

CalcuIemus

(Ejercicios de demostración con Isabelle/HOL y Lean)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 9 de agosto de 2021

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

Se permite:

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

Bajo las condiciones siguientes:

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



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



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

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

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

Índice general

1	Introducción	7
2	Ejercicios de mayo de 2021	9
2.1	Propiedad de monotonía de la intersección	9
2.2	Propiedad semidistributiva de la intersección sobre la unión . . .	13
2.3	Diferencia de diferencia de conjuntos	17
2.4	2ª propiedad semidistributiva de la intersección sobre la unión .	21
2.5	2ª diferencia de diferencia de conjuntos	24
2.6	Conmutatividad de la intersección	29
2.7	Intersección con su unión	35
2.8	Unión con su intersección	39
2.9	Unión con su diferencia	42
2.10	Diferencia de unión e intersección	47
2.11	Unión de los conjuntos de los pares e impares	56
3	Ejercicios de junio de 2021	59
3.1	Intersección de los primos y los mayores que dos	59
3.2	Distributiva de la intersección respecto de la unión general	61
3.3	Intersección de intersecciones	65
3.4	Unión con intersección general	70
3.5	Imagen inversa de la intersección	77
3.6	Imagen de la unión	83
3.7	Imagen inversa de la imagen	91
3.8	Subconjunto de la imagen inversa	94
3.9	Imagen inversa de la imagen de aplicaciones inyectivas	97
3.10	Imagen de la imagen inversa	100
3.11	Imagen de imagen inversa de aplicaciones suprayectivas	104
3.12	Monotonía de la imagen de conjuntos	107
3.13	Monotonía de la imagen inversa	110
3.14	Imagen inversa de la unión	114
3.15	Imagen de la intersección	120

3.16 Imagen de la intersección de aplicaciones inyectivas	124
3.17 Imagen de la diferencia de conjuntos	128
3.18 Imagen inversa de la diferencia	132
3.19 Intersección con la imagen	135
3.20 Unión con la imagen	142
3.21 Intersección con la imagen inversa	146
3.22 Unión con la imagen inversa	149
3.23 Imagen de la unión general	153
3.24 Imagen de la intersección general	158
3.25 Imagen de la intersección general mediante inyectiva	161
3.26 Imagen inversa de la unión general	165
3.27 Imagen inversa de la intersección general	168
3.28 Teorema de Cantor	172
3.29 En los monoides, los inversos a la izquierda y a la derecha son iguales	177
3.30 Producto_de_potencias_de_la_misma_base_en_monoides	181
4 Ejercicios de julio de 2021	187
4.1 Equivalencia de inversos iguales al neutro	187
4.2 Unicidad de inversos en monoides	191
4.3 Caracterización de producto igual al primer factor	194
4.4 Unicidad del elemento neutro en los grupos	197
4.5 Unicidad de los inversos en los grupos	200
4.6 Inverso del producto	203
4.7 Inverso del inverso en grupos	207
4.8 Propiedad cancelativa en grupos	211
4.9 Potencias de potencias en monoides	217
4.10 Los monoides booleanos son conmutativos	222
4.11 Límite de sucesiones constantes	226
4.12 Unicidad del límite de las sucesiones convergentes	230
4.13 Límite cuando se suma una constante	234
4.14 Límite de la suma de sucesiones convergentes	237
4.15 Límite multiplicado por una constante	243
4.16 El límite de u es a si y sólo si el de $u-a$ es 0	247
4.17 Producto de sucesiones convergentes a cero	251
4.18 Teorema del emparejado	255
4.19 La composición de crecientes es creciente	260
4.20 La composición de una función creciente y una decreciente es decreciente	264
4.21 Una función creciente e involutiva es la identidad	268
4.22 Si ' f $x \leq f y \rightarrow x \leq y$ ', entonces f es inyectiva	273

4.23 Los supremos de las sucesiones crecientes son sus límites	275
4.24 Un número es par syss lo es su cuadrado	280
4.25 Acotación de sucesiones convergente	285
4.26 La paradoja del barbero	288
4.27 Propiedad de la densidad de los reales	291
4.28 Propiedad cancelativa del producto de números naturales	296
4.29 Límite de sucesión menor que otra sucesión	301
4.30 Las sucesiones acotadas por cero son nulas	308
4.31 Producto de una sucesión acotada por otra convergente a cero .	311
5 Ejercicios de agosto de 2021	317
5.1 La congruencia módulo 2 es una relación de equivalencia	317
5.2 Las funciones con inversa por la izquierda son inyectivas	323
5.3 Las funciones inyectivas tienen inversa por la izquierda	326
5.4 Una función tiene inversa por la izquierda si y solo si es inyectiva	330
5.5 Las funciones con inversa por la derecha son suprayectivas	333
5.6 Las funciones suprayectivas tienen inversa por la derecha	337
5.7 Una función tiene inversa por la derecha si y solo si es suprayectiva	341
5.8 Las funciones con inversa son biyectivas	343
Índice alfabético	347
Bibliografía	350

Capítulo 1

Introducción

En el blog [Calculemus](#) se han ido proponiendo ejercicios de demostración de resultados matemáticos usando [sistemas de demostración interactiva](#). En este libro se hace una recopilación de las soluciones a dichos ejercicios usando [Isabelle/HOL](#) (versión de 2021) y [Lean](#) (versión 3.30.0). La ordenación de los ejercicios es simplemente temporal según su fecha de publicación en [Calculemus](#) y el orden de los ejercicios en [Calculemus](#) responde a los que me voy encontrando en mis [lecturas](#). En futuras versiones del libro está previsto cambiar la ordenación por otra temática; de momento, he añadido al final un índice temático.

Por otra parte, este libro es una continuación del [DAO \(Demostración Asistida por Ordenador\) con Lean](#) con el que comparte el objetivo de usarse 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.

En cada ejercicio, se exponen distintas soluciones ordenadas desde las más detalladas a las más automáticas. En primer lugar, se presentan las demostraciones con Isabelle (que al estar escritas con Isar su formato se aproxima a las de lenguaje natural) y a continuación se presentan las demostraciones con Lean (además, para facilitar su lectura, se proporciona un enlace que al pulsarlo abre las demostraciones en Lean Web (en una sesión del navegador) de forma que se puede navegar por las pruebas y editar otras alternativas),

Las soluciones del libro están en [este repositorio de GitHub](#).

El libro se irá actualizando periódicamente con los nuevos ejercicios que se proponen diariamente en [Calculemus](#).

Capítulo 2

Ejercicios de mayo de 2021

2.1. Propiedad de monotonía de la intersección

2.1.1. Demostraciones con Isabelle/HOL

```
theory Propiedad_de_monotonia_de_la_interseccion
imports Main
begin

(* -----
-- Demostrar que si
--    $s \subseteq t$ 
-- entonces
--    $s \cap u \subseteq t \cap u$ 
-- ----- *)

(* 1ª solución *)
lemma
  assumes "s ⊆ t"
  shows "s ∩ u ⊆ t ∩ u"
proof (rule subsetI)
  fix x
  assume hx: "x ∈ s ∩ u"
  have xs: "x ∈ s"
    using hx
    by (simp only: IntD1)
  then have xt: "x ∈ t"
    using assms
    by (simp only: subset_eq)
  have xu: "x ∈ u"
    using hx
```

```

    by (simp only: IntD2)
  show "x ∈ t ∩ u"
    using xt xu
    by (simp only: Int_iff)
qed

```

```

(* 2 solución *)

```

```

lemma

```

```

  assumes "s ⊆ t"
  shows "s ∩ u ⊆ t ∩ u"

```

```

proof

```

```

  fix x
  assume hx: "x ∈ s ∩ u"
  have xs: "x ∈ s"
    using hx
    by simp
  then have xt: "x ∈ t"
    using assms
    by auto
  have xu: "x ∈ u"
    using hx
    by simp
  show "x ∈ t ∩ u"
    using xt xu
    by simp

```

```

qed

```

```

(* 3ª solución *)

```

```

lemma

```

```

  assumes "s ⊆ t"
  shows "s ∩ u ⊆ t ∩ u"

```

```

using assms

```

```

by auto

```

```

(* 4ª solución *)

```

```

lemma

```

```

  "s ⊆ t ⇒ s ∩ u ⊆ t ∩ u"

```

```

by auto

```

```

end

```

2.1.2. Demostraciones con Lean

```

-- -----
-- Demostrar que si
--    $s \subseteq t$ 
-- entonces
--    $s \cap u \subseteq t \cap u$ 
-- -----

import data.set.basic
open set

variable {α : Type}
variables s t u : set α

-- 1ª demostración
-- =====

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
begin
  rw subset_def,
  rw inter_def,
  rw inter_def,
  dsimp,
  intros x h,
  cases h with xs xu,
  split,
  { rw subset_def at h,
    apply h,
    assumption },
  { assumption },
end

-- 2ª demostración
-- =====

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
begin
  rw [subset_def, inter_def, inter_def],
  dsimp,
  rintros x ⟨xs, xu⟩,

```

```

    rw subset_def at h,
    exact ⟨h _ xs, xu⟩,
end

-- 3ª demostración
-- =====

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
begin
  simp only [subset_def, mem_inter_eq] at *,
  rintro x ⟨xs, xu⟩,
  exact ⟨h _ xs, xu⟩,
end

-- 4ª demostración
-- =====

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
begin
  intro x xsu,
  exact ⟨h xsu.1, xsu.2⟩,
end

-- 5ª demostración
-- =====

example
  (h : s ⊆ t)
  : s ∩ u ⊆ t ∩ u :=
inter_subset_inter_left u h

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.2. Propiedad semidistributiva de la intersección sobre la unión

2.2.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--    $s \cap (t \cup u) \subseteq (s \cap t) \cup (s \cap u)$ 
-- ----- *)

theory Propiedad_semidistributiva_de_la_interseccion_sobre_la_union
imports Main
begin

(* 1ª demostración *)
lemma "s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)"
proof (rule subsetI)
  fix x
  assume hx : "x ∈ s ∩ (t ∪ u)"
  then have xs : "x ∈ s"
    by (simp only: IntD1)
  have xtu : "x ∈ t ∪ u"
    using hx by (simp only: IntD2)
  then have "x ∈ t ∨ x ∈ u"
    by (simp only: Un_iff)
  then show "x ∈ s ∩ t ∨ s ∩ u"
  proof (rule disjE)
    assume xt : "x ∈ t"
    have xst : "x ∈ s ∩ t"
      using xs xt by (simp only: Int_iff)
    then show "x ∈ (s ∩ t) ∪ (s ∩ u)"
      by (simp only: UnI1)
  next
    assume xu : "x ∈ u"
    have xsu : "x ∈ s ∩ u"
      using xs xu by (simp only: Int_iff)
    then show "x ∈ (s ∩ t) ∪ (s ∩ u)"
      by (simp only: UnI2)
  qed
qed

(* 2ª demostración *)
lemma "s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)"

```

```

proof
  fix x
  assume hx : "x ∈ s n (t u u)"
  then have xs : "x ∈ s"
    by simp
  have xtu: "x ∈ t u u"
    using hx by simp
  then have "x ∈ t v x ∈ u"
    by simp
  then show " x ∈ s n t u s n u"
  proof
    assume xt : "x ∈ t"
    have xst : "x ∈ s n t"
      using xs xt by simp
    then show "x ∈ (s n t) u (s n u)"
      by simp
  next
    assume xu : "x ∈ u"
    have xst : "x ∈ s n u"
      using xs xu by simp
    then show "x ∈ (s n t) u (s n u)"
      by simp
  qed
qed

(* 3ª demostración *)
lemma "s n (t u u) ⊆ (s n t) u (s n u)"
proof (rule subsetI)
  fix x
  assume hx : "x ∈ s n (t u u)"
  then have xs : "x ∈ s"
    by (simp only: IntD1)
  have xtu: "x ∈ t u u"
    using hx by (simp only: IntD2)
  then show " x ∈ s n t u s n u"
  proof (rule UnE)
    assume xt : "x ∈ t"
    have xst : "x ∈ s n t"
      using xs xt by (simp only: Int_iff)
    then show "x ∈ (s n t) u (s n u)"
      by (simp only: UnI1)
  next
    assume xu : "x ∈ u"
    have xst : "x ∈ s n u"
      using xs xu by (simp only: Int_iff)

```

```

    then show "x ∈ (s ∩ t) ∪ (s ∩ u)"
      by (simp only: UnI2)
  qed
qed

(* 4ª demostración *)
lemma "s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)"
proof
  fix x
  assume hx : "x ∈ s ∩ (t ∪ u)"
  then have xs : "x ∈ s"
    by simp
  have xtu : "x ∈ t ∪ u"
    using hx by simp
  then show "x ∈ s ∩ t ∪ s ∩ u"
  proof (rule UnE)
    assume xt : "x ∈ t"
    have xst : "x ∈ s ∩ t"
      using xs xt by simp
    then show "x ∈ (s ∩ t) ∪ (s ∩ u)"
      by simp
  next
    assume xu : "x ∈ u"
    have xsu : "x ∈ s ∩ u"
      using xs xu by simp
    then show "x ∈ (s ∩ t) ∪ (s ∩ u)"
      by simp
  qed
qed

(* 5ª demostración *)
lemma "s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)"
by (simp only: Int_Un_distrib)

(* 6ª demostración *)
lemma "s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ (s ∩ u)"
by auto

end

```

2.2.2. Demostraciones con Lean

```
-- Demostrar que
--    $s \cap (t \cup u) \subseteq (s \cap t) \cup (s \cap u)$ 
--
```

```
import data.set.basic
```

```
open set
```

```
variable {α : Type}
```

```
variables s t u : set α
```

```
-- 1ª demostración
```

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

```
example :
```

```
   $s \cap (t \cup u) \subseteq (s \cap t) \cup (s \cap u) :=$ 
```

```
begin
```

```
  intros x hx,
```

```
  have xs : x ∈ s := hx.1,
```

```
  have xtu : x ∈ t ∪ u := hx.2,
```

```
  clear hx,
```

```
  cases xtu with xt xu,
```

```
  { left,
```

```
    show x ∈ s ∩ t,
```

```
    exact ⟨xs, xt⟩ },
```

```
  { right,
```

```
    show x ∈ s ∩ u,
```

```
    exact ⟨xs, xu⟩ },
```

```
end
```

```
-- 2ª demostración
```

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

```
example :
```

```
   $s \cap (t \cup u) \subseteq (s \cap t) \cup (s \cap u) :=$ 
```

```
begin
```

```
  rintros x ⟨xs, xt | xu⟩,
```

```
  { left,
```

```
    exact ⟨xs, xt⟩ },
```

```
  { right,
```

```
    exact ⟨xs, xu⟩ },
```

```
end
```



```

-- 3ª demostración
-- =====

example :
  s \ (t ∪ u) ⊆ (s \ t) ∪ (s \ u) :=
begin
  intros x hx,
  by finish
end

-- 4ª demostración
-- =====

example :
  s \ (t ∪ u) ⊆ (s \ t) ∪ (s \ u) :=
by rw inter_union_distrib_left

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.3. Diferencia de diferencia de conjuntos

2.3.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   (s - t) - u ⊆ s - (t ∪ u)
-- ----- *)

theory Diferencia_de_diferencia_de_conjuntos
imports Main
begin

(* 1ª demostración *)
lemma "(s - t) - u ⊆ s - (t ∪ u)"
proof (rule subsetI)
  fix x
  assume hx : "x ∈ (s - t) - u"
  then show "x ∈ s - (t ∪ u)"
proof (rule DiffE)
  assume xst : "x ∈ s - t"
  assume xnu : "x ∉ u"
  note xst

```

```

then show "x ∈ s - (t ∪ u)"
proof (rule DiffE)
  assume xs : "x ∈ s"
  assume xnt : "x ∉ t"
  have xntu : "x ∉ t ∪ u"
  proof (rule notI)
    assume xtu : "x ∈ t ∪ u"
    then show False
    proof (rule UnE)
      assume xt : "x ∈ t"
      with xnt show False
      by (rule notE)
    next
      assume xu : "x ∈ u"
      with xnu show False
      by (rule notE)
    qed
  qed
  show "x ∈ s - (t ∪ u)"
  using xs xntu by (rule DiffI)
qed
qed
qed

(* 2ª demostración *)
lemma "(s - t) - u ⊆ s - (t ∪ u)"
proof
  fix x
  assume hx : "x ∈ (s - t) - u"
  then have xst : "x ∈ (s - t)"
    by simp
  then have xs : "x ∈ s"
    by simp
  have xnt : "x ∉ t"
    using xst by simp
  have xnu : "x ∉ u"
    using hx by simp
  have xntu : "x ∉ t ∪ u"
    using xnt xnu by simp
  then show "x ∈ s - (t ∪ u)"
    using xs by simp
qed

(* 3ª demostración *)
lemma "(s - t) - u ⊆ s - (t ∪ u)"

```

```

proof
  fix x
  assume "x ∈ (s - t) - u"
  then show "x ∈ s - (t ∪ u)"
    by simp
qed

(* 4ª demostración *)
lemma "(s - t) - u ⊆ s - (t ∪ u)"
by auto

end

```

2.3.2. Demostraciones con Lean

```

-----
-- Demostrar que
--   (s \ t) \ u ⊆ s \ (t ∪ u)
-----

import data.set.basic
open set

variable {α : Type}
variables s t u : set α

-- 1ª demostración
-- =====

example : (s \ t) \ u ⊆ s \ (t ∪ u) :=
begin
  intros x xstu,
  have xs : x ∈ s := xstu.1.1,
  have xnt : x ∉ t := xstu.1.2,
  have xnu : x ∉ u := xstu.2,
  split,
  { exact xs },
  { dsimp,
    intro xtu,
    cases xtu with xt xu,
    { show false, from xnt xt },
    { show false, from xnu xu }},
end

