 ▸ rfl

-- 6ª demostración
example
  (h1 : x = y)
  : y = x :=
-- by library_search
eq.symm h1

-- 7ª demostración
example
  (h1 : x = y)
  : y = x :=
begin
```

```
  rw h1,
end

-- 8ª demostración
example
  (h1 : x = y)
  : y = x :=
by rw h1

-- 9ª demostración
example
  (h1 : x = y)
  : y = x :=
by simp *

-- 10ª demostración
example
  (h1 : x = y)
  : y = x :=
-- by hint
by tauto

-- 11ª demostración
example
  (h1 : x = y)
  : y = x :=
by finish

-- 12ª demostración
example
  (h1 : x = y)
  : y = x :=
by solve_by_elim
```

3.4.4. Pruebas de y = x → y = z → x = z

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

```
-- Pruebas de y = x → y = z → x = z
-- ================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
```

```
--    y = x → y = z → x = z
-- ----------------------------------------------------

import tactic

variables (U : Type)
variables (x y z : U)

-- 1ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
have h3 : x = y, from eq.symm h1,
show x = z,      from eq.trans h3 h2

-- 2ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
have h3 : x = y, from eq.symm h1,
eq.trans h3 h2

-- 3ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
eq.trans (eq.symm h1) h2

-- 4ª demostración
example : y = x → y = z → x = z :=
λ h1 h2, eq.trans (eq.symm h1) h2

-- 5ª demostración
example : y = x → y = z → x = z :=
λ h1 h2, eq.trans h1.symm h2

-- 6ª demostración
example : y = x → y = z → x = z :=
-- by library_search
λ h, h.congr_left.mp

-- 7ª demostración
example : y = x → y = z → x = z :=
begin
  intros h1 h2,
```

```
  rwa ←h1,
end

-- 8ª demostración
example : y = x → y = z → x = z :=
begin
  intros h1 h2,
  rw h1 at h2,
  assumption,
end

-- 9ª demostración
example : y = x → y = z → x = z :=
begin
  intros h1 h2,
  rwa h1 at h2,
end

-- 10ª demostración
example : y = x → y = z → x = z :=
begin
  intros h1 h2,
  calc x = y : h1.symm
     ... = z : h2,
end

-- 11ª demostración
example : y = x → y = z → x = z :=
-- by hint
by finish

-- 12ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
show x = z,
  begin
    rw ←h1,
    rw h2
  end

-- 13ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
```

```
show x = z,
  begin
    rw [←h1, h2]
  end

-- 14ª demostración
example : y = x → y = z → x = z :=
assume h1 : y = x,
assume h2 : y = z,
show x = z, by rw [←h1, h2]
```

3.4.5. Pruebas de (x + y) + z = (x + z) + y

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

```
-- Pruebas de (x + y) + z = (x + z) + y
-- ====================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que para todo x, y, z en ℤ se tiene
--    (x + y) + z = (x + z) + y
-- ----------------------------------------------------

import tactic
import data.int.basic

variables (x y z : ℤ)

-- 1ª demostración
example : (x + y) + z = (x + z) + y :=
calc
   (x + y) + z = x + (y + z) : add_assoc x y z
           ... = x + (z + y) : eq.subst (add_comm y z) rfl
           ... = (x + z) + y : eq.symm (add_assoc x z y)

-- 2ª demostración
example : (x + y) + z = (x + z) + y :=
calc
  (x + y) + z = x + (y + z) : by rw add_assoc
          ... = x + (z + y) : by rw [add_comm y z]
          ... = (x + z) + y : by rw add_assoc
```

```
-- 3ª demostración
example : (x + y) + z = (x + z) + y :=
begin
  rw add_assoc,
  rw add_comm y z,
  rw add_assoc,
end

example : (x + y) + z = (x + z) + y :=
begin
  rw [add_assoc, add_comm y z, add_assoc],
end

-- 4ª demostración
example : (x + y) + z = (x + z) + y :=
by rw [add_assoc, add_comm y z, add_assoc]

-- 5ª demostración
example : (x + y) + z = (x + z) + y :=
-- by library_search
add_right_comm x y z

-- 6ª demostración
example : (x + y) + z = (x + z) + y :=
-- by hint
by omega

-- 7ª demostración
example : (x + y) + z = (x + z) + y :=
by linarith

-- 8ª demostración
example : (x + y) + z = (x + z) + y :=
by nlinarith

-- 9ª demostración
example : (x + y) + z = (x + z) + y :=
by ring

-- 10ª demostración
example : (x + y) + z = (x + z) + y :=
by finish
```

3.4.6. Pruebas de desarrollo de producto de sumas

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

```
-- Pruebas de desarrollo de producto de sumas
-- ==========================================

-- ----------------------------------------------------
-- Ej. 1. Sean a, b, c y d números enteros. Demostrar que
--    (a + b) * (c + d) = a * c + b * c + a * d + b * d
-- ----------------------------------------------------

import tactic
import data.int.basic

variables a b c d : ℤ

-- 1ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
calc
  (a + b) * (c + d)
      = (a + b) * c + (a + b) * d         : by rw left_distrib
  ... = (a * c + b * c) + (a + b) * d     : by rw right_distrib
  ... = (a * c + b * c) + (a * d + b * d) : by rw right_distrib
  ... = a * c + b * c + a * d + b * d     : by rw ←add_assoc

-- 2ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
by rw [left_distrib, right_distrib, right_distrib, ←add_assoc]

-- 3ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
begin
  rw left_distrib,
  rw right_distrib,
  rw right_distrib,
  rw ←add_assoc,
end

-- 4ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
calc
  (a + b) * (c + d)
      = (a + b) * c + (a + b) * d         : by rw mul_add
  ... = (a * c + b * c) + (a + b) * d     : by rw add_mul
```

```
  ... = (a * c + b * c) + (a * d + b * d) : by rw add_mul
  ... = a * c + b * c + a * d + b * d     : by rw ←add_assoc

-- 5ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
-- by hint
by linarith

-- 6ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
by nlinarith

-- 7ª demostración
example : (a + b) * (c + d) = a * c + b * c + a * d + b * d :=
by ring
```

Capítulo 4

Conjuntos

4.1. Elementos básicos sobre conjuntos

4.1.1. Pruebas de la reflexividad de la inclusión de conjuntos

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

```
-- Prueba de la reflexividad de la inclusión de conjuntos
-- ======================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    A ⊆ A
-- ----------------------------------------------------

import data.set
variable  U : Type
variable  x : U
variables A B C : set U

-- #reduce x ∈ A
-- #reduce B ⊆ C

-- 1ª demostración
example : A ⊆ A :=
begin
  intros x h,
  exact h,
end
```

```
-- 2ª demostración
example : A ⊆ A :=
assume x,
assume h : x ∈ A,
show x ∈ A, from h

-- 3ª demostración
example : A ⊆ A :=
assume x,
assume h : x ∈ A,
h

-- 4ª demostración
example : A ⊆ A :=
assume x,
λ h : x ∈ A, h

-- 5ª demostración
example : A ⊆ A :=
assume x,
id

-- 6ª demostración
example : A ⊆ A :=
λ x, id

-- 7ª demostración
example : A ⊆ A :=
-- by library_search
set.subset.rfl

open set

-- 8ª demostración
example : A ⊆ A :=
subset.rfl

-- 9ª demostración
example : A ⊆ A :=
-- by hint
by tauto

-- 10ª demostración
example : A ⊆ A :=
```

```
by finish

-- 11ª demostración
example : A ⊆ A :=
by refl
```

4.1.2. Pruebas de la antisimetría de la inclusión de conjuntos

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

```
-- Pruebas de la antisimetría de la inclusión de conjuntos
-- =======================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    A ⊆ B, B ⊆ A ⊢ A = B
-- ----------------------------------------------------

import data.set

variable  U : Type
variables A B : set U

open set

-- 1ª demostración
example
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
begin
  ext,
  split,
  { intro h,
    exact h1 h, },
  { intro h,
    exact h2 h, },
end

-- 2ª demostración
example
```

```
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
ext
( assume x,
  iff.intro
  ( assume h : x ∈ A,
    show x ∈ B, from h1 h)
  ( assume h : x ∈ B,
    show x ∈ A, from h2 h))

-- 3ª demostración
example
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
ext
(λ x,
 iff.intro
 (λ h, h1 h)
 (λ h, h2 h))

-- 4ª demostración
example
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
eq_of_subset_of_subset
  ( assume x,
    assume h : x ∈ A,
    show x ∈ B, from h1 h)
  ( assume x,
    assume h : x ∈ B,
    show x ∈ A, from h2 h)

-- 5ª demostración
example
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
eq_of_subset_of_subset h1 h2

-- 6ª demostración
```

```
example
  (h1 : A ⊆ B)
  (h2 : B ⊆ A)
  : A = B :=
-- by library_search
subset.antisymm h1 h2
```

4.1.3. Introducción de la intersección

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

```
-- Introducción de la intersección
-- ===============================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    x ∈ A → x ∈ B → x ∈ A ∩ B
-- ----------------------------------------------------

import data.set

variable  U : Type
variables A B : set U
variable  x : U

open set

-- #reduce x ∈ A ∩ B

-- 1ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
begin
  intros h1 h2,
  simp,
  split,
  { exact h1, },
  { exact h2, },
end

-- 2ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
begin
```

```
  intros h1 h2,
  split,
  { exact h1, },
  { exact h2, },
end

-- 3ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
assume h1 : x ∈ A,
assume h2 : x ∈ B,
show x ∈ A ∩ B, from and.intro h1 h2

-- 4ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
assume h1 : x ∈ A,
assume h2 : x ∈ B,
show x ∈ A ∩ B, from ⟨h1, h2⟩

-- 5ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
assume h1 : x ∈ A,
assume h2 : x ∈ B,
⟨h1, h2⟩

-- 6ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
λ h1 h2, ⟨h1, h2⟩

-- 7ª demostración
example : x ∈ A → x ∈ B → x ∈ A ∩ B :=
-- by library_search
mem_inter
```

4.1.4. Introducción de la unión

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

```
-- Regla de introducción de la unión
-- =================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
```

```
--     A ⊆ A ∪ B
-- ----------------------------------------------------

import data.set

variable  U : Type
variables A B : set U
variable  x : U

open set

-- #reduce x ∈ A ∪ B

-- 1ª demostración
example : A ⊆ A ∪ B :=
begin
  intros x h,
  simp,
  left,
  exact h,
end

-- 2ª demostración
example : A ⊆ A ∪ B :=
begin
  intros x h,
  left,
  exact h,
end

-- 3ª demostración
example : A ⊆ A ∪ B :=
assume x,
assume h : x ∈ A,
show x ∈ A ∪ B, from or.inl h

-- 4ª demostración
example : A ⊆ A ∪ B :=
assume x,
assume h : x ∈ A,
or.inl h

-- 5ª demostración
example : A ⊆ A ∪ B :=
assume x,
```

```
λ h : x ∈ A, or.inl h

-- 6ª demostración
example : A ⊆ A ∪ B :=
assume x, or.inl

-- 7ª demostración
example : A ⊆ A ∪ B :=
λ x, or.inl

-- 8ª demostración
example : A ⊆ A ∪ B :=
-- by library_search
subset_union_left A B

-- 9ª demostración
example : A ⊆ A ∪ B :=
λ x, mem_union_left B

-- 10ª demostración
example : A ⊆ A ∪ B :=
-- by hint
by finish

-- 11ª demostración
example : A ⊆ A ∪ B :=
by simp
```

4.1.5. El conjunto vacío

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

```
-- Regla del conjunto vacío
-- ========================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    ∅ ⊆ A
-- ----------------------------------------------------

import data.set
```

```
variable  U : Type
variables A : set U
variable  x : U

open set

-- #reduce (∅ : set U)
-- #reduce x ∈ (∅ : set U)

-- 1ª demostración
example : ∅ ⊆ A :=
begin
  intros x h,
  simp at h,
  exfalso,
  exact h,
end

-- 2ª demostración
example : ∅ ⊆ A :=
begin
  intros x h,
  exfalso,
  exact h,
end

-- 3ª demostración
example : ∅ ⊆ A :=
assume x,
assume h : x ∈ (∅ : set U),
show x ∈ A, from false.elim h

-- 4ª demostración
example : ∅ ⊆ A :=
λ x, λ h, false.elim h

-- 5ª demostración
example : ∅ ⊆ A :=
λ _, false.elim

-- 6ª demostración
example : ∅ ⊆ A :=
-- by library_search
empty_subset A
```

```
-- 7ª demostración
example : ∅ ⊆ A :=
assume x,
assume h : x ∈ (∅ : set U),
show x ∈ A, from absurd h (not_mem_empty x)

-- 8ª demostración
example : ∅ ⊆ A :=
λ x h, absurd h (not_mem_empty x)

-- 9ª demostración
example : ∅ ⊆ A :=
-- by hint
by tauto

-- 10ª demostración
example : ∅ ⊆ A :=
by finish

-- 11ª demostración
example : ∅ ⊆ A :=
by simp
```

4.1.6. Diferencia de conjuntos: $A \setminus B \subseteq A$

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

```
-- Diferencia de conjuntos: A \ B ⊆ A
-- ==================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    A \ B ⊆ A
-- ----------------------------------------------------

import data.set

variable  U : Type
variables A B : set U
variable  x : U

open set
```

```
-- #reduce (A \ B)
-- #reduce x ∈ A \ B

-- 1ª demostración
example : A \ B ⊆ A :=
begin
  intros x h,
  simp at h,
  exact h.left,
end

-- 2ª demostración
example : A \ B ⊆ A :=
begin
  intros x h,
  exact h.left,
end

-- 3ª demostración
example : A \ B ⊆ A :=
assume x,
assume h : x ∈ A \ B,
show x ∈ A, from h.left

-- 4ª demostración
example : A \ B ⊆ A :=
assume x,
assume h : x ∈ A \ B,
and.left h

-- 5ª demostración
example : A \ B ⊆ A :=
assume x,
λ h, and.left h

-- 6ª demostración
example : A \ B ⊆ A :=
assume x, and.left

-- 7ª demostración
example : A \ B ⊆ A :=
λ _, and.left

-- 8ª demostración
```

```
example : A \ B ⊆ A :=
-- by library_search
diff_subset A B

-- 9ª demostración
example : A \ B ⊆ A :=
assume x,
assume h : x ∈ A \ B,
show x ∈ A, from mem_of_mem_diff h

-- 10ª demostración
example : A \ B ⊆ A :=
λ _, mem_of_mem_diff

-- 11ª demostración
example : A \ B ⊆ A :=
by finish [subset_def]
```

4.1.7. Complementario de un conjunto: Pruebas de $A \setminus B \subseteq B^c$

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

```
-- Complementario de un conjunto: Pruebas de A \ B ⊆ Bᶜ
-- ====================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    A \ B ⊆ Bᶜ
-- ----------------------------------------------------

import data.set

variable  {U : Type}
variables A B : set U
variable  x : U

open set

-- #reduce x ∈ Bᶜ
-- #reduce Bᶜ
```

```
-- 1ª demostración
example : A \ B ⊆ Bᶜ :=
begin
  intros x h,
  simp at *,
  exact h.right,
end

-- 2ª demostración
example : A \ B ⊆ Bᶜ :=
begin
  intros x h,
  exact h.right,
end

-- 3ª demostración
example : A \ B ⊆ Bᶜ :=
assume x,
assume h1 : x ∈ A \ B,
have h2 : x ∉ B, from and.right h1,
show x ∈ Bᶜ,     from h2

-- 4ª demostración
example : A \ B ⊆ Bᶜ :=
assume x,
assume h1 : x ∈ A \ B,
show x ∈ Bᶜ, from and.right h1

-- 5ª demostración
example : A \ B ⊆ Bᶜ :=
assume x,
λ h1, and.right h1

-- 6ª demostración
example : A \ B ⊆ Bᶜ :=
assume x,
and.right

-- 7ª demostración
example : A \ B ⊆ Bᶜ :=
λ _, and.right

-- 8ª demostración
example : A \ B ⊆ Bᶜ :=
```

```
λ _, not_mem_of_mem_diff
```

4.1.8. Pruebas de la conmutatividad de la intersección

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

```
-- Pruebas de la conmutatividad de la intersección
-- ===============================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    A ∩ B ⊆ B ∩ A
-- ----------------------------------------------------

import data.set

variable  {U : Type}
variables A B : set U
variable  x : U

open set

-- 1ª demostración
example : A ∩ B ⊆ B ∩ A :=
begin
  intros x h,
  simp at *,
  split,
  { exact h.right, },
  { exact h.left,  },
end

-- 2ª demostración
example : A ∩ B ⊆ B ∩ A :=
begin
  intros x h,
  split,
  { exact h.right, },
  { exact h.left,  },
end

-- 3ª demostración
example : A ∩ B ⊆ B ∩ A :=
```

```
begin
  rintros x ⟨h1, h2⟩,
  split,
  { exact h2, },
  { exact h1, },
end

-- 4ª demostración
example : A ∩ B ⊆ B ∩ A :=
begin
  rintros x ⟨h1, h2⟩,
  exact ⟨h2, h1⟩,
end

-- 5ª demostración
example : A ∩ B ⊆ B ∩ A :=
assume x,
assume h : x ∈ A ∩ B,
have h1 : x ∈ A, from and.left h,
have h2 : x ∈ B, from and.right h,
show x ∈ B ∩ A,  from and.intro h2 h1

-- 6ª demostración
example : A ∩ B ⊆ B ∩ A :=
assume x,
assume h : x ∈ A ∩ B,
have h1 : x ∈ A ∧ x ∈ B, from h,
have h2 : x ∈ B ∧ x ∈ A, from and.comm.mp h1,
show x ∈ B ∩ A,          from h2

-- 7ª demostración
example : A ∩ B ⊆ B ∩ A :=
assume x,
assume h : x ∈ A ∩ B,
show x ∈ B ∩ A, from and.comm.mp h

-- 8ª demostración
example : A ∩ B ⊆ B ∩ A :=
assume x,
assume h : x ∈ A ∩ B,
and.comm.mp h

-- 9ª demostración
example : A ∩ B ⊆ B ∩ A :=
```

```
assume x,
λ h, and.comm.mp h

-- 10ª demostración
example : A ∩ B ⊆ B ∩ A :=
assume x,
and.comm.mp

-- 10ª demostración
example : A ∩ B ⊆ B ∩ A :=
λ _, and.comm.mp

-- 11ª demostración
example : A ∩ B ⊆ B ∩ A :=
-- by hint
by finish

-- 12ª demostración
lemma aux : A ∩ B ⊆ B ∩ A :=
by simp

-- -----------------------------------------------------
-- Ej. 2. Demostrar
--    A ∩ B = B ∩ A
-- -----------------------------------------------------

-- 1ª demostración
example : A ∩ B = B ∩ A :=
begin
  apply eq_of_subset_of_subset,
  { exact aux A B, },
  { exact aux B A, },
end

-- 2ª demostración
example : A ∩ B = B ∩ A :=
eq_of_subset_of_subset (aux A B) (aux B A)

-- 3ª demostración
example : A ∩ B = B ∩ A :=
-- by library_search
inter_comm A B

-- 4ª demostración
example : A ∩ B = B ∩ A :=
```

```
-- by hint
by finish
```

4.2. Identidades conjuntistas

4.2.1. Pruebas de la propiedad distributiva de la intersección sobre la unión.

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

```
-- Pruebas de A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)
-- ===========================================

import data.set
open set

variable  {U : Type}
variables A B C : set U

-- ------------------------------------------------------
-- Ej. 1. Demostrar
--    A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C)
-- ------------------------------------------------------

-- 1ª demostración
example :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
begin
  intros x h,
  cases h with ha hbc,
  cases hbc with hb hc,
  { left,
    split,
    { exact ha, },
    { exact hb, }},
  { right,
    split,
    { exact ha, },
    { exact hc, }},
end
```

```
-- 2ª demostración
example :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
begin
  intros x h,
  cases h with ha hbc,
  cases hbc with hb hc,
  { left,
    split,
    { assumption, },
    { assumption, }},
  { right,
    split,
    { assumption, },
    { assumption, }},
end

-- 3ª demostración
example :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
begin
  intros x h,
  cases h with ha hbc,
  cases hbc with hb hc,
  { left,
    split,
    assumption', },
  { right,
    split,
    assumption', },
end

-- 4ª demostración
example :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
begin
  rintros x ⟨ha, (hb | hc)⟩,
  { left,
    split,
    assumption', },
  { right,
    split,
    assumption', },
end
```

```
-- 5ª demostración
example :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
assume x,
assume h : x ∈ A ∩ (B ∪ C),
have x ∈ A, from and.left h,
have x ∈ B ∪ C, from and.right h,
or.elim (‹x ∈ B ∪ C›)
  ( assume : x ∈ B,
    have x ∈ A ∩ B, from and.intro ‹x ∈ A› ‹x ∈ B›,
    show x ∈ (A ∩ B) ∪ (A ∩ C), from or.inl this)
  ( assume : x ∈ C,
    have x ∈ A ∩ C, from and.intro ‹x ∈ A› ‹x ∈ C›,
    show x ∈ (A ∩ B) ∪ (A ∩ C), from or.inr this)

-- 6ª demostración
lemma inter_union_l1 :
  A ∩ (B ∪ C) ⊆ (A ∩ B) ∪ (A ∩ C) :=
assume x,
assume h : x ∈ A ∩ (B ∪ C),
have ha : x ∈ A, from and.left h,
have hbc : x ∈ B ∪ C, from and.right h,
or.elim hbc
  ( assume hb : x ∈ B,
    have hab: x ∈ A ∩ B, from and.intro ha hb,
    show x ∈ (A ∩ B) ∪ (A ∩ C), from or.inl hab)
  ( assume hc : x ∈ C,
    have hac : x ∈ A ∩ C, from and.intro ha hc,
    show x ∈ (A ∩ B) ∪ (A ∩ C), from or.inr hac)

-- ----------------------------------------------------
-- Ej. 2. Demostrar
--    (A ∩ B) ∪ (A ∩ C) ⊆ A ∩ (B ∪ C)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (A ∩ B) ∪ (A ∩ C) ⊆ A ∩ (B ∪ C) :=
begin
  intros x h,
  cases h with hab hac,
  { split,
    { exact hab.left, },
```

```
    { left,
      exact hab.right, }},
  { split,
    { exact hac.left, },
    { right,
      exact hac.right, }},
end

-- 2ª demostración
example :
  (A ∩ B) ∪ (A ∩ C) ⊆ A ∩ (B ∪ C) :=
begin
  rintros x (⟨ha, hb⟩ | ⟨ha, hc⟩),
  { split,
    { exact ha, },
    { left,
      exact hb, }},
  { split,
    { exact ha, },
    { right,
      exact hc, }},
end

-- 3ª demostración
lemma inter_union_l2 :
  (A ∩ B) ∪ (A ∩ C) ⊆ A ∩ (B ∪ C) :=
assume x,
assume : x ∈ (A ∩ B) ∪ (A ∩ C),
or.elim this
  ( assume h : x ∈ A ∩ B,
    have x ∈ A, from and.left h,
    have x ∈ B, from and.right h,
    have x ∈ B ∪ C, from or.inl this,
    show x ∈ A ∩ (B ∪ C), from and.intro ‹x ∈ A› this)
  ( assume h : x ∈ A ∩ C,
    have x ∈ A, from and.left h,
    have x ∈ C, from and.right h,
    have x ∈ B ∪ C, from or.inr this,
    show x ∈ A ∩ (B ∪ C), from and.intro ‹x ∈ A› this)

-- ---------------------------------------------------
-- Ej. 3. Demostrar
--    (A ∩ B) ∪ (A ∩ C) = A ∩ (B ∪ C)
-- ---------------------------------------------------
```

```
-- 1ª demostración
example :
  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) :=
-- by library_search
inter_distrib_left A B C

-- 2ª demostración
theorem inter_union :
  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) :=
eq_of_subset_of_subset
  (inter_union_l1 A B C)
  (inter_union_l2 A B C)

-- 3ª demostración
example :
  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) :=
begin
  ext,
  simp,
  exact and_or_distrib_left,
end

-- 4ª demostración
example :
  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) :=
begin
  ext,
  exact and_or_distrib_left,
end

-- 5ª demostración
example :
  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) :=
ext (λ x, and_or_distrib_left)
```

4.2.2. Pruebas de $(A \cap B^c) \cup B = A \cup B$

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