```

```

-- 2ª demostración
-- =====

example : (s  $\wedge$  t)  $\wedge$  u  $\subseteq$  s  $\wedge$  (t  $\vee$  u) :=
begin
  rintros x ⟨xs, xnt⟩, xnu,
  use xs,
  rintros (xt | xu); contradiction
end

-- 3ª demostración
-- =====

example : (s  $\wedge$  t)  $\wedge$  u  $\subseteq$  s  $\wedge$  (t  $\vee$  u) :=
begin
  intros x xstu,
  simp at *,
  finish,
end

-- 4ª demostración
-- =====

example : (s  $\wedge$  t)  $\wedge$  u  $\subseteq$  s  $\wedge$  (t  $\vee$  u) :=
begin
  intros x xstu,
  finish,
end

-- 5ª demostración
-- =====

example : (s  $\wedge$  t)  $\wedge$  u  $\subseteq$  s  $\wedge$  (t  $\vee$  u) :=
by rw diff_diff

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.4. 2ª propiedad semidistributiva de la intersección sobre la unión

2.4.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--       $(s \cap t) \cup (s \cap u) \subseteq s \cap (t \cup u)$ 
-- ----- *)

theory Propiedad_semidistributiva_de_la_interseccion_sobre_la_union_2
imports Main
begin

(* 1ª demostración *)
lemma "(s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)"
proof (rule subsetI)
  fix x
  assume "x ∈ (s ∩ t) ∪ (s ∩ u)"
  then show "x ∈ s ∩ (t ∪ u)"
  proof (rule UnE)
    assume xst : "x ∈ s ∩ t"
    then have xs : "x ∈ s"
      by (simp only: IntD1)
    have xt : "x ∈ t"
      using xst by (simp only: IntD2)
    then have xtu : "x ∈ t ∪ u"
      by (simp only: UnI1)
    show "x ∈ s ∩ (t ∪ u)"
      using xs xtu by (simp only: IntI)
  next
    assume xsu : "x ∈ s ∩ u"
    then have xs : "x ∈ s"
      by (simp only: IntD1)
    have xu : "x ∈ u"
      using xsu by (simp only: IntD2)
    then have xtu : "x ∈ t ∪ u"
      by (simp only: UnI2)
    show "x ∈ s ∩ (t ∪ u)"
      using xs xtu by (simp only: IntI)
  qed
qed
qed
```

```

(* 2ª demostración *)
lemma "(s n t) u (s n u) ⊆ s n (t u u)"
proof
  fix x
  assume "x ∈ (s n t) u (s n u)"
  then show "x ∈ s n (t u u)"
  proof
    assume xst : "x ∈ s n t"
    then have xs : "x ∈ s"
      by simp
    have xt : "x ∈ t"
      using xst by simp
    then have xtu : "x ∈ t u u"
      by simp
    show "x ∈ s n (t u u)"
      using xs xtu by simp
  next
    assume xsu : "x ∈ s n u"
    then have xs : "x ∈ s"
      by (simp only: IntD1)
    have xt : "x ∈ u"
      using xsu by simp
    then have xtu : "x ∈ t u u"
      by simp
    show "x ∈ s n (t u u)"
      using xs xtu by simp
  qed
qed

(* 3ª demostración *)
lemma "(s n t) u (s n u) ⊆ s n (t u u)"
proof
  fix x
  assume "x ∈ (s n t) u (s n u)"
  then show "x ∈ s n (t u u)"
  proof
    assume "x ∈ s n t"
    then show "x ∈ s n (t u u)"
      by simp
  next
    assume "x ∈ s n u"
    then show "x ∈ s n (t u u)"
      by simp
  qed
qed

```

```

(* 4ª demostración *)
lemma "(s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)"
proof
  fix x
  assume "x ∈ (s ∩ t) ∪ (s ∩ u)"
  then show "x ∈ s ∩ (t ∪ u)"
    by auto
qed

(* 5ª demostración *)
lemma "(s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)"
by auto

(* 6ª demostración *)
lemma "(s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)"
by (simp only: distrib_inf_le)

end

```

2.4.2. Demostraciones con Lean

```

-----
-- Demostrar que
--   (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u)
-----

import data.set.basic
open set

variable {α : Type}
variables s t u : set α

-- 1ª demostración
-- =====

example : (s ∩ t) ∪ (s ∩ u) ⊆ s ∩ (t ∪ u) :=
begin
  intros x hx,
  cases hx with xst xsu,
  { split,
    { exact xst.1 },
    { left,
      exact xst.2 }},

```

```

{ split,
  { exact xsu.1 },
  { right,
    exact xsu.2 }},
end

-- 2ª demostración
-- =====

example : (s ⊆ t) ∪ (s ⊆ u) ⊆ s ⊆ (t ∪ u) :=
begin
  rintros x (⟨xs, xt⟩ | ⟨xs, xu⟩),
  { use xs,
    left,
    exact xt },
  { use xs,
    right,
    exact xu },
end

-- 3ª demostración
-- =====

example : (s ⊆ t) ∪ (s ⊆ u) ⊆ s ⊆ (t ∪ u) :=
by rw inter_distrib_left s t u

-- 4ª demostración
-- =====

example : (s ⊆ t) ∪ (s ⊆ u) ⊆ s ⊆ (t ∪ u) :=
begin
  intros x hx,
  finish
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.5. 2ª diferencia de diferencia de conjuntos

2.5.1. Demostraciones con Isabelle/HOL


```

(* -----
-- Demostrar que
--    $s - (t \cup u) \subseteq (s - t) - u$ 
-- ----- *)

theory Diferencia_de_diferencia_de_conjuntos_2
imports Main
begin

(* 1ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
proof (rule subsetI)
  fix x
  assume "x ∈ s - (t ∪ u)"
  then show "x ∈ (s - t) - u"
  proof (rule DiffE)
    assume "x ∈ s"
    assume "x ∉ t ∪ u"
    have "x ∉ u"
    proof (rule notI)
      assume "x ∈ u"
      then have "x ∈ t ∪ u"
        by (simp only: UnI2)
      with "x ∉ t ∪ u" show False
        by (rule notE)
    qed
    have "x ∉ t"
    proof (rule notI)
      assume "x ∈ t"
      then have "x ∈ t ∪ u"
        by (simp only: UnI1)
      with "x ∉ t ∪ u" show False
        by (rule notE)
    qed
    with "x ∈ s" have "x ∈ s - t"
      by (rule DiffI)
    then show "x ∈ (s - t) - u"
      using "x ∉ u" by (rule DiffI)
  qed
qed

(* 2ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
proof
  fix x

```

```

assume "x ∈ s - (t ∪ u)"
then show "x ∈ (s - t) - u"
proof
  assume "x ∈ s"
  assume "x ∉ t ∪ u"
  have "x ∉ u"
  proof
    assume "x ∈ u"
    then have "x ∈ t ∪ u"
    by simp
    with ⟨x ∉ t ∪ u⟩ show False
    by simp
  qed
  have "x ∉ t"
  proof
    assume "x ∈ t"
    then have "x ∈ t ∪ u"
    by simp
    with ⟨x ∉ t ∪ u⟩ show False
    by simp
  qed
  with ⟨x ∈ s⟩ have "x ∈ s - t"
  by simp
  then show "x ∈ (s - t) - u"
  using ⟨x ∉ u⟩ by simp
qed
qed

(* 3ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
proof
  fix x
  assume "x ∈ s - (t ∪ u)"
  then show "x ∈ (s - t) - u"
  proof
    assume "x ∈ s"
    assume "x ∉ t ∪ u"
    then have "x ∉ u"
    by simp
    have "x ∉ t"
    using ⟨x ∉ t ∪ u⟩ by simp
    with ⟨x ∈ s⟩ have "x ∈ s - t"
    by simp
    then show "x ∈ (s - t) - u"
    using ⟨x ∉ u⟩ by simp
  qed

```

```

qed
qed

(* 4ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
proof
  fix x
  assume "x ∈ s - (t ∪ u)"
  then show "x ∈ (s - t) - u"
  proof
    assume "x ∈ s"
    assume "x ∉ t ∪ u"
    then show "x ∈ (s - t) - u"
    using ⟨x ∈ s⟩ by simp
  qed
qed

(* 5ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
proof
  fix x
  assume "x ∈ s - (t ∪ u)"
  then show "x ∈ (s - t) - u"
  by simp
qed

(* 6ª demostración *)
lemma "s - (t ∪ u) ⊆ (s - t) - u"
by auto

end

```

2.5.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   s \ (t ∪ u) ⊆ (s \ t) \ u
-- -----

import data.set.basic
open set

variable {α : Type}

```

```

variables s t u : set  $\alpha$ 

-- 1ª demostración
-- =====

example : s  $\cap$  (t  $\cup$  u)  $\subseteq$  (s  $\cap$  t)  $\cup$  :=
begin
  intros x hx,
  split,
  { split,
    { exact hx.1, },
    { dsimp,
      intro xt,
      apply hx.2,
      left,
      exact xt, }},
  { dsimp,
    intro xu,
    apply hx.2,
    right,
    exact xu, },
end

-- 2ª demostración
-- =====

example : s  $\cap$  (t  $\cup$  u)  $\subseteq$  (s  $\cap$  t)  $\cup$  :=
begin
  rintros x (xs, xntu),
  split,
  { split,
    { exact xs, },
    { intro xt,
      exact xntu (or.inl xt), }},
  { intro xu,
    exact xntu (or.inr xu), },
end

-- 3ª demostración
-- =====

example : s  $\cap$  (t  $\cup$  u)  $\subseteq$  (s  $\cap$  t)  $\cup$  :=
begin
  rintros x (xs, xntu),
  use xs,

```

```

{ intro xt,
  exact xntu (or.inl xt) },
{ intro xu,
  exact xntu (or.inr xu) },
end

-- 4ª demostración
-- =====

example : s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ u :=
begin
  rintros x ⟨xs, xntu⟩;
  finish,
end

-- 5ª demostración
-- =====

example : s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ u :=
by intro ; finish

-- 6ª demostración
-- =====

example : s ∩ (t ∪ u) ⊆ (s ∩ t) ∪ u :=
by rw diff_diff

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.6. Conmutatividad de la intersección

2.6.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   s ∩ t = t ∩ s
-- ----- *)

theory Conmutatividad_de_la_interseccion
imports Main
begin

```

```

(* 1ª demostración *)
lemma "s n t = t n s"
proof (rule set_eqI)
  fix x
  show "x ∈ s n t ↔ x ∈ t n s"
  proof (rule iffI)
    assume h : "x ∈ s n t"
    then have xs : "x ∈ s"
      by (simp only: IntD1)
    have xt : "x ∈ t"
      using h by (simp only: IntD2)
    then show "x ∈ t n s"
      using xs by (rule IntI)
  next
    assume h : "x ∈ t n s"
    then have xt : "x ∈ t"
      by (simp only: IntD1)
    have xs : "x ∈ s"
      using h by (simp only: IntD2)
    then show "x ∈ s n t"
      using xt by (rule IntI)
  qed
qed

(* 2ª demostración *)
lemma "s n t = t n s"
proof (rule set_eqI)
  fix x
  show "x ∈ s n t ↔ x ∈ t n s"
  proof
    assume h : "x ∈ s n t"
    then have xs : "x ∈ s"
      by simp
    have xt : "x ∈ t"
      using h by simp
    then show "x ∈ t n s"
      using xs by simp
  next
    assume h : "x ∈ t n s"
    then have xt : "x ∈ t"
      by simp
    have xs : "x ∈ s"
      using h by simp
    then show "x ∈ s n t"
      using xt by simp
  qed
qed

```

```

qed
qed

(* 3ª demostración *)
lemma "s ∩ t = t ∩ s"
proof (rule equalityI)
  show "s ∩ t ⊆ t ∩ s"
  proof (rule subsetI)
    fix x
    assume h : "x ∈ s ∩ t"
    then have xs : "x ∈ s"
      by (simp only: IntD1)
    have xt : "x ∈ t"
      using h by (simp only: IntD2)
    then show "x ∈ t ∩ s"
      using xs by (rule IntI)
  qed
qed
next
  show "t ∩ s ⊆ s ∩ t"
  proof (rule subsetI)
    fix x
    assume h : "x ∈ t ∩ s"
    then have xt : "x ∈ t"
      by (simp only: IntD1)
    have xs : "x ∈ s"
      using h by (simp only: IntD2)
    then show "x ∈ s ∩ t"
      using xt by (rule IntI)
  qed
qed

(* 4ª demostración *)
lemma "s ∩ t = t ∩ s"
proof
  show "s ∩ t ⊆ t ∩ s"
  proof
    fix x
    assume h : "x ∈ s ∩ t"
    then have xs : "x ∈ s"
      by simp
    have xt : "x ∈ t"
      using h by simp
    then show "x ∈ t ∩ s"
      using xs by simp
  qed
qed

```

```

next
  show "t n s ⊆ s n t"
  proof
    fix x
    assume h : "x ∈ t n s"
    then have xt : "x ∈ t"
      by simp
    have xs : "x ∈ s"
      using h by simp
    then show "x ∈ s n t"
      using xt by simp
  qed
qed

(* 5ª demostración *)
lemma "s n t = t n s"
proof
  show "s n t ⊆ t n s"
  proof
    fix x
    assume "x ∈ s n t"
    then show "x ∈ t n s"
      by simp
  qed
next
  show "t n s ⊆ s n t"
  proof
    fix x
    assume "x ∈ t n s"
    then show "x ∈ s n t"
      by simp
  qed
qed

(* 6ª demostración *)
lemma "s n t = t n s"
by (fact Int_commute)

(* 7ª demostración *)
lemma "s n t = t n s"
by (fact inf_commute)

(* 8ª demostración *)
lemma "s n t = t n s"
by auto

```



```
end
```

2.6.2. Demostraciones con Lean

```

-- -----
--  Demostrar que
--     $s \cap t = t \cap s$ 
-- -----

import data.set.basic
open set

variable {α : Type}
variables s t u : set α

-- 1ª demostración
-- =====

example : s ∩ t = t ∩ s :=
begin
  ext x,
  simp only [mem_inter_eq],
  split,
  { intro h,
    split,
    { exact h.2, },
    { exact h.1, }},
  { intro h,
    split,
    { exact h.2, },
    { exact h.1, }},
end

-- 2ª demostración
-- =====

example : s ∩ t = t ∩ s :=
begin
  ext,
  simp only [mem_inter_eq],
  exact (λ h, ⟨h.2, h.1⟩,
        λ h, ⟨h.2, h.1⟩),
end

```

```

-- 3ª demostración
-- =====

example : s  $\cap$  t = t  $\cap$  s :=
begin
  ext,
  exact ( $\lambda$  h, (h.2, h.1),
         $\lambda$  h, (h.2, h.1)),
end

-- 4ª demostración
-- =====

example : s  $\cap$  t = t  $\cap$  s :=
begin
  ext x,
  simp only [mem_inter_eq],
  split,
  { rintros (xs, xt),
    exact (xt, xs) },
  { rintros (xt, xs),
    exact (xs, xt) },
end

-- 5ª demostración
-- =====

example : s  $\cap$  t = t  $\cap$  s :=
begin
  ext x,
  exact and.comm,
end

-- 6ª demostración
-- =====

example : s  $\cap$  t = t  $\cap$  s :=
ext ( $\lambda$  x, and.comm)

-- 7ª demostración
-- =====

example : s  $\cap$  t = t  $\cap$  s :=
by ext x; simp [and.comm]

```

```

-- 8ª demostración
-- =====

example : s ∩ t = t ∩ s :=
inter_comm s t

-- 9ª demostración
-- =====

example : s ∩ t = t ∩ s :=
by finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.7. Intersección con su unión

2.7.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   s ∩ (s ∪ t) = s
-- ----- *)

theory Interseccion_con_su_union
imports Main
begin

(* 1ª demostración *)
lemma "s ∩ (s ∪ t) = s"
proof (rule equalityI)
  show "s ∩ (s ∪ t) ⊆ s"
  proof (rule subsetI)
    fix x
    assume "x ∈ s ∩ (s ∪ t)"
    then show "x ∈ s"
    by (simp only: IntD1)
  qed
qed
next
  show "s ⊆ s ∩ (s ∪ t)"
  proof (rule subsetI)
    fix x
    assume "x ∈ s"

```

```

    then have "x ∈ s ∪ t"
    by (simp only: UnI1)
    with ⟨x ∈ s⟩ show "x ∈ s ∩ (s ∪ t)"
    by (rule IntI)
  qed
qed

(* 2ª demostración *)
lemma "s ∩ (s ∪ t) = s"
proof
  show "s ∩ (s ∪ t) ⊆ s"
  proof
    fix x
    assume "x ∈ s ∩ (s ∪ t)"
    then show "x ∈ s"
    by simp
  qed
next
  show "s ⊆ s ∩ (s ∪ t)"
  proof
    fix x
    assume "x ∈ s"
    then have "x ∈ s ∪ t"
    by simp
    then show "x ∈ s ∩ (s ∪ t)"
    using ⟨x ∈ s⟩ by simp
  qed
qed

(* 3ª demostración *)
lemma "s ∩ (s ∪ t) = s"
by (fact Un_Int_eq)

(* 4ª demostración *)
lemma "s ∩ (s ∪ t) = s"
by auto

```

2.7.2. Demostraciones con Lean

```

-- Demostrar que
--   s ∩ (s ∪ t) = s

```

```

import data.set.basic
open set

variable {α : Type}
variables s t : set α

-- 1ª demostración
-- =====

example : s ∩ (s ∪ t) = s :=
begin
  ext x,
  split,
  { intros h,
    dsimp at h,
    exact h.1, },
  { intro xs,
    dsimp,
    split,
    { exact xs, },
    { left,
      exact xs, }},
end

-- 2ª demostración
-- =====

example : s ∩ (s ∪ t) = s :=
begin
  ext x,
  split,
  { intros h,
    exact h.1, },
  { intro xs,
    split,
    { exact xs, },
    { left,
      exact xs, }},
end

-- 3ª demostración
-- =====

example : s ∩ (s ∪ t) = s :=

```

```

begin
  ext x,
  split,
  { intros h,
    exact h.l, },
  { intro xs,
    split,
    { exact xs, },
    { exact (or.inl xs), }},
end

-- 4ª demostración
-- =====

example : s ⊓ (s ⊔ t) = s :=
begin
  ext,
  exact (λ h, h.l,
    λ xs, {xs, or.inl xs}),
end

-- 5ª demostración
-- =====

example : s ⊓ (s ⊔ t) = s :=
begin
  ext,
  exact (and.left,
    λ xs, {xs, or.inl xs}),
end

-- 6ª demostración
-- =====

example : s ⊓ (s ⊔ t) = s :=
begin
  ext x,
  split,
  { rintros {xs, _},
    exact xs },
  { intro xs,
    use xs,
    left,
    exact xs },
end