```
-- Pruebas de (A ∩ Bᶜ) ∪ B = A ∪ B
-- ===============================
```

```
-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    (A ∩ Bᶜ) ∪ B = A ∪ B
-- ----------------------------------------------------

import data.set

open set

variable  U : Type
variables A B C : set U

-- 1ª demostración
-- ===============

example : (A ∩ Bᶜ) ∪ B = A ∪ B :=
calc
  (A ∩ Bᶜ) ∪ B = (A ∪ B) ∩ (Bᶜ ∪ B) : by rw union_distrib_right
           ... = (A ∪ B) ∩ univ     : by rw compl_union_self
           ... = A ∪ B              : by rw inter_univ

example : (A ∩ B) ∪ C = (A ∪ C) ∩ (B ∪ C) :=
-- by library_search
union_distrib_right A B C

example : Bᶜ ∪ B = univ :=
-- by library_search
compl_union_self B

example : A ∩ univ = A :=
-- by library_search
inter_univ A

-- 2ª demostración
-- ===============

example : (A ∩ Bᶜ) ∪ B = A ∪ B :=
begin
  rw union_distrib_right,
  rw compl_union_self,
  rw inter_univ,
end

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

```
example : (A ∩ Bᶜ) ∪ B = A ∪ B :=
by rw [union_distrib_right, compl_union_self, inter_univ]

-- 4ª demostración
-- ===============

example : (A ∩ Bᶜ) ∪ B = A ∪ B :=
by simp [union_distrib_right]
```

4.3. Familias de conjuntos

4.3.1. Unión e intersección de familias de conjuntos

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

```
-- Unión e intersección de familias de conjuntos
-- =============================================

-- ----------------------------------------------------
-- Ej. 1. Declarar I y U como variables de tipo.
-- ----------------------------------------------------

variables {I U : Type}

-- ----------------------------------------------------
-- Ej. 2. Definir la función
--    Union : (I → set U) → set U
-- tal que (Union A) es la unión de de los conjuntos de
-- la familia A.
-- ----------------------------------------------------

def Union (A : I → set U) : set U :=
  { x | ∃ i : I, x ∈ A i }

-- ----------------------------------------------------
-- Ej. 3. Definir la función
--    Inter : (I → set U) → set U
-- tal que (Inter A) es la intersección de de los
-- conjuntos de la familia A.
-- ----------------------------------------------------
```

```
def Inter (A : I → set U) : set U :=
  { x | ∀ i : I, x ∈ A i }

-- ----------------------------------------------------
-- Ej. 4. Declarar
-- + x como una variable sobre U y
-- + A como una variable sobre familas de conjuntos de
--   U con índice en A.
-- ----------------------------------------------------

variable x : U
variable (A : I → set U)

-- ----------------------------------------------------
-- Ej. 5. Demostrar que
--    x ∈ Union A ⊢ ∃ i, x ∈ A i
-- ----------------------------------------------------

example
  (h : x ∈ Union A)
  : ∃ i, x ∈ A i :=
h

-- ----------------------------------------------------
-- Ej. 6. Demostrar que
--    x ∈ x ∈ Inter A ⊢ ∀ i, x ∈ A i
-- ----------------------------------------------------

example
  (h : x ∈ Inter A)
  : ∀ i, x ∈ A i :=
h

-- ----------------------------------------------------
-- Ej 7. Usar (⋃ i, A i) como notación para (Union A).
-- ----------------------------------------------------

notation `⋃` binders `, ` r:(scoped f, Union f) := r

-- ----------------------------------------------------
-- Ej 8. Usar (⋂ i, A i) como notación para (Inter A).
-- ----------------------------------------------------

notation `⋂` binders `, ` r:(scoped f, Inter f) := r
```

```
-- ----------------------------------------------------
-- Ej. 9. Demostrar que
--    x ∈ ⋃ i, A i ⊢ ∃ i, x ∈ A i
-- ----------------------------------------------------

example
  (h : x ∈ ⋃ i, A i)
  : ∃ i, x ∈ A i :=
h

-- ----------------------------------------------------
-- Ej. 10. Demostrar que
--    x ∈ x ∈ ⋂ i, A i ⊢ ∀ i, x ∈ A i
-- ----------------------------------------------------

example
  (h : x ∈ ⋂ i, A i)
  : ∀ i, x ∈ A i :=
h
```

4.3.2. Pertenencia a uniones e intersecciones de familias

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

```
-- Pertenencia a uniones e intersecciones de familias
-- ==================================================

import data.set

open set

variables {I U : Type}
variables {A : I → set U}
variable  {x : U}

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--    (x ∈ ⋃ i, A i) ↔ (∃ i, x ∈ A i)
-- ----------------------------------------------------
```

```
-- 1ª demostración
example :
  (x ∈ ⋃ i, A i) ↔ (∃ i, x ∈ A i) :=
-- by library_search
mem_Union

-- 2ª demostración
example :
  (x ∈ ⋃ i, A i) ↔ (∃ i, x ∈ A i) :=
by simp

-- ----------------------------------------------------
-- Ej. 2. Demostrar que
--    (x ∈ ⋂ i, A i) ↔ (∀ i, x ∈ A i)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (x ∈ ⋂ i, A i) ↔ (∀ i, x ∈ A i) :=
-- by library_search
mem_Inter

-- 2ª demostración
example :
  (x ∈ ⋂ i, A i) ↔ (∀ i, x ∈ A i) :=
by simp
```

4.3.3. Pruebas de la distributiva de la intersección general sobre la intersección

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

```
-- Pruebas de la distributiva de la intersección general sobre la intersección
-- ===========================================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i)
-- ----------------------------------------------------

import data.set
import tactic
```

```
open set

variables {I U : Type}
variables {A B : I → set U}

-- 1ª demostración
example :
  (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
begin
  ext,
  split,
  { intro h,
    rw mem_Inter at h,
    split,
    { rw mem_Inter,
      intro i,
      exact (h i).left, },
    { rw mem_Inter,
      intro i,
      exact (h i).right, }},
  { rintro ⟨h1, h2⟩,
    rw mem_Inter at *,
    intro i,
    exact ⟨h1 i, h2 i⟩, },
end

-- 2ª demostración
example :
  (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
ext $
assume x : U,
iff.intro
( assume h : x ∈ ⋂ i, A i ∩ B i,
  have h1 : ∀ i, x ∈ A i ∩ B i,
    from mem_Inter.mp h,
  have h2 : ∀ i, x ∈ A i,
    from assume i, and.left (h1 i),
  have h3 : ∀ i, x ∈ B i,
    from assume i, and.right (h1 i),
  have h4 : x ∈ ⋂ i, A i,
    from mem_Inter.mpr h2,
  have h5 : x ∈ ⋂ i, B i,
    from mem_Inter.mpr h3,
  show x ∈ (⋂ i, A i) ∩ (⋂ i, B i),
```

```
    from and.intro h4 h5)
( assume h : x ∈ (⋂ i, A i) ∩ (⋂ i, B i),
  have h1 : ∀ i, x ∈ A i,
    from mem_Inter.mp (and.left h),
  have h2 : ∀ i, x ∈ B i,
    from mem_Inter.mp (and.right h),
  have h3 : ∀ i, x ∈ A i ∩ B i,
    from assume i, and.intro (h1 i) (h2 i),
  show x ∈ ⋂ i, A i ∩ B i,
    from mem_Inter.mpr h3)

-- 3ª demostración
example :
  (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
-- by library_search
Inter_inter_distrib A B

-- 4ª demostración
example :
  (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i) :=
ext (by finish)
```

4.3.4. Reglas de la intersección general

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

```
-- Reglas de la intersección general
-- ==================================

import data.set
open set

section
variables {I U : Type}
variables {A : I → set U}
variable  {x : U}

-- Regla de introducción de la intersección
-- ========================================

-- 1ª demostración
example
```

```
  (h : ∀ i, x ∈ A i)
  : x ∈ ⋂ i, A i :=
begin
  simp,
  assumption,
end

-- 2ª demostración
theorem Inter.intro
  (h : ∀ i, x ∈ A i)
  : x ∈ ⋂ i, A i :=
by simp; assumption

-- Regla de eliminación de la intersección
-- ========================================

-- 1ª demostración
example
  (h : x ∈ ⋂ i, A i)
  (i : I)
  : x ∈ A i :=
begin
  simp at h,
  apply h,
end

-- 2ª demostración
@[elab_simple]
theorem Inter.elim
  (h : x ∈ ⋂ i, A i)
  (i : I)
  : x ∈ A i :=
by simp at h; apply h

end
```

4.3.5. Reglas de la unión general

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

```
-- Reglas de la unión general
-- ==========================
```

```
import data.set

open set

variables {I U : Type}
variables {A : I → set U}
variable  {x : U}
variable  (i : I)

-- Regla de introducción de la unión
-- ==================================

-- 1ª demostración
example
  (h : x ∈ A i)
  : x ∈ ⋃ i, A i :=
begin
  simp,
  existsi i,
  exact h
end

-- 2ª demostración
theorem Union.intro
  (h : x ∈ A i)
  : x ∈ ⋃ i, A i :=
by {simp, existsi i, exact h}

-- Regla de eliminación de la unión
-- =================================

-- 1ª demostración
example
  {b : Prop}
  (h₁ : x ∈ ⋃ i, A i)
  (h₂ : ∀ (i : I), x ∈ A i → b)
  : b :=
begin
  simp at h₁,
  cases h₁ with i h,
  exact h₂ i h,
end

-- 2ª demostración
```

```
theorem Union.elim
  {b : Prop}
  (h₁ : x ∈ ⋃ i, A i)
  (h₂ : ∀ (i : I), x ∈ A i → b)
  : b :=
by {simp at h₁, cases h₁ with i h, exact h₂ i h}
```

4.3.6. Pruebas de intersección sobre unión general

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

```
-- Pruebas de intersección sobre unión general
-- ===========================================

import data.set

open set

variables {I U : Type}
variables {A : I → set U}
variable  {C : set U}

-- ------------------------------------------------------
-- Ej. 1. Demostrar
--    C ∩ (⋃i, A i) ⊆ (⋃ i, C ∩ A i)
-- ------------------------------------------------------

-- 1ª demostración
example :
  C ∩ (⋃i, A i) ⊆ (⋃ i, C ∩ A i)  :=
begin
  rintros x ⟨hC, hU⟩,
  rw mem_Union at hU,
  cases hU with i hA,
  apply mem_Union.mpr,
  use i,
  split,
  assumption',
end

-- 2ª demostración
example :
```

```
  C ∩ (⋃i, A i) ⊆ (⋃ i, C ∩ A i)  :=
begin
  intros x h,
  simp * at *,
end

-- 3ª demostración
lemma inter_Uni_l1 :
  C ∩ (⋃i, A i) ⊆ (⋃ i, C ∩ A i)  :=
by {intros x h, simp * at *}

-- ----------------------------------------------------
-- Ej. 2. Demostrar
--    (⋃ i, C ∩ A i) ⊆ C ∩ (⋃i, A i)
-- ----------------------------------------------------

-- 1ª demostración
example :
  (⋃ i, C ∩ A i) ⊆ C ∩ (⋃i, A i) :=
begin
  intros x h,
  rw mem_Union at h,
  cases h with i hi,
  cases hi with hC hA,
  split,
  { exact hC, },
  { apply mem_Union.mpr,
    use i,
    exact hA, },
end

-- 2ª demostración
example : (⋃ i, C ∩ A i) ⊆ C ∩ (⋃i, A i) :=
begin
  intros x h,
  rw mem_Union at h,
  rcases h with ⟨i, hC, hA⟩,
  split,
  { exact hC, },
  { apply mem_Union.mpr,
    use i,
    exact hA, },
end

-- 3ª demostración
```

```
example :
  (⋃ i, C ∩ A i) ⊆ C ∩ (⋃i, A i) :=
begin
  intros x h,
  simp * at *,
end

-- 4ª demostración
lemma inter_Uni_l2 :
  (⋃ i, C ∩ A i) ⊆ C ∩ (⋃i, A i) :=
by {intros x h, simp * at *}

-- ------------------------------------------------------
-- Ej. 3. Demostrar
--    C ∩ (⋃i, A i) = (⋃ i, C ∩ A i)
-- ------------------------------------------------------

-- 1ª demostración
example :
  C ∩ (⋃i, A i) = (⋃ i, C ∩ A i) :=
eq_of_subset_of_subset inter_Uni_l1 inter_Uni_l2

-- 2ª demostración
example :
  C ∩ (⋃i, A i) = (⋃ i, C ∩ A i) :=
-- by library_search
inter_Union C A

-- 3ª demostración
example :
  C ∩ (⋃i, A i) = (⋃ i, C ∩ A i) :=
ext $ by simp

-- 4ª demostración
example :
  C ∩ (⋃i, A i) = (⋃ i, C ∩ A i) :=
by {ext, simp}
```

4.3.7. Pruebas de (⋃i, ⋂j, A i j) ⊆ (⋂j, ⋃i, A i j)

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

```
-- Pruebas de (⋃i, ⋂j, A i j) ⊆ (⋂j, ⋃i, A i j)
-- ============================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar
--    (⋃i, ⋂j, A i j) ⊆ (⋂j, ⋃i, A i j)
-- ----------------------------------------------------

import data.set

open set

variables {I J U : Type}
variables (A : I → J → set U)

-- 1ª demostración
example : (⋃i, ⋂j, A i j) ⊆ (⋂j, ⋃i, A i j) :=
begin
  intros x h,
  rw mem_Union at h,
  cases h with i hi,
  rw mem_Inter at hi,
  apply mem_Inter.mpr,
  intro j,
  apply mem_Union.mpr,
  use i,
  exact (hi j),
end

-- 2ª demostración
example : (⋃i, ⋂j, A i j) ⊆ (⋂j, ⋃i, A i j) :=
begin
  intros x h,
  simp * at *,
  cases h with i hi,
  intro j,
  use i,
  exact (hi j),
end
```

4.4. Conjunto potencia

4.4.1. Definición del conjunto potencia

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

```
-- Definición del conjunto potencia
-- ================================

variable {U : Type}

def powerset (A : set U) : set (set U) :=
  {B : set U | B ⊆ A}

example
  (A B : set U)
  (h : B ∈ powerset A)
  : B ⊆ A :=
h
```

4.4.2. Pruebas de A ∈ ℘ (A ∪ B)

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

```
-- Pruebas de A ∈ ℘ (A ∪ B)
-- ========================

import data.set
open set

variable  {U : Type}
variables (A B : set U)

#reduce powerset A
#reduce B ∈ powerset A
#reduce ℘ A
#reduce B ∈ ℘ A

-- ?ª demostración
example : A ∈ ℘ (A ∪ B) :=
begin
```

```
  intros x h,
  simp,
  left,
  exact h,
end

-- ?ª demostración
example : A ⊆ ℘ (A ∪ B) :=
begin
  intros x h,
  exact or.inl h,
end

-- ?ª demostración
example : A ⊆ ℘ (A ∪ B) :=
λ x, or.inl

-- ?ª demostración
example : A ⊆ ℘ (A ∪ B) :=
assume x,
assume : x ∈ A,
show x ∈ A ∪ B, from or.inl ‹x ∈ A›
```

4.4.3. Monotonía del conjunto potencia: $\wp$ A $\subseteq$ $\wp$ B $\leftrightarrow$ A $\subseteq$ B

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

```
-- Monotonía del conjunto potencia: ℘ A ⊆ ℘ B ↔ A ⊆ B
-- ====================================================

import data.set
open set

variable  {U : Type}
variables {A B C : set U}

-- #reduce ℘ A
-- #reduce B ∈ ℘ A

-- ----------------------------------------------------
-- Ej. 1. Demostrar
```

```
--    ℘ A ⊆ ℘ B → A ⊆ B
-- ----------------------------------------------------

-- 1ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
begin
  intro h,
  apply subset_of_mem_powerset,
  apply h,
  apply mem_powerset,
  exact subset.rfl,
end

-- 2ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
begin
  intro h,
  apply h,
  exact subset.rfl,
end

-- 3ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
begin
  intro h,
  exact (h subset.rfl),
end

-- 4ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
λ h, h subset.rfl

-- 5ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
assume h1 : ℘ A ⊆ ℘ B,
have h2 : A ⊆ A, from subset.rfl,
have h3 : A ∈ ℘ A, from h2,
have h4 : A ∈ ℘ B, from h1 h3,
show A ⊆ B, from h4

-- 6ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
assume h1 : ℘ A ⊆ ℘ B,
have h2 : A ⊆ A, from subset.rfl,
```

```
have h3 : A ∈ ℘ A, from h2,
h1 h3

-- 7ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
assume h1 : ℘ A ⊆ ℘ B,
have h2 : A ⊆ A, from subset.rfl,
h1 h2

-- 8ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
assume h1 : ℘ A ⊆ ℘ B,
h1 subset.rfl

-- 9ª demostración
lemma aux1 : ℘ A ⊆ ℘ B → A ⊆ B :=
λ h, h subset.rfl

-- 10ª demostración
example : ℘ A ⊆ ℘ B → A ⊆ B :=
powerset_mono.mp

-- ----------------------------------------------------
-- Ej. 2. Demostrar
--    A ⊆ B → ℘ A ⊆ ℘ B
-- ----------------------------------------------------

-- 1ª demostración
example : A ⊆ B → ℘ A ⊆ ℘ B :=
begin
  intro h,
  intros C hCA,
  apply mem_powerset,
  apply subset.trans hCA h,
end

-- 2ª demostración
example : A ⊆ B → ℘ A ⊆ ℘ B :=
begin
  intros h C hCA,
  apply subset.trans hCA h,
end

-- 3ª demostración
```

```
lemma aux2 : A ⊆ B → ℘ A ⊆ ℘ B :=
λ h C hCA, subset.trans hCA h

-- 4ª demostración
example : A ⊆ B → ℘ A ⊆ ℘ B :=
powerset_mono.mpr

-- ----------------------------------------------------
-- Ej. 3. Demostrar
--     ℘ A ⊆ ℘ B ↔ A ⊆ B
-- ----------------------------------------------------

-- 1ª demostración
example : ℘ A ⊆ ℘ B ↔ A ⊆ B :=
iff.intro aux1 aux2

-- 2ª demostración
example : ℘ A ⊆ ℘ B ↔ A ⊆ B :=
-- by library_search
powerset_mono

-- 3ª demostración
example : ℘ A ⊆ ℘ B ↔ A ⊆ B :=
-- by hint
by finish

-- 4ª demostración
example : ℘ A ⊆ ℘ B ↔ A ⊆ B :=
by simp
```

Capítulo 5

Relaciones

5.1. Relaciones de orden

5.1.1. Las irreflexivas y transitivas son asimétricas

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

```
-- Las irreflexivas y transitivas son asimétricas
-- ==============================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que las relaciones irreflexivas y
-- transitivas son asimétricas.
-- ----------------------------------------------------

variable A : Type
variable R : A → A → Prop

-- #reduce irreflexive R
-- #reduce transitive R

-- 1ª demostración
example
  (h1 : irreflexive R)
  (h2 : transitive R)
  : ∀ x y, R x y → ¬ R y x :=
begin
  intros x y h3 h4,
  apply h1 x,
  apply h2 h3 h4,
end
```

```
-- 2ª demostración
example
  (h1 : irreflexive R)
  (h2 : transitive R)
  : ∀ x y, R x y → ¬ R y x :=
begin
  intros x y h3 h4,
  apply (h1 x) (h2 h3 h4),
end

-- 3ª demostración
example
  (h1 : irreflexive R)
  (h2 : transitive R)
  : ∀ x y, R x y → ¬ R y x :=
λ x y h3 h4, (h1 x) (h2 h3 h4)

-- 4ª demostración
example
  (h1 : irreflexive R)
  (h2 : transitive R)
  : ∀ x y, R x y → ¬ R y x :=
assume x y,
assume h3 : R x y,
assume h4 : R y x,
have h5 : R x x, from h2 h3 h4,
have h6 : ¬ R x x, from h1 x,
show false, from h6 h5
```

5.1.2. Las partes estrictas son irreflexivas

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

```
-- Las partes estrictas son irreflexivas
-- =====================================

-- ------------------------------------------------------
-- Ej. 1. La parte estricta de una relación R es la
-- relación R' definida por
--    R' a b := R a b ∧ a ≠ b
--
-- Demostrar que la parte estricta de cualquier
```

```
-- relación es irreflexiva.
-- ----------------------------------------------------

import tactic

section

parameter {A : Type}
parameter (R : A → A → Prop)

definition R' (a b : A) : Prop :=
  R a b ∧ a ≠ b

#reduce irreflexive R

-- 1ª demostración
example :
  irreflexive R' :=
begin
  intros a h,
  cases h with h1 h2,
  apply h2,
  refl,
end

-- 2ª demostración
example :
  irreflexive R' :=
assume a,
assume : R' a a,
have a ≠ a, from and.right this,
have a = a, from rfl,
show false, from ‹a ≠ a› ‹a = a›

end
```

5.1.3. Las partes estrictas de los órdenes parciales son transitivas

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

```
-- Las partes estrictas de los órdenes parciales son transitivas
-- ==============================================================

-- ----------------------------------------------------
-- Ej. 1. La parte estricta de una relación R es la
-- relación R' definida por
--    R' a b := R a b ∧ a ≠ b
--
-- Demostrar que si R es un orden parcial, entonces su
-- parte estricta es transitiva.
-- ----------------------------------------------------

import tactic

section

parameter {A : Type}
parameter (R : A → A → Prop)
parameter (reflR    : reflexive R)
parameter (transR   : transitive R)
parameter (antisimR : anti_symmetric R)
variables {a b c : A}

definition R' (a b : A) : Prop :=
  R a b ∧ a ≠ b

include transR
include antisimR

-- 1ª demostración
example : transitive R' :=
begin
  rintros a b c ⟨h1,h2⟩ ⟨h3,h4⟩,
  split,
  { apply (transR h1 h3), },
  { intro h5,
    apply h4,
    apply (antisimR h3),
    rw ←h5,
    exact h1, },
end

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

```
local infix ≤ := R
local infix < := R'

example : transitive (<) :=
assume a b c,
assume h₁ : a < b,
assume h₂ : b < c,
have a ≤ b, from and.left h₁,
have a ≠ b, from and.right h₁,
have b ≤ c, from and.left h₂,
have b ≠ c, from and.right h₂,
have a ≤ c, from transR ‹a ≤ b› ‹b ≤ c›,
have a ≠ c, from
    assume : a = c,
    have c ≤ b, from eq.subst ‹a = c› ‹a ≤ b›,
    have b = c, from antisimR ‹b ≤ c› ‹c ≤ b›,
    show false, from ‹b ≠ c› ‹b = c›,
show a < c, from and.intro ‹a ≤ c› ‹a ≠ c›

end
```

5.1.4. Las partes simétricas de las reflexivas son reflexivas

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

```
-- Las partes simétricas de las reflexivas son reflexivas
-- ======================================================

-- ----------------------------------------------------
-- Ej. 1. La parte simétrica de una relación R es la
-- relación S definida por
--    S x y := R x y ∧ R y x
--
-- Demostrar que la parte simétrica de una relación
-- reflexiva es reflexiva.
-- ----------------------------------------------------

section

parameter A : Type
parameter R : A → A → Prop
```

```
parameter reflR : reflexive R

include reflR

def S (x y : A) := R x y ∧ R y x

-- 1ª demostración
example : reflexive S :=
begin
  intro x,
  split,
  { exact (reflR x), },
  { exact (reflR x), },
end

-- 2ª demostración
example : reflexive S :=
assume x,
have R x x, from reflR x,
show S x x, from and.intro this this

-- 3ª demostración
example : reflexive S :=
assume x,
show S x x, from and.intro (reflR x) (reflR x)

-- 4ª demostración
example : reflexive S :=
assume x,
and.intro (reflR x) (reflR x)

-- 5ª demostración
example : reflexive S :=
λ x, ⟨reflR x, reflR x⟩

end
```

5.1.5. Las partes simétricas son simétricas

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

```
-- Las partes simétricas son simétricas
-- ====================================

-- ----------------------------------------------------
-- Ej. 1. La parte simétrica de una relación R es la
-- relación S definida por
--    S x y := R x y ∧ R y x
--
-- Demostrar que la parte simétrica de cualquier
-- relación es simétrica.
-- ----------------------------------------------------

section
parameter A : Type
parameter R : A → A → Prop

def S (x y : A) := R x y ∧ R y x

-- 1ª demostración
example : symmetric S :=
begin
  intros x y h,
  split,
  { exact h.right, },
  { exact h.left, },
end

-- 2ª demostración
example : symmetric S :=
begin
  intros x y h,
  exact ⟨h.right, h.left⟩,
end

-- 3ª demostración
example : symmetric S :=
λ x y h, ⟨h.right, h.left⟩

-- 4ª demostración
example : symmetric S :=
assume x y,
assume h : S x y,
have h1 : R x y, from h.left,
have h2 : R y x, from h.right,
show S y x, from ⟨h2, h1⟩
```