```

```
-- 7ª demostración
-- =====

example : s ∩ (s ∪ t) = s :=
inf_sup_self
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.8. Unión con su intersección

2.8.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--   s ∪ (s ∩ t) = s
-- ----- *)

theory Union_con_su_interseccion
imports Main
begin

(* 1ª demostración *)
lemma "s ∪ (s ∩ t) = s"
proof (rule equalityI)
  show "s ∪ (s ∩ t) ⊆ s"
  proof (rule subsetI)
    fix x
    assume "x ∈ s ∪ (s ∩ t)"
    then show "x ∈ s"
    proof
      assume "x ∈ s"
      then show "x ∈ s"
      by this
    next
      assume "x ∈ s ∩ t"
      then show "x ∈ s"
      by (simp only: IntD1)
    qed
  qed
next
  show "s ⊆ s ∪ (s ∩ t)"
  proof (rule subsetI)
```

```

    fix x
    assume "x ∈ s"
    then show "x ∈ s ∪ (s ∩ t)"
    by (simp only: UnI1)
  qed
qed

(* 2ª demostración *)
lemma "s ∪ (s ∩ t) = s"
proof
  show "s ∪ s ∩ t ⊆ s"
  proof
    fix x
    assume "x ∈ s ∪ (s ∩ t)"
    then show "x ∈ s"
    proof
      assume "x ∈ s"
      then show "x ∈ s"
      by this
    next
      assume "x ∈ s ∩ t"
      then show "x ∈ s"
      by simp
    qed
  qed
next
  show "s ⊆ s ∪ (s ∩ t)"
  proof
    fix x
    assume "x ∈ s"
    then show "x ∈ s ∪ (s ∩ t)"
    by simp
  qed
qed

(* 3ª demostración *)
lemma "s ∪ (s ∩ t) = s"
  by auto
end

```


2.8.2. Demostraciones con Lean

```

-- -----
--  Demostrar que
--     $s \cup (s \cap t) = s$ 
-- -----

import data.set.basic
open set

variable {α : Type}
variables s t : set α

-- 1ª demostración
-- =====

example : s ∪ (s ∩ t) = s :=
begin
  ext x,
  split,
  { intro hx,
    cases hx with xs xst,
    { exact xs, },
    { exact xst.1, }},
  { intro xs,
    left,
    exact xs, },
end

-- 2ª demostración
-- =====

example : s ∪ (s ∩ t) = s :=
begin
  ext x,
  exact (λ hx, or.dcases_on hx id and.left,
        λ xs, or.inl xs),
end

-- 3ª demostración
-- =====

example : s ∪ (s ∩ t) = s :=
begin
  ext x,

```

```

split,
{ rintros (xs | {xs, xt});
  exact xs },
{ intro xs,
  left,
  exact xs },
end

-- 4ª demostración
-- =====

example : s ∪ (s ∩ t) = s :=
sup_inf_self

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.9. Unión con su diferencia

2.9.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
-- (s \ t) ∪ t = s ∪ t
-- ----- *)

theory Union_con_su_diferencia
imports Main
begin

(* 1ª demostración *)

lemma "(s - t) ∪ t = s ∪ t"
proof (rule equalityI)
  show "(s - t) ∪ t ⊆ s ∪ t"
  proof (rule subsetI)
    fix x
    assume "x ∈ (s - t) ∪ t"
    then show "x ∈ s ∪ t"
    proof (rule UnE)
      assume "x ∈ s - t"
      then have "x ∈ s"
        by (simp only: DiffD1)
    end
  end
end

```

```

    then show "x ∈ s ∪ t"
      by (simp only: UnI1)
  next
    assume "x ∈ t"
    then show "x ∈ s ∪ t"
      by (simp only: UnI2)
  qed
qed
next
show "s ∪ t ⊆ (s - t) ∪ t"
proof (rule subsetI)
  fix x
  assume "x ∈ s ∪ t"
  then show "x ∈ (s - t) ∪ t"
  proof (rule UnE)
    assume "x ∈ s"
    show "x ∈ (s - t) ∪ t"
    proof (cases ⟨x ∈ t⟩)
      assume "x ∈ t"
      then show "x ∈ (s - t) ∪ t"
        by (simp only: UnI2)
    next
      assume "x ∉ t"
      with ⟨x ∈ s⟩ have "x ∈ s - t"
        by (rule DiffI)
      then show "x ∈ (s - t) ∪ t"
        by (simp only: UnI1)
    qed
  next
    assume "x ∈ t"
    then show "x ∈ (s - t) ∪ t"
      by (simp only: UnI2)
  qed
qed
qed

(* 2ª demostración *)

lemma "(s - t) ∪ t = s ∪ t"
proof
  show "(s - t) ∪ t ⊆ s ∪ t"
  proof
    fix x
    assume "x ∈ (s - t) ∪ t"
    then show "x ∈ s ∪ t"

```

```

proof
  assume "x ∈ s - t"
  then have "x ∈ s"
    by simp
  then show "x ∈ s ∪ t"
    by simp
next
  assume "x ∈ t"
  then show "x ∈ s ∪ t"
    by simp
qed
qed
next
show "s ∪ t ⊆ (s - t) ∪ t"
proof
  fix x
  assume "x ∈ s ∪ t"
  then show "x ∈ (s - t) ∪ t"
  proof
    assume "x ∈ s"
    show "x ∈ (s - t) ∪ t"
    proof
      assume "x ∉ t"
      with <x ∈ s> show "x ∈ s - t"
      by simp
    qed
  next
    assume "x ∈ t"
    then show "x ∈ (s - t) ∪ t"
    by simp
  qed
qed
qed

(* 3ª demostración *)

lemma "(s - t) ∪ t = s ∪ t"
by (fact Un_Diff_cancel2)

(* 4ª demostración *)

lemma "(s - t) ∪ t = s ∪ t"
by auto

end

```

2.9.2. Demostraciones con Lean

```

-----
-- Demostrar que
--    $(s \setminus t) \cup t = s \cup t$ 
-----

import data.set.basic
open set

variable {α : Type}
variables s t : set α

-- 1ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=
begin
  ext x,
  split,
  { intro hx,
    cases hx with xst xt,
    { left,
      exact xst.1, },
    { right,
      exact xt }},
  { by_cases h : x ∈ t,
    { intro _,
      right,
      exact h },
    { intro hx,
      cases hx with xs xt,
      { left,
        split,
        { exact xs, },
        { dsimp,
          exact h, }},
      { right,
        exact xt, }}}},
end

-- 2ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=

```

```

begin
  ext x,
  split,
  { rintros ((xs, nxt) | xt),
    { left,
      exact xs},
    { right,
      exact xt }},
  { by_cases h : x ∈ t,
    { intro _,
      right,
      exact h },
    { rintros (xs | xt),
      { left,
        use [xs, h] },
      { right,
        use xt }}}},
end

-- 3ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=
begin
  rw ext_iff,
  intro,
  rw iff_def,
  finish,
end

-- 4ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=
by finish [ext_iff, iff_def]

-- 5ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=
diff_union_self

-- 6ª definición
-- =====

```

```

example : (s \ t) ∪ t = s ∪ t :=
begin
  ext,
  simp,
end

-- 7ª definición
-- =====

example : (s \ t) ∪ t = s ∪ t :=
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.10. Diferencia de unión e intersección

2.10.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   (s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)
-- -----

theory Diferencia_de_union_e_interseccion
imports Main
begin

(* 1ª demostración *)

lemma "(s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)"
proof (rule equalityI)
  show "(s - t) ∪ (t - s) ⊆ (s ∪ t) - (s ∩ t)"
  proof (rule subsetI)
    fix x
    assume "x ∈ (s - t) ∪ (t - s)"
    then show "x ∈ (s ∪ t) - (s ∩ t)"
    proof (rule UnE)
      assume "x ∈ s - t"
      then show "x ∈ (s ∪ t) - (s ∩ t)"
      proof (rule DiffE)
        assume "x ∈ s"
        assume "x ∉ t"

```

```

have "x ∈ s ∪ t"
  using ⟨x ∈ s⟩ by (simp only: UnI1)
moreover
have "x ∉ s ∩ t"
proof (rule notI)
  assume "x ∈ s ∩ t"
  then have "x ∈ t"
    by (simp only: IntD2)
  with ⟨x ∉ t⟩ show False
    by (rule notE)
qed
ultimately show "x ∈ (s ∪ t) - (s ∩ t)"
  by (rule DiffI)
qed
next
assume "x ∈ t - s"
then show "x ∈ (s ∪ t) - (s ∩ t)"
proof (rule DiffE)
  assume "x ∈ t"
  assume "x ∉ s"
  have "x ∈ s ∪ t"
    using ⟨x ∈ t⟩ by (simp only: UnI2)
  moreover
  have "x ∉ s ∩ t"
  proof (rule notI)
    assume "x ∈ s ∩ t"
    then have "x ∈ s"
      by (simp only: IntD1)
    with ⟨x ∉ s⟩ show False
      by (rule notE)
  qed
  ultimately show "x ∈ (s ∪ t) - (s ∩ t)"
    by (rule DiffI)
  qed
qed
qed
next
show "(s ∪ t) - (s ∩ t) ⊆ (s - t) ∪ (t - s)"
proof (rule subsetI)
  fix x
  assume "x ∈ (s ∪ t) - (s ∩ t)"
  then show "x ∈ (s - t) ∪ (t - s)"
  proof (rule DiffE)
    assume "x ∈ s ∪ t"
    assume "x ∉ s ∩ t"

```



```

note <x ∈ s ∪ t>
then show "x ∈ (s - t) ∪ (t - s)"
proof (rule UnE)
  assume "x ∈ s"
  have "x ∉ t"
  proof (rule notI)
    assume "x ∈ t"
    with <x ∈ s> have "x ∈ s ∩ t"
    by (rule IntI)
    with <x ∉ s ∩ t> show False
    by (rule notE)
  qed
  with <x ∈ s> have "x ∈ s - t"
  by (rule DiffI)
  then show "x ∈ (s - t) ∪ (t - s)"
  by (simp only: UnI1)
next
  assume "x ∈ t"
  have "x ∉ s"
  proof (rule notI)
    assume "x ∈ s"
    then have "x ∈ s ∩ t"
    using <x ∈ t> by (rule IntI)
    with <x ∉ s ∩ t> show False
    by (rule notE)
  qed
  with <x ∈ t> have "x ∈ t - s"
  by (rule DiffI)
  then show "x ∈ (s - t) ∪ (t - s)"
  by (rule UnI2)
qed
qed
qed
qed

(* 2ª demostración *)

lemma "(s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)"
proof
  show "(s - t) ∪ (t - s) ⊆ (s ∪ t) - (s ∩ t)"
  proof
    fix x
    assume "x ∈ (s - t) ∪ (t - s)"
    then show "x ∈ (s ∪ t) - (s ∩ t)"

```

```

proof
  assume "x ∈ s - t"
  then show "x ∈ (s ∪ t) - (s ∩ t)"
  proof
    assume "x ∈ s"
    assume "x ∉ t"
    have "x ∈ s ∪ t"
      using ⟨x ∈ s⟩ by simp
    moreover
    have "x ∉ s ∩ t"
    proof
      assume "x ∈ s ∩ t"
      then have "x ∈ t"
        by simp
      with ⟨x ∉ t⟩ show False
        by simp
    qed
    ultimately show "x ∈ (s ∪ t) - (s ∩ t)"
      by simp
  qed
next
assume "x ∈ t - s"
then show "x ∈ (s ∪ t) - (s ∩ t)"
proof
  assume "x ∈ t"
  assume "x ∉ s"
  have "x ∈ s ∪ t"
    using ⟨x ∈ t⟩ by simp
  moreover
  have "x ∉ s ∩ t"
  proof
    assume "x ∈ s ∩ t"
    then have "x ∈ s"
      by simp
    with ⟨x ∉ s⟩ show False
      by simp
  qed
  ultimately show "x ∈ (s ∪ t) - (s ∩ t)"
    by simp
qed
qed
qed
next
show "(s ∪ t) - (s ∩ t) ⊆ (s - t) ∪ (t - s)"
proof

```

```

fix x
assume "x ∈ (s ∪ t) - (s ∩ t)"
then show "x ∈ (s - t) ∪ (t - s)"
proof
  assume "x ∈ s ∪ t"
  assume "x ∉ s ∩ t"
  note ⟨x ∈ s ∪ t⟩
  then show "x ∈ (s - t) ∪ (t - s)"
  proof
    assume "x ∈ s"
    have "x ∉ t"
    proof
      assume "x ∈ t"
      with ⟨x ∈ s⟩ have "x ∈ s ∩ t"
      by simp
      with ⟨x ∉ s ∩ t⟩ show False
      by simp
    qed
    with ⟨x ∈ s⟩ have "x ∈ s - t"
    by simp
    then show "x ∈ (s - t) ∪ (t - s)"
    by simp
  next
    assume "x ∈ t"
    have "x ∉ s"
    proof
      assume "x ∈ s"
      then have "x ∈ s ∩ t"
      using ⟨x ∈ t⟩ by simp
      with ⟨x ∉ s ∩ t⟩ show False
      by simp
    qed
    with ⟨x ∈ t⟩ have "x ∈ t - s"
    by simp
    then show "x ∈ (s - t) ∪ (t - s)"
    by simp
  qed
qed
qed
qed

(* 3ª demostración *)

lemma "(s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)"
proof

```

```

show "(s - t) u (t - s) ⊆ (s u t) - (s n t)"
proof
  fix x
  assume "x ∈ (s - t) u (t - s)"
  then show "x ∈ (s u t) - (s n t)"
  proof
    assume "x ∈ s - t"
    then show "x ∈ (s u t) - (s n t)" by simp
  next
    assume "x ∈ t - s"
    then show "x ∈ (s u t) - (s n t)" by simp
  qed
qed
next
show "(s u t) - (s n t) ⊆ (s - t) u (t - s)"
proof
  fix x
  assume "x ∈ (s u t) - (s n t)"
  then show "x ∈ (s - t) u (t - s)"
  proof
    assume "x ∈ s u t"
    assume "x ∉ s n t"
    note ⟨x ∈ s u t⟩
    then show "x ∈ (s - t) u (t - s)"
    proof
      assume "x ∈ s"
      then show "x ∈ (s - t) u (t - s)"
        using ⟨x ∉ s n t⟩ by simp
    next
      assume "x ∈ t"
      then show "x ∈ (s - t) u (t - s)"
        using ⟨x ∉ s n t⟩ by simp
    qed
  qed
qed
qed

(* 4a demostración *)

lemma "(s - t) u (t - s) = (s u t) - (s n t)"
proof
  show "(s - t) u (t - s) ⊆ (s u t) - (s n t)"
  proof
    fix x
    assume "x ∈ (s - t) u (t - s)"

```

```

    then show "x ∈ (s ∪ t) - (s ∩ t)" by auto
  qed
next
  show "(s ∪ t) - (s ∩ t) ⊆ (s - t) ∪ (t - s)"
  proof
    fix x
    assume "x ∈ (s ∪ t) - (s ∩ t)"
    then show "x ∈ (s - t) ∪ (t - s)" by auto
  qed
qed

(* 5ª demostración *)

lemma "(s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)"
proof
  show "(s - t) ∪ (t - s) ⊆ (s ∪ t) - (s ∩ t)" by auto
next
  show "(s ∪ t) - (s ∩ t) ⊆ (s - t) ∪ (t - s)" by auto
qed

(* 6ª demostración *)

lemma "(s - t) ∪ (t - s) = (s ∪ t) - (s ∩ t)"
  by auto

end

```

2.10.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t)
-- -----

import data.set.basic
open set

variable {α : Type}
variables s t : set α

-- 1ª demostración
-- =====

```

```
example : (s ⊔ t) ⊔ (t ⊔ s) = (s ⊔ t) ⊔ (s ⊔ t) :=
```

```
begin
```

```
  ext x,
  split,
  { rintros (⟨xs, xnt⟩ | ⟨xt, xns⟩),
    { split,
      { left,
        exact xs },
      { rintros (_, xt),
        contradiction }},
    { split ,
      { right,
        exact xt },
      { rintros (xs, _),
        contradiction }},
    { rintros (xs | xt, nxst),
      { left,
        use xs,
        intro xt,
        apply nxst,
        split; assumption },
      { right,
        use xt,
        intro xs,
        apply nxst,
        split; assumption }},
```

```
end
```

```
-- 2ª demostración
```

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

```
example : (s ⊔ t) ⊔ (t ⊔ s) = (s ⊔ t) ⊔ (s ⊔ t) :=
```

```
begin
```

```
  ext x,
  split,
  { rintros (⟨xs, xnt⟩ | ⟨xt, xns⟩),
    { finish, },
    { finish, }},
  { rintros (xs | xt, nxst),
    { finish, },
    { finish, }},
```

```
end
```

```
-- 3ª demostración
```

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

```

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
begin
  ext x,
  split,
  { rintros (⟨xs, xnt⟩ | ⟨xt, xns⟩) ; finish, },
  { rintros ⟨xs | xt, nxst⟩ ; finish, },
end

-- 4ª demostración
-- =====

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
begin
  ext,
  split,
  { finish, },
  { finish, },
end

-- 5ª demostración
-- =====

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
begin
  rw ext_iff,
  intro,
  rw iff_def,
  finish,
end

-- 6ª demostración
-- =====

example : (s \ t) ∪ (t \ s) = (s ∪ t) \ (s ∩ t) :=
by finish [ext_iff, iff_def]

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

2.11. Unión de los conjuntos de los pares e impares

2.11.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Los conjuntos de los números naturales, de los pares y de los impares
-- se definen por
--   def naturales : set ℕ := {n | true}
--   def pares     : set ℕ := {n | even n}
--   def impares   : set ℕ := {n | ¬ even n}
--
-- Demostrar que
--   pares ∪ impares = naturales
----- *)

theory Union_de_pares_e_impares
imports Main
begin

definition naturales :: "nat set" where
  "naturales = {n ∈ ℕ . True}"

definition pares :: "nat set" where
  "pares = {n ∈ ℕ . even n}"

definition impares :: "nat set" where
  "impares = {n ∈ ℕ . ¬ even n}"

(* 1ª demostración *)

lemma "pares ∪ impares = naturales"
proof -
  have "∀ n ∈ ℕ . even n ∨ ¬ even n ↔ True"
  by simp
  then have "{n ∈ ℕ. even n} ∪ {n ∈ ℕ. ¬ even n} = {n ∈ ℕ. True}"
  by auto
  then show "pares ∪ impares = naturales"
  by (simp add: naturales_def pares_def impares_def)
qed