```
-- 5ª demostración
example : symmetric S :=
assume x y,
assume h : S x y,
show S y x, from ⟨h.right, h.left⟩

-- 6ª demostración
example : symmetric S :=
λ x y h, ⟨h.right, h.left⟩

end
```

5.2. Órdenes sobre números

5.2.1. Pruebas de n + 1 ≤ m ⊢ n <m + 1

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

```
-- Pruebas de n + 1 ≤ m ⊢ n < m + 1
-- ================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que si n y m son números naturales
-- tales que n + 1 ≤ m, entonces n < m + 1.
-- ----------------------------------------------------

import tactic

variables n m : ℕ

-- 1ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
calc
  n   < n + 1 : lt_add_one n
  ... ≤ m     : h
  ... < m + 1 : lt_add_one m

-- 2ª demostración
example
```

```
  (h : n + 1 ≤ m)
  : n < m + 1 :=
have h1 : n < n + 1, from lt_add_one n,
have h2 : n < m,     from lt_of_lt_of_le h1 h,
have h3 : m < m + 1, from lt_add_one m,
show n < m + 1,      from lt.trans h2 h3

-- 3ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
begin
  apply lt_trans (lt_add_one n),
  apply lt_of_le_of_lt h (lt_add_one m),
end

-- 4ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
lt_trans (lt_add_one n) (lt_of_le_of_lt h (lt_add_one m))

-- 5ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
-- by suggest
nat.lt.step h

-- 6ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
-- by hint
by omega

-- 7ª demostración
example
  (h : n + 1 ≤ m)
  : n < m + 1 :=
by linarith

-- 8ª demostración
example
  (h : n + 1 ≤ m)
```

```
  : n < m + 1 :=
by nlinarith
```

5.3. Relaciones de equivalencia

5.3.1. Las equivalencias son preórdenes simétricos

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

```
-- Las equivalencias son preórdenes simétricos
-- ===========================================

-- ----------------------------------------------------
-- Ej. 1. Un preorden es una relación reflexiva y
-- transitiva.
--
-- Demostrar que las relaciones de equivalencias son
-- los prórdenes simétricos.
-- ----------------------------------------------------

import tactic

variable {A : Type}
variable R : A → A → Prop

def preorden (R : A → A → Prop) : Prop :=
  reflexive R ∧ transitive R

-- #print equivalence
-- #print symmetric

-- 1ª demostración
example :
  equivalence R ↔ preorden R ∧ symmetric R :=
begin
  split,
  { rintros ⟨h1, h2, h3⟩,
    exact ⟨⟨h1, h3⟩, h2⟩, },
  { rintros ⟨⟨h1, h3⟩, h2⟩,
    exact ⟨h1, h2, h3⟩, },
end
```

```
-- 2ª demostración
example :
  equivalence R ↔ preorden R ∧ symmetric R :=
⟨λ ⟨h1, h2, h3⟩, ⟨⟨h1, h3⟩, h2⟩,
 λ ⟨⟨h1, h3⟩, h2⟩, ⟨h1, h2, h3⟩⟩

-- 3ª demostración
example :
  equivalence R ↔ preorden R ∧ symmetric R :=
iff.intro
  ( assume h1 : equivalence R,
    have h2 : reflexive R, from and.left h1,
    have h3 : symmetric R, from and.left (and.right h1),
    have h4 : transitive R, from and.right (and.right h1),
    show preorden R ∧ symmetric R,
      from and.intro (and.intro h2 h4) h3)
  ( assume h1 : preorden R ∧ symmetric R,
    have h2 : preorden R, from and.left h1,
    show equivalence R,
      from and.intro (and.left h2)
             (and.intro (and.right h1) (and.right h2)))

-- 4ª demostración
example :
  equivalence R ↔ preorden R ∧ symmetric R :=
begin
  unfold equivalence preorden,
  tauto,
end

-- 5ª demostración
example :
  equivalence R ↔ preorden R ∧ symmetric R :=
by finish [preorden]
```

5.3.2. Las relaciones reflexivas y euclídeas son de equivalencia

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

```
-- Las relaciones reflexivas y euclídeas son de equivalencia
-- ==========================================================
```

```
-- ----------------------------------------------------
-- Una relación binaria (≈) es euclídea si
--    ∀ {a b c}, a ≈ b → c ≈ b → a ≈ c
--
-- El objetivo de esta teoría es demostrar que si una
-- relación es reflexiva y euclídea, entonces es de
-- equivalencia.
-- ----------------------------------------------------

import tactic

section

parameter {A : Type}
parameter (R : A → A → Prop)

local infix ≈ := R

parameter reflexivaR : reflexive (≈)
parameter euclideaR : ∀ {a b c}, a ≈ b → c ≈ b → a ≈ c

include reflexivaR euclideaR

-- ----------------------------------------------------
-- Ej. 1. Demostrar que las relaciones reflexivas y
-- y euclídeas son simétricas.
-- ----------------------------------------------------

-- 1ª demostración
example : symmetric (≈) :=
begin
  intros a b h,
  exact euclideaR (reflexivaR b) h,
end

-- 2ª demostración
example : symmetric (≈) :=
λ a b h, euclideaR (reflexivaR b) h

-- 3ª demostración
lemma simetricaR : symmetric (≈) :=
assume a b (h1 : a ≈ b),
have h2 : b ≈ b, from (reflexivaR b),
show b ≈ a, from euclideaR h2 h1
```

```
-- ----------------------------------------------------
-- Ej. 2. Demostrar que las relaciones reflexivas y
-- y euclídeas son transitivas.
-- ----------------------------------------------------

-- 1ª demostración
example : transitive (≈) :=
begin
  rintros a b c h1 h2,
  apply euclideaR h1,
  exact euclideaR (reflexivaR c) h2,
end

-- 2ª demostración
lemma transitivaR : transitive (≈) :=
λ a b c h1 h2, (euclideaR h1) (euclideaR (reflexivaR c) h2)

-- 3ª demostración
example : transitive (≈) :=
assume a b c (h1 : a ≈ b) (h2 : b ≈ c),
have h3 : c ≈ b, from euclideaR (reflexivaR c) h2,
show a ≈ c, from euclideaR h1 h3

-- ----------------------------------------------------
-- Ej. 3. Demostrar que las relaciones reflexivas y
-- y euclídeas son de equivalencia.
-- ----------------------------------------------------

-- 1ª demostración
example : equivalence (≈) :=
begin
  unfold equivalence,
  exact ⟨reflexivaR, simetricaR, transitivaR⟩,
end

-- 2ª demostración
example : equivalence (≈) :=
⟨reflexivaR, simetricaR, transitivaR⟩

end
```

Capítulo 6

Funciones

6.1. Funciones en Lean

6.1.1. Definición de la composición de funciones

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

```
-- Definición de la composición de funciones
-- =========================================

-- ----------------------------------------------------
-- Ej. 1. Definir la composición de dos funciones.
-- ----------------------------------------------------

namespace reservado

variables {X Y Z : Type}

def comp (f : Y → Z) (g : X → Y) : X → Z :=
λ x, f (g x)

infixr ` ∘ ` := comp

end reservado

#print notation ∘
#print function.comp
```

6.1.2. Definición de la función identidad

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

```
-- Definición de la función identidad
-- ==================================

-- ----------------------------------------------------
-- Ej. 1. Definir la función identidad.
-- ----------------------------------------------------

namespace reservado
variable {X : Type}

def id (x : X) : X :=
x

end reservado
```

6.1.3. Extensionalidad funcional

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

```
-- Extensionalidad funcional
-- =========================

-- ----------------------------------------------------
-- Ej. 1. Sean f y g funciones de X en Y. Demostrar que
-- si
--    ∀ (x : X), f x = g x
-- entonces, las funciones f y g son iguales.
-- ----------------------------------------------------

variables {X Y : Type}
variables (f g : X → Y)

example
  (h : ∀ x, f x = g x)
  : f = g :=
funext h
```

6.1.4. Propiedades de la composición de funciones (elemento neutro y asociatividad)

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

```
-- Propiedades de la composición de funciones
-- ==========================================

import tactic

open function

variables {X Y Z W : Type}

-- ----------------------------------------------------
-- Ej. 1. Demostrar que
--    id ∘ f = f
-- ----------------------------------------------------

-- 1ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
begin
  ext,
  calc (id ∘ f) x = id (f x) : by rw comp_app
       ...        = f x      : by rw id.def,
end

-- 2ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
begin
  ext,
  rw comp_app,
  rw id.def,
end

-- 3ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
begin
```

```
  ext,
  rw [comp_app, id.def],
end

-- 4ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
begin
  ext,
  calc (id ∘ f) x = id (f x) : rfl
       ...        = f x      : rfl,
end

-- 5ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
rfl

-- 6ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
-- by library_search
left_id f

-- 7ª demostración
example
  (f : X → Y)
  : id ∘ f = f :=
comp.left_id f

-- ----------------------------------------------------
-- Ej. 2. Demostrar que
--    f ∘ id = f
-- ----------------------------------------------------

-- 1ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
begin
  ext,
  calc (f ∘ id) x = f (id x) : by rw comp_app
```

```
       ...          = f x       : by rw id.def,
end

-- 2ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
begin
  ext,
  rw comp_app,
  rw id.def,
end

-- 3ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
begin
  ext,
  rw [comp_app, id.def],
end

-- 4ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
begin
  ext,
  calc (f ∘ id) x = f (id x) : rfl
       ...          = f x       : rfl,
end

-- 5ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
rfl

-- 6ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
-- by library_search
right_id f
```

```
-- 7ª demostración
example
  (f : X → Y)
  : f ∘ id = f :=
comp.right_id f

-- ----------------------------------------------------
-- Ej. 3. Demostrar que
--    (f ∘ g) ∘ h = f ∘ (g ∘ h)
-- ----------------------------------------------------

-- 1ª demostración
example
  (f : Z → W)
  (g : Y → Z)
  (h : X → Y)
  : (f ∘ g) ∘ h = f ∘ (g ∘ h) :=
begin
  ext,
  calc ((f ∘ g) ∘ h) x
           = (f ∘ g) (h x)   : by rw comp_app
       ... = f (g (h x))     : by rw comp_app
       ... = f ((g ∘ h) x)   : by rw comp_app
       ... = (f ∘ (g ∘ h)) x : by rw comp_app
end

-- 2ª demostración
example
  (f : Z → W)
  (g : Y → Z)
  (h : X → Y)
  : (f ∘ g) ∘ h = f ∘ (g ∘ h) :=
begin
  ext,
  rw comp_app,
end

-- 3ª demostración
example
  (f : Z → W)
  (g : Y → Z)
  (h : X → Y)
  : (f ∘ g) ∘ h = f ∘ (g ∘ h) :=
begin
```

```
  ext,
  calc ((f ∘ g) ∘ h) x
           = (f ∘ g) (h x)   : rfl
       ... = f (g (h x))     : rfl
       ... = f ((g ∘ h) x)   : rfl
       ... = (f ∘ (g ∘ h)) x : rfl
end

-- 4ª demostración
example
  (f : Z → W)
  (g : Y → Z)
  (h : X → Y)
  : (f ∘ g) ∘ h = f ∘ (g ∘ h) :=
rfl

-- 5ª demostración
example
  (f : Z → W)
  (g : Y → Z)
  (h : X → Y)
  : (f ∘ g) ∘ h = f ∘ (g ∘ h) :=
comp.assoc f g h
```

6.1.5. Funciones inyectivas, suprayectivas y biyectivas

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

```
-- Funciones inyectivas, suprayectivas y biyectivas
-- ================================================

-- ----------------------------------------------------
-- Ej. 1. Definir las funciones inyectivas,
-- suprayectivas y biyectivas.
-- ----------------------------------------------------

variables {X Y Z : Type}

def injective (f : X → Y) : Prop :=
∀ ⦃x₁ x₂⦄, f x₁ = f x₂ → x₁ = x₂

def surjective (f : X → Y) : Prop :=
```

```
∀ y, ∃ x, f x = y

def bijective (f : X → Y) : Prop :=
injective f ∧ surjective f
```

6.1.6. La identidad es biyectiva

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

```
-- La identidad es biyectiva
-- =========================

import tactic

open function

variables {X : Type}

-- #print injective
-- #print surjective
-- #print bijective

-- ----------------------------------------------------
-- Ej. 1. Demostrar que la identidad es inyectiva.
-- ----------------------------------------------------

-- 1ª demostración
example : injective (@id X) :=
begin
  intros x₁ x₂ h,
  exact h,
end

-- 2ª demostración
example : injective (@id X) :=
λ x₁ x₂ h, h

-- 3ª demostración
example : injective (@id X) :=
λ x₁ x₂, id

-- 4ª demostración
example : injective (@id X) :=
```

```
assume x₁ x₂,
assume h : id x₁ = id x₂,
show x₁ = x₂, from h

-- 5ª demostración
example : injective (@id X) :=
--by library_search
injective_id

-- 6ª demostración
example : injective (@id X) :=
-- by hint
by tauto

-- 7ª demostración
example : injective (@id X) :=
-- by hint
by finish

-- ----------------------------------------------------
-- Ej. 2. Demostrar que la identidad es suprayectiva.
-- ----------------------------------------------------

-- 1ª demostración
example : surjective (@id X) :=
begin
  intro x,
  use x,
  exact rfl,
end

-- 2ª demostración
example : surjective (@id X) :=
begin
  intro x,
  exact ⟨x, rfl⟩,
end

-- 3ª demostración
example : surjective (@id X) :=
λ x, ⟨x, rfl⟩

-- 4ª demostración
example : surjective (@id X) :=
assume y,
```

```
show ∃ x, id x = y, from exists.intro y rfl

-- 5ª demostración
example : surjective (@id X) :=
-- by library_search
surjective_id

-- 6ª demostración
example : surjective (@id X) :=
-- by hint
by tauto

-- ----------------------------------------------------
-- Ej. 3. Demostrar que la identidad es biyectiva.
-- ----------------------------------------------------

-- 1ª demostración
example : bijective (@id X) :=
and.intro injective_id surjective_id

-- 2ª demostración
example : bijective (@id X) :=
⟨injective_id, surjective_id⟩

-- 3ª demostración
example : bijective (@id X) :=
-- by library_search
bijective_id
```

6.1.7. La composición de funciones inyectivas es inyectiva

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

```
-- La composición de funciones inyectivas es inyectiva
-- ===================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que la composición de dos funciones
-- inyectivas es una función inyectiva.
-- ----------------------------------------------------
```

```
import tactic

open function

variables {X Y Z : Type}
variable  {f : X → Y}
variable  {g : Y → Z}

-- 1ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
begin
  intros x y h,
  apply Hf,
  apply Hg,
  exact h,
end

-- 2ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
begin
  intros x y h,
  apply Hf,
  exact Hg h,
end

-- 3ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
begin
  intros x y h,
  exact Hf (Hg h),
end

-- 4ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
```

```
  : injective (g ∘ f) :=
λ x y h, Hf (Hg h)

-- 5ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
assume x y,
assume h1 : (g ∘ f) x = (g ∘ f) y,
have h2 : f x = f y, from Hg h1,
show x = y, from Hf h2

-- 6ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
assume x y,
assume h1 : (g ∘ f) x = (g ∘ f) y,
show x = y, from Hf (Hg h1)

-- 7ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
assume x y,
assume h1 : (g ∘ f) x = (g ∘ f) y,
Hf (Hg h1)

-- 8ª demostración
example
  (Hf : injective f)
  (Hg : injective g)
  : injective (g ∘ f) :=
λ x y h1, Hf (Hg h1)

-- 9ª demostración
example
  (Hg : injective g)
  (Hf : injective f)
  : injective (g ∘ f) :=
-- by library_search
injective.comp Hg Hf
```

```
-- 10ª demostración
example
  (Hg : injective g)
  (Hf : injective f)
  : injective (g ∘ f) :=
-- by hint
by tauto
```

6.1.8. La composición de funciones suprayectivas es suprayectiva

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

```
-- La composición de funciones suprayectivas es suprayectiva
-- ==========================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que la composición de dos funciones
-- suprayectivas es una función suprayectiva.
-- ----------------------------------------------------

import tactic

open function

variables {X Y Z : Type}
variable  {f : X → Y}
variable  {g : Y → Z}

-- 1ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
begin
  intro z,
  cases hg z with y hy,
  cases hf y with x hx,
  use x,
  simp,
  rw hx,
  exact hy,
```

```
end

-- 2ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
begin
  intro z,
  cases hg z with y hy,
  cases hf y with x hx,
  use x,
  calc (g ∘ f) x = g (f x) : by rw comp_app
             ... = g y     : congr_arg g hx
             ... = z       : hy,
end

-- 3ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
assume z,
exists.elim (hg z)
  ( assume y (hy : g y = z),
    exists.elim (hf y)
    ( assume x (hx : f x = y),
      have g (f x) = z, from eq.subst (eq.symm hx) hy,
      show ∃ x, g (f x) = z, from exists.intro x this))

-- 4ª demostración
example
  (hf : surjective f)
  (hg : surjective g)
  : surjective (g ∘ f) :=
-- by library_search
surjective.comp hg hf

-- 5ª 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, exists.intro x
         (show g (f x) = z,
            from (eq.trans (congr_arg g hx) hy))))
```

6.1.9. La composición de funciones biyectivas es biyectiva

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

```
-- La composición de funciones biyectivas es biyectiva
-- ====================================================

-- ----------------------------------------------------
-- Ej. 1. Demostrar que la composición de dos funciones
-- biyectivas es una función biyectiva.
-- ----------------------------------------------------

import tactic

open function

variables {X Y Z : Type}
variable  {f : X → Y}
variable  {g : Y → Z}

-- 1ª demostración
example
  (hf : bijective f)
  (hg : bijective g)
  : bijective (g ∘ f) :=
begin
  split,
  { apply injective.comp,
    { exact hg.left, },
    { exact hf.left, }},
  { apply surjective.comp,
    { exact hg.right, },
    { exact hf.right, }},
end

-- 2ª demostración
example
```

```
  (hf : bijective f)
  (hg : bijective g)
  : bijective (g ∘ f) :=
begin
  split,
  { exact injective.comp hg.1 hf.1, },
  { exact surjective.comp hg.2 hf.2, },
end

-- 3ª demostración
example
  (hf : bijective f)
  (hg : bijective g)
  : bijective (g ∘ f) :=
⟨injective.comp  hg.1 hf.1,
 surjective.comp hg.2 hf.2⟩

-- 4ª demostración
example
  (hf : bijective f)
  (hg : bijective g)
  : bijective (g ∘ f) :=
have giny : injective g, from hg.left,
have gsupr : surjective g, from hg.right,
have finy : injective f, from hf.left,
have fsupr : surjective f, from hf.right,
show bijective (g ∘ f),
  from and.intro (injective.comp giny finy)
                 (surjective.comp gsupr fsupr)

-- 5ª demostración
example
  (hf : bijective f)
  (hg : bijective g)
  : bijective (g ∘ f) :=
-- by library_search
bijective.comp hg hf

-- 6ª demostración
example :
  bijective f → bijective g → bijective (g ∘ f) :=
begin
  rintros ⟨f_iny,f_supr⟩ ⟨g_iny,g_supr⟩,
  exact ⟨injective.comp  g_iny f_iny,
         surjective.comp g_supr f_supr⟩,
```

```
end

-- 7ª demostración
example :
  bijective f → bijective g → bijective (g ∘ f) :=
begin
  rintros ⟨f_iny,f_supr⟩ ⟨g_iny,g_supr⟩,
  exact ⟨g_iny.comp f_iny,
         g_supr.comp f_supr⟩,
end

-- 8ª demostración
example :
  bijective f → bijective g → bijective (g ∘ f)
| ⟨f_iny,f_supr⟩ ⟨g_iny,g_supr⟩ :=
  ⟨g_iny.comp f_iny, g_supr.comp f_supr⟩

-- 9ª demostración
example :
  bijective f → bijective g → bijective (g ∘ f) :=
λ ⟨f_iny,f_supr⟩ ⟨g_iny,g_supr⟩,
  ⟨g_iny.comp f_iny, g_supr.comp f_supr⟩
```

6.1.10. Las composiciones con las inversas son la identidad

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

```
-- Las composiciones con las inversas son la identidad
-- ===================================================

import tactic

open function

variables {X Y Z : Type}
variable  {f : X → Y}
variable  {g : Y → X}

#reduce left_inverse g f
#reduce right_inverse g f
```

```
-- ----------------------------------------------------
-- Ej. 1. Demostrar que si g es una inversa por la
-- izquierda de f, entonces
--    g ∘ f = id
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : left_inverse g f)
  : g ∘ f = id :=
begin
  apply funext,
  intro x,
  rw comp_app,
  rw id.def,
  rw h,
end

-- 2ª demostración
example
  (h : left_inverse g f)
  : g ∘ f = id :=
begin
  apply funext,
  intro x,
  calc (g ∘ f) x
          = g (f x) : by rw comp_app
      ... = x       : by rw h
      ... = id x    : by rw id.def,
end

-- 3ª demostración
example
  (h : left_inverse g f)
  : g ∘ f = id :=
begin
  funext,
  dsimp,
  rw h,
end

-- 4ª demostración
example
  (h : left_inverse g f)
  : g ∘ f = id :=
```

```
funext h

-- 5ª demostración
example
  (h : left_inverse g f)
  : g ∘ f = id :=
-- by library_search
left_inverse.id h

-- ---------------------------------------------------
-- Ej. 2. Demostrar que si g es una inversa por la
-- derecha de f, entonces
--    f ∘ g = id
-- ---------------------------------------------------

-- 1ª demostración
example
  (h : right_inverse g f)
  : f ∘ g = id :=
begin
  apply funext,
  intro x,
  rw comp_app,
  rw id.def,
  rw h,
end

-- 2ª demostración
example
  (h : right_inverse g f)
  : f ∘ g = id :=
begin
  apply funext,
  intro x,
  calc (f ∘ g) x
         = f (g x) : by rw comp_app
     ... = x       : by rw h
     ... = id x    : by rw id.def,
end

-- 3ª demostración
example
  (h : right_inverse g f)
  : f ∘ g = id :=
begin
```

```
  funext,
  dsimp,
  rw h,
end

-- 4ª demostración
example
  (h : right_inverse g f)
  : f ∘ g = id :=
funext h

-- 5ª demostración
example
  (h : right_inverse g f)
  : f ∘ g = id :=
-- by library_search
right_inverse.id h
```

6.1.11. Las funciones con inversa por la izquierda son inyectivas

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

```
-- Las funciones con inversa por la izquierda son inyectivas
-- =========================================================

import tactic

open function

variables {X Y : Type}
variable  {f : X → Y}
variable  {g : Y → X}

-- ----------------------------------------------------
-- Ej. 1. Demostrar que si f tiene inversa por la
-- izquierda, entonces f es inyectiva.
-- ----------------------------------------------------

-- 1ª demostración
example
  (h : left_inverse g f)
```

```
  : injective f :=
begin
  intros x₁ x₂ h1,
  rw ←(h x₁),
  rw ←(h x₂),
  rw h1,
end

-- 2ª demostración
example
  (h : left_inverse g f)
  : injective f :=
begin
  intros x₁ x₂ h1,
  calc x₁ = g (f x₁) : (h x₁).symm
      ... = g (f x₂) : congr_arg g h1
      ... = x₂       : h x₂,
end

-- 3ª demostración
example
  (h : left_inverse g f)
  : injective f :=
begin
  intros x₁ x₂ h1,
  calc x₁ = g (f x₁) : by rw h
      ... = g (f x₂) : by rw h1
      ... = x₂       : by rw h
end

-- 4ª demostración
example
  (h : left_inverse g f)
  : injective f :=
assume x₁ x₂,
assume h1 : f x₁ = f x₂,
show x₁ = x₂, from
  calc x₁ = g (f x₁) : by rw h
      ... = g (f x₂) : by rw h1
      ... = x₂       : by rw h

-- 5ª demostración
example
  (h : left_inverse g f)
  : injective f :=
```

```
-- by library_search
left_inverse.injective h

-- 6ª demostración
example
  (h : left_inverse g f)
  : injective f :=
assume x₁ x₂,
assume h1 : f x₁ = f x₂,
show x₁ = x₂, from
  calc x₁ = g (f x₁) : (h x₁).symm
      ... = g (f x₂) : congr_arg g h1
      ... = x₂       : h x₂

-- 7ª demostración
example
  (h : left_inverse g f)
  : injective f :=
λ x₁ x₂ h1, (trans (h x₁).symm
                   (trans (congr_arg g h1)
                          (h x₂)))
```

6.1.12. Las funciones con inversa por la derecha son suprayectivas

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