(* 2ª demostración *)
```



```

lemma "pares u impares = naturales"
  unfolding naturales_def pares_def impares_def
  by auto

end

```

2.11.2. Demostraciones con Lean

```

-----
-- Los conjuntos de los números naturales, de los pares y de los impares
-- se definen por
--   def naturales : set ℕ := {n | true}
--   def pares     : set ℕ := {n | even n}
--   def impares   : set ℕ := {n | ¬ even n}
--
-- Demostrar que
--   pares u impares = naturales
-----

import data.nat.parity
import data.set.basic
import tactic

open set

def naturales : set ℕ := {n | true}
def pares     : set ℕ := {n | even n}
def impares   : set ℕ := {n | ¬ even n}

-- 1ª demostración
-- =====

example : pares u impares = naturales :=
begin
  unfold pares impares naturales,
  ext n,
  simp,
  apply classical.em,
end

-- 2ª demostración
-- =====

```

```
example : pares  $\sqcup$  impares = naturales :=
begin
  unfold pares impares naturales,
  ext n,
  finish,
end

-- 3ª demostración
-- =====

example : pares  $\sqcup$  impares = naturales :=
by finish [pares, impares, naturales, ext_iff]
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

Capítulo 3

Ejercicios de junio de 2021

3.1. Intersección de los primos y los mayores que dos

3.1.1. Demostraciones con Isabelle/HOL

```
(* -----  
-- Los números primos, los mayores que 2 y los impares se definen por  
--   def primos      : set ℕ := {n | prime n}  
--   def mayoresQue2 : set ℕ := {n | n > 2}  
--   def impares     : set ℕ := {n | ¬ even n}  
--  
-- Demostrar que  
--   primos ∩ mayoresQue2 ⊆ impares  
-- ----- *)  
  
theory Interseccion_de_los_primos_y_los_mayores_que_dos  
imports Main "HOL-Number_Theory.Number_Theory"  
begin  
  
definition primos :: "nat set" where  
  "primos = {n ∈ ℕ . prime n}"  
  
definition mayoresQue2 :: "nat set" where  
  "mayoresQue2 = {n ∈ ℕ . n > 2}"  
  
definition impares :: "nat set" where  
  "impares = {n ∈ ℕ . ¬ even n}"  
  
(* 1ª demostración *)
```

```

lemma "primos n mayoresQue2  $\subseteq$  impares"
proof
  fix x
  assume "x  $\in$  primos n mayoresQue2"
  then have "x  $\in \mathbb{N} \wedge \text{prime } x \wedge 2 < x$ "
    by (simp add: primos_def mayoresQue2_def)
  then have "x  $\in \mathbb{N} \wedge \text{odd } x$ "
    by (simp add: prime_odd_nat)
  then show "x  $\in$  impares"
    by (simp add: impares_def)
qed

(* 2ª demostración *)

lemma "primos n mayoresQue2  $\subseteq$  impares"
  unfolding primos_def mayoresQue2_def impares_def
  by (simp add: Collect_mono_iff Int_def prime_odd_nat)

(* 3ª demostración *)

lemma "primos n mayoresQue2  $\subseteq$  impares"
  unfolding primos_def mayoresQue2_def impares_def
  by (auto simp add: prime_odd_nat)

end

```

3.1.2. Demostraciones con Lean

```

-----
-- Los números primos, los mayores que 2 y los impares se definen por
--   def primos      : set  $\mathbb{N}$  := {n | prime n}
--   def mayoresQue2 : set  $\mathbb{N}$  := {n | n > 2}
--   def impares     : set  $\mathbb{N}$  := {n |  $\neg$  even n}
--
-- Demostrar que
--   primos n mayoresQue2  $\subseteq$  impares
-----

import data.nat.parity
import data.nat.prime
import tactic

open nat

```

```

def primos      : set ℕ := {n | prime n}
def mayoresQue2 : set ℕ := {n | n > 2}
def impares     : set ℕ := {n | ¬ even n}

example : primos ∩ mayoresQue2 ⊆ impares :=
begin
  unfold primos mayoresQue2 impares,
  intro n,
  simp,
  intro hn,
  cases prime.eq_two_or_odd hn with h h,
  { rw h,
    intro,
    linarith, },
  { rw even_iff,
    rw h,
    norm_num },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.2. Distributiva de la intersección respecto de la unión general

3.2.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   s n (⋃ i, A i) = ⋃ i, (A i n s)
-- ----- *)

theory Distributiva_de_la_interseccion_respecto_de_la_union_general
imports Main
begin

(* 1ª demostración *)

lemma "s n (⋃ i ∈ I. A i) = (⋃ i ∈ I. (A i n s))"
proof (rule equalityI)
  show "s n (⋃ i ∈ I. A i) ⊆ (⋃ i ∈ I. (A i n s))"
  proof (rule subsetI)

```

```

fix x
  assume "x ∈ s ∧ (⋃ i ∈ I. A i)"
  then have "x ∈ s"
    by (simp only: IntD1)
  have "x ∈ (⋃ i ∈ I. A i)"
    using <x ∈ s ∧ (⋃ i ∈ I. A i)> by (simp only: IntD2)
  then show "x ∈ (⋃ i ∈ I. (A i ∩ s))"
  proof (rule UN_E)
    fix i
    assume "i ∈ I"
    assume "x ∈ A i"
    then have "x ∈ A i ∩ s"
      using <x ∈ s> by (rule IntI)
    with <i ∈ I> show "x ∈ (⋃ i ∈ I. (A i ∩ s))"
      by (rule UN_I)
  qed
qed
next
show "(⋃ i ∈ I. (A i ∩ s)) ⊆ s ∧ (⋃ i ∈ I. A i)"
proof (rule subsetI)
  fix x
  assume "x ∈ (⋃ i ∈ I. A i ∩ s)"
  then show "x ∈ s ∧ (⋃ i ∈ I. A i)"
  proof (rule UN_E)
    fix i
    assume "i ∈ I"
    assume "x ∈ A i ∩ s"
    then have "x ∈ A i"
      by (rule IntD1)
    have "x ∈ s"
      using <x ∈ A i ∩ s> by (rule IntD2)
    moreover
    have "x ∈ (⋃ i ∈ I. A i)"
      using <i ∈ I> <x ∈ A i> by (rule UN_I)
    ultimately show "x ∈ s ∧ (⋃ i ∈ I. A i)"
      by (rule IntI)
  qed
qed
qed

(* 2ª demostración *)

lemma "s ∧ (⋃ i ∈ I. A i) = (⋃ i ∈ I. (A i ∩ s))"
proof
  show "s ∧ (⋃ i ∈ I. A i) ⊆ (⋃ i ∈ I. (A i ∩ s))"

```

```

proof
  fix x
  assume "x ∈ s n (⋃ i ∈ I. A i)"
  then have "x ∈ s"
    by simp
  have "x ∈ (⋃ i ∈ I. A i)"
    using <x ∈ s n (⋃ i ∈ I. A i)> by simp
  then show "x ∈ (⋃ i ∈ I. (A i n s))"
  proof
    fix i
    assume "i ∈ I"
    assume "x ∈ A i"
    then have "x ∈ A i n s"
      using <x ∈ s> by simp
    with <i ∈ I> show "x ∈ (⋃ i ∈ I. (A i n s))"
      by (rule UN_I)
  qed
qed
next
show "(⋃ i ∈ I. (A i n s)) ⊆ s n (⋃ i ∈ I. A i)"
proof
  fix x
  assume "x ∈ (⋃ i ∈ I. A i n s)"
  then show "x ∈ s n (⋃ i ∈ I. A i)"
  proof
    fix i
    assume "i ∈ I"
    assume "x ∈ A i n s"
    then have "x ∈ A i"
      by simp
    have "x ∈ s"
      using <x ∈ A i n s> by simp
    moreover
      have "x ∈ (⋃ i ∈ I. A i)"
        using <i ∈ I> <x ∈ A i> by (rule UN_I)
    ultimately show "x ∈ s n (⋃ i ∈ I. A i)"
      by simp
  qed
qed
qed

(* 3ª demostración *)

lemma "s n (⋃ i ∈ I. A i) = (⋃ i ∈ I. (A i n s))"
  by auto

```

end

3.2.2. Demostraciones con Lean

```

-- Demostrar que
--   s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s)
-- -----

import data.set.basic
import data.set.lattice
import tactic

open set

variable {α : Type}
variable s : set α
variable A : ℕ → set α

-- 1ª demostración
-- =====

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
begin
  ext x,
  split,
  { intro h,
    rw mem_Union,
    cases h with xs xUAi,
    rw mem_Union at xUAi,
    cases xUAi with i xAi,
    use i,
    split,
    { exact xAi, },
    { exact xs, }},
  { intro h,
    rw mem_Union at h,
    cases h with i hi,
    cases hi with xAi xs,
    split,
    { exact xs, },
    { rw mem_Union,
      use i,

```



```

    exact xAi, }},
end

-- 2ª demostración
-- =====

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
begin
  ext x,
  simp only [mem_inter_eq, mem_Union],
  split,
  { rintro ⟨xs, ⟨i, xAi⟩⟩,
    exact ⟨i, xAi, xs⟩, },
  { rintro ⟨i, xAi, xs⟩,
    exact ⟨xs, ⟨i, xAi⟩⟩ },
end

-- 3ª demostración
-- =====

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
begin
  ext x,
  finish [mem_inter_eq, mem_Union],
end

-- 4ª demostración
-- =====

example : s ∩ (⋃ i, A i) = ⋃ i, (A i ∩ s) :=
by finish [mem_inter_eq, mem_Union, ext_iff]

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.3. Intersección de intersecciones

3.3.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   (⋂ i, A i ∩ B i) = (⋂ i, A i) ∩ (⋂ i, B i)
-- ----- *)

```

```

theory Interseccion_de_intersecciones
imports Main
begin

(* la demostración *)

lemma "( $\bigcap i \in I. A\ i \cap B\ i$ ) = ( $\bigcap i \in I. A\ i$ )  $\cap$  ( $\bigcap i \in I. B\ i$ )"
proof (rule equalityI)
  show "( $\bigcap i \in I. A\ i \cap B\ i$ )  $\subseteq$  ( $\bigcap i \in I. A\ i$ )  $\cap$  ( $\bigcap i \in I. B\ i$ )"
  proof (rule subsetI)
    fix x
    assume h1 : " $x \in (\bigcap i \in I. A\ i \cap B\ i)$ "
    have " $x \in (\bigcap i \in I. A\ i)$ "
    proof (rule INT_I)
      fix i
      assume "i  $\in I$ "
      with h1 have " $x \in A\ i \cap B\ i$ "
      by (rule INT_D)
      then show " $x \in A\ i$ "
      by (rule IntD1)
    qed
    moreover
    have " $x \in (\bigcap i \in I. B\ i)$ "
    proof (rule INT_I)
      fix i
      assume "i  $\in I$ "
      with h1 have " $x \in A\ i \cap B\ i$ "
      by (rule INT_D)
      then show " $x \in B\ i$ "
      by (rule IntD2)
    qed
    ultimately show " $x \in (\bigcap i \in I. A\ i) \cap (\bigcap i \in I. B\ i)$ "
    by (rule IntI)
  qed
next
show "( $\bigcap i \in I. A\ i$ )  $\cap$  ( $\bigcap i \in I. B\ i$ )  $\subseteq$  ( $\bigcap i \in I. A\ i \cap B\ i$ )"
proof (rule subsetI)
  fix x
  assume h2 : " $x \in (\bigcap i \in I. A\ i) \cap (\bigcap i \in I. B\ i)$ "
  show " $x \in (\bigcap i \in I. A\ i \cap B\ i)$ "
  proof (rule INT_I)
    fix i
    assume "i  $\in I$ "
    have " $x \in A\ i$ "

```

```

proof -
  have "x ∈ (⋂ i ∈ I. A i)"
    using h2 by (rule IntD1)
  then show "x ∈ A i"
    using <i ∈ I> by (rule INT_D)
qed
moreover
have "x ∈ B i"
proof -
  have "x ∈ (⋂ i ∈ I. B i)"
    using h2 by (rule IntD2)
  then show "x ∈ B i"
    using <i ∈ I> by (rule INT_D)
qed
ultimately show "x ∈ A i n B i"
  by (rule IntI)
qed
qed
qed

(* 2ª demostración *)

lemma "(⋂ i ∈ I. A i n B i) = (⋂ i ∈ I. A i) n (⋂ i ∈ I. B i)"
proof
  show "(⋂ i ∈ I. A i n B i) ⊆ (⋂ i ∈ I. A i) n (⋂ i ∈ I. B i)"
  proof
    fix x
    assume h1 : "x ∈ (⋂ i ∈ I. A i n B i)"
    have "x ∈ (⋂ i ∈ I. A i)"
    proof
      fix i
      assume "i ∈ I"
      then show "x ∈ A i"
        using h1 by simp
    qed
    moreover
    have "x ∈ (⋂ i ∈ I. B i)"
    proof
      fix i
      assume "i ∈ I"
      then show "x ∈ B i"
        using h1 by simp
    qed
    ultimately show "x ∈ (⋂ i ∈ I. A i) n (⋂ i ∈ I. B i)"
      by simp
  qed
end

```

```

qed
next
show "(\ i \in I. A i) \cap (\ i \in I. B i) \subseteq (\ i \in I. A i \cap B i)"
proof
  fix x
  assume h2 : "x \in (\ i \in I. A i) \cap (\ i \in I. B i)"
  show "x \in (\ i \in I. A i \cap B i)"
  proof
    fix i
    assume "i \in I"
    then have "x \in A i"
      using h2 by simp
    moreover
    have "x \in B i"
      using <i \in I> h2 by simp
    ultimately show "x \in A i \cap B i"
      by simp
  qed
qed
qed

(* 3ª demostración *)

lemma "(\ i \in I. A i \cap B i) = (\ i \in I. A i) \cap (\ i \in I. B i)"
  by auto

end

```

3.3.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   (\ i, A i \cap B i) = (\ i, A i) \cap (\ i, B i)
-- -----

import data.set.basic
import tactic

open set

variable {α : Type}
variables A B : ℕ → set α

```

```

-- 1ª demostración
-- =====

example : (⋂ i, A i ⋂ B i) = (⋂ i, A i) ⋂ (⋂ i, B i) :=
begin
  ext x,
  simp only [mem_inter_eq, mem_Inter],
  split,
  { intro h,
    split,
    { intro i,
      exact (h i).1 },
    { intro i,
      exact (h i).2 }},
  { intros h i,
    cases h with h1 h2,
    split,
    { exact h1 i },
    { exact h2 i }},
end

-- 2ª demostración
-- =====

example : (⋂ i, A i ⋂ B i) = (⋂ i, A i) ⋂ (⋂ i, B i) :=
begin
  ext x,
  simp only [mem_inter_eq, mem_Inter],
  exact (λ h, (λ i, (h i).1, λ i, (h i).2),
        λ (h1, h2) i, (h1 i, h2 i)),
end

-- 3ª demostración
-- =====

example : (⋂ i, A i ⋂ B i) = (⋂ i, A i) ⋂ (⋂ i, B i) :=
begin
  ext,
  simp only [mem_inter_eq, mem_Inter],
  finish,
end

-- 4ª demostración
-- =====

```

```

example : (⋂ i, A i ∧ B i) = (⋂ i, A i) ∧ (⋂ i, B i) :=
begin
  ext,
  finish [mem_inter_eq, mem_Inter],
end

-- 5ª demostración
-- =====

example : (⋂ i, A i ∧ B i) = (⋂ i, A i) ∧ (⋂ i, B i) :=
by finish [mem_inter_eq, mem_Inter, ext_iff]

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.4. Unión con intersección general

3.4.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   s ∪ (⋂ i. A i) = (⋂ i. A i ∪ s)
-- ----- *)

theory Union_con_interseccion_general
imports Main
begin

(* 1ª demostración *)

lemma "s ∪ (⋂ i ∈ I. A i) = (⋂ i ∈ I. A i ∪ s)"
proof (rule equalityI)
  show "s ∪ (⋂ i ∈ I. A i) ⊆ (⋂ i ∈ I. A i ∪ s)"
  proof (rule subsetI)
    fix x
    assume "x ∈ s ∪ (⋂ i ∈ I. A i)"
    then show "x ∈ (⋂ i ∈ I. A i ∪ s)"
    proof (rule UnE)
      assume "x ∈ s"
      show "x ∈ (⋂ i ∈ I. A i ∪ s)"
      proof (rule INT_I)
        fix i
        assume "i ∈ I"

```

```

    show "x ∈ A i u s"
    using <x ∈ s> by (rule UnI2)
  qed
next
  assume h1 : "x ∈ (⋂ i ∈ I. A i)"
  show "x ∈ (⋂ i ∈ I. A i u s)"
  proof (rule INT_I)
    fix i
    assume "i ∈ I"
    with h1 have "x ∈ A i"
      by (rule INT_D)
    then show "x ∈ A i u s"
      by (rule UnI1)
  qed
qed
qed
next
  show "(⋂ i ∈ I. A i u s) ⊆ s u (⋂ i ∈ I. A i)"
  proof (rule subsetI)
    fix x
    assume h2 : "x ∈ (⋂ i ∈ I. A i u s)"
    show "x ∈ s u (⋂ i ∈ I. A i)"
    proof (cases "x ∈ s")
      assume "x ∈ s"
      then show "x ∈ s u (⋂ i ∈ I. A i)"
        by (rule UnI1)
    next
      assume "x ∉ s"
      have "x ∈ (⋂ i ∈ I. A i)"
      proof (rule INT_I)
        fix i
        assume "i ∈ I"
        with h2 have "x ∈ A i u s"
          by (rule INT_D)
        then show "x ∈ A i"
          proof (rule UnE)
            assume "x ∈ A i"
            then show "x ∈ A i"
              by this
          next
            assume "x ∈ s"
            with <x ∉ s> show "x ∈ A i"
              by (rule notE)
          qed
        qed
      qed
    next
      assume "x ∈ s"
      with <x ∉ s> show "x ∈ A i"
        by (rule notE)
    qed
  qed

```