```
-- Las funciones con inversa por la derecha son suprayectivas
-- ==========================================================

import tactic

open function

variables {X Y : Type}
variable  {f : X → Y}
variable  {g : Y → X}

-- ----------------------------------------------------
-- Ej. 1. Demostrar que si f tiene inversa por la
-- derecha, entonces f es suprayectiva
-- ----------------------------------------------------
```

```
-- 1ª demostración
example
  (h : right_inverse g f)
  : surjective f :=
begin
  intro y,
  use g y,
  exact h y,
end

-- 2ª demostración
example
  (h : right_inverse g f)
  : surjective f :=
λ y, ⟨g y, h y⟩

-- 3ª demostración
example
  (h : right_inverse g f)
  : surjective f :=
assume y,
show ∃ x, f x = y,
  from exists.intro (g y) (h y)

-- 4ª demostración
example
  (h : right_inverse g f)
  : surjective f :=
-- by library_search
right_inverse.surjective h
```

6.2. La función inversa

6.2.1. Las funciones inyectivas tienen inversa por la izquierda

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

```
-- Las funciones inyectivas tienen inversa por la izquierda
-- =========================================================
```

```
import tactic

open classical function

local attribute [instance] prop_decidable

variables {X Y : Type}

-- ---------------------------------------------------
-- Ej. 1. Definir la función
--    inversa :: (X → Y) → X → (Y → X)
-- tal que (inversa f d) es la función que a cada
-- y ∈ Y le asigna
-- + alguno de los elementos x tales que f(x) = y, si
--   existen dichos elementos y
-- + d, en caso contrario.
-- ---------------------------------------------------

noncomputable def inversa (f : X → Y) (d : X) : Y → X :=
λ y, if h : ∃ x, f x = y then some h else d

-- Notación:
variable (f : X → Y)
variable (d : X)
variable (y : Y)

-- ---------------------------------------------------
-- Ej. 2. Demostrar que si
--    ∃ x, f x = y
-- entonces,
--    f ((inversa f d) y) = y
-- ---------------------------------------------------

-- 1ª demostración
example
  (h : ∃ x, f x = y)
  : f ((inversa f d) y) = y :=
calc f ((inversa f d) y)
         = f (some h)   : congr rfl (dif_pos h)
     ... = y            : some_spec h

-- 2ª demostración
lemma inversa_cuando_existe
  (h : ∃ x, f x = y)
  : f ((inversa f d) y) = y :=
```

```
have h1 : (inversa f d) y = some h,
  from dif_pos h,
have h2 : f (some h) = y,
  from some_spec h,
show f (inversa f d y) = y,
  from eq.subst (eq.symm h1) h2

-- ------------------------------------------------------
-- Ej. 3. Demostrar que si f es inyectiva, entonces
-- (inversa f d) es inversa de f por la izquierda.
-- ------------------------------------------------------

-- 1ª demostración
example
  (h : injective f)
  : left_inverse (inversa f d) f :=
begin
  intro x,
  apply h,
  rw inversa_cuando_existe f d,
  use x,
end

-- 2ª demostración
lemma inversa_es_inversa_por_la_izquierda
  (h : injective f)
  : left_inverse (inversa f d) f :=
let g := (inversa f d) in
assume x,
have h1 : ∃ x', f x' = f x,
  from exists.intro x rfl,
have h2 : f (g (f x)) = f x,
  from inversa_cuando_existe f d (f x) h1,
show g (f x) = x,
  from h h2

-- ------------------------------------------------------
-- Ej. 4. Demostrar que si f es inyectiva, entonces
-- f tiene inversa por la izquierda.
-- ------------------------------------------------------

-- 1ª demostración
example
  (d : X)
  (h : injective f)
```

```
  : has_left_inverse f :=
begin
  unfold has_left_inverse,
  use (inversa f d),
  exact (inversa_es_inversa_por_la_izquierda f d h),
end

-- 2ª demostración
example
  (d : X)
  (h : injective f)
  : has_left_inverse f :=
have h1 : left_inverse (inversa f d) f,
  from inversa_es_inversa_por_la_izquierda f d h,
have h2 : ∃ g, left_inverse g f,
  from exists.intro (inversa f d) h1,
show has_left_inverse f,
  from h2

-- 3ª demostración
example
  (d : X)
  (h : injective f)
  : has_left_inverse f :=
have h1 : left_inverse (inversa f d) f,
  from inversa_es_inversa_por_la_izquierda f d h,
show has_left_inverse f,
  from exists.intro (inversa f d) h1
```

6.3. Funciones y conjuntos

6.3.1. La composición de inyectivas parciales es inyectiva

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

```
-- La composición de inyectivas parciales es inyectiva
-- ===================================================

-- ----------------------------------------------------
-- Ej. 1. Sean A un conjunto de elementos de X, B un
```

```
-- conjunto de elementos de Y, f una función de X en Y
-- y g una función de Y en Z. Si B contiene a la imagen
-- de A por f, f es inyectiva sobre A y g es inyectiva
-- sobre B, entonces (g ∘ f) es inyectiva sobre A.
-- ----------------------------------------------------

import data.set
open set function

variables {X Y Z : Type}
variable  (A : set X)
variable  (B : set Y)
variable  (f : X → Y)
variable  (g : Y → Z)

-- #reduce maps_to f A B
-- #reduce inj_on f A
-- #reduce inj_on g B

-- 1ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
begin
  intros x1 x1A x2 x2A h1,
  apply hf x1A x2A,
  apply hg,
  { exact h x1A, },
  { exact h x2A, },
  exact h1,
end

-- 2ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
begin
  intros x1 x1A x2 x2A h1,
  apply hf x1A x2A,
  apply hg (h x1A) (h x2A),
  exact h1,
```

```
end

-- 3ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
begin
  intros x1 x1A x2 x2A h1,
  apply hf x1A x2A,
  exact (hg (h x1A) (h x2A)) h1,
end

-- 4ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
begin
  intros x1 x1A x2 x2A h1,
  exact hf x1A x2A (hg (h x1A) (h x2A) h1),
end

-- 5ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
λ x1 x1A x2 x2A h1, hf x1A x2A (hg (h x1A) (h x2A) h1)

-- 6ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
assume x1 : X,
assume x1A : x1 ∈ A,
assume x2 : X,
assume x2A : x2 ∈ A,
have fx1B : f x1 ∈ B, from h x1A,
```

```
have fx2B : f x2 ∈ B, from h x2A,
assume h1 : g (f x1) = g (f x2),
have h2 : f x1 = f x2, from hg fx1B fx2B h1,
show x1 = x2, from hf x1A x2A h2

-- 7ª demostración
example
  (h  : maps_to f A B)
  (hf : inj_on f A)
  (hg : inj_on g B)
  : inj_on (g ∘ f) A :=
-- by library_search
inj_on.comp hg hf h
```

6.3.2. La composición de suprayectivas parciales es suprayectiva

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

```
-- La composición de suprayectivas parciales es suprayectiva
-- ==========================================================

-- ----------------------------------------------------
-- Ej. 1. Sean A un conjunto de elementos de X, B un
-- conjunto de elementos de Y, C un conjunto de
-- elementos de Z, f una función de X en Y y g una
-- función de Y en Z. Si f es suprayectiva de A en B y
-- g es suprayectiva de B en C, entonces (g ∘ f) es
-- suprayectiva de A en C.
-- ----------------------------------------------------

import data.set
open set function

variables {X Y Z : Type}
variable  (A : set X)
variable  (B : set Y)
variable  (C : set Z)
variable  (f : X → Y)
variable  (g : Y → Z)

-- #reduce surj_on f A B
```

```
-- #reduce surj_on g B C
-- #reduce surj_on (g ∘ f) A C

-- 1ª demostración
example
  (hf: surj_on f A B)
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
begin
  intros z zC,
  simp,
  specialize (hg zC),
  simp at hg,
  cases hg with y h1,
  specialize (hf h1.left),
  simp at hf,
  cases hf with x h2,
  use x,
  split,
  { exact h2.left, },
  { calc g (f x) = g y : by rw h2.right
             ... = z   : by rw h1.right },
end

-- 2ª demostración
example
  (hf: surj_on f A B)
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
begin
  intros z zC,
  specialize (hg zC),
  cases hg with y h1,
  specialize (hf h1.left),
  cases hf with x h2,
  use x,
  split,
  { exact h2.left, },
  { calc g (f x) = g y : by rw h2.right
             ... = z   : by rw h1.right },
end

-- 3ª demostración
example
  (hf: surj_on f A B)
```

```
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
begin
  intros z zC,
  cases (hg zC) with y h1,
  cases (hf h1.left) with x h2,
  use x,
  split,
  { exact h2.left, },
  { calc g (f x) = g y : by rw h2.right
             ... = z   : by rw h1.right },
end

-- 4ª demostración
example
  (hf: surj_on f A B)
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
begin
  intros z zC,
  cases (hg zC) with y h1,
  cases (hf h1.left) with x h2,
  exact ⟨x,
        ⟨h2.left,
         calc g (f x) = g y : by rw h2.right
                  ... = z   : by rw h1.right⟩⟩,
end

-- 5ª demostración
example
  (hf: surj_on f A B)
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
assume z,
assume zC : z ∈ C,
exists.elim (hg zC)
  ( assume y (h1 : y ∈ B ∧ g y = z),
    exists.elim (hf (and.left h1))
    ( assume x (h2 : x ∈ A ∧ f x = y),
      show ∃x, x ∈ A ∧ g (f x) = z, from
        exists.intro x
          (and.intro
            (and.left h2)
            (calc
              g (f x) = g y : by rw and.right h2
```

```
                  ... = z   : by rw and.right h1))))

-- 6ª demostración
example
  (hf: surj_on f A B)
  (hg : surj_on g B C)
  : surj_on (g ∘ f) A C :=
-- by library_search
surj_on.comp hg hf
```

6.3.3. La imagen de la unión es la unión de las imágenes

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

```
-- La imagen de la unión es la unión de las imágenes
-- ==================================================

import data.set
open set function

variables {X Y : Type}
variable  (f : X → Y)
variables (A₁ A₂ : set X)

-- #reduce image f A₁
-- #reduce f '' A₁

-- 1ª demostración
example :
  f '' (A₁ ∪ A₂) = f '' A₁ ∪ f '' A₂ :=
begin
  ext y,
  split,
  { intro h,
    cases h with x hx,
    cases hx with xA₁A₂ fxy,
    cases xA₁A₂ with xA₁ xA₂,
    { left,
      -- simp,
      use x,
      exact ⟨xA₁, fxy⟩, },
    { right,
      use x,
```

```
      exact ⟨xA₂, fxy⟩, }},
  { intro h,
    cases h with yifA₁ yifA₂,
    { cases yifA₁ with  x h1,
      cases h1 with xA₁ fxy,
      use x,
      split,
      { left,
        exact xA₁, },
      { exact fxy, }},
    { cases yifA₂ with x h1,
      cases h1 with xA₂ fxy,
      use x,
      split,
      { right,
        exact xA₂, },
      { exact fxy, }}},
end

-- 2ª demostración
example :
  f '' (A₁ ∪ A₂) = f '' A₁ ∪ f '' A₂ :=
begin
  ext y,
  split,
  { rintro ⟨x, (xA₁ | xA₂), fxy⟩,
    { exact or.inl ⟨x, xA₁, fxy⟩, },
    { exact or.inr ⟨x, xA₂, fxy⟩, }},
  { rintro (⟨x, xA₁, fxy⟩ | ⟨x, xA₂, fxy⟩),
    { exact ⟨x, or.inl xA₁, fxy⟩, },
    { exact ⟨x, or.inr xA₂, fxy⟩, }},
end

-- 3ª demostración
example :
  f '' (A₁ ∪ A₂) = f '' A₁ ∪ f '' A₂ :=
ext (assume y, iff.intro
  ( assume h : y ∈ f '' (A₁ ∪ A₂),
    exists.elim h
      ( assume x h1,
        have xA₁A₂ : x ∈ A₁ ∪ A₂, from h1.left,
        have fxy : f x = y, from h1.right,
        or.elim xA₁A₂
          ( assume xA₁, or.inl ⟨x, xA₁, fxy⟩)
          ( assume xA₂, or.inr ⟨x, xA₂, fxy⟩)))
```

```
  ( assume h : y ∈ f '' A₁ ∪ f '' A₂,
    or.elim h
      ( assume yifA₁ : y ∈ f '' A₁,
        exists.elim yifA₁
          ( assume x h1,
            have xA₁ : x ∈ A₁, from h1.left,
            have fxy : f x = y, from h1.right,
            ⟨x, or.inl xA₁, fxy⟩))
      ( assume yifA₂ : y ∈ f '' A₂,
        exists.elim yifA₂
          ( assume x h1,
            have xA₂ : x ∈ A₂, from h1.left,
            have fxy : f x = y, from h1.right,
            ⟨x, (or.inr xA₂), fxy⟩))))

-- 4ª demostración
example :
  f '' (A₁ ∪ A₂) = f '' A₁ ∪ f '' A₂ :=
-- by library_search
image_union f A₁ A₂

-- 5ª demostración
example :
  f '' (A₁ ∪ A₂) = f '' A₁ ∪ f '' A₂ :=
by finish [ext_iff, iff_def]
```

Capítulo 7

Números naturales, recursión e inducción

7.1. Definiciones por recursión

7.1.1. Definiciones por recursión sobre los naturales

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

```
-- Definiciones por recursión sobre los naturales
-- ==============================================

-- ----------------------------------------------------
-- Ej. 1. Definir el tipo nat de los números naturales
-- con los constructores zero (para el número cero) y
-- succ (para el sucesor).
-- ----------------------------------------------------

namespace reservado

inductive nat : Type
| zero : nat
| succ : nat → nat

end reservado

-- ----------------------------------------------------
-- Ej. 2. Abrir el espacio de nombre de los números
-- naturales.
-- ----------------------------------------------------
```

```
open nat

-- ----------------------------------------------------
-- Ej. 3. Definir la función
--    factorial : ℕ → ℕ
-- tal que (factorial n) es el factorial de n. Por ejemplo,
--    factorial 3 == 6
-- ----------------------------------------------------

-- 1ª definición
def factorial : ℕ → ℕ
| 0        := 1
| (succ n) := (succ n) * factorial n

-- 2ª definición
def factorial2 : ℕ → ℕ
| 0       := 1
| (n + 1) := (n + 1) * factorial2 n

-- ----------------------------------------------------
-- Ej. 4. Calcular el factorial de 3 y de 30.
-- ----------------------------------------------------

-- El cáculo es
--    #eval factorial 3
--    #eval factorial 300

-- ----------------------------------------------------
-- Ej. 5. Demostrar que el factorial de 3 es 6.
-- ----------------------------------------------------

example : factorial 3 = 6 := rfl

-- ----------------------------------------------------
-- Ej. 6. Definir por recurión la función
--    potencia_de_dos : ℕ → ℕ
-- tal que (potencia_de_dos n) es 2^n. Por ejemplo,
-- ----------------------------------------------------

-- 1ª definición
def potencia_de_dos : ℕ → ℕ
| 0        := 1
| (succ n) := 2 * potencia_de_dos n
```

```
-- 1ª definición
def potencia_de_dos2 : ℕ → ℕ
| 0       := 1
| (n + 1) := 2 * potencia_de_dos2 n

-- ----------------------------------------------------
-- Ej. 7. Calcular (potencia_de_dos 3) y
-- (potencia_de_dos 10).
-- ----------------------------------------------------

-- El cálculo es
--    #eval potencia_de_dos 3
--    #eval potencia_de_dos 10

-- ----------------------------------------------------
-- Ej. 8. Demostrar que (potencia_de_dos 3) es 8.
-- ----------------------------------------------------

example : potencia_de_dos 3 = 8 := rfl

-- ----------------------------------------------------
-- Ej. 9. Definir la función
--    potencia : ℕ → ℕ → ℕ
-- tal que (potencia m n) es m^n. Por ejemplo,
--    potencia 2 3  =  8
--    potencia 3 2  =  9
-- ----------------------------------------------------

def potencia : ℕ → ℕ → ℕ
| m 0       := 1
| m (n + 1) := potencia m n * m

-- #eval potencia 2 3
-- #eval potencia 3 2

-- #eval pow 2 3
-- #eval pow 3 2

-- #eval 2^3
-- #eval 3^2

-- ----------------------------------------------------
-- Ej. 10. Sean m, n ∈ ℕ. Demostrar,
--    m^0 = 1
--    m^(n+1) = m^n * m
```

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

variables m n : ℕ

-- 1ª demostración
example : m^0 = 1 := nat.pow_zero m

-- 2ª demostración
example : m^0 = 1 := rfl

-- 1ª demostración
example : m^(n+1) = m^n * m := nat.pow_succ m n

-- 2ª demostración
example : m^(n+1) = m^n * m := rfl

-- ----------------------------------------------------
-- Ej. 11. Definir la función
--    fib : ℕ → ℕ
-- tal que (fib n) es el n-ésimo número de Fibonacci.
-- Por ejemplo,
--    fib 4 = 3
--    fib 5 = 5
--    fib 6 = 8
-- ----------------------------------------------------

def fib : ℕ → ℕ
| 0       := 0
| 1       := 1
| (n + 2) := fib (n + 1) + fib n

-- #eval fib 4
-- #eval fib 5
-- #eval fib 6
```

7.1.2. Operaciones aritméticas definidas

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

```
-- Operaciones aritméticas definidas
-- =================================

import data.nat.basic
```

```
open nat

variables m n : ℕ

-- Suma
#check @nat.add_zero m   -- : m + 0 = 0
#check @nat.add_succ m n -- : m + succ n = succ (m + n)

-- Producto
#check @nat.mul_zero m   -- : m * 0 = 0
#check @nat.mul_succ m n -- : m * succ n = n * n + m

-- Potencia
#check @nat.pow_zero m   -- : m ^ 0 = 1
#check @nat.pow_succ m n -- : m ^ succ n = m ^n * m

-- Predecesor
#check @pred_zero        -- : pred 0 = 0
#check @pred_succ n      -- : pred (succ n) = n
```

7.2. Recursión e inducción

7.2.1. Prueba por inducción 1: (∀ n ∈ ℕ) 0 + n = n

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

```
-- Pruebas por inducción 1: 0 + n = n
-- ==================================

-- ----------------------------------------------------
-- Ej. 1. Sean m y n números naturales. Demostrar que
--    m + 0 = m
-- ----------------------------------------------------

import tactic

open nat

variables (m n : ℕ)

-- 1ª demostración
example : m + 0 = m :=
```

```
nat.add_zero m

-- 2ª demostración
example : m + 0 = m :=
rfl

-- ----------------------------------------------------
-- Ej. 2. Sean m y n números naturales. Demostrar que
--    m + (n + 1) = (m + n) + 1
-- ----------------------------------------------------

-- 1ª demostración
example : m + (n + 1) = (m + n) + 1 :=
add_succ m n

-- 2ª demostración
example : m + (n + 1) = (m + n) + 1 :=
rfl

-- ----------------------------------------------------
-- Ej. 3. Sean n un número natural. Demostrar que
--    0 + n = n
-- ----------------------------------------------------

-- 1ª demostración
example : 0 + n = n :=
begin
  induction n with n HI,
  { rw nat.add_zero, },
  { rw add_succ,
    rw HI, },
end

-- 2ª demostración
example : 0 + n = n :=
begin
  induction n with n HI,
  { rw nat.add_zero, },
  { rw [add_succ, HI], },
end

-- 3ª demostración
example : 0 + n = n :=
begin
  induction n with n HI,
```

```
  { simp, },
  { simp [add_succ, HI], },
end

-- 4ª demostración
example : 0 + n = n :=
begin
  induction n with n HI,
  { simp only [nat.add_zero], },
  { simp only [add_succ, HI], },
end

-- 5ª demostración
example : 0 + n = n :=
by induction n;
   simp only [*,
              nat.add_zero,
              add_succ]

-- 6ª demostración
example : 0 + n = n :=
by induction n;
   simp

-- 7ª demostración
example : 0 + n = n :=
nat.rec_on n
  ( show 0 + 0 = 0, from nat.add_zero 0)
  ( assume n,
    assume HI : 0 + n = n,
    show 0 + succ n = succ n, from
      calc
        0 + succ n = succ (0 + n) : by rw add_succ
               ... = succ n       : by rw HI)

-- 8ª demostración
example : 0 + n = n :=
nat.rec_on n
  ( show 0 + 0 = 0, from rfl)
  ( assume n,
    assume HI : 0 + n = n,
    show 0 + succ n = succ n, from
      calc
        0 + succ n = succ (0 + n) : rfl
               ... = succ n       : by rw HI)
```

```
-- 9ª demostración
example : 0 + n = n :=
nat.rec_on n rfl (λ n HI, by rw [add_succ, HI])

-- 10ª demostración
example : 0 + n = n :=
nat.rec_on n rfl (λ n HI, by simp only [add_succ, HI])

-- 11ª demostración
example : 0 + n = n :=
-- by library_search
zero_add n

-- 12ª demostración
example : 0 + n = n :=
-- by hint
by simp

-- 13ª demostración
example : 0 + n = n :=
by finish

-- 14ª demostración
example : 0 + n = n :=
by linarith

-- 15ª demostración
example : 0 + n = n :=
by nlinarith

-- 16ª demostración
example : 0 + n = n :=
by norm_num

-- 17ª demostración
example : 0 + n = n :=
by ring

-- 18ª demostración
example : 0 + n = n :=
by omega

-- 19ª demostración
example : 0 + n = n :=
```

```
by tidy

-- 20ª demostración
example : 0 + n = n :=
-- by tidy ?
by simp at *

-- 21ª demostración
lemma cero_mas : ∀ n : ℕ, 0 + n = n
| 0     := rfl
| (n+1) := congr_arg succ (cero_mas n)

-- 22ª demostración
lemma cero_mas2 : ∀ n : ℕ, 0 + n = n
| 0     := by simp
| (n+1) := by simp

-- 23ª demostración
lemma cero_mas3 : ∀ n : ℕ, 0 + n = n
| 0     := by simp only [add_zero]
| (n+1) := by simp only [add_zero, add_succ, cero_mas3 n]

-- 24ª demostración
lemma cero_mas4 : ∀ n : ℕ, 0 + n = n
| 0     := by rw [add_zero]
| (n+1) := by rw [add_succ, cero_mas4 n]
```

7.2.2. Prueba por inducción 2: (∀ m n k ∈ ℕ) (m + n) + k = m + (n + k)

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

```
-- Prueba por inducción 2: ∀ m n k : ℕ, (m + n) + k = m + (n + k)
-- ================================================================

-- ----------------------------------------------------
-- Ej. 1. Sean m, n y k números naturales. Demostrar que
--    (m + n) + k = m + (n + k)
-- ----------------------------------------------------

import tactic
```

```
open nat

variables (m n k : ℕ)

-- 1ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k' HI,
  { rw nat.add_zero,
    rw nat.add_zero, },
  { rw add_succ,
    rw HI,
    rw add_succ, },
end

-- 2ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k' HI,
  { rw [nat.add_zero, nat.add_zero], },
  { rw [add_succ, HI, add_succ], },
end

-- 3ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k HI,
  { simp, },
  { simp [add_succ, HI], },
end

-- 4ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k HI,
  { simp only [add_zero], },
  { simp only [add_succ, HI], },
end

-- 5ª demostración
example : (m + n) + k = m + (n + k) :=
by induction k;
   simp only [*, add_zero, add_succ]

-- 6ª demostración
```

```
example : (m + n) + k = m + (n + k) :=
by induction k;
   simp [*, add_succ]

-- 7ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k HI,
  { calc
      (m + n) + 0
          = m + n       : by rw nat.add_zero
      ... = m + (n + 0) : by rw nat.add_zero, },
  { calc
      (m + n) + succ k
          = succ ((m + n) + k) : by rw add_succ
      ... = succ (m + (n + k)) : by rw HI
      ... = m + succ (n + k)   : by rw add_succ
      ... = m + (n + succ k)   : by rw add_succ, },
end

-- 8ª demostración
example : (m + n) + k = m + (n + k) :=
begin
  induction k with k HI,
  { calc
      (m + n) + 0 = m + (n + 0) : rfl, },
  { calc
      (m + n) + succ k
          = succ ((m + n) + k) : rfl
      ... = succ (m + (n + k)) : by rw HI
      ... = m + succ (n + k)   : rfl
      ... = m + (n + succ k)   : rfl, },
end

-- 9ª demostración
example : (m + n) + k = m + (n + k) :=
nat.rec_on k
  ( show (m + n) + 0 = m + (n + 0), from rfl )
  ( assume k,
    assume HI : (m + n) + k = m + (n + k),
    show (m + n) + succ k = m + (n + succ k), from
      calc
        (m + n) + succ k
            = succ ((m + n) + k) : rfl
        ... = succ (m + (n + k)) : by rw HI
```

```
        ... = m + succ (n + k)    : rfl
        ... = m + (n + succ k)    : rfl )

-- 10ª demostración
example : (m + n) + k = m + (n + k) :=
nat.rec_on k rfl (λ k HI, by simp only [add_succ, HI])

-- 11ª demostración
example : (m + n) + k = m + (n + k) :=
-- by library_search
add_assoc m n k

-- 12ª demostración
example : (m + n) + k = m + (n + k) :=
-- by hint
by finish

-- 13ª demostración
example : (m + n) + k = m + (n + k) :=
by linarith

-- 14ª demostración
example : (m + n) + k = m + (n + k) :=
by nlinarith

-- 15ª demostración
example : (m + n) + k = m + (n + k) :=
by omega

-- 16ª demostración
example : (m + n) + k = m + (n + k) :=
by ring

-- 17ª demostración
lemma asociativa_suma :
  ∀ k : ℕ, (m + n) + k = m + (n + k)
| 0       := rfl
| (k + 1) := congr_arg succ (asociativa_suma k)

-- 18ª demostración
lemma asociativa_suma2 :
  ∀ k : ℕ, (m + n) + k = m + (n + k)
| 0       := by simp
| (k + 1) := by simp [add_succ, asociativa_suma2 k]
```

```
-- 19ª demostración
lemma asociativa_suma3 :
  ∀ k : ℕ, (m + n) + k = m + (n + k)
| 0       := by simp only [nat.add_zero]
| (k + 1) := by simp only [nat.add_zero, add_succ, asociativa_suma3 k]
```

7.2.3. Prueba por inducción 3: (∀ m n ∈ ℕ) succ m + n = succ (m + n)

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

```
-- Prueba por inducción 3: ∀ m n : ℕ, succ m + n = succ (m + n)
-- ============================================================

-- ----------------------------------------------------
-- Ej. 1. Sean m y n números naturales. Demostrar que
--    succ m + n = succ (m + n)
-- ----------------------------------------------------

import tactic

open nat

variables (m n : ℕ)

-- 1ª demostración
example : succ m + n = succ (m + n) :=
begin
  induction n with n HI,
  { rw nat.add_zero,
    rw nat.add_zero, },
  { rw add_succ,
    rw HI, },
end

-- 2ª demostración
example : succ m + n = succ (m + n) :=
begin
  induction n with n HI,
  { simp only [nat.add_zero], },
  { simp only [add_succ, HI], },
end
```

```
-- 3ª demostración
example : succ m + n = succ (m + n) :=
by induction n;
   simp only [*, nat.add_zero, add_succ]

-- 4ª demostración
example : succ m + n = succ (m + n) :=
by induction n;
   simp [*, add_succ]

-- 5ª demostración
example : succ m + n = succ (m + n) :=
nat.rec_on n
  (show succ m + 0 = succ (m + 0), from
    calc
      succ m + 0
          = succ m       : by rw nat.add_zero
      ... = succ (m + 0) : by rw nat.add_zero)
  (assume n,
    assume HI : succ m + n = succ (m + n),
    show succ m + succ n = succ (m + succ n), from
      calc
        succ m + succ n
            = succ (succ m + n)   : by rw add_succ
        ... = succ (succ (m + n)) : by rw HI
        ... = succ (m + succ n)   : by rw add_succ)

-- 6ª demostración
example : succ m + n = succ (m + n) :=
nat.rec_on n
  (show succ m + 0 = succ (m + 0), from rfl)
  (assume n,
    assume HI : succ m + n = succ (m + n),
    show succ m + succ n = succ (m + succ n), from
      calc
        succ m + succ n
            = succ (succ m + n)   : rfl
        ... = succ (succ (m + n)) : by rw HI
        ... = succ (m + succ n)   : rfl)

-- 7ª demostración
example : succ m + n = succ (m + n) :=
nat.rec_on n rfl (λ n HI, by simp only [add_succ, HI])
```

```
-- 8ª demostración
example : succ m + n = succ (m + n) :=
-- by library_search
succ_add m n

-- 9ª demostración
example : succ m + n = succ (m + n) :=
-- by hint
by omega

-- 10ª demostración
lemma suc_suma : ∀ n m : ℕ, (succ n) + m = succ (n + m)
| n 0     := rfl
| n (m+1) := congr_arg succ (suc_suma n m)

-- 11ª demostración
lemma suc_suma2 : ∀ n m : ℕ, (succ n) + m = succ (n + m)
| n 0     := by simp
| n (m+1) := by simp [add_succ, suc_suma2 n m]

-- 12ª demostración
lemma suc_suma3 : ∀ n m : ℕ, (succ n) + m = succ (n + m)
| n 0     := by simp only [add_zero]
| n (m+1) := by simp only [add_zero, add_succ, suc_suma3 n m]
```

7.2.4. Prueba por inducción 4: ($\forall$ m n $\in \mathbb{N}$) m + n = n + m

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

```
-- Prueba por inducción 4: ∀ m n : ℕ, m + n = n + m
-- ================================================

-- ----------------------------------------------------
-- Ej. 1. Sean m y n números naturales. Demostrar que
--    m + n = n + m
-- ----------------------------------------------------

import tactic

open nat

variables (m n : ℕ)
```

```
-- #check nat.add_zero
-- #check nat.add_succ
-- #check nat.zero_add
-- #check nat.succ_add

-- 1ª demostración
example : m + n = n + m :=
begin
  induction n with n HI,
  { rw nat.add_zero,
    rw nat.zero_add, },
  { rw add_succ,
    rw HI,
    rw succ_add, },
end

-- 2ª demostración
example : m + n = n + m :=
begin
  induction n with n HI,
  { simp only [nat.add_zero, nat.zero_add] },
  { simp only [add_succ, HI, succ_add] },
end

-- 3ª demostración
example : m + n = n + m :=
by induction n;
   simp only [*, nat.add_zero, add_succ, succ_add, nat.zero_add]

-- 4ª demostración
example : m + n = n + m :=
by induction n;
   simp [*, add_succ, succ_add]

-- 5ª demostración
example : m + n = n + m :=
nat.rec_on n
  (show m + 0 = 0 + m, from
    calc m + 0
             = m     : by rw nat.add_zero
         ... = 0 + m : by rw nat.zero_add )
  (assume n,
   assume HI : m + n = n + m,
   show m + n.succ = n.succ + m, from
     calc
```

```
       m + succ n
           = succ (m + n) : by rw add_succ
       ... = succ (n + m) : by rw HI
       ... = succ n + m   : by rw succ_add)

-- 6ª demostración
example : m + n = n + m :=
nat.rec_on n
  (show m + 0 = 0 + m, by rw [nat.zero_add, nat.add_zero])
  (assume n,
   assume HI : m + n = n + m,
   calc
     m + succ n = succ (m + n) : rfl
       ... = succ (n + m)      : by rw HI
       ... = succ n + m        : by rw succ_add)

-- 7ª demostración
example : m + n = n + m :=
nat.rec_on n
  (by simp only [nat.zero_add, nat.add_zero])
  (λ n HI, by simp only [add_succ, HI, succ_add])

-- 8ª demostración
example : m + n = n + m :=
nat.rec_on n
  (by simp)
  (λ n HI, by simp [add_succ, HI, succ_add])

-- 9ª demostración
example : m + n = n + m :=
-- by library_search
nat.add_comm m n

-- 10ª demostración
example : m + n = n + m :=
-- by hint
by finish

-- 11ª demostración
example : m + n = n + m :=
by linarith

-- 12ª demostración
example : m + n = n + m :=
by nlinarith
```

```
-- 13ª demostración
example : m + n = n + m :=
by ring

-- 14ª demostración
example : m + n = n + m :=
by omega

-- 15ª demostración
lemma conmutativa : ∀ m n : ℕ, m + n = n + m
| m 0     := by simp
| m (n+1) := by simp [add_succ, conmutativa m n, succ_add]

-- 16ª demostración
lemma conmutativa2 : ∀ m n : ℕ, m + n = n + m
| m 0     := by simp only [nat.add_zero, nat.zero_add]
| m (n+1) := by simp only [nat.add_zero, add_succ, conmutativa2 m n, succ_add]
```

7.2.5. Prueba por inducción 5: (∀ m n ∈ ℕ) m^(n+1) = m * m^n

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

```
-- Prueba por inducción 5: m^(succ n) = m * m^n
-- ============================================

-- ----------------------------------------------------
-- Ej. 1. Sean m y n números naturales. Demostrar que
--    m^(succ n) = m * m^n
-- ----------------------------------------------------

import data.nat.basic
open nat

variables (m n : ℕ)

-- #check nat.pow_zero
-- #check nat.pow_succ
-- #check nat.mul_one
-- #check nat.one_mul
-- #check nat.mul_assoc
-- #check nat.mul_comm
```

```
-- 1ª demostración
example : m^(succ n) = m * m^n :=
begin
  induction n with n HI,
  { rw nat.pow_succ,
    rw nat.pow_zero,
    rw nat.one_mul,
    rw nat.mul_one, },
  { rw nat.pow_succ,
    rw HI,
    rw nat.mul_assoc,
    rw nat.mul_comm (m^n), },
end

-- 2ª demostración
example : m^(succ n) = m * m^n :=
begin
  induction n with n HI,
  rw [nat.pow_succ, nat.pow_zero, one_mul, mul_one],
  rw [nat.pow_succ, HI, mul_assoc, mul_comm (m^n)],
end

-- 3ª demostración
example : m^(succ n) = m * m^n :=
begin
  induction n with n HI,
  { simp only [nat.pow_succ, nat.pow_zero, one_mul, mul_one]},
  { simp only [nat.pow_succ, HI, mul_assoc, mul_comm (m^n)]},
end

-- 4ª demostración
example : m^(succ n) = m * m^n :=
by induction n;
   simp only [*,
              nat.pow_succ,
              nat.pow_zero,
              nat.one_mul,
              nat.mul_one,
              nat.mul_assoc,
              nat.mul_comm]

-- 5ª demostración
example : m^(succ n) = m * m^n :=
by induction n;
```

```
   simp [*,
         nat.pow_succ,
         mul_comm]

-- 6ª demostración
example : m^(succ n) = m * m^n :=
begin
  induction n with n HI,
  { simp, },
  { simp [nat.pow_succ, HI],
    cc, },
end

-- 7ª demostración
example : m^(succ n) = m * m^n :=
begin
  induction n with n HI,
  { calc
      m^(succ 0)
          = m^0 * m : by rw nat.pow_succ
      ... = 1 * m   : by rw nat.pow_zero
      ... = m       : by rw nat.one_mul
      ... = m * 1   : by rw nat.mul_one
      ... = m * m^0 : by rw nat.pow_zero, },
  { calc
      m^(succ (succ n))
          = m^(succ n) * m   : by rw nat.pow_succ
      ... = (m * m^n) * m    : by rw HI
      ... = m * (m^n * m)    : by rw nat.mul_assoc
      ... = m * m^(succ n)   : by rw nat.pow_succ, },
end

-- 8ª demostración
example : m^(succ n) = m * m^n :=
nat.rec_on n
  (show m^(succ 0) = m * m^0, from calc
    m^(succ 0) = m^0 * m : by rw nat.pow_succ
           ... = 1 * m   : by rw nat.pow_zero
           ... = m       : by rw one_mul
           ... = m * 1   : by rw mul_one
           ... = m * m^0 : by rw nat.pow_zero)
  (assume n,
    assume HI : m^(succ n) = m * m^n,
    show m^(succ (succ n)) = m * m^(succ n), from calc
```

```
      m^(succ (succ n)) = m^(succ n) * m    : by rw nat.pow_succ
                    ... = (m * m^n) * m     : by rw HI
                    ... = m * (m^n * m)     : by rw mul_assoc
                    ... = m * m^(succ n)    : by rw nat.pow_succ)

-- 9ª demostración
example : m^(succ n) = m * (m^n) :=
nat.rec_on n
  (show m^(succ 0) = m * m^0,
    by rw [nat.pow_succ, nat.pow_zero, mul_one, one_mul])
  (assume n,
    assume HI : m^(succ n) = m * m^n,
    show m^(succ (succ n)) = m * m^(succ n),
      by rw [nat.pow_succ, HI, mul_assoc, mul_comm (m^n)])

-- 10ª demostración
example : m^(succ n) = m * (m^n) :=
nat.rec_on n
  (show m^(succ 0) = m * m^0,
    by simp )
  (assume n,
    assume HI : m^(succ n) = m * m^n,
    show m^(succ (succ n)) = m * m^(succ n),
      by finish [nat.pow_succ, HI] )

-- 11ª demostración
example : m^(succ n) = m * (m^n) :=
nat.rec_on n
  (by simp)
  (λ n HI, by finish [nat.pow_succ, HI])

-- 12ª demostración
lemma aux : ∀ m n : ℕ, m^(succ n) = m * (m^n)
| m 0     := by simp
| m (n+1) := by simp [nat.pow_succ,
                      aux m n,
                      mul_assoc,
                      mul_comm (m^n)]

-- 13ª demostración
lemma aux2 : ∀ m n : ℕ, m^(succ n) = m * (m^n)
| m 0     := by simp only [nat.pow_succ,
                           nat.pow_zero,
                           one_mul,
```

```
                             mul_one]
| m (n+1) := by simp only [nat.pow_succ,
                             aux2 m n,
                             mul_assoc,
                             mul_comm (m^n)]

-- 14ª demostración
lemma aux3 : ∀ m n : ℕ, m^(succ n) = m * (m^n)
| m 0     := by simp
| m (n+1) := by simp [nat.pow_succ, aux3 m n] ; cc
```

7.2.6. Prueba por inducción 6: (∀ m n k ∈ ℕ) m^(n + k) = m^n * m^k

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

```
-- Prueba por inducción 6: (∀ m n k ∈ ℕ) m^(n + k) = m^n * m^k
-- ==============================================================

import data.nat.basic
import tactic
open nat

variables  (m n k : ℕ)

-- 1ª demostración
example : m^(n + k) = m^n * m^k :=
begin
  induction k with k HI,
  { rw add_zero,
    rw nat.pow_zero,
    rw mul_one, },
  { rw add_succ,
    rw nat.pow_succ,
    rw HI,
    rw nat.pow_succ,
    rw mul_assoc, },
end

-- 2ª demostración
example : m^(n + k) = m^n * m^k :=
begin
```

```
  induction k with k HI,
  { calc
      m^(n + 0)
          = m^n       : by rw add_zero
      ... = m^n * 1   : by rw mul_one
      ... = m^n * m^0 : by rw nat.pow_zero, },
  { calc
      m^(n + succ k)
          = m^(succ (n + k)) : by rw nat.add_succ
      ... = m^(n + k) * m    : by rw nat.pow_succ
      ... = m^n * m^k * m    : by rw HI
      ... = m^n * (m^k * m)  : by rw mul_assoc
      ... = m^n * m^(succ k) : by rw nat.pow_succ, },
end

-- 3ª demostración
example : m^(n + k) = m^n * m^k :=
begin
  induction k with k HI,
  { rw [add_zero,
        nat.pow_zero,
        mul_one], },
  { rw [add_succ,
        nat.pow_succ,
        HI,
        nat.pow_succ,
        mul_assoc], },
end

-- 4ª demostración
example : m^(n + k) = m^n * m^k :=
begin
  induction k with k HI,
  { simp only [add_zero,
               nat.pow_zero,
               mul_one], },
  { simp only [add_succ,
               nat.pow_succ,
               HI,
               nat.pow_succ,
               mul_assoc], },
end

-- 5ª demostración
```

```
example : m^(n + k) = m^n * m^k :=
begin
  induction k with k HI,
  { simp, },
  { simp [HI,
          nat.pow_succ,
          mul_assoc], },
end

-- 6ª demostración
example : m^(n + k) = m^n * m^k :=
by induction k; simp [*, nat.pow_succ, mul_assoc]

-- 7ª demostración
example : m^(n + k) = m^n * m^k :=
nat.rec_on k
  (show m^(n + 0) = m^n * m^0, from
    calc
      m^(n + 0)
          = m^n       : by rw add_zero
      ... = m^n * 1   : by rw mul_one
      ... = m^n * m^0 : by rw nat.pow_zero)
  (assume k,
    assume HI : m^(n + k) = m^n * m^k,
    show m^(n + succ k) = m^n * m^(succ k), from
      calc
        m^(n + succ k)
            = m^(succ (n + k)) : by rw nat.add_succ
        ... = m^(n + k) * m    : by rw nat.pow_succ
        ... = m^n * m^k * m    : by rw HI
        ... = m^n * (m^k * m)  : by rw mul_assoc
        ... = m^n * m^(succ k) : by rw nat.pow_succ)

-- 8ª demostración
example : m^(n + k) = m^n * m^k :=
nat.rec_on k
  (by simp)
  (λ n HI, by simp [HI, nat.pow_succ, mul_assoc])

-- 9ª demostración
example : m^(n + k) = m^n * m^k :=
-- by library_search
pow_add m n k
```

7.2.7. Prueba por inducción 7: $(\forall\ n \in \mathbb{N})\ n \neq 0 \to$ succ (pred n) = n

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

```
-- Prueba por induccion 7: (∀ n ∈ ℕ) n ≠ 0 → succ (pred n) = n
-- ===========================================================

import data.nat.basic
open nat

variable (n : ℕ)

-- ?ª demostración
example : n ≠ 0 → succ (pred n) = n :=
begin
  cases n,
  { intro h,
    contradiction, },
  { intro h,
    rw pred_succ, },
end

-- ?ª demostración
example : n ≠ 0 → succ (pred n) = n :=
by cases n; simp

-- ?ª demostración
example : n ≠ 0 → succ (pred n) = n :=
nat.cases_on n
  (assume h : 0 ≠ 0,
    show succ (pred 0) = 0,
      from absurd rfl h)
  (assume n,
    assume h : succ n ≠ 0,
    show succ (pred (succ n)) = succ n,
      by rw pred_succ)

-- ?ª demostración
example : n ≠ 0 → succ (pred n) = n :=
nat.cases_on n
  (λ h, absurd rfl h)
  (λ n h, by rw pred_succ)
```

Capítulo 8

Razonamiento sobre programas

8.1. Razonamiento ecuacional

8.1.1. Razonamiento ecuacional sobre longitudes de listas

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

```
-- Razonamiento ecuacional sobre longitudes de listas
-- ===================================================

import data.list.basic
open list

variable {α : Type}
variable x : α
variable (xs : list α)

-- ----------------------------------------------------
-- Ejercicio 1. Definir, por recursión, la función
--    longitud : list α → ℕ
-- tal que (longitud xs) es la longitud de la lista
-- xs. Por ejemplo,
--    longitud [a,c,d] = 3
-- ----------------------------------------------------

def longitud : list α → ℕ
| []        := 0
| (_ :: xs) := longitud xs + 1

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

```
-- Ejercicio 2. Calcular
--    longitud [4,2,5]
-- ----------------------------------------------------

-- #eval longitud [4,2,5]
-- da 3

-- ----------------------------------------------------
-- Ejercicio 3. Demostrar los siguientes lemas
-- + longitud_nil :
--     longitud ([] : list α) = 0
-- + longitud_cons :
--     longitud (x :: xs) = longitud xs + 1
-- ----------------------------------------------------

@[simp]
lemma longitud_nil :
  longitud ([] : list α) = 0 :=
rfl

@[simp]
lemma longitud_cons :
  longitud (x :: xs) = longitud xs + 1 :=
rfl

-- ----------------------------------------------------
-- Ejercicio 4. Demostrar que
--    longitud [a,b,c] = 3
-- ----------------------------------------------------

-- 1ª demostración
example
  (a b c : α)
  : longitud [a,b,c] = 3 :=
calc
  longitud [a,b,c]
      = longitud [b,c] + 1           : by rw longitud_cons
  ... = (longitud [c] + 1) + 1       : by rw longitud_cons
  ... = ((longitud [] + 1) + 1) + 1 : by rw longitud_cons
  ... = ((0 + 1) + 1) + 1           : by rw longitud_nil
  ... = 3                            : rfl

-- 2ª demostración
example
  (a b c : α)
```

```
  : longitud [a,b,c] = 3 :=
calc
  longitud [a,b,c]
      = longitud [b,c] + 1          : rfl
  ... = (longitud [c] + 1) + 1      : rfl
  ... = ((longitud [] + 1) + 1) + 1 : rfl
  ... = ((0 + 1) + 1) + 1           : rfl
  ... = 3                           : rfl

-- 3ª demostración
example
  (a b c : α)
  : longitud [a,b,c] = 3 :=
begin
  rw longitud_cons,
  rw longitud_cons,
  rw longitud_cons,
  rw longitud_nil,
end

-- 4ª demostración
example
  (a b c : α)
  : longitud [a,b,c] = 3 :=
by rw [longitud_cons,
       longitud_cons,
       longitud_cons,
       longitud_nil]

-- 5ª demostración
example
  (a b c : α)
  : longitud [a,b,c] = 3 :=
by simp only [longitud_cons,
              longitud_nil]

-- 4ª demostración
example
  (a b c : α)
  : longitud [a,b,c] = 3 :=
by simp

-- 6ª demostración
example
  (a b c : α)
```

```
  : longitud [a,b,c] = 3 :=
rfl

-- Comentarios sobre la función length:
-- + Es equivalente a la función longitud.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval length [4,2,5,7,2]
-- + Se puede demostrar. Por ejemplo,
--      example
--        (a b c : α)
--        : length [a,b,c] = 3 :=
--      rfl
```

8.1.2. Razonamiento ecuacional sobre intercambio en pares

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

```
-- Razonamiento ecuacional sobre intercambio en pares
-- ==================================================

import data.prod
open prod

variables {α : Type*} {β : Type*}
variable  (x : α)
variable  (y : β)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    intercambia :: α × β → β × α
-- tal que (intercambia p) es el par obtenido
-- intercambiando las componentes del par p. Por
-- ejemplo,
--    intercambia (5,7) = (7,5)
-- ----------------------------------------------------

def intercambia : α × β → β × α
```

```
| (x,y) := (y, x)

-- #eval intercambia (5,7)

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar el lema
--    intercambia_simp : intercambia p = (p.2, p.1)
-- ---------------------------------------------------

@[simp]
lemma intercambia_simp :
  intercambia (x,y) = (y,x) :=
rfl

-- ---------------------------------------------------
-- Ejercicio 3. (p.6) Demostrar que
--    intercambia (intercambia (x,y)) = (x,y)
-- ---------------------------------------------------

-- 1ª demostración
example :
  intercambia (intercambia (x,y)) = (x,y) :=
calc intercambia (intercambia (x,y))
         = intercambia (y,x)          : by rw intercambia_simp
     ... = (x,y)                      : by rw intercambia_simp

-- 2ª demostración
example :
  intercambia (intercambia (x,y)) = (x,y) :=
calc intercambia (intercambia (x,y))
         = intercambia (y,x)          : by simp
     ... = (x,y)                      : by simp

-- 3ª demostración
example :
  intercambia (intercambia (x,y)) = (x,y) :=
by simp

-- 4ª demostración
example :
  intercambia (intercambia (x,y)) = (x,y) :=
rfl

-- 5ª demostración
example :
```

```
  intercambia (intercambia (x,y)) = (x,y) :=
begin
  rw intercambia_simp,
  rw intercambia_simp,
end

-- Comentarios sobre la función swap:
-- + Es equivalente a la función intercambia.
-- + Para usarla hay que importar la librería data.prod
--   y abrir espacio de nombre prod el escribiendo al
--   principio del fichero
--      import data.prod
--      open prod
-- + Se puede evaluar. Por ejemplo,
--      #eval swap (5,7)
-- + Se puede demostrar. Por ejemplo,
--      example :
--        swap (swap (x,y)) = (x,y) :=
--      rfl
--
--      example :
--        swap (swap (x,y)) = (x,y) :=
--      -- by library_search
--      swap_swap (x,y)
```

8.1.3. Razonamiento ecuacional sobre la inversa de listas unitarias

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

```
-- Razonamiento ecuacional sobre la inversa de listas unitarias
-- ============================================================

import data.list.basic
open list

variable  {α : Type*}
variable  x : α
variables (xs : list α)

-- ----------------------------------------------------
-- Ejercicio 1. Definir, por recursión, la función
--    inversa :: list α → list α
```