```

    then show "x ∈ s ∪ (⋂ i ∈ I. A i)"
    by (rule UnI2)
  qed
qed
qed

(* 2ª demostración *)

lemma "s ∪ (⋂ i ∈ I. A i) = (⋂ i ∈ I. A i ∪ s)"
proof
  show "s ∪ (⋂ i ∈ I. A i) ⊆ (⋂ i ∈ I. A i ∪ s)"
  proof
    fix x
    assume "x ∈ s ∪ (⋂ i ∈ I. A i)"
    then show "x ∈ (⋂ i ∈ I. A i ∪ s)"
    proof
      assume "x ∈ s"
      show "x ∈ (⋂ i ∈ I. A i ∪ s)"
      proof
        fix i
        assume "i ∈ I"
        show "x ∈ A i ∪ s"
        using <x ∈ s> by simp
      qed
    qed
  next
    assume h1 : "x ∈ (⋂ i ∈ I. A i)"
    show "x ∈ (⋂ i ∈ I. A i ∪ s)"
    proof
      fix i
      assume "i ∈ I"
      with h1 have "x ∈ A i"
      by simp
      then show "x ∈ A i ∪ s"
      by simp
    qed
  qed
qed
next
show "(⋂ i ∈ I. A i ∪ s) ⊆ s ∪ (⋂ i ∈ I. A i)"
proof
  fix x
  assume h2 : "x ∈ (⋂ i ∈ I. A i ∪ s)"
  show "x ∈ s ∪ (⋂ i ∈ I. A i)"
  proof (cases "x ∈ s")
    assume "x ∈ s"

```



```

    then show "x ∈ s ∪ (⋂ i ∈ I. A i)"
      by simp
  next
    assume "x ∉ s"
    have "x ∈ (⋂ i ∈ I. A i)"
    proof
      fix i
      assume "i ∈ I"
      with h2 have "x ∈ A i ∪ s"
        by (rule INT_D)
      then show "x ∈ A i"
      proof
        assume "x ∈ A i"
        then show "x ∈ A i"
          by this
      next
        assume "x ∈ s"
        with <x ∉ s> show "x ∈ A i"
          by simp
      qed
    qed
    then show "x ∈ s ∪ (⋂ i ∈ I. A i)"
      by simp
  qed
qed
qed
(* 3ª demostración *)

lemma "s ∪ (⋂ i ∈ I. A i) = (⋂ i ∈ I. A i ∪ s)"
proof
  show "s ∪ (⋂ i ∈ I. A i) ⊆ (⋂ i ∈ I. A i ∪ s)"
  proof
    fix x
    assume "x ∈ s ∪ (⋂ i ∈ I. A i)"
    then show "x ∈ (⋂ i ∈ I. A i ∪ s)"
    proof
      assume "x ∈ s"
      then show "x ∈ (⋂ i ∈ I. A i ∪ s)"
        by simp
    next
      assume "x ∈ (⋂ i ∈ I. A i)"
      then show "x ∈ (⋂ i ∈ I. A i ∪ s)"
        by simp
    qed
  qed
qed

```

```

qed
next
show " $(\bigcap i \in I. A\ i \cup s) \subseteq s \cup (\bigcap i \in I. A\ i)$ "
proof
  fix x
  assume h2 : " $x \in (\bigcap i \in I. A\ i \cup s)$ "
  show " $x \in s \cup (\bigcap i \in I. A\ i)$ "
  proof (cases " $x \in s$ ")
    assume " $x \in s$ "
    then show " $x \in s \cup (\bigcap i \in I. A\ i)$ "
    by simp
  next
    assume " $x \notin s$ "
    then show " $x \in s \cup (\bigcap i \in I. A\ i)$ "
    using h2 by simp
  qed
qed
qed

(* 4ª demostración *)

lemma " $s \cup (\bigcap i \in I. A\ i) = (\bigcap i \in I. A\ i \cup s)$ "
proof
  show " $s \cup (\bigcap i \in I. A\ i) \subseteq (\bigcap i \in I. A\ i \cup s)$ "
  proof
    fix x
    assume " $x \in s \cup (\bigcap i \in I. A\ i)$ "
    then show " $x \in (\bigcap i \in I. A\ i \cup s)$ "
    proof
      assume " $x \in s$ "
      then show ?thesis by simp
    next
      assume " $x \in (\bigcap i \in I. A\ i)$ "
      then show ?thesis by simp
    qed
  qed
next
show " $(\bigcap i \in I. A\ i \cup s) \subseteq s \cup (\bigcap i \in I. A\ i)$ "
proof
  fix x
  assume h2 : " $x \in (\bigcap i \in I. A\ i \cup s)$ "
  show " $x \in s \cup (\bigcap i \in I. A\ i)$ "
  proof (cases " $x \in s$ ")
    case True
    then show ?thesis by simp

```

```

next
  case False
  then show ?thesis using h2 by simp
qed
qed
qed

(* 5ª demostración *)

lemma "s ∪ (⋂ i ∈ I. A i) = (⋂ i ∈ I. A i ∪ s)"
  by auto

end

```

3.4.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s)
-- -----

import data.set.basic
import tactic

open set

variable {α : Type}
variable s : set α
variables A : ℕ → set α

-- 1ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
begin
  ext x,
  simp only [mem_union, mem_Inter],
  split,
  { intros h i,
    cases h with xs xAi,
    { right,
      exact xs },

```

```

    { left,
      exact xAi i, }},
  { intro h,
    by_cases xs : x ∈ s,
    { left,
      exact xs },
    { right,
      intro i,
      cases h i with xAi xs,
      { exact xAi, },
      { contradiction, }},
  end

-- 2ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
begin
  ext x,
  simp only [mem_union, mem_Inter],
  split,
  { rintros (xs | xI) i,
    { right,
      exact xs },
    { left,
      exact xI i }},
  { intro h,
    by_cases xs : x ∈ s,
    { left,
      exact xs },
    { right,
      intro i,
      cases h i,
      { assumption },
      { contradiction }},
  end

-- 3ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
begin
  ext x,
  simp only [mem_union, mem_Inter],
  split,

```

```

    { finish, },
    { finish, },
end

-- 4ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
begin
  ext,
  simp only [mem_union, mem_Inter],
  split ; finish,
end

-- 5ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
begin
  ext,
  simp only [mem_union, mem_Inter],
  finish [iff_def],
end

-- 6ª demostración
-- =====

example : s ∪ (⋂ i, A i) = ⋂ i, (A i ∪ s) :=
by finish [ext_iff, mem_union, mem_Inter, iff_def]

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.5. Imagen inversa de la intersección

3.5.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, la imagen inversa de un conjunto s (de elementos de
-- tipo β) por la función f (de tipo α → β) es el conjunto 'f -' s' de
-- elementos x (de tipo α) tales que 'f x ∈ s'.
--
-- Demostrar que

```

```

--       $f^{-1'}(u \cap v) = f^{-1'} u \cap f^{-1'} v$ 
--      ----- *)

theory Imagen_inversa_de_la_interseccion
imports Main
begin

(* 1ª demostración *)

lemma "f -' (u ∩ v) = f -' u ∩ f -' v"
proof (rule equalityI)
  show "f -' (u ∩ v) ⊆ f -' u ∩ f -' v"
  proof (rule subsetI)
    fix x
    assume "x ∈ f -' (u ∩ v)"
    then have h : "f x ∈ u ∩ v"
      by (simp only: vimage_eq)
    have "x ∈ f -' u"
    proof -
      have "f x ∈ u"
        using h by (rule IntD1)
      then show "x ∈ f -' u"
        by (rule vimageI2)
    qed
    moreover
    have "x ∈ f -' v"
    proof -
      have "f x ∈ v"
        using h by (rule IntD2)
      then show "x ∈ f -' v"
        by (rule vimageI2)
    qed
    ultimately show "x ∈ f -' u ∩ f -' v"
      by (rule IntI)
  qed
next
  show "f -' u ∩ f -' v ⊆ f -' (u ∩ v)"
  proof (rule subsetI)
    fix x
    assume h2 : "x ∈ f -' u ∩ f -' v"
    have "f x ∈ u"
    proof -
      have "x ∈ f -' u"
        using h2 by (rule IntD1)
      then show "f x ∈ u"

```

```

      by (rule vimageD)
    qed
  moreover
  have "f x ∈ v"
  proof -
    have "x ∈ f -' v"
      using h2 by (rule IntD2)
    then show "f x ∈ v"
      by (rule vimageD)
  qed
  ultimately have "f x ∈ u ∩ v"
    by (rule IntI)
  then show "x ∈ f -' (u ∩ v)"
    by (rule vimageI2)
  qed
qed

(* 2ª demostración *)

lemma "f -' (u ∩ v) = f -' u ∩ f -' v"
proof
  show "f -' (u ∩ v) ⊆ f -' u ∩ f -' v"
  proof
    fix x
    assume "x ∈ f -' (u ∩ v)"
    then have h : "f x ∈ u ∩ v"
      by simp
    have "x ∈ f -' u"
    proof -
      have "f x ∈ u"
        using h by simp
      then show "x ∈ f -' u"
        by simp
    qed
    moreover
    have "x ∈ f -' v"
    proof -
      have "f x ∈ v"
        using h by simp
      then show "x ∈ f -' v"
        by simp
    qed
    ultimately show "x ∈ f -' u ∩ f -' v"
      by simp
  qed
qed

```

```

next
  show "f -' u n f -' v ⊆ f -' (u n v)"
  proof
    fix x
    assume h2 : "x ∈ f -' u n f -' v"
    have "f x ∈ u"
    proof -
      have "x ∈ f -' u"
      using h2 by simp
      then show "f x ∈ u"
      by simp
    qed
    moreover
    have "f x ∈ v"
    proof -
      have "x ∈ f -' v"
      using h2 by simp
      then show "f x ∈ v"
      by simp
    qed
    ultimately have "f x ∈ u n v"
    by simp
    then show "x ∈ f -' (u n v)"
    by simp
  qed
qed

(* 3ª demostración *)

lemma "f -' (u n v) = f -' u n f -' v"
proof
  show "f -' (u n v) ⊆ f -' u n f -' v"
  proof
    fix x
    assume h1 : "x ∈ f -' (u n v)"
    have "x ∈ f -' u" using h1 by simp
    moreover
    have "x ∈ f -' v" using h1 by simp
    ultimately show "x ∈ f -' u n f -' v" by simp
  qed
qed
next
  show "f -' u n f -' v ⊆ f -' (u n v)"
  proof
    fix x
    assume h2 : "x ∈ f -' u n f -' v"

```



```

    have "f x ∈ u" using h2 by simp
  moreover
    have "f x ∈ v" using h2 by simp
  ultimately have "f x ∈ u ∩ v" by simp
  then show "x ∈ f ⁻¹ (u ∩ v)" by simp
qed
qed

(* 4ª demostración *)

lemma "f ⁻¹ (u ∩ v) = f ⁻¹ u ∩ f ⁻¹ v"
  by (simp only: vimage_Int)

(* 5ª demostración *)

lemma "f ⁻¹ (u ∩ v) = f ⁻¹ u ∩ f ⁻¹ v"
  by auto

end

```

3.5.2. Demostraciones con Lean

```

-----
-- En Lean, la imagen inversa de un conjunto s (de elementos de tipo β)
-- por la función f (de tipo α → β) es el conjunto 'f ⁻¹ s' de
-- elementos x (de tipo α) tales que 'f x ∈ s'.
--
-- Demostrar que
--   f ⁻¹ (u ∩ v) = f ⁻¹ u ∩ f ⁻¹ v
-----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables u v : set β

-- 1ª demostración
-- =====

example : f ⁻¹ (u ∩ v) = f ⁻¹ u ∩ f ⁻¹ v :=

```

```

begin
  ext x,
  split,
  { intro h,
    split,
    { apply mem_preimage.mpr,
      rw mem_preimage at h,
      exact mem_of_mem_inter_left h, },
    { apply mem_preimage.mpr,
      rw mem_preimage at h,
      exact mem_of_mem_inter_right h, }},
  { intro h,
    apply mem_preimage.mpr,
    split,
    { apply mem_preimage.mp,
      exact mem_of_mem_inter_left h, },
    { apply mem_preimage.mp,
      exact mem_of_mem_inter_right h, }},
end

-- 2ª demostración
-- =====

example : f -1 (u ∩ v) = f -1 u ∩ f -1 v :=
begin
  ext x,
  exact (λ h, ⟨mem_preimage.mpr (mem_of_mem_inter_left h),
    mem_preimage.mpr (mem_of_mem_inter_right h)⟩,
    λ h, ⟨mem_preimage.mp (mem_of_mem_inter_left h),
    mem_preimage.mp (mem_of_mem_inter_right h)⟩),
end

-- 3ª demostración
-- =====

example : f -1 (u ∩ v) = f -1 u ∩ f -1 v :=
begin
  ext,
  refl,
end

-- 4ª demostración
-- =====

example : f -1 (u ∩ v) = f -1 u ∩ f -1 v :=

```

```

by {ext, refl}

-- 5ª demostración
-- =====

example : f -1 (u ∩ v) = f -1 u ∩ f -1 v :=
rfl

-- 6ª demostración
-- =====

example : f -1 (u ∩ v) = f -1 u ∩ f -1 v :=
preimage_inter

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.6. Imagen de la unión

3.6.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle, la imagen de un conjunto s por una función f se
-- representa por
--   f ' s = {y | ∃ x, x ∈ s ∧ f x = y}
-- Demostrar que
--   f ' (s ∪ t) = f ' s ∪ f ' t
----- *)

theory Imagen_de_la_union
imports Main
begin

(* 1ª demostración *)

lemma "f ' (s ∪ t) = f ' s ∪ f ' t"
proof (rule equalityI)
  show "f ' (s ∪ t) ⊆ f ' s ∪ f ' t"
  proof (rule subsetI)
    fix y
    assume "y ∈ f ' (s ∪ t)"
    then show "y ∈ f ' s ∪ f ' t"
    proof (rule imageE)

```

```

fix x
  assume "y = f x"
  assume "x ∈ s u t"
  then show "y ∈ f ' s u f ' t"
  proof (rule UnE)
    assume "x ∈ s"
    with <y = f x> have "y ∈ f ' s"
      by (simp only: image_eqI)
    then show "y ∈ f ' s u f ' t"
      by (rule UnI1)
  next
    assume "x ∈ t"
    with <y = f x> have "y ∈ f ' t"
      by (simp only: image_eqI)
    then show "y ∈ f ' s u f ' t"
      by (rule UnI2)
  qed
qed
qed
next
show "f ' s u f ' t ⊆ f ' (s u t)"
proof (rule subsetI)
  fix y
  assume "y ∈ f ' s u f ' t"
  then show "y ∈ f ' (s u t)"
  proof (rule UnE)
    assume "y ∈ f ' s"
    then show "y ∈ f ' (s u t)"
    proof (rule imageE)
      fix x
      assume "y = f x"
      assume "x ∈ s"
      then have "x ∈ s u t"
        by (rule UnI1)
      with <y = f x> show "y ∈ f ' (s u t)"
        by (simp only: image_eqI)
    qed
  next
    assume "y ∈ f ' t"
    then show "y ∈ f ' (s u t)"
    proof (rule imageE)
      fix x
      assume "y = f x"
      assume "x ∈ t"
      then have "x ∈ s u t"

```

```

      by (rule UnI2)
    with <y = f x> show "y ∈ f ' (s u t)"
      by (simp only: image_eqI)
  qed
qed
qed
qed

(* 2ª demostración *)

lemma "f ' (s u t) = f ' s u f ' t"
proof
  show "f ' (s u t) ⊆ f ' s u f ' t"
  proof
    fix y
    assume "y ∈ f ' (s u t)"
    then show "y ∈ f ' s u f ' t"
    proof
      fix x
      assume "y = f x"
      assume "x ∈ s u t"
      then show "y ∈ f ' s u f ' t"
      proof
        assume "x ∈ s"
        with <y = f x> have "y ∈ f ' s"
          by simp
        then show "y ∈ f ' s u f ' t"
          by simp
      next
        assume "x ∈ t"
        with <y = f x> have "y ∈ f ' t"
          by simp
        then show "y ∈ f ' s u f ' t"
          by simp
      qed
    qed
  qed
next
  show "f ' s u f ' t ⊆ f ' (s u t)"
  proof
    fix y
    assume "y ∈ f ' s u f ' t"
    then show "y ∈ f ' (s u t)"
    proof
      assume "y ∈ f ' s"

```

```

then show "y ∈ f ' (s u t)"
proof
  fix x
  assume "y = f x"
  assume "x ∈ s"
  then have "x ∈ s u t"
    by simp
  with <y = f x> show "y ∈ f ' (s u t)"
    by simp
qed
next
assume "y ∈ f ' t"
then show "y ∈ f ' (s u t)"
proof
  fix x
  assume "y = f x"
  assume "x ∈ t"
  then have "x ∈ s u t"
    by simp
  with <y = f x> show "y ∈ f ' (s u t)"
    by simp
qed
qed
qed
qed

(* 3ª demostración *)

Lemma "f ' (s u t) = f ' s u f ' t"
  by (simp only: image_Un)

(* 4ª demostración *)

Lemma "f ' (s u t) = f ' s u f ' t"
  by auto

end

```

3.6.2. Demostraciones con Lean

```

-- -----
-- En Lean, la imagen de un conjunto s por una función f se representa
-- por 'f '' s'; es decir,

```

```

--      f '' s = {y | ∃ x, x ∈ s ∧ f x = y}
--
-- Demostrar que
--      f '' (s ∪ t) = f '' s ∪ f '' t
-----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables s t : set α

-- 1ª demostración
-- =====

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
begin
  ext y,
  split,
  { intro h1,
    cases h1 with x hx,
    cases hx with xst fxy,
    rw ← fxy,
    cases xst with xs xt,
    { left,
      apply mem_image_of_mem,
      exact xs, },
    { right,
      apply mem_image_of_mem,
      exact xt, }},
  { intro h2,
    cases h2 with yfs yft,
    { cases yfs with x hx,
      cases hx with xs fxy,
      rw ← fxy,
      apply mem_image_of_mem,
      left,
      exact xs, },
    { cases yft with x hx,
      cases hx with xt fxy,
      rw ← fxy,
      apply mem_image_of_mem,