```
-- tal que (inversa xs) es la lista obtenida
-- invirtiendo el orden de los elementos de xs.
-- Por ejemplo,
--    inversa [3,2,5] = [5,2,3]
-- ---------------------------------------------------

def inversa : list α → list α
| []        := []
| (x :: xs) := inversa xs ++ [x]

-- #eval inversa [3,2,5]

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + inversa_nil :
--     inversa ([] : list α) = []
-- + inversa_cons :
--     inversa (x :: xs) = inversa xs ++ [x]
-- ---------------------------------------------------

@[simp]
lemma inversa_nil :
  inversa ([] : list α) = [] :=
rfl

@[simp]
lemma inversa_cons :
  inversa (x :: xs) = inversa xs ++ [x] :=
rfl

-- ---------------------------------------------------
-- Ejercicio 3. (p. 9) Demostrar que
--    inversa [x] = [x]
-- ---------------------------------------------------

-- 1ª demostración
example : inversa [x] = [x] :=
calc inversa [x]
         = inversa ([] : list α) ++ [x] : by rw inversa_cons
     ... = ([] : list α) ++ [x]         : by rw inversa_nil
     ... = [x]                          : by rw nil_append


-- 2ª demostración
example : inversa [x] = [x] :=
```

```
calc inversa [x]
         = inversa ([] : list α) ++ [x] : by simp
     ... = ([] : list α) ++ [x]         : by simp
     ... = [x]                          : by simp

-- 3ª demostración
example : inversa [x] = [x] :=
by simp

-- 4ª demostración
example : inversa [x] = [x] :=
begin
  rw inversa_cons,
  rw inversa_nil,
  rw nil_append,
end

-- 5ª demostración
example : inversa [x] = [x] :=
by rw [inversa_cons,
       inversa_nil,
       nil_append]

-- 6ª demostración
example : inversa [x] = [x] :=
rfl

-- Comentarios sobre la función reverse:
-- + Es equivalente a la función inversa
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval reverse  [3,2,5]
-- + Se puede demostrar. Por ejemplo,
--      example : reverse [x] = [x] :=
--      -- by library_search
--      reverse_singleton x
```

8.2. Razonamiento por inducción sobre los naturales

8.2.1. Pruebas de longitud (repite n x) = n

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

```
-- Pruebas de longitud (repite n x) = n
-- ====================================

import data.nat.basic
open nat
open list

variable {α : Type}
variable (x : α)
variable (xs : list α)
variable (n : ℕ)

-- ----------------------------------------------------
-- Nota. Se usará la función longitud y sus propiedades
-- estudiadas anteriormente.
-- ----------------------------------------------------

def longitud : list α → nat
| []        := 0
| (x :: xs) := longitud xs + 1

@[simp]
lemma longitud_nil :
  longitud ([] : list α) = 0 :=
rfl

@[simp]
lemma longitud_cons :
  longitud (x :: xs) = longitud xs + 1 :=
rfl

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    repite :: ℕ → α → list α
-- tal que (repite n x) es la lista formada por n
-- copias del elemento x. Por ejemplo,
--    repite 3 7 = [7,7,7]
```

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

def repite : ℕ → α → list α
| 0 _        := []
| (succ n) x := x :: repite n x

-- #eval repite 3 7

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + repite_cero :
--     repite 0 x = []
-- + repite_suc :
--     repite (succ n) x = x :: repite n x
-- ----------------------------------------------------

@[simp]
lemma repite_cero :
  repite 0 x = [] :=
rfl

@[simp]
lemma repite_suc :
  repite (succ n) x = x :: repite n x :=
rfl

-- ----------------------------------------------------
-- Ejercicio 3. (p. 18) Demostrar que
--    longitud (repite n x) = n
-- ----------------------------------------------------

-- 1ª demostración
example :
  longitud (repite n x) = n :=
begin
  induction n with n HI,
  { calc
      longitud (repite 0 x)
          = longitud []                : by rw repite_cero
      ... = 0                          : by rw longitud_nil },
  { calc
      longitud (repite (succ n) x)
          = longitud (x :: repite n x) : by rw repite_suc
      ... = longitud (repite n x) + 1  : by rw longitud_cons
      ... = n + 1                      : by rw HI
```

```
      ... = succ n                    : rfl, },
end

-- 2ª demostración
example : longitud (repite n x) = n :=
begin
  induction n with n HI,
  { rw repite_cero,
    rw longitud_nil, },
  { rw repite_suc,
    rw longitud_cons,
    rw HI, },
end

-- 3ª demostración
example : longitud (repite n x) = n :=
begin
  induction n with n HI,
  { simp only [repite_cero, longitud_nil], },
  { simp only [repite_suc, longitud_cons, HI], },
end

-- 4ª demostración
example : longitud (repite n x) = n :=
begin
  induction n with n HI,
  { simp, },
  { simp [HI], },
end

-- 5ª demostración
example : longitud (repite n x) = n :=
by induction n ; simp [*]

-- 6ª demostración
example : longitud (repite n x) = n :=
nat.rec_on n
  ( show longitud (repite 0 x) = 0, from
      calc
        longitud (repite 0 x)
            = longitud []             : by rw repite_cero
        ... = 0                       : by rw longitud_nil )
  ( assume n,
    assume HI : longitud (repite n x) = n,
    show longitud (repite (succ n) x) = succ n, from
```

```
      calc
      longitud (repite (succ n) x)
          = longitud (x :: repite n x) : by rw repite_suc
      ... = longitud (repite n x) + 1  : by rw longitud_cons
      ... = n + 1                      : by rw HI
      ... = succ n                     : rfl )

-- 7ª demostración
example : longitud (repite n x) = n :=
nat.rec_on n
  ( by simp )
  ( λ n HI, by simp [HI])

-- 7ª demostración
lemma longitud_repite_1 :
  ∀ n, longitud (repite n x) = n
| 0 := by calc
    longitud (repite 0 x)
        = longitud ([] : list α) : by rw repite_cero
    ... = 0                      : by rw longitud_nil
| (n+1) := by calc
    longitud (repite (n + 1) x)
        = longitud (x :: repite n x) : by rw repite_suc
    ... = longitud (repite n x) + 1  : by rw longitud_cons
    ... = n + 1                      : by rw longitud_repite_1

-- 8ª demostración
lemma longitud_repite_2 :
  ∀ n, longitud (repite n x) = n
| 0     := by simp
| (n+1) := by simp [*]

-- Comentarios sobre la función (repeat x n)
-- + Es equivalente a la función (repite n x).
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval list.repeat 7 3
-- + Se puede demostrar. Por ejemplo,
--      example : length (repeat x n) = n :=
--      by induction n ; simp [*]
--
```

```
--      example : length (repeat x n) = n :=
--      -- by library_search
--      length_repeat x n
```

8.3. Razonamiento por inducción sobre listas

8.3.1. Pruebas de la asociatividad de la concatenación

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

```
-- Prueba de la asociatividad de la concatenación
-- ==============================================

import tactic
open list

variable  {α : Type}
variable  (x : α)
variables (xs ys zs : list α)

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    conc :: list α → list α → list α
-- tal que (conc xs ys) es la concatención de las
-- listas xs e ys. Por ejemplo,
--    conc [1,4] [2,4,1,3] = [1,4,2,4,1,3]
-- ----------------------------------------------------

def conc : list α → list α → list α
| []        ys := ys
| (x :: xs) ys := x :: (conc xs ys)

-- #eval conc [1,4] [2,4,1,3]

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + conc_nil :
--     conc ([] : list α) ys = ys
-- + conc_cons :
--     conc (x :: xs) ys = x :: (conc xs ys)
-- ----------------------------------------------------
```

```
@[simp]
lemma conc_nil :
  conc ([] : list α) ys = ys :=
rfl

@[simp]
lemma conc_cons :
  conc (x :: xs) ys = x :: (conc xs ys) :=
rfl

-- ----------------------------------------------------
-- Ejercicio 3. (p. 24) Demostrar que
--    conc xs (conc ys zs) = conc (conc xs ys) zs
-- ----------------------------------------------------

-- 1ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
begin
  induction xs with a as HI,
  { calc conc [] (conc ys zs)
         = conc ys zs                  : by rw conc_nil
     ... = conc (conc [] ys) zs        : by rw conc_nil, },
  { calc conc (a :: as) (conc ys zs)
         = a :: conc as (conc ys zs)   : by rw conc_cons
     ... = a :: conc (conc as ys) zs   : by rw HI
     ... = conc (a :: conc as ys) zs   : by rw conc_cons
     ... = conc (conc (a :: as) ys) zs : by rw ←conc_cons, },
end

-- 2ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
begin
  induction xs with a as HI,
  { calc conc [] (conc ys zs)
         = conc ys zs                  : by simp
     ... = conc (conc [] ys) zs        : by simp, },
  { calc conc (a :: as) (conc ys zs)
         = a :: conc as (conc ys zs)   : by simp
     ... = a :: conc (conc as ys) zs   : by simp [HI]
     ... = conc (a :: conc as ys) zs   : by simp
     ... = conc (conc (a :: as) ys) zs : by simp, },
end
```

```
-- 3ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
begin
  induction xs with a as HI,
  { by simp, },
  { by simp [HI], },
end

-- 4ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
by induction xs ; simp [*]

-- 5ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
begin
  induction xs with a as HI,
  { rw conc_nil,
    rw conc_nil, },
  { rw conc_cons,
    rw HI,
    rw conc_cons,
    rw conc_cons, },
end

-- 6ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
list.rec_on xs
  ( show conc [] (conc ys zs) = conc (conc [] ys) zs,
      from calc
        conc [] (conc ys zs)
            = conc ys zs            : by rw conc_nil
        ... = conc (conc [] ys) zs : by rw conc_nil )
    ( assume a as,
      assume HI : conc as (conc ys zs) = conc (conc as ys) zs,
      show conc (a :: as) (conc ys zs) = conc (conc (a :: as) ys) zs,
        from calc
          conc (a :: as) (conc ys zs)
               = a :: conc as (conc ys zs)  : by rw conc_cons
          ... = a :: conc (conc as ys) zs   : by rw HI
          ... = conc (a :: conc as ys) zs   : by rw conc_cons
          ... = conc (conc (a :: as) ys) zs : by rw ←conc_cons)
```

```
-- 7ª demostración
example :
  conc xs (conc ys zs) = conc (conc xs ys) zs :=
list.rec_on xs
  (by simp)
  (by simp [*])

-- 8ª demostración
lemma conc_asoc_1 :
  ∀ xs, conc xs (conc ys zs) = conc (conc xs ys) zs
| [] := by calc
    conc [] (conc ys zs)
        = conc ys zs            : by rw conc_nil
    ... = conc (conc [] ys) zs : by rw conc_nil
| (a :: as) := by calc
    conc (a :: as) (conc ys zs)
        = a :: conc as (conc ys zs)    : by rw conc_cons
    ... = a :: conc (conc as ys) zs    : by rw conc_asoc_1
    ... = conc (a :: conc as ys) zs    : by rw conc_cons
    ... = conc (conc (a :: as) ys) zs : by rw ←conc_cons

-- 9ª demostración
lemma conc_asoc_2 :
  ∀ xs, conc xs (conc ys zs) = conc (conc xs ys) zs
| []         := by simp
| (a :: as)  := by simp [conc_asoc_2 as]

-- Comentarios sobre la función (++)
-- + Es equivalente a la función conc.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval [1,4] ++ [2,4,1,3]
-- + Se puede demostrar. Por ejemplo,
--      example :
--        xs ++ (ys ++ zs) = (xs ++ ys) ++ zs :=
--      -- by library_search
--      (append_assoc xs ys zs).symm
--
--      example :
--        (xs ++ ys) ++ zs = xs ++ (ys ++ zs) :=
```

```
--      -- by library_search
--      append_assoc xs ys zs
--
--      example :
--        (xs ++ ys) ++ zs = xs ++ (ys ++ zs) :=
--      by induction xs ; simp [*]
```

8.3.2. Pruebas del elemento neutro por la derecha de la concatenación

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

```
-- Prueba del elemento neutro por la derecha de la concatenación
-- ==============================================================

import tactic
import data.list.basic
open list

variable  {α : Type}
variable  (x : α)
variables (xs ys : list α)

-- ----------------------------------------------------
-- Nota. Se usará la función conc y sus propiedades
-- estudiadas anteriormente.
-- ----------------------------------------------------

def conc : list α → list α → list α
| []        ys := ys
| (x :: xs) ys := x :: (conc xs ys)

@[simp]
lemma conc_nil :
  conc ([] : list α) ys = ys :=
rfl

@[simp]
lemma conc_cons :
  conc (x :: xs) ys = x :: (conc xs ys) :=
rfl

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

```
-- Ejercicio 1. (p. 28) Demostrar que
--    conc xs [] = xs
-- ----------------------------------------------------

-- 1ª demostración
example : conc xs [] = xs :=
begin
  induction xs with a as HI,
  { rw conc_nil, },
  { rw conc_cons,
    rw HI, },
end

-- 2ª demostración
example : conc xs [] = xs :=
begin
  induction xs with x xs HI,
  { rw [conc_nil], },
  { rw [conc_cons, HI], },
end

-- 3ª demostración
example : conc xs [] = xs :=
begin
  induction xs with x xs HI,
  { simp only [conc_nil], },
  { simp only [conc_cons, HI],
    cc, },
end

-- 4ª demostración
example : conc xs [] = xs :=
begin
  induction xs with x xs HI,
  { simp , },
  { simp [HI], },
end

-- 5ª demostración
example : conc xs [] = xs :=
by induction xs ; simp [*]

-- 6ª demostración
example : conc xs [] = xs :=
begin
```

```
  induction xs with a as HI,
  { rw conc_nil, },
  { calc
      conc (a :: as) []
          = a :: (conc as []) : by rw conc_cons
      ... = a :: as           : by rw HI, },
end

-- 7ª demostración
example : conc xs [] = xs :=
list.rec_on xs
  ( show conc [] [] = [], from calc
      conc [] [] = [] : by rw conc_nil )
  ( assume a as,
    assume HI : conc as [] = as,
    show conc (a :: as) [] = a :: as, from calc
      conc (a :: as) []
          = a :: (conc as []) : by rw conc_cons
      ... = a :: as           : by rw HI)

-- 8ª demostración
example : conc xs [] = xs :=
list.rec_on xs
  ( show conc [] [] = [], by simp)
  ( assume a as,
    assume HI : conc as [] = as,
    show conc (a :: as) [] = a :: as, by simp [HI])

-- 9ª demostración
example : conc xs [] = xs :=
list.rec_on xs
  (by simp)
  (λ a as HI, by simp [HI])

-- 10ª demostración
lemma conc_nil_1:
  ∀ xs : list α, conc xs [] = xs
| []        := by rw conc_nil
| (a :: as) := by calc
    conc (a :: as) []
        = a :: conc as [] : by rw conc_cons
    ... = a :: as         : by rw conc_nil_1

-- 11ª demostración
lemma conc_nil_2:
```

```
  ∀ xs : list α, conc xs [] = xs
| []        := by simp
| (a :: as) := by simp [conc_nil_2 as]

-- Comentarios sobre la función (++)
-- + Es equivalente a la función conc.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval [1,4] ++ [2,4,1,3]
-- + Se puede demostrar. Por ejemplo,
example : xs ++ [] = xs :=
by induction xs ; simp [*]

example : xs ++ [] = xs :=
-- by library_search
append_nil xs

example : xs ++ [] = xs :=
by simp
```

8.3.3. Pruebas de longitud (conc xs ys) = longitud xs + longitud ys

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

```
-- Pruebas de la relación entre longitud y conc
-- ============================================

import tactic
open list

variable  {α : Type}
variable  (x : α)
variables (xs ys zs : list α)

-- ------------------------------------------------------
-- Nota. Se usarán las definiciones y propiedades de
-- las funciones longitud y conc estudiadas
-- anteriormente.
```

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

def longitud : list α → nat
| []        := 0
| (x :: xs) := longitud xs + 1

@[simp]
lemma longitud_nil :
  longitud ([] : list α) = 0 :=
rfl

@[simp]
lemma longitud_cons :
  longitud (x :: xs) = longitud xs + 1 :=
rfl

def conc : list α → list α → list α
| []        ys := ys
| (x :: xs) ys := x :: (conc xs ys)

@[simp]
lemma conc_nil :
  conc ([] : list α) ys = ys :=
rfl

@[simp]
lemma conc_cons :
  conc (x :: xs) ys = x :: (conc xs ys) :=
rfl

-- ----------------------------------------------------
-- Ejercicio 1. (p. 30) Demostrar que
--    longitud (conc xs ys) = longitud xs + longitud ys
-- ----------------------------------------------------

-- 1ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { rw conc_nil,
    rw longitud_nil,
    rw zero_add, },
  { rw conc_cons,
    rw longitud_cons,
```

```
    rw HI,
    rw longitud_cons,
    rw add_assoc,
    rw add_comm (longitud ys),
    rw add_assoc, },
end

-- 2ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { rw conc_nil,
    rw longitud_nil,
    rw zero_add, },
  { rw conc_cons,
    rw longitud_cons,
    rw HI,
    rw longitud_cons,
    -- library_search,
    exact add_right_comm (longitud as) (longitud ys) 1 },
end

-- 3ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { rw conc_nil,
    rw longitud_nil,
    rw zero_add, },
  { rw conc_cons,
    rw longitud_cons,
    rw HI,
    rw longitud_cons,
    -- by hint,
    linarith, },
end

-- 4ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { simp, },
```

```
  { simp [HI],
    linarith, },
end

-- 5ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { simp, },
  { finish [HI],},
end

-- 6ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
by induction xs ; finish [*]

-- 7ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
begin
  induction xs with a as HI,
  { calc longitud (conc [] ys)
         = longitud ys                      : by rw conc_nil
     ... = 0 + longitud ys                  : by exact (zero_add (longitud ys)).symm
     ... = longitud [] + longitud ys        : by rw longitud_nil },
  { calc longitud (conc (a :: as) ys)
         = longitud (a :: conc as ys)       : by rw conc_cons
     ... = longitud (conc as ys) + 1        : by rw longitud_cons
     ... = (longitud as + longitud ys) + 1  : by rw HI
     ... = (longitud as + 1) + longitud ys  : by exact add_right_comm (longitud as) (lon
     ... = longitud (a :: as) + longitud ys : by rw longitud_cons, },
end

-- 8ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
list.rec_on xs
  ( show longitud (conc [] ys) = longitud [] + longitud ys, from
      calc longitud (conc [] ys)
           = longitud ys                      : by rw conc_nil
       ... = 0 + longitud ys                  : by exact (zero_add (longitud ys)).symm
       ... = longitud [] + longitud ys        : by rw longitud_nil )
  ( assume a as,
```

```
    assume HI : longitud (conc as ys) = longitud as + longitud ys,
    show longitud (conc (a :: as) ys) = longitud (a :: as) + longitud ys, from
      calc longitud (conc (a :: as) ys)
           = longitud (a :: conc as ys)       : by rw conc_cons
       ... = longitud (conc as ys) + 1        : by rw longitud_cons
       ... = (longitud as + longitud ys) + 1  : by rw HI
       ... = (longitud as + 1) + longitud ys  : by exact add_right_comm (longitud as) (l
       ... = longitud (a :: as) + longitud ys : by rw longitud_cons)

-- 9ª demostración
example :
  longitud (conc xs ys) = longitud xs + longitud ys :=
list.rec_on xs
  ( by simp)
  ( λ a as HI, by simp [HI, add_right_comm])

-- 10ª demostración
lemma longitud_conc_1 :
  ∀ xs, longitud (conc xs ys) = longitud xs + longitud ys
| [] := by calc
    longitud (conc [] ys)
        = longitud ys                       : by rw conc_nil
    ... = 0 + longitud ys                   : by rw zero_add
    ... = longitud [] + longitud ys         : by rw longitud_nil
| (a :: as) := by calc
    longitud (conc (a :: as) ys)
        = longitud (a :: conc as ys)        : by rw conc_cons
    ... = longitud (conc as ys) + 1         : by rw longitud_cons
    ... = (longitud as + longitud ys) + 1   : by rw longitud_conc_1
    ... = (longitud as + 1) + longitud ys   : by exact add_right_comm (longitud as) (long
    ... = longitud (a :: as) + longitud ys  : by rw longitud_cons

-- 11ª demostración
lemma longitud_conc_2 :
  ∀ xs, longitud (conc xs ys) = longitud xs + longitud ys
| []        := by simp
| (a :: as) := by simp [longitud_conc_2 as, add_right_comm]

-- Comentarios sobre las funciones length y (++)
-- + Para usarlas hay que abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      open list
-- + Se puede demostrar. Por ejemplo,
example :
  length (xs ++ ys) = length xs + length ys :=
```

```
by induction xs ; finish [*]

example :
  length (xs ++ ys) = length xs + length ys :=
-- by library_search
length_append xs ys

example :
  length (xs ++ ys) = length xs + length ys :=
by simp
```

8.4. Inducción con patrones para funciones recursivas generales

8.4.1. Pruebas de conc (coge n xs) (elimina n xs) = xs

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

```
-- Pruebas de conc (coge n xs) (elimina n xs) = xs
-- ===============================================

import tactic
open list
open nat

variable  {α : Type}
variable  (x : α)
variables (xs ys : list α)
variable  (n : ℕ)

-- ----------------------------------------------------
-- Nota. Se usará la definición y propiedades de la
-- función conc estudiadas anteriormente.
-- ----------------------------------------------------

def conc : list α → list α → list α
| []       ys := ys
| (x :: xs) ys := x :: (conc xs ys)

@[simp]
lemma conc_nil :
  conc ([] : list α) ys = ys :=
```

```
rfl

@[simp]
lemma conc_cons :
  conc (x :: xs) ys = x :: (conc xs ys) :=
rfl

-- ---------------------------------------------------
-- Ejercicio 1. Definir la función
--    coge : ℕ → list α → list α
-- tal que (coge n xs) es la lista de los n primeros
-- elementos de xs. Por ejemplo,
--    coge 2 [1,4,2,7,0] =  [1,4]
-- ---------------------------------------------------

def coge : ℕ → list α → list α
| 0        _         := []
| (succ _) []        := []
| (succ n) (x :: xs) := x :: coge n xs

-- #eval coge 2 [1,4,2,7]

-- ---------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + coge_cero :
--      coge 0 xs = []
-- + coge_nil :
--      coge n [] = []
-- + coge_cons :
--      coge (succ n) (x :: xs) = x :: coge n xs
-- ---------------------------------------------------

@[simp]
lemma coge_cero :
  coge 0 xs = [] :=
rfl

@[simp]
lemma coge_nil :
  ∀ n, coge n ([] : list α) = []
| 0     := rfl
| (n+1) := rfl

@[simp]
lemma coge_cons :
```

```
  coge (succ n) (x :: xs) = x :: coge n xs :=
rfl

-- ---------------------------------------------------
-- Ejercicio 3. Definir la función
--    elimina : ℕ → list α → list α
-- tal que (elimina n xs) es la lista obtenida eliminando los n primeros
-- elementos de xs. Por ejemplo,
--    elimina 2 [1,4,2,7,0] = [2,7,0]
-- ---------------------------------------------------

def elimina : ℕ → list α → list α
| 0        xs        := xs
| (succ n) []        := []
| (succ n) (x :: xs) := elimina n xs

-- #eval elimina 2 [1,4,2,7,0]

-- ---------------------------------------------------
-- Ejercicio 4. Demostrar los siguientes lemas
-- + elimina_cero :
--      elimina 0 xs = xs
-- + elimina_nil :
--      elimina n [] = []
-- + elimina_cons :
--      elimina (succ n) (x :: xs) = elimina n xs
-- ---------------------------------------------------

@[simp]
lemma elimina_cero :
  elimina 0 xs = xs :=
rfl

@[simp]
lemma elimina_nil :
  ∀ n, elimina n ([] : list α) = []
| 0     := rfl
| (n+1) := rfl

@[simp]
lemma elimina_cons :
  elimina (succ n) (x :: xs) = elimina n xs :=
rfl

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

```
-- Ejercicio 5. (p. 35) Demostrar que
--    conc (coge n xs) (elimina n xs) = xs
-- ----------------------------------------------------

-- 1ª demostración
example :
  ∀ xs : list α, conc (coge n xs) (elimina n xs) = xs :=
begin
  induction n with m HI1,
  { intro,
    rw coge_cero,
    rw elimina_cero,
    rw conc_nil, },
  { intro,
    induction xs with a as HI2,
    { rw coge_nil,
      rw elimina_nil,
      rw conc_nil, },
    { rw coge_cons,
      rw elimina_cons,
      rw conc_cons,
      rw (HI1 as), }, },
end

-- 2ª demostración
example :
  ∀ xs : list α, conc (coge n xs) (elimina n xs) = xs :=
begin
  induction n with m HI1,
  { intro,
    calc conc (coge 0 xs) (elimina 0 xs)
         = conc [] (elimina 0 xs) : by rw coge_cero
     ... = conc [] xs             : by rw elimina_cero
     ... = xs                     : by rw conc_nil, },
  { intro,
    induction xs with a as HI2,
    { calc conc (coge (succ m) []) (elimina (succ m) [])
           = conc ([] : list α) (elimina (succ m) []) : by rw coge_nil
       ... = conc [] []                               : by rw elimina_nil
       ... = []                                       : by rw conc_nil, },
    { calc conc (coge (succ m) (a :: as)) (elimina (succ m) (a :: as))
           = conc (a :: coge m as) (elimina (succ m) (a :: as)) :by rw coge_cons
       ... = conc (a :: coge m as) (elimina m as)               :by rw elimina_cons
       ... = a :: conc (coge m as) (elimina m as)               :by rw conc_cons
       ... = a :: as                                            :by rw (HI1 as), }, },
```

```
end

-- 3ª demostración
example :
  ∀ xs : list α, conc (coge n xs) (elimina n xs) = xs :=
begin
  induction n with m HI1,
  { intro,
    simp, },
  { intro,
    induction xs with a as HI2,
    { simp, },
    { simp [HI1 as], }, },
end

-- 4ª demostración
example :
  ∀ xs : list α, conc (coge n xs) (elimina n xs) = xs :=
nat.rec_on n
  ( assume xs,
    show conc (coge 0 xs) (elimina 0 xs) = xs, from
      calc conc (coge 0 xs) (elimina 0 xs)
           = conc ([] : list α) (elimina 0 xs) : by rw coge_cero
       ... = conc [] xs                        : by rw elimina_cero
       ... = xs                                : by rw conc_nil)
  ( assume m,
    assume HI1 : ∀ xs, conc (coge m xs) (elimina m xs) = xs,
    assume xs,
    show conc (coge (succ m) xs) (elimina (succ m) xs) = xs, from
      list.rec_on xs
      ( show conc (coge (succ m) []) (elimina (succ m) []) = [], from
          calc conc (coge (succ m) []) (elimina (succ m) [])
                    = conc ([] : list α) (elimina (succ m) []) : by rw coge_nil
                ... = conc [] []                               : by rw elimina_nil
                ... = []                                       : by rw conc_nil)
      ( assume a as,
        assume HI2 : conc (coge (succ m) as) (elimina (succ m) as) = as,
        show conc (coge (succ m) (a :: as)) (elimina (succ m) (a :: as)) = a :: as, from
          calc conc (coge (succ m) (a :: as)) (elimina (succ m) (a :: as))
                    = conc (a :: coge m as) (elimina (succ m) (a :: as)) :by rw coge_co
                ... = conc (a :: coge m as) (elimina m as)               :by rw elimina
                ... = a :: conc (coge m as) (elimina m as)               :by rw conc_co
                ... = a :: as                                            :by rw (HI1 as

-- 5ª demostración
```

```
example :
  ∀ xs : list α, conc (coge n xs) (elimina n xs) = xs :=
nat.rec_on n
  (by simp)
  (λ m HI1 xs, list.rec_on xs
    (by simp)
    (by simp [HI1]))

-- 6ª demostración
lemma conc_coge_elimina_1 :
  ∀ (n : ℕ) (xs : list α), conc (coge n xs) (elimina n xs) = xs
| 0 xs := by calc
    conc (coge 0 xs) (elimina 0 xs)
        = conc [] (elimina 0 xs)                          : by rw coge_cero
    ... = conc [] xs                                      : by rw elimina_cero
    ... = xs                                              : by rw conc_nil
| (succ m) [] := by calc
    conc (coge (succ m) []) (elimina (succ m) [])
        = conc ([] : list α) (elimina (succ m) [])        : by rw coge_nil
    ... = conc [] []                                      : by rw elimina_nil
    ... = []                                              : by rw conc_nil
| (succ m) (a :: as) := by calc
    conc (coge (succ m) (a :: as)) (elimina (succ m) (a :: as))
        = conc (a :: coge m as) (elimina (succ m) (a :: as))    : by rw coge_cons
    ... = conc (a :: coge m as) (elimina m as)            : by rw elimina_cons
    ... = a :: conc (coge m as) (elimina m as)            : by rw conc_cons
    ... = a :: as                                         : by rw conc_coge_elimin

-- 7ª demostración
lemma conc_coge_elimina_2 :
  ∀ (n : ℕ) (xs : list α), conc (coge n xs) (elimina n xs) = xs
| 0        xs        := by simp
| (succ m) []        := by simp
| (succ m) (a :: as) := by simp [conc_coge_elimina_2]

-- 8ª demostración
lemma conc_coge_elimina_3 :
  ∀ (n : ℕ) (xs : list α), conc (coge n xs) (elimina n xs) = xs
| 0        xs        := rfl
| (succ m) []        := rfl
| (succ m) (a :: as) := congr_arg (cons a) (conc_coge_elimina_3 m as)

-- Comentarios sobre las funciones take y drop:
-- + Para usarlas hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
```

```
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede calcular. Por ejemplo,
--      #eval take 2 [1,4,2,7]
--      #eval drop 2 [1,4,2,7,0]
-- + Se puede demostrar. Por ejemplo,
example : take n xs ++ drop n xs = xs :=
-- by library_search
take_append_drop n xs

lemma take_drop_1 :
  ∀ (n : ℕ) (xs : list α), take n xs ++ drop n xs = xs
| 0        xs        := by simp
| (succ n) []        := by simp
| (succ n) (x :: xs) := by simp [take_drop_1]

example : take n xs ++ drop n xs = xs :=
by simp
```

8.5. Razonamiento por casos

8.5.1. Pruebas de esVacia xs = esVacia (conc xs xs)

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

```
-- Pruebas de esVacia xs = esVacia (conc xs xs)
-- ============================================

import data.list.basic
open list

variable  {α : Type}
variable  (x : α)
variables (xs ys : list α)

-- ----------------------------------------------------
-- Nota. Se usará la función conc y sus propiedades
-- estudiadas anteriormente.
-- ----------------------------------------------------

def conc : list α → list α → list α
```

```
| []        ys := ys
| (x :: xs) ys := x :: (conc xs ys)

@[simp]
lemma conc_nil :
  conc ([] : list α) ys = ys :=
rfl

@[simp]
lemma conc_cons :
  conc (x :: xs) ys = x :: (conc xs ys) :=
rfl

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    esVacia : list α → bool
-- tal que (esVacia xs) se verifica si xs es la lista
-- vacía. Por ejemplo,
--    esVacia []  = tt
--    esVacia [1] = ff
-- ----------------------------------------------------

def esVacia : list α → bool
| [] := tt
| _  := ff

-- #eval esVacia ([] : list ℕ)
-- #eval esVacia [1,5]

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + esVacia_nil :
--      esVacia ([] : list α) = tt :=
-- + esVacia_cons :
--      esVacia (x :: xs) = ff :=
-- ----------------------------------------------------

@[simp]
lemma esVacia_nil :
  esVacia ([] : list α) = tt :=
rfl

@[simp]
lemma esVacia_cons :
  esVacia (x :: xs) = ff :=
```

```
rfl

-- ----------------------------------------------------
-- Ejercicio 3 (p. 39) . Demostrar que
--    esVacia xs = esVacia (conc xs xs)
-- ----------------------------------------------------

-- 1ª demostración
example : esVacia xs = esVacia (conc xs xs) :=
begin
  cases xs with a as,
  { rw conc_nil, },
  { rw conc_cons,
    rw esVacia_cons,
    rw esVacia_cons, },
end

-- 2ª demostración
example : esVacia xs = esVacia (conc xs xs) :=
begin
  cases xs with a as,
  { simp, },
  { simp, },
end

-- 3ª demostración
example : esVacia xs = esVacia (conc xs xs) :=
by cases xs ; simp

-- 4ª demostración
example : esVacia xs = esVacia (conc xs xs) :=
list.cases_on xs
  (show esVacia ([] : list α) = esVacia (conc [] []),
     from congr_arg esVacia (conc_nil []))
  (assume a as,
   show esVacia (a :: as) = esVacia (conc (a :: as) (a :: as)),
     from calc
       esVacia (a :: as)
           = ff                                : by rw esVacia_cons
       ... = esVacia (a :: conc as (a :: as))    : by rw esVacia_cons
       ... = esVacia (conc (a :: as) (a :: as)) : by rw conc_cons)

-- 5ª demostración
example : esVacia xs = esVacia (conc xs xs) :=
list.cases_on xs
```

```
  (by simp)
  (by simp)

-- 6ª demostración
lemma esVacia_conc_1
  : ∀ xs : list α, esVacia xs = esVacia (conc xs xs)
| []        := by calc
    esVacia [] = esVacia (conc [] []) : by rw conc_nil
| (a :: as) := by calc
    esVacia (a :: as)
        = ff                             : by rw esVacia_cons
    ... = esVacia (a :: conc as (a :: as))   : by rw esVacia_cons
    ... = esVacia (conc (a :: as) (a :: as)) : by rw conc_cons

-- 7ª demostración
lemma esVacia_conc_2
  : ∀ xs : list α, esVacia xs = esVacia (conc xs xs)
| []        := by simp
| (a :: as) := by simp

-- Comentarios sobre la función is_nil.
-- + Es equivalente a la función esVacia.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval is_nil ([] : list ℕ)
--      #eval is_nil [1]
-- + Se puede demostrar. Por ejemplo,
example : is_nil xs = is_nil (xs ++ xs) :=
by cases xs ; finish
```

8.6. Heurística de generalización

8.6.1. Pruebas de equivalencia entre definiciones de inversa (Heurística de generalización)

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

```
-- Pruebas de la equivalencia entre definiciones de inversa
-- ===========================================================

import data.list.basic
open list

variable  {α : Type*}
variable  (x : α)
variables (xs ys : list α)

-- -------------------------------------------------------
-- Nota. Se usará la función inversa y sus propiedades
-- estudiadas anteriormente.
-- -------------------------------------------------------

def inversa : list α → list α
| []        := []
| (x :: xs) := inversa xs ++ [x]

@[simp]
lemma inversa_nil :
  inversa ([] : list α) = [] :=
rfl

@[simp]
lemma inversa_cons :
  inversa (x :: xs) = inversa xs ++ [x] :=
rfl

-- -------------------------------------------------------
-- Ejercicio 1. Definir la función
--    inversaAc : list α → list α
-- tal que (inversaAc xs) es a inversa de xs calculada
-- usando acumuladores. Por ejemplo,
--    inversaAc [1,3,2,5] = [5,3,2,1]
-- -------------------------------------------------------

def inversaAcAux : list α → list α → list α
| []        ys := ys
| (x :: xs) ys := inversaAcAux xs (x :: ys)

@[simp]
def inversaAc : list α → list α :=
λ xs, inversaAcAux xs []
```

```
-- #eval inversaAc [1,3,2,5]

-- ---------------------------------------------------
-- Ejercicio 2.Demostrar los siguientes lemas
-- + inversaAcAux_nil :
--      inversaAcAux [] ys = ys
-- + inversaAcAux_cons :
--      inversaAcAux (x :: xs) ys =
--      inversaAcAux xs (x :: ys)
-- ---------------------------------------------------

@[simp]
lemma inversaAcAux_nil :
  inversaAcAux [] ys = ys :=
rfl

@[simp]
lemma inversaAcAux_cons :
  inversaAcAux (x :: xs) ys = inversaAcAux xs (x :: ys) :=
rfl

-- ---------------------------------------------------
-- Ejercicio 3. Demostrar que
--    inversaAc [1,2,3] = inversa [1,2,3]
-- ---------------------------------------------------

example : inversaAc [1,2,3] = inversa [1,2,3] :=
rfl

-- ---------------------------------------------------
-- Ejercicio 4. (p. 44) Demostrar que
--    inversaAcAux xs ys = (inversa xs) ++ ys
-- ---------------------------------------------------

-- 1ª demostración
example :
  ∀ ys, inversaAcAux xs ys = (inversa xs) ++ ys :=
begin
  induction xs with a as HI,
  { intro,
    rw inversaAcAux_nil,
    rw inversa_nil,
    rw nil_append, },
  { intro,
    rw inversaAcAux_cons,
```

```
    rw (HI (a :: ys)),
    rw inversa_cons,
    rw append_assoc,
    rw singleton_append, },
end

-- 2ª demostración
example :
  ∀ ys, inversaAcAux xs ys = (inversa xs) ++ ys :=
begin
  induction xs with a as HI,
  { intro,
    calc inversaAcAux [] ys
         = ys                 : by rw inversaAcAux_nil
     ... = [] ++ ys           : by rw nil_append
     ... = inversa [] ++ ys : by rw inversa_nil },
  { intro,
    calc inversaAcAux (a :: as) ys
         = inversaAcAux as (a :: ys) : by rw inversaAcAux_cons
     ... = inversa as ++ (a :: ys)   : by rw (HI (a :: ys))
     ... = inversa as ++ ([a] ++ ys) : by rw singleton_append
     ... = (inversa as ++ [a]) ++ ys : by rw append_assoc
     ... = inversa (a :: as) ++ ys   : by rw inversa_cons },
end

-- 3ª demostración
example :
  inversaAcAux xs ys = (inversa xs) ++ ys :=
begin
  induction xs with a as HI generalizing ys,
  { rw inversaAcAux_nil,
    rw inversa_nil,
    rw nil_append, },
  { rw inversaAcAux_cons,
    rw (HI (a :: ys)),
    rw inversa_cons,
    rw append_assoc,
    rw singleton_append, },
end

-- 4ª demostración
example :
  inversaAcAux xs ys = (inversa xs) ++ ys :=
begin
  induction xs with a as HI generalizing ys,
```

```
  { calc inversaAcAux [] ys
        = ys               : by rw inversaAcAux_nil
    ... = [] ++ ys         : by rw nil_append
    ... = inversa [] ++ ys : by rw inversa_nil },
  { calc inversaAcAux (a :: as) ys
        = inversaAcAux as (a :: ys) : by rw inversaAcAux_cons
    ... = inversa as ++ (a :: ys)   : by rw (HI (a :: ys))
    ... = inversa as ++ ([a] ++ ys) : by rw singleton_append
    ... = (inversa as ++ [a]) ++ ys : by rw append_assoc
    ... = inversa (a :: as) ++ ys   : by rw inversa_cons },
end

-- 5ª demostración
example :
  inversaAcAux xs ys = (inversa xs) ++ ys :=
begin
  induction xs with a as HI generalizing ys,
  { simp, },
  { simp [HI (a :: ys)], },
end

-- 6ª demostración
example :
  inversaAcAux xs ys = (inversa xs) ++ ys :=
by induction xs generalizing ys ; simp [*]

-- 7ª demostración
@[simp]
lemma inversa_equiv :
  ∀ xs : list α, ∀ ys, inversaAcAux xs ys = (inversa xs) ++ ys
| []         := by simp
| (a :: as)  := by simp [inversa_equiv as]

-- ----------------------------------------------------
-- Ejercicio 5. (p. 43) Demostrar que
--    inversaAc xs = inversa xs
-- ----------------------------------------------------

-- 1ª demostración
example : inversaAc xs = inversa xs :=
calc inversaAc xs
     = inversaAcAux xs [] : rfl
 ... = inversa xs ++ []   : by rw inversa_equiv
 ... = inversa xs         : by rw append_nil
```

```
-- 2ª demostración
example : inversaAc xs = inversa xs :=
by simp [inversa_equiv]

-- 3ª demostración
example : inversaAc xs = inversa xs :=
by simp
```

8.7. Inducción para funciones de orden superior

8.7.1. Pruebas de la relación entre length y map.

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

```
-- Pruebas de la relación entre length y map
-- ==========================================

import data.nat.basic
open nat
open list

variables {α : Type*} {β : Type*}
variable  (x : α)
variables (xs : list α)

-- ------------------------------------------------------
-- Nota. Se usarán la función longitud y sus
-- propiedades estudiadas anteriormente.
-- ------------------------------------------------------

def longitud : list α → nat
| []        := 0
| (x :: xs) := longitud xs + 1

@[simp]
lemma longitud_nil :
  longitud ([] : list α) = 0 :=
rfl

@[simp]
lemma longitud_cons :
```

```
  longitud (x :: xs) = longitud xs + 1 :=
rfl

-- ----------------------------------------------------
-- Ejercicio 3. Definir la función
--    aplica : (α → β) → list α → 'b list
-- tal que (aplica f xs) es la lista obtenida
-- aplicando la función f a los elementos de xs. Por
-- ejemplo,
--    aplica (λx, 2*x) [3,2,5] = [6,4,10]
--    aplica ((*) 2)   [3,2,5] = [6,4,10]
--    aplica ((+) 2)   [3,2,5] = [5,4,7]
-- ----------------------------------------------------

def aplica : (α → β) → list α → list β
| f []        := []
| f (x :: xs) := (f x) :: aplica f xs

-- #eval aplica (λx, 2*x) [3,2,5]
-- #eval aplica ((*) 2) [3,2,5]
-- #eval aplica ((+) 2) [3,2,5]

-- ----------------------------------------------------
-- Ejercicio 4. Demostrar los siguientes lemas
-- + aplica_nil :
--      aplica f [] = []
-- + aplica_cons :
--      aplica f (x :: xs) = (f x) :: aplica f xs
-- ----------------------------------------------------

@[simp]
lemma aplica_nil
  (f : α → β)
  : aplica f [] = [] :=
rfl

@[simp]
lemma aplica_cons
  (f : α → β)
  : aplica f (x :: xs) = (f x) :: aplica f xs :=
rfl

-- ----------------------------------------------------
-- Ejercicio 1. (p. 48) Demostrar que
--    longitud (aplica f xs) = longitud xs
```

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

-- 1ª demostración
example
  (f : α → β)
  : longitud (aplica f xs) = longitud xs :=
begin
  induction xs with a as HI,
  { rw aplica_nil,
    rw longitud_nil,
    rw longitud_nil, },
  { rw aplica_cons,
    rw longitud_cons,
    rw HI,
    rw longitud_cons, },
end

-- 2ª demostración
example
  (f : α → β)
  : longitud (aplica f xs) = longitud xs :=
begin
  induction xs with a as HI,
  { calc longitud (aplica f [])
         = longitud []                    : by rw aplica_nil
     ... = 0                              : by rw longitud_nil
     ... = longitud []                    : by rw longitud_nil, },
  { calc longitud (aplica f (a :: as))
         = longitud (f a :: aplica f as) : by rw aplica_cons
     ... = longitud (aplica f as) + 1     : by rw longitud_cons
     ... = longitud as + 1                : by rw HI
     ... = longitud (a :: as)             : by rw longitud_cons, },
end

-- 3ª demostración
example
  (f : α → β)
  : longitud (aplica f xs) = longitud xs :=
begin
  induction xs with x xs HI,
  { simp, },
  { simp [HI], },
end

-- 4ª demostración
```

```
example
  (f : α → β)
  : longitud (aplica f xs) = longitud xs :=
by induction xs ; simp [*]

-- 5ª demostración
lemma longitud_aplica
  (f : α → β)
  : ∀ xs, longitud (aplica f xs) = longitud xs
| []        := by simp
| (a :: as) := by simp [longitud_aplica as]

-- Comentarios sobre las función map:
-- + Es equivalente a la función aplica.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
--      open list
-- + Se puede evaluar. Por ejemplo,
--      #eval map (λx, 2*x) [3,2,5]
--      #eval map ((*) 2) [3,2,5]
--      #eval map ((+) 2) [3,2,5]
-- + Se puede demostrar. Por ejemplo,
example
  (f : α → β)
  : length (map f xs) = length xs :=
by induction xs ; simp [*]

example
  (f : α → β)
  : length (map f xs) = length xs :=
-- by suggest
length_map f xs

example
  (f : α → β)
  : length (map f xs) = length xs :=
by simp
```

8.7.2. Pruebas de la distributiva del producto sobre sumas

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

```
-- Pruebas de la distributiva del producto sobre sumas
-- ===================================================

import data.nat.basic
open nat
open list

variables {α : Type*} {β : Type*}
variable  (x : α)
variables (xs : list α)
variable  (n : ℕ)
variable  (ns : list ℕ)

-- ----------------------------------------------------
-- Nota. Se usará la función aplica y sus propiedades
-- estudiadas anteriormente.
-- ----------------------------------------------------

def aplica : (α → β) → list α → list β
| f []        := []
| f (x :: xs) := (f x) :: aplica f xs

@[simp]
lemma aplica_nil
  (f : α → β)
  : aplica f [] = [] :=
rfl

@[simp]
lemma aplica_cons
  (f : α → β)
  : aplica f (x :: xs) = (f x) :: aplica f xs :=
rfl

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    suma : list ℕ → ℕ
-- tal que (suma xs) es la suma de los elementos de
-- xs. Por ejemplo,
--    suma [3,2,5] = 10
```

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

def suma : list ℕ → ℕ
| []        := 0
| (n :: ns) := n + suma ns

-- #eval suma [3,2,5]

-- ----------------------------------------------------
-- Ejercicio 2. Demostrar los siguientes lemas
-- + suma_nil :
--      suma ([] : list ℕ) = 0 :=
-- + suma_cons :
--      suma (n :: ns) = n + suma ns :=
-- ----------------------------------------------------

@[simp]
lemma suma_nil :
  suma ([] : list ℕ) = 0 :=
rfl

@[simp]
lemma suma_cons :
  suma (n :: ns) = n + suma ns :=
rfl

-- ----------------------------------------------------
-- Ejercicio 3. (p. 45) Demostrar que
--    suma (aplica (λ x, 2*x) ns) = 2 * (suma ns)
-- ----------------------------------------------------

-- 1ª demostración
example :
  suma (aplica (λ x, 2*x) ns) = 2 * (suma ns) :=
begin
  induction ns with m ms HI,
  { rw aplica_nil,
    rw suma_nil,
    rw nat.mul_zero, },
  { rw aplica_cons,
    rw suma_cons,
    rw HI,
    rw suma_cons,
    rw mul_add, },
end
```

```
-- 2ª demostración
example :
  suma (aplica (λ x, 2*x) ns) = 2 * (suma ns) :=
begin
  induction ns with m ms HI,
  { calc suma (aplica (λ (x : ℕ), 2 * x) [])
         = suma []                              : by rw aplica_nil
     ... = 0                                    : by rw suma_nil
     ... = 2 * 0                                : by rw nat.mul_zero
     ... = 2 * suma []                          : by rw suma_nil, },
  { calc suma (aplica (λ x, 2 * x) (m :: ms))
         = suma (2 * m :: aplica (λ x, 2 * x) ms) : by rw aplica_cons
     ... = 2 * m + suma (aplica (λ x, 2 * x) ms)  : by rw suma_cons
     ... = 2 * m + 2 * suma ms                  : by rw HI
     ... = 2 * (m + suma ms)                    : by rw mul_add
     ... = 2 * suma (m :: ms)                   : by rw suma_cons, },
end

-- 3ª demostración
example :
  suma (aplica (λ x, 2*x) ns) = 2 * (suma ns) :=
begin
  induction ns with m ms HI,
  { simp, },
  { simp [HI, mul_add], },
end

-- 4ª demostración
example :
  suma (aplica (λ x, 2*x) ns) = 2 * (suma ns) :=
by induction ns ; simp [*, mul_add]

-- 5ª demostración
lemma suma_aplica :
  ∀ ns, suma (aplica (λ x, 2*x) ns) = 2 * (suma ns)
| []        := by simp
| (m :: ms) := by simp [suma_aplica ms, mul_add]

-- Comentarios sobre las funciones sum y map:
-- + Son equivalentes a las funciones suma y aplica.
-- + Para usarla hay que importar la librería
--   data.list.basic y abrir el espacio de nombre
--   list escribiendo al principio del fichero
--      import data.list.basic
```

```
--       open list
-- + Se puede evaluar. Por ejemplo,
--       #eval sum [3,2,5]
--       #eval map (λx, 2*x) [3,2,5]
--       #eval map ((*) 2) [3,2,5]
--       #eval map ((+) 2) [3,2,5]
```

Capítulo 9

Tipos inductivos

9.1. Tipos abreviados

9.1.1. Razonamiento con tipos abreviados: Posiciones

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