```

```

        right,
        exact xt, }},
end

-- 2ª demostración
-- =====

example : f '' (s U t) = f '' s U f '' t :=
begin
  ext y,
  split,
  { rintro (x, xst, fxy),
    rw ← fxy,
    cases xst with xs xt,
    { left,
      exact mem_image_of_mem f xs, },
    { right,
      exact mem_image_of_mem f xt, }},
  { rintros (yfs | yft),
    { rcases yfs with (x, xs, fxy),
      rw ← fxy,
      apply mem_image_of_mem,
      left,
      exact xs, },
    { rcases yft with (x, xt, fxy),
      rw ← fxy,
      apply mem_image_of_mem,
      right,
      exact xt, }},
end

-- 3ª demostración
-- =====

example : f '' (s U t) = f '' s U f '' t :=
begin
  ext y,
  split,
  { rintro (x, xst, rfl),
    cases xst with xs xt,
    { left,
      exact mem_image_of_mem f xs, },
    { right,
      exact mem_image_of_mem f xt, }},
  { rintros (yfs | yft),

```



```

{ rcases yfs with (x, xs, rfl),
  apply mem_image_of_mem,
  left,
  exact xs, },
{ rcases yft with (x, xt, rfl),
  apply mem_image_of_mem,
  right,
  exact xt, }},
end

-- 4ª demostración
-- =====

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
begin
  ext y,
  split,
  { rintro (x, xst, rfl),
    cases xst with xs xt,
    { left,
      use [x, xs], },
    { right,
      use [x, xt], }},
  { rintros (yfs | yft),
    { rcases yfs with (x, xs, rfl),
      use [x, or.inl xs], },
    { rcases yft with (x, xt, rfl),
      use [x, or.inr xt], }},
end

-- 5ª demostración
-- =====

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
begin
  ext y,
  split,
  { rintros (x, xs | xt, rfl),
    { left,
      use [x, xs] },
    { right,
      use [x, xt] }},
  { rintros ((x, xs, rfl) | (x, xt, rfl)),
    { use [x, or.inl xs] },
    { use [x, or.inr xt] }},
end

```

end

-- 6ª demostración

-- =====

example : f '' (s U t) = f '' s U f '' t :=

begin

```
  ext y,
  split,
  { rintros ⟨x, xs | xt, rfl⟩,
    { finish, },
    { finish, }},
  { rintros (⟨x, xs, rfl⟩ | ⟨x, xt, rfl⟩),
    { finish, },
    { finish, }},
```

end

-- 7ª demostración

-- =====

example : f '' (s U t) = f '' s U f '' t :=

begin

```
  ext y,
  split,
  { rintros ⟨x, xs | xt, rfl⟩ ; finish, },
  { rintros (⟨x, xs, rfl⟩ | ⟨x, xt, rfl⟩) ; finish, },
```

end

-- 8ª demostración

-- =====

example : f '' (s U t) = f '' s U f '' t :=

begin

```
  ext y,
  split,
  { finish, },
  { finish, },
```

end

-- 9ª demostración

-- =====

example : f '' (s U t) = f '' s U f '' t :=

begin

```
  ext y,
```

```

rw iff_def,
finish,
end

-- 10ª demostración
-- =====

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
by finish [ext_iff, iff_def, mem_image_eq]

-- 11ª demostración
-- =====

example : f '' (s ∪ t) = f '' s ∪ f '' t :=
image_union f s t

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.7. Imagen inversa de la imagen

3.7.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si s es un subconjunto del dominio de la función f,
-- entonces s está contenido en la imagen inversa de la imagen de s
-- por f; es decir,
--   s ⊆ f-1[f[s]]
----- *)

theory Imagen_inversa_de_la_imagen
imports Main
begin

(* 1ª demostración *)

lemma "s ⊆ f-1 (f '' s)"
proof (rule subsetI)
  fix x
  assume "x ∈ s"
  then have "f x ∈ f '' s"
    by (simp only: imageI)
  then show "x ∈ f-1 (f '' s)"

```

```

    by (simp only: vimageI)
qed

(* 2ª demostración *)

lemma "s ⊆ f ⁻¹ (f ' s)"
proof
  fix x
  assume "x ∈ s"
  then have "f x ∈ f ' s" by simp
  then show "x ∈ f ⁻¹ (f ' s)" by simp
qed

(* 3ª demostración *)

lemma "s ⊆ f ⁻¹ (f ' s)"
  by auto

end

```

3.7.2. Demostraciones con Lean

```

-----
-- Demostrar que si s es un subconjunto del dominio de la función f,
-- entonces s está contenido en la [imagen inversa](https://bit.ly/3ckseBL)
-- de la [imagen de s por f](https://bit.ly/3x2Jxij); es decir,
--   s ⊆ f-1[f[s]]
-----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variable s : set α

-- 1ª demostración
-- =====

example : s ⊆ f-1 (f ' s) :=
begin
  intros x xs,

```

```

    apply mem_preimage.mpr,
    apply mem_image_of_mem,
    exact xs,
end

-- 2ª demostración
-- =====

example : s ⊆ f ⁻¹' (f '' s) :=
begin
  intros x xs,
  apply mem_image_of_mem,
  exact xs,
end

-- 3ª demostración
-- =====

example : s ⊆ f ⁻¹' (f '' s) :=
λ x, mem_image_of_mem f

-- 4ª demostración
-- =====

example : s ⊆ f ⁻¹' (f '' s) :=
begin
  intros x xs,
  show f x ∈ f '' s,
  use [x, xs],
end

-- 5ª demostración
-- =====

example : s ⊆ f ⁻¹' (f '' s) :=
begin
  intros x xs,
  use [x, xs],
end

-- 6ª demostración
-- =====

example : s ⊆ f ⁻¹' (f '' s) :=
subset_preimage_image f s

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.8. Subconjunto de la imagen inversa

3.8.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--    $f[s] \subseteq u \leftrightarrow s \subseteq f^{-1}[u]$ 
-- ----- *)

theory Subconjunto_de_la_imagen_inversa
imports Main
begin

(* 1ª demostración *)

lemma "f ' s  $\subseteq$  u  $\leftrightarrow$  s  $\subseteq$  f - ' u"
proof (rule iffI)
  assume "f ' s  $\subseteq$  u"
  show "s  $\subseteq$  f - ' u"
  proof (rule subsetI)
    fix x
    assume "x  $\in$  s"
    then have "f x  $\in$  f ' s"
      by (simp only: imageI)
    then have "f x  $\in$  u"
      using <f ' s  $\subseteq$  u> by (rule set_rev_mp)
    then show "x  $\in$  f - ' u"
      by (simp only: vimageI)
  qed
next
  assume "s  $\subseteq$  f - ' u"
  show "f ' s  $\subseteq$  u"
  proof (rule subsetI)
    fix y
    assume "y  $\in$  f ' s"
    then show "y  $\in$  u"
  proof
    fix x
    assume "y = f x"
    assume "x  $\in$  s"
    then have "x  $\in$  f - ' u"
```

```

    using <s ⊆ f -' u> by (rule set_rev_mp)
    then have "f x ∈ u"
      by (rule vimageD)
    with <y = f x> show "y ∈ u"
      by (rule ssubst)
  qed
qed
qed

(* 2ª demostración *)

lemma "f ' s ⊆ u ↔ s ⊆ f -' u"
proof
  assume "f ' s ⊆ u"
  show "s ⊆ f -' u"
  proof
    fix x
    assume "x ∈ s"
    then have "f x ∈ f ' s"
      by simp
    then have "f x ∈ u"
      using <f ' s ⊆ u> by (simp add: set_rev_mp)
    then show "x ∈ f -' u"
      by simp
  qed
next
  assume "s ⊆ f -' u"
  show "f ' s ⊆ u"
  proof
    fix y
    assume "y ∈ f ' s"
    then show "y ∈ u"
    proof
      fix x
      assume "y = f x"
      assume "x ∈ s"
      then have "x ∈ f -' u"
        using <s ⊆ f -' u> by (simp only: set_rev_mp)
      then have "f x ∈ u"
        by simp
      with <y = f x> show "y ∈ u"
        by simp
    qed
  qed
qed

```

```

(* 3ª demostración *)

lemma "f ' s ⊆ u ↔ s ⊆ f -' u"
  by (simp only: image_subset_iff_subset_vimage)

(* 4ª demostración *)

lemma "f ' s ⊆ u ↔ s ⊆ f -' u"
  by auto

end

```

3.8.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f[s] ⊆ u ↔ s ⊆ f-1[u]
-- -----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variable s : set α
variable u : set β

-- 1ª demostración
-- =====

example : f '' s ⊆ u ↔ s ⊆ f-1 u :=
begin
  split,
  { intros h x xs,
    apply mem_preimage.mpr,
    apply h,
    apply mem_image_of_mem,
    exact xs, },
  { intros h y hy,
    rcases hy with ⟨x, xs, fxy⟩,
    rw ← fxy,
    exact h xs, },

```



```

end

-- 2ª demostración
-- =====

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
begin
  split,
  { intros h x xs,
    apply h,
    apply mem_image_of_mem,
    exact xs, },
  { rintros h y (x, xs, rfl),
    exact h xs, },
end

-- 3ª demostración
-- =====

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
image_subset_iff

-- 4ª demostración
-- =====

example : f '' s ⊆ u ↔ s ⊆ f ⁻¹' u :=
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.9. Imagen inversa de la imagen de aplicaciones inyectivas

3.9.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si f es inyectiva, entonces
--   f⁻¹[f[s]] ⊆ s
-- ----- *)

theory Imagen_inversa_de_la_imagen_de_aplicaciones_inyectivas
imports Main

```

```

begin

(* 1ª demostración *)

lemma
  assumes "inj f"
  shows "f -' (f ' s)  $\subseteq$  s"
proof (rule subsetI)
  fix x
  assume "x  $\in$  f -' (f ' s)"
  then have "f x  $\in$  f ' s"
    by (rule vimageD)
  then show "x  $\in$  s"
  proof (rule imageE)
    fix y
    assume "f x = f y"
    assume "y  $\in$  s"
    have "x = y"
      using <inj f> <f x = f y> by (rule injD)
    then show "x  $\in$  s"
      using <y  $\in$  s> by (rule ssubst)
  qed
qed

(* 2ª demostración *)

lemma
  assumes "inj f"
  shows "f -' (f ' s)  $\subseteq$  s"
proof
  fix x
  assume "x  $\in$  f -' (f ' s)"
  then have "f x  $\in$  f ' s"
    by simp
  then show "x  $\in$  s"
  proof
    fix y
    assume "f x = f y"
    assume "y  $\in$  s"
    have "x = y"
      using <inj f> <f x = f y> by (rule injD)
    then show "x  $\in$  s"
      using <y  $\in$  s> by simp
  qed
qed

```

```

(* 3ª demostración *)

lemma
  assumes "inj f"
  shows "f ⁻¹ (f ' s) ⊆ s"
  using assms
  unfolding inj_def
  by auto

(* 4ª demostración *)

lemma
  assumes "inj f"
  shows "f ⁻¹ (f ' s) ⊆ s"
  using assms
  by (simp only: inj_vimage_image_eq)

end

```

3.9.2. Demostraciones con Lean

```

-----
-- Demostrar que si f es inyectiva, entonces
--    $f^{-1}[f[s]] \subseteq s$ 
-----

```

```

import data.set.basic

open set function

variables {α : Type*} {β : Type*}
variable f : α → β
variable s : set α

-- 1ª demostración
-- =====

example
  (h : injective f)
  : f ⁻¹ (f ' s) ⊆ s :=
begin
  intros x hx,
  rw mem_preimage at hx,

```

```

rw mem_image_eq at hx,
cases hx with y hy,
cases hy with ys fyx,
unfold injective at h,
have h1 : y = x := h fyx,
rw h1,
exact ys,
end

-- 2ª demostración
-- =====

example
  (h : injective f)
  : f ⁻¹ (f '' s) ⊆ s :=
begin
  intros x hx,
  rw mem_preimage at hx,
  rcases hx with (y, ys, fyx),
  rw h fyx,
  exact ys,
end

-- 3ª demostración
-- =====

example
  (h : injective f)
  : f ⁻¹ (f '' s) ⊆ s :=
begin
  rintros x (y, ys, hy),
  rw h hy,
  exact ys,
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.10. Imagen de la imagen inversa

3.10.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--    $f^{-1}(f^{-1}u) \subseteq u$ 
-- ----- *)

theory Imagen_de_la_imagen_inversa
imports Main
begin

(* 1ª demostración *)

lemma "f-1 (f-1 u) ⊆ u"
proof (rule subsetI)
  fix y
  assume "y ∈ f-1 (f-1 u)"
  then show "y ∈ u"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume "x ∈ f-1 u"
    then have "f x ∈ u"
      by (rule vimageD)
    with <y = f x> show "y ∈ u"
      by (rule ssubst)
  qed
qed

(* 2ª demostración *)

lemma "f-1 (f-1 u) ⊆ u"
proof
  fix y
  assume "y ∈ f-1 (f-1 u)"
  then show "y ∈ u"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ f-1 u"
    then have "f x ∈ u"
      by simp
    with <y = f x> show "y ∈ u"
      by simp
  qed
qed

```

```

(* 3ª demostración *)

lemma "f ' (f -' u) ⊆ u"
  by (simp only: image_vimage_subset)

(* 4ª demostración *)

lemma "f ' (f -' u) ⊆ u"
  by auto

end

```

3.10.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f '' (f-1 u) ⊆ u
-- -----

import data.set.basic
open set

variables {α : Type*} {β : Type*}
variable f : α → β
variable u : set β

-- 1ª demostración
-- =====

example : f '' (f-1 u) ⊆ u :=
begin
  intros y h,
  cases h with x h2,
  cases h2 with hx fxy,
  rw ← fxy,
  exact hx,
end

-- 2ª demostración
-- =====

example : f '' (f-1 u) ⊆ u :=
begin

```

```

    intros y h,
    rcases h with ⟨x, hx, fxy⟩,
    rw ← fxy,
    exact hx,
end

-- 3ª demostración
-- =====

example : f '' (f⁻¹' u) ⊆ u :=
begin
  rintros y ⟨x, hx, fxy⟩,
  rw ← fxy,
  exact hx,
end

-- 4ª demostración
-- =====

example : f '' (f⁻¹' u) ⊆ u :=
begin
  rintros y ⟨x, hx, rfl⟩,
  exact hx,
end

-- 5ª demostración
-- =====

example : f '' (f⁻¹' u) ⊆ u :=
image_preimage_subset f u

-- 6ª demostración
-- =====

example : f '' (f⁻¹' u) ⊆ u :=
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.11. Imagen de imagen inversa de aplicaciones suprayectivas

3.11.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que si f es suprayectiva, entonces
--    $u \subseteq f^{-1}(f \cdot u)$ 
*----- *)

theory Imagen_de_imagen_inversa_de_aplicaciones_suprayectivas
imports Main
begin

(* 1ª demostración *)

lemma
  assumes "surj f"
  shows " $u \subseteq f^{-1}(f \cdot u)$ "
proof (rule subsetI)
  fix y
  assume "y ∈ u"
  have " $\exists x. y = f x$ "
    using <surj f> by (rule surjD)
  then obtain x where "y = f x"
    by (rule exE)
  then have "f x ∈ u"
    using <y ∈ u> by (rule subst)
  then have "x ∈ f-1 u"
    by (simp only: vimage_eq)
  then have "f x ∈ f-1 (f-1 u)"
    by (rule imageI)
  with <y = f x> show "y ∈ f-1 (f-1 u)"
    by (rule ssubst)
qed

(* 2ª demostración *)

lemma
  assumes "surj f"
  shows " $u \subseteq f^{-1}(f \cdot u)$ "
proof
  fix y
```



```

assume "y ∈ u"
have "∃x. y = f x"
  using ⟨surj f⟩ by (rule surjD)
then obtain x where "y = f x"
  by (rule exE)
then have "f x ∈ u"
  using ⟨y ∈ u⟩ by simp
then have "x ∈ f ⁻¹ u"
  by simp
then have "f x ∈ f ⁻¹ (f ⁻¹ u)"
  by simp
with ⟨y = f x⟩ show "y ∈ f ⁻¹ (f ⁻¹ u)"
  by simp
qed

```

(* 3ª demostración *)

Lemma

```

assumes "surj f"
shows "u ⊆ f ⁻¹ (f ⁻¹ u)"
using assms
by (simp only: surj_image_vimage_eq)

```

(* 4ª demostración *)

Lemma

```

assumes "surj f"
shows "u ⊆ f ⁻¹ (f ⁻¹ u)"
using assms
unfolding surj_def
by auto

```

(* 5ª demostración *)

Lemma

```

assumes "surj f"
shows "u ⊆ f ⁻¹ (f ⁻¹ u)"
using assms
by auto

```

end

3.11.2. Demostraciones con Lean

```
-- Demostrar que si f es suprayectiva, entonces
--  $u \subseteq f^{-1}(f^{-1} u)$ 
```

```
import data.set.basic
```

```
open set function
```

```
variables {α : Type*} {β : Type*}
```

```
variable f : α → β
```

```
variable u : set β
```

```
-- 1ª demostración
```

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

```
example
```

```
(h : surjective f)
```

```
: u ⊆ f ⁻¹ (f ⁻¹ u) :=
```

```
begin
```

```
intros y yu,
```

```
cases h y with x fxy,
```

```
use x,
```

```
split,
```

```
{ apply mem_preimage.mpr,
```

```
rw fxy,
```

```
exact yu },
```

```
{ exact fxy },
```

```
end
```

```
-- 2ª demostración
```

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

```
example
```

```
(h : surjective f)
```

```
: u ⊆ f ⁻¹ (f ⁻¹ u) :=
```

```
begin
```

```
intros y yu,
```

```
cases h y with x fxy,
```

```
use x,
```

```
split,
```

```
{ show f x ∈ u,
```

```
rw fxy,
```

```
exact yu },
```

```

{ exact fxy },
end

-- 3ª demostración
-- =====

example
  (h : surjective f)
  : u ⊆ f '' (f⁻¹' u) :=
begin
  intros y yu,
  cases h y with x fxy,
  by finish,
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.12. Monotonía de la imagen de conjuntos

3.12.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si  $s \subseteq t$ , entonces
--  $f' s \subseteq f' t$ 
-- ----- *)

theory Monotonia_de_la_imagen_de_conjuntos
imports Main
begin

(* 1ª demostración *)

lemma
  assumes "s ⊆ t"
  shows "f' s ⊆ f' t"
proof (rule subsetI)
  fix y
  assume "y ∈ f' s"
  then show "y ∈ f' t"
proof (rule imageE)
  fix x
  assume "y = f x"
  assume "x ∈ s"

```

```

    then have "x ∈ t"
      using <s ⊆ t> by (simp only: set_rev_mp)
    then have "f x ∈ f ' t"
      by (rule imageI)
    with <y = f x> show "y ∈ f ' t"
      by (rule ssubst)
  qed
qed

```

(* 2ª demostración *)

Lemma

```

  assumes "s ⊆ t"
  shows "f ' s ⊆ f ' t"

```

proof

```

  fix y
  assume "y ∈ f ' s"
  then show "y ∈ f ' t"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s"
    then have "x ∈ t"
      using <s ⊆ t> by (simp only: set_rev_mp)
    then have "f x ∈ f ' t"
      by simp
    with <y = f x> show "y ∈ f ' t"
      by simp
  qed

```

qed

qed

(* 3ª demostración *)

Lemma

```

  assumes "s ⊆ t"
  shows "f ' s ⊆ f ' t"
  using assms
  by blast

```

(* 4ª demostración *)

Lemma

```

  assumes "s ⊆ t"
  shows "f ' s ⊆ f ' t"
  using assms

```

```

by (simp only: image_mono)

end

```

3.12.2. Demostraciones con Lean

```

-- Demostrar que si  $s \subseteq t$ , entonces
--  $f '' s \subseteq f '' t$ 

```

```

import data.set.basic
import tactic

```

```

open set

```

```

variables {α : Type*} {β : Type*}
variable f : α → β
variables s t : set α

```

```

-- 1ª demostración
-- =====

```

```

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
begin
  intros y hy,
  rw mem_image at hy,
  cases hy with x hx,
  cases hx with xs fxy,
  use x,
  split,
  { exact h xs, },
  { exact fxy, },
end

```

```

-- 2ª demostración
-- =====

```

```

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=

```

```

begin
  intros y hy,
  rcases hy with ⟨x, xs, fxy⟩,
  use x,
  exact ⟨h xs, fxy⟩,
end

-- 3ª demostración
-- =====

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
begin
  rintros y ⟨x, xs, fxy⟩,
  use [x, h xs, fxy],
end

-- 4ª demostración
-- =====

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
by finish [subset_def, mem_image_eq]

-- 5ª demostración
-- =====

example
  (h : s ⊆ t)
  : f '' s ⊆ f '' t :=
image_subset f h

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.13. Monotonía de la imagen inversa

3.13.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si  $u \subseteq v$ , entonces

```

```

--       $f^{-1} u \subseteq f^{-1} v$ 
----- *)

theory Monotonia_de_la_imagen_inversa
imports Main
begin

(* 1ª demostración *)

lemma
  assumes "u  $\subseteq$  v"
  shows "f-1 u  $\subseteq$  f-1 v"
proof (rule subsetI)
  fix x
  assume "x  $\in$  f-1 u"
  then have "f x  $\in$  u"
    by (rule vimageD)
  then have "f x  $\in$  v"
    using <u  $\subseteq$  v> by (rule set_rev_mp)
  then show "x  $\in$  f-1 v"
    by (simp only: vimage_eq)
qed

(* 2ª demostración *)

lemma
  assumes "u  $\subseteq$  v"
  shows "f-1 u  $\subseteq$  f-1 v"
proof
  fix x
  assume "x  $\in$  f-1 u"
  then have "f x  $\in$  u"
    by simp
  then have "f x  $\in$  v"
    using <u  $\subseteq$  v> by (rule set_rev_mp)
  then show "x  $\in$  f-1 v"
    by simp
qed

(* 3ª demostración *)

lemma
  assumes "u  $\subseteq$  v"
  shows "f-1 u  $\subseteq$  f-1 v"
  using assms

```

```

by (simp only: vimage_mono)

(* 4ª demostración *)

lemma
  assumes "u ⊆ v"
  shows "f ⁻¹' u ⊆ f ⁻¹' v"
  using assms
  by blast

end

```

3.13.2. Demostraciones con Lean

```

-- -----
-- Demostrar que si  $u \subseteq v$ , entonces
--  $f^{-1}' u \subseteq f^{-1}' v$ 
-- -----

```

```

import data.set.basic
open set

```

```

variables {α : Type*} {β : Type*}
variable f : α → β
variables u v : set β

```

```

-- 1ª demostración
-- =====

```

```

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
begin
  intros x hx,
  apply mem_preimage.mpr,
  apply h,
  apply mem_preimage.mp,
  exact hx,
end

```

```

-- 2ª demostración
-- =====

```



```

example
  (h : u  $\subseteq$  v)
  : f-1 u  $\subseteq$  f-1 v :=
begin
  intros x hx,
  apply h,
  exact hx,
end

```

```

-- 3ª demostración
-- =====

```

```

example
  (h : u  $\subseteq$  v)
  : f-1 u  $\subseteq$  f-1 v :=
begin
  intros x hx,
  exact h hx,
end

```

```

-- 4ª demostración
-- =====

```

```

example
  (h : u  $\subseteq$  v)
  : f-1 u  $\subseteq$  f-1 v :=
λ x hx, h hx

```

```

-- 5ª demostración
-- =====

```

```

example
  (h : u  $\subseteq$  v)
  : f-1 u  $\subseteq$  f-1 v :=
by intro x; apply h

```

```

-- 6ª demostración
-- =====

```

```

example
  (h : u  $\subseteq$  v)
  : f-1 u  $\subseteq$  f-1 v :=
preimage_mono h

```

```
-- 7ª demostración
-- =====

example
  (h : u ⊆ v)
  : f ⁻¹' u ⊆ f ⁻¹' v :=
by tauto
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.14. Imagen inversa de la unión

3.14.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--   f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v
-- ----- *)

theory Imagen_inversa_de_la_union
imports Main
begin

(* 1ª demostración *)

lemma "f ⁻¹' (u ∪ v) = f ⁻¹' u ∪ f ⁻¹' v"
proof (rule equalityI)
  show "f ⁻¹' (u ∪ v) ⊆ f ⁻¹' u ∪ f ⁻¹' v"
  proof (rule subsetI)
    fix x
    assume "x ∈ f ⁻¹' (u ∪ v)"
    then have "f x ∈ u ∪ v"
      by (rule vimageD)
    then show "x ∈ f ⁻¹' u ∪ f ⁻¹' v"
    proof (rule UnE)
      assume "f x ∈ u"
      then have "x ∈ f ⁻¹' u"
        by (rule vimageI2)
      then show "x ∈ f ⁻¹' u ∪ f ⁻¹' v"
        by (rule UnI1)
    next
      assume "f x ∈ v"

```

```

    then have "x ∈ f -' v"
    by (rule vimageI2)
    then show "x ∈ f -' u ∪ f -' v"
    by (rule UnI2)
  qed
qed
next
show "f -' u ∪ f -' v ⊆ f -' (u ∪ v)"
proof (rule subsetI)
  fix x
  assume "x ∈ f -' u ∪ f -' v"
  then show "x ∈ f -' (u ∪ v)"
  proof (rule UnE)
    assume "x ∈ f -' u"
    then have "f x ∈ u"
    by (rule vimageD)
    then have "f x ∈ u ∪ v"
    by (rule UnI1)
    then show "x ∈ f -' (u ∪ v)"
    by (rule vimageI2)
  next
    assume "x ∈ f -' v"
    then have "f x ∈ v"
    by (rule vimageD)
    then have "f x ∈ u ∪ v"
    by (rule UnI2)
    then show "x ∈ f -' (u ∪ v)"
    by (rule vimageI2)
  qed
qed
qed

(* 2ª demostración *)

lemma "f -' (u ∪ v) = f -' u ∪ f -' v"
proof
  show "f -' (u ∪ v) ⊆ f -' u ∪ f -' v"
  proof
    fix x
    assume "x ∈ f -' (u ∪ v)"
    then have "f x ∈ u ∪ v" by simp
    then show "x ∈ f -' u ∪ f -' v"
    proof
      assume "f x ∈ u"
      then have "x ∈ f -' u" by simp
    qed
  qed

```

```

    then show "x ∈ f -' u u f -' v" by simp
  next
    assume "f x ∈ v"
    then have "x ∈ f -' v" by simp
    then show "x ∈ f -' u u f -' v" by simp
  qed
qed
next
show "f -' u u f -' v ⊆ f -' (u u v)"
proof
  fix x
  assume "x ∈ f -' u u f -' v"
  then show "x ∈ f -' (u u v)"
  proof
    assume "x ∈ f -' u"
    then have "f x ∈ u" by simp
    then have "f x ∈ u u v" by simp
    then show "x ∈ f -' (u u v)" by simp
  next
    assume "x ∈ f -' v"
    then have "f x ∈ v" by simp
    then have "f x ∈ u u v" by simp
    then show "x ∈ f -' (u u v)" by simp
  qed
qed
qed

(* 3ª demostración *)

lemma "f -' (u u v) = f -' u u f -' v"
  by (simp only: vimage_Un)

(* 4ª demostración *)

lemma "f -' (u u v) = f -' u u f -' v"
  by auto

end

```

3.14.2. Demostraciones con Lean

```

-----
-- Demostrar que
--    $f^{-1'} (u \cup v) = f^{-1'} u \cup f^{-1'} v$ 
-----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables u v : set β

-- 1ª demostración
-- =====

example : f-1' (u ∪ v) = f-1' u ∪ f-1' v :=
begin
  ext x,
  split,
  { intros h,
    rw mem_preimage at h,
    cases h with fxu fxv,
    { left,
      apply mem_preimage.mpr,
      exact fxu, },
    { right,
      apply mem_preimage.mpr,
      exact fxv, }},
  { intro h,
    rw mem_preimage,
    cases h with xfu xfv,
    { rw mem_preimage at xfu,
      left,
      exact xfu, },
    { rw mem_preimage at xfv,
      right,
      exact xfv, }},
end

-- 2ª demostración
-- =====

example : f-1' (u ∪ v) = f-1' u ∪ f-1' v :=
begin

```

```

ext x,
split,
{ intros h,
  cases h with f xu fxv,
  { left,
    exact fxu, },
  { right,
    exact fxv, }},
{ intro h,
  cases h with x fu xfv,
  { left,
    exact x fu, },
  { right,
    exact x fv, }},
end

-- 3ª demostración
-- =====

example : f -1' (u  $\sqcup$  v) = f -1' u  $\sqcup$  f -1' v :=
begin
  ext x,
  split,
  { rintro (fxu | fxv),
    { exact or.inl fxu, },
    { exact or.inr fxv, }},
  { rintro (xfu | xfv),
    { exact or.inl xfu, },
    { exact or.inr xfv, }},
end

-- 4ª demostración
-- =====

example : f -1' (u  $\sqcup$  v) = f -1' u  $\sqcup$  f -1' v :=
begin
  ext x,
  split,
  { finish, },
  { finish, } ,
end

-- 5ª demostración
-- =====

```

```

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
begin
  ext x,
  finish,
end

-- 6ª demostración
-- =====

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
by ext; finish

-- 7ª demostración
-- =====

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
by ext; refl

-- 8ª demostración
-- =====

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
refl

-- 9ª demostración
-- =====

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
preimage_union

-- 10ª demostración
-- =====

example : f ⁻¹ (u ∪ v) = f ⁻¹ u ∪ f ⁻¹ v :=
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.15. Imagen de la intersección

3.15.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--  $f' (s \cap t) \subseteq f' s \cap f' t$ 
-- ----- *)

theory Imagen_de_la_interseccion
imports Main
begin

(* 1ª demostración *)

lemma "f' (s ∩ t) ⊆ f' s ∩ f' t"
proof (rule subsetI)
  fix y
  assume "y ∈ f' (s ∩ t)"
  then have "y ∈ f' s"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume "x ∈ s ∩ t"
    have "x ∈ s"
      using "x ∈ s ∩ t" by (rule IntD1)
    then have "f x ∈ f' s"
      by (rule imageI)
    with "y = f x" show "y ∈ f' s"
      by (rule ssubst)
  qed
  moreover
  note "y ∈ f' (s ∩ t)"
  then have "y ∈ f' t"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume "x ∈ s ∩ t"
    have "x ∈ t"
      using "x ∈ s ∩ t" by (rule IntD2)
    then have "f x ∈ f' t"
      by (rule imageI)
    with "y = f x" show "y ∈ f' t"
      by (rule ssubst)
  qed
end
```



```

qed
ultimately show "y ∈ f ' s n f ' t"
  by (rule IntI)
qed

(* 2ª demostración *)

Lemma "f ' (s n t) ⊆ f ' s n f ' t"
proof
  fix y
  assume "y ∈ f ' (s n t)"
  then have "y ∈ f ' s"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s n t"
    have "x ∈ s"
      using ⟨x ∈ s n t⟩ by simp
    then have "f x ∈ f ' s"
      by simp
    with ⟨y = f x⟩ show "y ∈ f ' s"
      by simp
  qed
  moreover
  note ⟨y ∈ f ' (s n t)⟩
  then have "y ∈ f ' t"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s n t"
    have "x ∈ t"
      using ⟨x ∈ s n t⟩ by simp
    then have "f x ∈ f ' t"
      by simp
    with ⟨y = f x⟩ show "y ∈ f ' t"
      by simp
  qed
  ultimately show "y ∈ f ' s n f ' t"
    by simp
qed

(* 3ª demostración *)

Lemma "f ' (s n t) ⊆ f ' s n f ' t"
proof

```

```

fix y
  assume "y ∈ f ' (s n t)"
  then obtain x where hx : "y = f x ∧ x ∈ s n t" by auto
  then have "y = f x" by simp
  have "x ∈ s" using hx by simp
  have "x ∈ t" using hx by simp
  have "y ∈ f ' s" using ⟨y = f x⟩ ⟨x ∈ s⟩ by simp
  moreover
  have "y ∈ f ' t" using ⟨y = f x⟩ ⟨x ∈ t⟩ by simp
  ultimately show "y ∈ f ' s n f ' t"
    by simp
qed

(* 4ª demostración *)

lemma "f ' (s n t) ⊆ f ' s n f ' t"
  by (simp only: image_Int_subset)

(* 5ª demostración *)

lemma "f ' (s n t) ⊆ f ' s n f ' t"
  by auto

end

```

3.15.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f '' (s n t) ⊆ f '' s n f '' t
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables s t : set α

-- 1ª demostración
-- =====

```

```
example : f '' (s n t) ⊆ f '' s n f '' t :=
begin
```

```
  intros y hy,
  cases hy with x hx,
  cases hx with xst fxy,
  split,
  { use x,
    split,
    { exact xst.1, },
    { exact fxy, }},
  { use x,
    split,
    { exact xst.2, },
    { exact fxy, }},
```

```
end
```

```
-- 2ª demostración
```

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

```
example : f '' (s n t) ⊆ f '' s n f '' t :=
begin
```

```
  intros y hy,
  rcases hy with (x, (xs, xt), fxy),
  split,
  { use x,
    exact (xs, fxy), },
  { use x,
    exact (xt, fxy), },
```

```
end
```

```
-- 3ª demostración
```

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

```
example : f '' (s n t) ⊆ f '' s n f '' t :=
begin
```

```
  rintros y (x, (xs, xt), fxy),
  split,
  { use [x, xs, fxy], },
  { use [x, xt, fxy], },
```

```
end
```

```
-- 4ª demostración
```

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

```

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
image_inter_subset f s t

-- 5ª demostración
-- =====

example : f '' (s ∩ t) ⊆ f '' s ∩ f '' t :=
by intro ; finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.16. Imagen de la intersección de aplicaciones inyectivas

3.16.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si f es inyectiva, entonces
--   f ' s ∩ f ' t ⊆ f ' (s ∩ t)
-- ----- *)

theory Imagen_de_la_interseccion_de_aplicaciones_inyectivas
imports Main
begin

(* 1ª demostración *)

lemma
  assumes "inj f"
  shows "f ' s ∩ f ' t ⊆ f ' (s ∩ t)"
proof (rule subsetI)
  fix y
  assume "y ∈ f ' s ∩ f ' t"
  then have "y ∈ f ' s"
    by (rule IntD1)
  then show "y ∈ f ' (s ∩ t)"
proof (rule imageE)
  fix x
  assume "y = f x"
  assume "x ∈ s"
  have "x ∈ t"
  proof -

```

```

have "y ∈ f ' t"
  using ⟨y ∈ f ' s ∧ f ' t⟩ by (rule IntD2)
then show "x ∈ t"
proof (rule imageE)
  fix z
  assume "y = f z"
  assume "z ∈ t"
  have "f x = f z"
    using ⟨y = f x⟩ ⟨y = f z⟩ by (rule subst)
  with ⟨inj f⟩ have "x = z"
    by (simp only: inj_eq)
  then show "x ∈ t"
    using ⟨z ∈ t⟩ by (rule ssubst)
qed
qed
with ⟨x ∈ s⟩ have "x ∈ s ∧ t"
  by (rule IntI)
with ⟨y = f x⟩ show "y ∈ f ' (s ∧ t)"
  by (rule image_eqI)
qed
qed

(* 2ª demostración *)

lemma
  assumes "inj f"
  shows "f ' s ∧ f ' t ⊆ f ' (s ∧ t)"
proof
  fix y
  assume "y ∈ f ' s ∧ f ' t"
  then have "y ∈ f ' s" by simp
  then show "y ∈ f ' (s ∧ t)"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s"
    have "x ∈ t"
    proof -
      have "y ∈ f ' t" using ⟨y ∈ f ' s ∧ f ' t⟩ by simp
      then show "x ∈ t"
      proof
        fix z
        assume "y = f z"
        assume "z ∈ t"
        have "f x = f z" using ⟨y = f x⟩ ⟨y = f z⟩ by simp

```

```

    with <inj f> have "x = z" by (simp only: inj_eq)
    then show "x ∈ t" using <z ∈ t> by simp
  qed
qed
with <x ∈ s> have "x ∈ s n t" by simp
with <y = f x> show "y ∈ f ' (s n t)" by simp
qed
qed

(* 3ª demostración *)

lemma
  assumes "inj f"
  shows "f ' s n f ' t ⊆ f ' (s n t)"
  using assms
  by (simp only: image_Int)