```
-- Razonamiento con tipos abreviados: Posiciones
-- ==============================================

import tactic

-- ----------------------------------------------------
-- Ejercicio 1. Definir el tipo Pos como una abreviatura
-- de pares de enteros para representar posiciones.
-- ----------------------------------------------------

abbreviation Pos : Type := ℤ × ℤ

-- ----------------------------------------------------
-- Ejercicio 2. Definir la posición origen.
-- ----------------------------------------------------

def origen : Pos :=
(0,0)

-- ----------------------------------------------------
-- Ejercicio 3. Definir la función
--    izquierda : Pos → Pos
-- tal que (izquierda p) esla posición que se encuentra
-- a la izquierda de p. Por ejemplo,
--    izquierda (3,5) = (2,5)
```

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

-- 1ª definición
@[simp]
def izquierda : Pos → Pos :=
λ ⟨x,y⟩, (x-1,y)

-- 2ª definición
@[simp]
def izquierda_2 : Pos → Pos
| (x,y) := (x-1,y)

-- #eval izquierda (3,5)
-- Da: (2,5)

-- ----------------------------------------------------
-- Ejercicio 4. Definir la función
--    derecha : Pos → Pos
-- tal que (derecha p) esla posición que se encuentra
-- a la derecha de p. Por ejemplo,
--    derecha (3,5) = (4,5)
-- ----------------------------------------------------

-- 1ª definición
@[simp]
def derecha : Pos → Pos :=
λ ⟨x,y⟩, (x+1,y)

-- 2ª definición
@[simp]
def derecha_2 : Pos → Pos
| (x,y) := (x+1,y)

-- #eval derecha (3,5)
-- Da: (4, 5)

-- ----------------------------------------------------
-- Ejercicio 5. Demostrar que para cualquier posición p,
--    izquierda (derecha p) = p
-- ----------------------------------------------------

-- 1ª demostración
lemma izquierda_derecha :
  ∀ p : Pos, izquierda (derecha p) = p
| (x,y) := by calc izquierda (derecha (x, y))
```

```
                   = izquierda (x+1, y)       :by simp
               ... = (x+1-1, y)               :by simp
               ... = (x, y)                   :by simp

-- 2ª demostración
lemma izquierda_derecha_2 :
  ∀ p : Pos, izquierda (derecha p) = p :=
λ ⟨x,y⟩, by simp

-- ----------------------------------------------------
-- Ejercicio 6. Definir el tipo Movimiento para
-- representar los movimientos com funciones desde la
-- posición inicial a la final.
-- ----------------------------------------------------

abbreviation Movimiento : Type := Pos → Pos
```

9.2. Tipos enumerados

9.2.1. Razonamiento con tipos enumerados: Los días de la semana

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

```
-- Razonamiento con tipos enumerados: Los días de la semana
-- ========================================================

-- ----------------------------------------------------
-- Ejercicio ?. Definir el tipo dia cuyos constructores
-- sean los días de la semana.
-- -------------------------------------------------------------------

inductive dia : Type
| lunes     : dia
| martes    : dia
| miercoles : dia
| jueves    : dia
| viernes   : dia
| sabado    : dia
| domingo   : dia

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

```
-- Ejercicio ?. Calcular el tipo del constructor lunes.
-- ----------------------------------------------------

-- #check dia.lunes
-- Da: dia

-- ----------------------------------------------------
-- Ejercicio ?. Abrir el espacio de nombre dia.
-- ----------------------------------------------------

namespace dia

-- ----------------------------------------------------
-- Ejercicio ?. Calcular el tipo del constructor lunes.
-- ----------------------------------------------------

-- #check lunes
-- Da: dia

-- ----------------------------------------------------
-- Ejercicio ?. Calcular la lista de las funciones
-- definidas en el espacio de nombres dia.
-- ----------------------------------------------------

-- #print prefix dia
-- Da:
--    dia : Type
--    dia.cases_on : □ {C : dia → Sort l} (n : dia),
--      C lunes → C martes → C miercoles → C jueves → C viernes → C sabado → C domi
--    dia.domingo : dia
--    dia.domingo.inj : domingo = domingo → true
--    dia.domingo.inj_arrow : domingo = domingo → □ ⦃P : Sort l⦄, (true → P) → P
--    dia.domingo.inj_eq : domingo = domingo = true
--    dia.domingo.sizeof_spec : domingo.sizeof = 1
--    dia.has_sizeof_inst : has_sizeof dia
--    dia.jueves : dia
--    dia.jueves.inj : jueves = jueves → true
--    dia.jueves.inj_arrow : jueves = jueves → □ ⦃P : Sort l⦄, (true → P) → P
--    dia.jueves.inj_eq : jueves = jueves = true
--    dia.jueves.sizeof_spec : jueves.sizeof = 1
--    dia.lunes : dia
--    dia.lunes.inj : lunes = lunes → true
--    dia.lunes.inj_arrow : lunes = lunes → □ ⦃P : Sort l⦄, (true → P) → P
--    dia.lunes.inj_eq : lunes = lunes = true
--    dia.lunes.sizeof_spec : lunes.sizeof = 1
```

```
--    dia.martes : dia
--    dia.martes.inj : martes = martes → true
--    dia.martes.inj_arrow : martes = martes → Π {⦃P : Sort l⦄}, (true → P) → P
--    dia.martes.inj_eq : martes = martes = true
--    dia.martes.sizeof_spec : martes.sizeof = 1
--    dia.miercoles : dia
--    dia.miercoles.inj : miercoles = miercoles → true
--    dia.miercoles.inj_arrow : miercoles = miercoles → Π {⦃P : Sort l⦄}, (true → P) → 
--    dia.miercoles.inj_eq : miercoles = miercoles = true
--    dia.miercoles.sizeof_spec : miercoles.sizeof = 1
--    dia.no_confusion : Π {P : Sort l} {v1 v2 : dia}, v1 = v2 → dia.no_confusion_type 
--    dia.no_confusion_type : Sort l → dia → dia → Sort l
--    dia.rec : Π {C : dia → Sort l},
--      C lunes → C martes → C miercoles → C jueves → C viernes → C sabado → C domi
--    dia.rec_on : Π {C : dia → Sort l} (n : dia),
--      C lunes → C martes → C miercoles → C jueves → C viernes → C sabado → C domi
--    dia.sabado : dia
--    dia.sabado.inj : sabado = sabado → true
--    dia.sabado.inj_arrow : sabado = sabado → Π {⦃P : Sort l⦄}, (true → P) → P
--    dia.sabado.inj_eq : sabado = sabado = true
--    dia.sabado.sizeof_spec : sabado.sizeof = 1
--    dia.sizeof : dia → ℕ
--    dia.viernes : dia
--    dia.viernes.inj : viernes = viernes → true
--    dia.viernes.inj_arrow : viernes = viernes → Π {⦃P : Sort l⦄}, (true → P) → P
--    dia.viernes.inj_eq : viernes = viernes = true
--    dia.viernes.sizeof_spec : viernes.sizeof = 1

-- ----------------------------------------------------
-- Ejercicio ?. Calcular el tipo de las reglas de
-- eliminación (recursores) del tipo dia: dia.rec,
-- dia.rec_on y dia.cases_on.
-- ----------------------------------------------------

-- Recursor (regla de eliminación):
-- #check @dia.rec
-- Da: Π {C : dia → Sort u_1},
--        C lunes →
--        C martes →
--        C miercoles →
--        C jueves →
--        C viernes →
--        C sabado →
--        C domingo
--        → Π (n : dia), C n
```

```
--
-- #check @dia.rec_on
-- Da: □ {C : dia → Sort u_1} (n : dia),
--       C lunes →
--       C martes →
--       C miercoles →
--       C jueves →
--       C viernes →
--       C sabado →
--       C domingo
--
-- #check @dia.cases_on
-- Da: □ {C : dia → Sort u_1} (n : dia),
--       C lunes →
--       C martes →
--       C miercoles →
--       C jueves →
--       C viernes →
--       C sabado →
--       C domingo

-- -------------------------------------------------------
-- Ejercicio ?. Definir la función
--    numero_del_dia : dia → ℕ
-- tal que (numero_del_dia d) es el número del día de
-- la semana de d. Por ejemplo,
--    numero_del_dia martes = 2
-- -------------------------------------------------------

-- 1ª definición
def numero_del_dia (d : dia) : ℕ :=
dia.rec_on d 1 2 3 4 5 6 7

-- 2ª definición
def numero_del_dia_2 (d : dia) : ℕ :=
dia.cases_on d 1 2 3 4 5 6 7

-- 3ª definición
def numero_del_dia_3 : dia → ℕ :=
λ d, dia.rec 1 2 3 4 5 6 7 d

-- -------------------------------------------------------
-- Ejercicio ?. Definir la función
--    siguiente : dia → dia
-- tal que (siguiente d) es el día siguiente a d. Por
```

```
-- ejemplo,
--    siguiente (siguiente jueves) = sabado
-- ----------------------------------------------------

-- 1ª definición
def siguiente : dia → dia :=
λ d, dia.cases_on d martes miercoles jueves viernes sabado domingo lunes

-- 2ª definición
def siguiente_2 (d : dia) : dia :=
dia.cases_on d martes miercoles jueves viernes sabado domingo lunes

-- 3ª definición
def siguiente_3 : dia → dia
| lunes     := martes
| martes    := miercoles
| miercoles := jueves
| jueves    := viernes
| viernes   := sabado
| sabado    := domingo
| domingo   := lunes

-- #reduce siguiente (siguiente jueves)
-- Da: sabado

-- ----------------------------------------------------
-- Ejercicio ?. Definir la función
--    anterior : dia → dia
-- tal que (anterior d) es el día anterior a d. Por
-- ejemplo,
--    siguiente (anterior jueves) = jueves
-- ----------------------------------------------------

-- 1ª definición
def anterior : dia → dia :=
λ d, dia.cases_on d domingo lunes martes miercoles jueves viernes sabado

-- 2ª definición
def anterior_2 (d : dia) : dia :=
dia.cases_on d domingo lunes martes miercoles jueves viernes sabado

-- 3ª definición
def anterior_3 : dia → dia
| lunes     := domingo
| martes    := lunes
```

```
| miercoles := martes
| jueves    := miercoles
| viernes   := jueves
| sabado    := viernes
| domingo   := sabado

-- #reduce siguiente (anterior jueves)
-- Da: jueves

-- ----------------------------------------------------
-- Ejercicio ?. Demostrar que
--    siguiente (anterior jueves) = jueves
-- ----------------------------------------------------

example : siguiente (anterior jueves) = jueves :=
rfl

-- ----------------------------------------------------
-- Ejercicio ?. Demostrar que, para cualquier día d,
--    siguiente (anterior d) = d
-- ----------------------------------------------------

-- 1ª demostración
example (d: dia) :
  siguiente (anterior d) = d :=
dia.cases_on d
  (show siguiente (anterior lunes)     = lunes,     from rfl)
  (show siguiente (anterior martes)    = martes,    from rfl)
  (show siguiente (anterior miercoles) = miercoles, from rfl)
  (show siguiente (anterior jueves)    = jueves,    from rfl)
  (show siguiente (anterior viernes)   = viernes,   from rfl)
  (show siguiente (anterior sabado)    = sabado,    from rfl)
  (show siguiente (anterior domingo)   = domingo,   from rfl)

-- 2ª demostración
example (d: dia) :
  siguiente (anterior d) = d :=
dia.cases_on d rfl rfl rfl rfl rfl rfl rfl

-- 3ª demostración
example (d: dia) :
  siguiente (anterior d) = d :=
begin
 apply dia.cases_on d,
 refl,
```

```
 refl,
 refl,
 refl,
 refl,
 refl,
 refl,
end

-- 4ª demostración
example (d: dia) :
  siguiente (anterior d) = d :=
by apply dia.cases_on d; refl

-- 5ª demostración
example (d: dia) :
  siguiente (anterior d) = d :=
by apply dia.rec_on d; refl

end dia
```

9.3. Árboles binarios

9.3.1. Razonamiento sobre árboles binarios: La función espejo es involutiva

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

```
-- Razonamiento sobre árboles binarios: La función espejo es involutiva
-- ====================================================================

import tactic

variable {α : Type}

-- ----------------------------------------------------
-- Ejercicio 1. Definir un tipo de dato para los
-- árboles binarios, con los constructores hoja y nodo.
-- ----------------------------------------------------

inductive arbol (α : Type) : Type
| hoja : α → arbol
| nodo : α → arbol → arbol → arbol
```

```
-- ----------------------------------------------------
-- Ejercicio 2. Abrir el espacio de nombres arbol
-- ----------------------------------------------------

namespace arbol

-- #print prefix arbol

-- ----------------------------------------------------
-- Ejercicio 3. Definir el árbol correspondiente a
--         3
--        / \
--       2   4
--      / \
--     1   5
-- ----------------------------------------------------

def ejArbol : arbol ℕ :=
  nodo 3 (nodo 2 (hoja 1) (hoja 5)) (hoja 4)

-- ----------------------------------------------------
-- Ejercicio 3. Definir la función
--     repr : arbol α → string
-- tal que (repr a) es la cadena que representa al
-- árbol a. Por ejemplo,
--      #eval repr ejArbol
--      Da: "N 3 (N 2 (H 1) (H 5)) (H 4)"
-- ----------------------------------------------------

def repr [has_repr α] : arbol α → string
| (hoja x)     := "H " ++ has_repr.repr x
| (nodo x i d) := "N " ++ has_repr.repr x ++ " (" ++ repr i ++ ") (" ++ repr d ++ ")"

-- #eval repr ejArbol

-- ----------------------------------------------------
-- Ejercicio 4. Declarar repr la función para
-- representar los árboles. Por ejemplo,
--     #eval ejArbol
--     -- Da: N 3 (N 2 (H 1) (H 5)) (H 4)
-- ----------------------------------------------------

instance [has_repr α] : has_repr (arbol α) := ⟨repr⟩
```

```
-- #eval ejArbol
-- -- Da: N 3 (N 2 (H 1) (H 5)) (H 4)

-- ---------------------------------------------------------------
-- Ejercicio 5. Definir la función
--    espejo : arbol α → arbol α
-- tal que (espejo a) es la imagen especular de a. Por ejmplo,
--    #eval espejo ejArbol
--    -- Da: N 3 (H 4) (N 2 (H 5) (H 1))
-- ----------------------------------------------------

def espejo : arbol α → arbol α
| (hoja x)     := hoja x
| (nodo x i d) := nodo x (espejo d) (espejo i)

-- #eval espejo ejArbol
-- -- Da: N 3 (H 4) (N 2 (H 5) (H 1))

-- ----------------------------------------------------
-- Ejercicio 6. Declarar las siguientes variables:
-- + a i d como variables sobre árboles de tipo α.
-- + x como variable sobre elementos de tipo α.
-- ----------------------------------------------------

variables (a i d : arbol α)
variable  (x : α)

-- ----------------------------------------------------
-- Ejercicio 7. Demostrar los siguientes lemas
-- + espejo_1 :
--      espejo (hoja x) = hoja x
-- + espejo_2 :
--      espejo (nodo x i d) = nodo x (espejo d) (espejo i)
-- ----------------------------------------------------

@[simp]
lemma espejo_1 :
  espejo (hoja x) = hoja x :=
espejo.equations._eqn_1 x

@[simp]
lemma espejo_2 :
  espejo (nodo x i d) = nodo x (espejo d) (espejo i) :=
espejo.equations._eqn_2 x i d
```

```
-- ----------------------------------------------------
-- Ejercicio 8. Demostrar que
--    espejo (espejo a) = a
-- ----------------------------------------------------

-- 1ª demostración
example :
  espejo (espejo a) = a :=
begin
  induction a with x x i d Hi Hd,
  { rw espejo_1,
    rw espejo_1, },
  { rw espejo_2,
    rw espejo_2,
    rw Hi,
    rw Hd, },
end

-- 2ª demostración
example :
  espejo (espejo a) = a :=
begin
  induction a with x x i d Hi Hd,
  { calc espejo (espejo (hoja x))
         = espejo (hoja x)
             : by exact congr_arg espejo (espejo_1 x)
     ... = hoja x
             : by rw espejo_1, },
  { calc espejo (espejo (nodo x i d))
         = espejo (nodo x (espejo d) (espejo i))
             :by exact congr_arg espejo (espejo_2 i d x)
     ... = nodo x (espejo (espejo i)) (espejo (espejo d))
             :by rw espejo_2
     ... = nodo x i (espejo (espejo d))
             :by rw Hi
     ... = nodo x i d
             :by rw Hd, },
end

-- 3ª demostración
example :
  espejo (espejo a) = a :=
begin
  induction a with _ x i d Hi Hd,
  { simp, },
```

```
  { simp [Hi, Hd], },
end

-- 4ª demostración
example :
  espejo (espejo a) = a :=
by induction a ; simp [*]

-- 5ª demostración
example :
  espejo (espejo a) = a :=
arbol.rec_on a
  ( assume x,
    calc espejo (espejo (hoja x))
         = espejo (hoja x)
             : by exact congr_arg espejo (espejo_1 x)
     ... = hoja x
             : by rw espejo_1 )
  ( assume x i d,
    assume Hi : espejo (espejo i) = i,
    assume Hd : espejo (espejo d) = d,
    calc espejo (espejo (nodo x i d))
         = espejo (nodo x (espejo d) (espejo i))
             :by exact congr_arg espejo (espejo_2 i d x)
     ... = nodo x (espejo (espejo i)) (espejo (espejo d))
             :by rw espejo_2
     ... = nodo x i (espejo (espejo d))
             :by rw Hi
     ... = nodo x i d
             :by rw Hd )

-- 6ª demostración
example :
  espejo (espejo a) = a :=
arbol.rec_on a
  (λ x, by simp )
  (λ x i d Hi Hd, by simp [Hi,Hd])

-- 7ª demostración
lemma espejo_espejo :
  ∀ a : arbol α, espejo (espejo a) = a
| (hoja x)     := by simp
| (nodo x i d) := by simp [espejo_espejo i, espejo_espejo d]

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

```
-- Ejercicio 9. Cerrar el espacio de nombres arbol.
-- ----------------------------------------------------

end arbol
```

9.3.2. Razonamiento sobre arboles binarios: Aplanamiento e imagen especular

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

```
-- Razonamiento sobre arboles binarios: Aplanamiento e imagen especular
-- ====================================================================

import tactic
open list

variable {α : Type}

-- ----------------------------------------------------
-- Nota. Se usarán las definiciones de árboles e
-- imagen especular estudiadas anteriormente.
-- ----------------------------------------------------

inductive arbol (α : Type) : Type
| hoja : α → arbol
| nodo : α → arbol → arbol → arbol

namespace arbol

def ejArbol : arbol ℕ :=
  nodo 3 (nodo 2 (hoja 1) (hoja 5)) (hoja 4)

def repr [has_repr α] : arbol α → string
| (hoja x)     := "H " ++ has_repr.repr x
| (nodo x i d) := "N " ++ has_repr.repr x ++ " (" ++ repr i ++ ") (" ++ repr d ++ ")"

instance [has_repr α] : has_repr (arbol α) := ⟨repr⟩

def espejo : arbol α → arbol α
| (hoja x)     := hoja x
| (nodo x i d) := nodo x (espejo d) (espejo i)

variables (a i d : arbol α)
```

```
variable  (x : α)

@[simp]
lemma espejo_1 :
  espejo (hoja x) = hoja x :=
espejo.equations._eqn_1 x

@[simp]
lemma espejo_2 :
  espejo (nodo x i d) = nodo x (espejo d) (espejo i) :=
espejo.equations._eqn_2 x i d

-- ----------------------------------------------------
-- Ejercicio 1. Definir la función
--    aplana : arbol α → list α
-- tal que (aplana a) es la lista obtenida aplanando
-- el árbole a recorriéndolo en orden infijo. Por
-- ejemplo,
--    #eval aplana ejArbol
--    -- Da: [1, 2, 5, 3, 4]
-- ----------------------------------------------------

def aplana : arbol α → list α
| (hoja x)     := [x]
| (nodo x i d) := (aplana i) ++ [x] ++ (aplana d)

-- #eval aplana ejArbol
-- -- Da: [1, 2, 5, 3, 4]

-- ----------------------------------------------------
-- Ejercicio 7. Demostrar los siguientes lemas
-- + aplana_1 :
--      aplana (hoja x) = [x]
-- + aplana_2 :
--      aplana (nodo x i d) = (aplana i) ++ [x] ++ (aplana d)
-- ----------------------------------------------------

@[simp]
lemma aplana_1 :
  aplana (hoja x) = [x] :=
aplana.equations._eqn_1 x

@[simp]
lemma aplana_2 :
  aplana (nodo x i d) = (aplana i) ++ [x] ++ (aplana d) :=
```

```
aplana.equations._eqn_2 x i d

-- ---------------------------------------------------
-- Ejercicio 3. Demostrar que
--    aplana (espejo a) = rev (aplana a)
-- ---------------------------------------------------

-- 1ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
begin
  induction a with x x i d Hi Hd,
  { rw espejo_1,
    rw aplana_1,
    rw reverse_singleton, },
  { rw espejo_2,
    rw aplana_2,
    rw [Hi, Hd],
    rw aplana_2,
    rw reverse_append,
    rw reverse_append,
    rw reverse_singleton,
    rw append_assoc, },
end

-- 2ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
begin
  induction a with x x i d Hi Hd,
  { calc aplana (espejo (hoja x))
         = aplana (hoja x)
             :by simp only [espejo_1]
     ... = [x]
             :by rw aplana_1
     ... = reverse [x]
             :by rw reverse_singleton
     ... = reverse (aplana (hoja x))
             :by simp only [aplana_1], },
  { calc aplana (espejo (nodo x i d))
         = aplana (nodo x (espejo d) (espejo i))
             :by simp only [espejo_2]
     ... = aplana (espejo d) ++ [x] ++ aplana (espejo i)
             :by rw aplana_2
     ... = reverse (aplana d) ++ [x] ++ reverse (aplana i)
```

```
            :by rw [Hi, Hd]
    ... = reverse (aplana d) ++ reverse [x] ++ reverse (aplana i)
            :by simp only [reverse_singleton]
    ... = reverse ([x] ++ aplana d) ++ reverse (aplana i)
            :by simp only [reverse_append]
    ... = reverse (aplana i ++ ([x] ++ aplana d))
            :by simp only [reverse_append]
    ... = reverse (aplana i ++ [x] ++ aplana d)
            :by simp only [append_assoc]
    ... = reverse (aplana (nodo x i d))
            :by simp only [aplana_2], },
end

-- 3ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
begin
  induction a with x x i d Hi Hd,
  { simp, },
  { simp [Hi, Hd], },
end

-- 4ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
by induction a ; simp [*]

-- 5ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
arbol.rec_on a
  ( assume x,
    calc aplana (espejo (hoja x))
         = aplana (hoja x)
            :by simp only [espejo_1]
     ... = [x]
            :by rw aplana_1
     ... = reverse [x]
            :by rw reverse_singleton
     ... = reverse (aplana (hoja x))
            :by simp only [aplana_1])
  ( assume x i d,
    assume Hi : aplana (espejo i) = reverse (aplana i),
    assume Hd : aplana (espejo d) = reverse (aplana d),
    calc aplana (espejo (nodo x i d))
```

```
        = aplana (nodo x (espejo d) (espejo i))
            :by simp only [espejo_2]
    ... = aplana (espejo d) ++ [x] ++ aplana (espejo i)
            :by rw aplana_2
    ... = reverse (aplana d) ++ [x] ++ reverse (aplana i)
            :by rw [Hi, Hd]
    ... = reverse (aplana d) ++ reverse [x] ++ reverse (aplana i)
            :by simp only [reverse_singleton]
    ... = reverse ([x] ++ aplana d) ++ reverse (aplana i)
            :by simp only [reverse_append]
    ... = reverse (aplana i ++ ([x] ++ aplana d))
            :by simp only [reverse_append]
    ... = reverse (aplana i ++ [x] ++ aplana d)
            :by simp only [append_assoc]
    ... = reverse (aplana (nodo x i d))
            :by simp only [aplana_2])

-- 6ª demostración
example :
  aplana (espejo a) = reverse (aplana a) :=
arbol.rec_on a
  (λ x, by simp)
  (λ x i d Hi Hd, by simp [Hi, Hd])

-- 7ª demostración
lemma aplana_espejo :
  ∀ a : arbol α, aplana (espejo a) = reverse (aplana a)
| (hoja x)     := by simp
| (nodo x i d) := by simp [aplana_espejo i,
                           aplana_espejo d]

-- -------------------------------------------------------
-- Ejercicio 4. Cerrar el espacio de nombres arbol.
-- -------------------------------------------------------

end arbol
```