(* 4ª demostración *)

lemma
  assumes "inj f"
  shows "f ' s n f ' t ⊆ f ' (s n t)"
  using assms
  unfolding inj_def
  by auto

end

```

3.16.2. Demostraciones con Lean

```

-----
-- Demostrar que si f es inyectiva, entonces
--   f '' s n f '' t ⊆ f '' (s n t)
-----

```

```

import data.set.basic

open set function

variables {α : Type*} {β : Type*}
variable f : α → β
variables s t : set α

```

```

-- 1ª demostración
-- =====

example
  (h : injective f)
  : f '' s ∩ f '' t ⊆ f '' (s ∩ t) :=
begin
  intros y hy,
  cases hy with hy1 hy2,
  cases hy1 with x1 hx1,
  cases hx1 with x1s fx1y,
  cases hy2 with x2 hx2,
  cases hx2 with x2t fx2y,
  use x1,
  split,
  { split,
    { exact x1s, },
    { convert x2t,
      apply h,
      rw ← fx2y at fx1y,
      exact fx1y, }},
  { exact fx1y, },
end

-- 2ª demostración
-- =====

example
  (h : injective f)
  : f '' s ∩ f '' t ⊆ f '' (s ∩ t) :=
begin
  rintros y ((x1, x1s, fx1y), (x2, x2t, fx2y)),
  use x1,
  split,
  { split,
    { exact x1s, },
    { convert x2t,
      apply h,
      rw ← fx2y at fx1y,
      exact fx1y, }},
  { exact fx1y, },
end

-- 3ª demostración
-- =====

```

```

example
  (h : injective f)
  : f ' s ∩ f ' t ⊆ f ' (s ∩ t) :=
begin
  rintros y <{(x1, x1s, fx1y), (x2, x2t, fx2y)}>,
  unfold injective at h,
  finish,
end

-- 4ª demostración
-- =====

```

```

example
  (h : injective f)
  : f ' s ∩ f ' t ⊆ f ' (s ∩ t) :=
by intro ; unfold injective at * ; finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.17. Imagen de la diferencia de conjuntos

3.17.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   f ' s - f ' t ⊆ f ' (s - t)
-- ----- *)

theory Imagen_de_la_diferencia_de_conjuntos
imports Main
begin

(* 1ª demostración *)

lemma "f ' s - f ' t ⊆ f ' (s - t)"
proof (rule subsetI)
  fix y
  assume hy : "y ∈ f ' s - f ' t"
  then show "y ∈ f ' (s - t)"
  proof (rule DiffE)
    assume "y ∈ f ' s"
    assume "y ∉ f ' t"

```



```

note <y ∈ f ' s>
then show "y ∈ f ' (s - t)"
proof (rule imageE)
  fix x
  assume "y = f x"
  assume "x ∈ s"
  have <x ∉ t>
  proof (rule notI)
    assume "x ∈ t"
    then have "f x ∈ f ' t"
      by (rule imageI)
    with <y = f x> have "y ∈ f ' t"
      by (rule ssubst)
    with <y ∉ f ' t> show False
      by (rule notE)
  qed
  with <x ∈ s> have "x ∈ s - t"
    by (rule DiffI)
  then have "f x ∈ f ' (s - t)"
    by (rule imageI)
  with <y = f x> show "y ∈ f ' (s - t)"
    by (rule ssubst)
  qed
qed

(* 2ª demostración *)

lemma "f ' s - f ' t ⊆ f ' (s - t)"
proof
  fix y
  assume hy : "y ∈ f ' s - f ' t"
  then show "y ∈ f ' (s - t)"
  proof
    assume "y ∈ f ' s"
    assume "y ∉ f ' t"
    note <y ∈ f ' s>
    then show "y ∈ f ' (s - t)"
    proof
      fix x
      assume "y = f x"
      assume "x ∈ s"
      have <x ∉ t>
      proof
        assume "x ∈ t"

```

```

    then have "f x ∈ f ' t" by simp
    with <y = f x> have "y ∈ f ' t" by simp
    with <y ∉ f ' t> show False by simp
  qed
with <x ∈ s> have "x ∈ s - t" by simp
then have "f x ∈ f ' (s - t)" by simp
with <y = f x> show "y ∈ f ' (s - t)" by simp
qed
qed
qed

(* 3ª demostración *)

lemma "f ' s - f ' t ⊆ f ' (s - t)"
  by (simp only: image_diff_subset)

(* 4ª demostración *)

lemma "f ' s - f ' t ⊆ f ' (s - t)"
  by auto

end

```

3.17.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f '' s \ f '' t ⊆ f '' (s \ t)
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables s t : set α

-- 1ª demostración
-- =====

example : f '' s \ f '' t ⊆ f '' (s \ t) :=

```

```

begin
  intros y hy,
  cases hy with yfs ynft,
  cases yfs with x hx,
  cases hx with xs fxy,
  use x,
  split,
  { split,
    { exact xs, },
    { dsimp,
      intro xt,
      apply ynft,
      rw ← fxy,
      apply mem_image_of_mem,
      exact xt, }},
    { exact fxy, },
  }
end

-- 2ª demostración
-- =====

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
begin
  rintros y ⟨(x, xs, fxy), ynft⟩,
  use x,
  split,
  { split,
    { exact xs, },
    { intro xt,
      apply ynft,
      use [x, xt, fxy], }},
    { exact fxy, },
  }
end

-- 3ª demostración
-- =====

example : f '' s \ f '' t ⊆ f '' (s \ t) :=
begin
  rintros y ⟨(x, xs, fxy), ynft⟩,
  use x,
  finish,
end

-- 4ª demostración

```

```
-- =====
example : f '' s \ f '' t ⊆ f '' (s \ t) :=
subset_image_diff f s t
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.18. Imagen inversa de la diferencia

3.18.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--   f -' u - f -' v ⊆ f -' (u - v)
-- ----- *)

theory Imagen_inversa_de_la_diferencia
imports Main
begin

(* 1ª demostración *)

lemma "f -' u - f -' v ⊆ f -' (u - v)"
proof (rule subsetI)
  fix x
  assume "x ∈ f -' u - f -' v"
  then have "f x ∈ u - v"
  proof (rule DiffE)
    assume "x ∈ f -' u"
    assume "x ∉ f -' v"
    have "f x ∈ u"
      using <x ∈ f -' u> by (rule vimageD)
    moreover
    have "f x ∉ v"
    proof (rule notI)
      assume "f x ∈ v"
      then have "x ∈ f -' v"
        by (rule vimageI2)
      with <x ∉ f -' v> show False
        by (rule notE)
    qed
    ultimately show "f x ∈ u - v"
      by (rule DiffI)
  end
end
```

```

qed
then show "x ∈ f -' (u - v)"
  by (rule vimageI2)
qed

(* 2ª demostración *)

lemma "f -' u - f -' v ⊆ f -' (u - v)"
proof
  fix x
  assume "x ∈ f -' u - f -' v"
  then have "f x ∈ u - v"
  proof
    assume "x ∈ f -' u"
    assume "x ∉ f -' v"
    have "f x ∈ u" using ⟨x ∈ f -' u⟩ by simp
    moreover
    have "f x ∉ v"
    proof
      assume "f x ∈ v"
      then have "x ∈ f -' v" by simp
      with ⟨x ∉ f -' v⟩ show False by simp
    qed
    ultimately show "f x ∈ u - v" by simp
  qed
  then show "x ∈ f -' (u - v)" by simp
qed

(* 3ª demostración *)

lemma "f -' u - f -' v ⊆ f -' (u - v)"
  by (simp only: vimage_Diff)

(* 4ª demostración *)

lemma "f -' u - f -' v ⊆ f -' (u - v)"
  by auto

end

```

3.18.2. Demostraciones con Lean

```

-----
-- Demostrar que
--  $f^{-1} u \setminus f^{-1} v \subseteq f^{-1} (u \setminus v)$ 
-----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variables u v : set β

-- 1ª demostración
-- =====

example : f ⁻¹ u \ f ⁻¹ v ⊆ f ⁻¹ (u \ v) :=
begin
  intros x hx,
  rw mem_preimage,
  split,
  { rw ← mem_preimage,
    exact hx.1, },
  { dsimp,
    rw ← mem_preimage,
    exact hx.2, },
end

-- 2ª demostración
-- =====

example : f ⁻¹ u \ f ⁻¹ v ⊆ f ⁻¹ (u \ v) :=
begin
  intros x hx,
  split,
  { exact hx.1, },
  { exact hx.2, },
end

-- 3ª demostración
-- =====

example : f ⁻¹ u \ f ⁻¹ v ⊆ f ⁻¹ (u \ v) :=

```

```

begin
  intros x hx,
  exact ⟨hx.1, hx.2⟩,
end

-- 4ª demostración
-- =====

example : f-1 u ∩ f-1 v ⊆ f-1 (u ∩ v) :=
begin
  rintros x ⟨h1, h2⟩,
  exact ⟨h1, h2⟩,
end

-- 5ª demostración
-- =====

example : f-1 u ∩ f-1 v ⊆ f-1 (u ∩ v) :=
subset rfl

-- 6ª demostración
-- =====

example : f-1 u ∩ f-1 v ⊆ f-1 (u ∩ v) :=
by finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.19. Intersección con la imagen

3.19.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   (f-1 s) ∩ v = f-1 (s ∩ f-1 v)
-- ----- *)

theory Interseccion_con_la_imagen
imports Main
begin

(* 1ª demostración *)

```

```

Lemma "(f ' s) n v = f ' (s n f -' v)"
proof (rule equalityI)
  show "(f ' s) n v  $\subseteq$  f ' (s n f -' v)"
  proof (rule subsetI)
    fix y
    assume "y  $\in$  (f ' s) n v"
    then show "y  $\in$  f ' (s n f -' v)"
    proof (rule IntE)
      assume "y  $\in$  v"
      assume "y  $\in$  f ' s"
      then show "y  $\in$  f ' (s n f -' v)"
      proof (rule imageE)
        fix x
        assume "x  $\in$  s"
        assume "y = f x"
        then have "f x  $\in$  v"
          using [x  $\in$  s] by (rule subst)
        then have "x  $\in$  f -' v"
          by (rule vimageI2)
        with [x  $\in$  s] have "x  $\in$  s n f -' v"
          by (rule IntI)
        then have "f x  $\in$  f ' (s n f -' v)"
          by (rule imageI)
        with [y = f x] show "y  $\in$  f ' (s n f -' v)"
          by (rule ssubst)
      qed
    qed
  qed
next
  show "f ' (s n f -' v)  $\subseteq$  (f ' s) n v"
  proof (rule subsetI)
    fix y
    assume "y  $\in$  f ' (s n f -' v)"
    then show "y  $\in$  (f ' s) n v"
    proof (rule imageE)
      fix x
      assume "y = f x"
      assume hx : "x  $\in$  s n f -' v"
      have "y  $\in$  f ' s"
      proof -
        have "x  $\in$  s"
          using hx by (rule IntD1)
        then have "f x  $\in$  f ' s"
          by (rule imageI)
      qed
    qed
  qed

```



```

    with ⟨y = f x⟩ show "y ∈ f ' s"
      by (rule ssubst)
  qed
  moreover
  have "y ∈ v"
  proof -
    have "x ∈ f -' v"
      using hx by (rule IntD2)
    then have "f x ∈ v"
      by (rule vimageD)
    with ⟨y = f x⟩ show "y ∈ v"
      by (rule ssubst)
  qed
  ultimately show "y ∈ (f ' s) ∩ v"
    by (rule IntI)
  qed
qed
qed

(* 2ª demostración *)

lemma "(f ' s) ∩ v = f ' (s ∩ f -' v)"
proof
  show "(f ' s) ∩ v ⊆ f ' (s ∩ f -' v)"
  proof
    fix y
    assume "y ∈ (f ' s) ∩ v"
    then show "y ∈ f ' (s ∩ f -' v)"
    proof
      assume "y ∈ v"
      assume "y ∈ f ' s"
      then show "y ∈ f ' (s ∩ f -' v)"
      proof
        fix x
        assume "x ∈ s"
        assume "y = f x"
        then have "f x ∈ v" using ⟨y ∈ v⟩ by simp
        then have "x ∈ f -' v" by simp
        with ⟨x ∈ s⟩ have "x ∈ s ∩ f -' v" by simp
        then have "f x ∈ f ' (s ∩ f -' v)" by simp
        with ⟨y = f x⟩ show "y ∈ f ' (s ∩ f -' v)" by simp
      qed
    qed
  qed
qed
qed
next

```

```

show "f ' (s n f -' v) ⊆ (f ' s) n v"
proof
  fix y
  assume "y ∈ f ' (s n f -' v)"
  then show "y ∈ (f ' s) n v"
  proof
    fix x
    assume "y = f x"
    assume hx : "x ∈ s n f -' v"
    have "y ∈ f ' s"
    proof -
      have "x ∈ s" using hx by simp
      then have "f x ∈ f ' s" by simp
      with <y = f x> show "y ∈ f ' s" by simp
    qed
    moreover
    have "y ∈ v"
    proof -
      have "x ∈ f -' v" using hx by simp
      then have "f x ∈ v" by simp
      with <y = f x> show "y ∈ v" by simp
    qed
    ultimately show "y ∈ (f ' s) n v" by simp
  qed
qed
qed

```

(* 2ª demostración *)

```

lemma "(f ' s) n v = f ' (s n f -' v)"
proof
  show "(f ' s) n v ⊆ f ' (s n f -' v)"
  proof
    fix y
    assume "y ∈ (f ' s) n v"
    then show "y ∈ f ' (s n f -' v)"
    proof
      assume "y ∈ v"
      assume "y ∈ f ' s"
      then show "y ∈ f ' (s n f -' v)"
      proof
        fix x
        assume "x ∈ s"
        assume "y = f x"
        then show "y ∈ f ' (s n f -' v)"
      qed
    qed
  qed

```

```

        using <x ∈ s> <y ∈ v> by simp
      qed
    qed
  qed
next
  show "f ' (s n f -' v) ⊆ (f ' s) n v"
  proof
    fix y
    assume "y ∈ f ' (s n f -' v)"
    then show "y ∈ (f ' s) n v"
    proof
      fix x
      assume "y = f x"
      assume hx : "x ∈ s n f -' v"
      then have "y ∈ f ' s" using <y = f x> by simp
      moreover
      have "y ∈ v" using hx <y = f x> by simp
      ultimately show "y ∈ (f ' s) n v" by simp
    qed
  qed
qed

(* 4ª demostración *)

lemma "(f ' s) n v = f ' (s n f -' v)"
  by auto

end

```

3.19.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   (f '' s) n v = f '' (s n f -1' v)
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*}
variable f : α → β

```

```

variable s : set  $\alpha$ 
variable v : set  $\beta$ 

-- 1ª demostración
-- =====

example : (f '' s)  $\cap$  v = f '' (s  $\cap$  f  $^{-1}$  v) :=
begin
  ext y,
  split,
  { intro hy,
    cases hy with hyfs yv,
    cases hyfs with x hx,
    cases hx with xs fxy,
    use x,
    split,
    { split,
      { exact xs, },
      { rw mem_preimage,
        rw fxy,
        exact yv, }},
    { exact fxy, }},
  { intro hy,
    cases hy with x hx,
    split,
    { use x,
      split,
      { exact hx.1.1, },
      { exact hx.2, }},
    { cases hx with hx1 fxy,
      rw  $\leftarrow$  fxy,
      rw  $\leftarrow$  mem_preimage,
      exact hx1.2, }},
end

-- 2ª demostración
-- =====

example : (f '' s)  $\cap$  v = f '' (s  $\cap$  f  $^{-1}$  v) :=
begin
  ext y,
  split,
  { rintros ((x, xs, fxy), yv),
    use x,
    split,

```

```

{ split,
  { exact xs, },
  { rw mem_preimage,
    rw fxy,
    exact yv, }},
{ exact fxy, }},
{ rintros (x, (xs, xv), fxy),
split,
{ use [x, xs, fxy], },
{ rw ← fxy,
  rw ← mem_preimage,
  exact xv, }},
end

-- 3ª demostración
-- =====

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
begin
  ext y,
  split,
  { rintros ((x, xs, fxy), yv),
    finish, },
  { rintros (x, (xs, xv), fxy),
    finish, },
end

-- 4ª demostración
-- =====

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by ext ; split ; finish

-- 5ª demostración
-- =====

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
by finish [ext_iff, iff_def]

-- 6ª demostración
-- =====

example : (f '' s) ∩ v = f '' (s ∩ f ⁻¹' v) :=
(image_inter_preimage f s v).symm

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.20. Unión con la imagen

3.20.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--    $f' (s \cup f^{-1} v) \subseteq f' s \cup v$ 
-- ----- *)

theory Union_con_la_imagen
imports Main
begin

(* 1ª demostración *)

lemma "f' (s ∪ f -' v) ⊆ f' s ∪ v"
proof (rule subsetI)
  fix y
  assume "y ∈ f' (s ∪ f -' v)"
  then show "y ∈ f' s ∪ v"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume "x ∈ s ∪ f -' v"
    then show "y ∈ f' s ∪ v"
    proof (rule UnE)
      assume "x ∈ s"
      then have "f x ∈ f' s"
        by (rule imageI)
      with <y = f x> have "y ∈ f' s"
        by (rule ssubst)
      then show "y ∈ f' s ∪ v"
        by (rule UnI1)
    next
      assume "x ∈ f -' v"
      then have "f x ∈ v"
        by (rule vimageD)
      with <y = f x> have "y ∈ v"
        by (rule ssubst)
      then show "y ∈ f' s ∪ v"
        by (rule UnI2)
    end
  end
end
```

```

qed
qed
qed

(* 2ª demostración *)

lemma "f ' (s u f -' v) ⊆ f ' s u v"
proof
  fix y
  assume "y ∈ f ' (s u f -' v)"
  then show "y ∈ f ' s u v"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s u f -' v"
    then show "y ∈ f ' s u v"
    proof
      assume "x ∈ s"
      then have "f x ∈ f ' s" by simp
      with <y = f x> have "y ∈ f ' s" by simp
      then show "y ∈ f ' s u v" by simp
    next
      assume "x ∈ f -' v"
      then have "f x ∈ v" by simp
      with <y = f x> have "y ∈ v" by simp
      then show "y ∈ f ' s u v" by simp
    qed
  qed
qed

(* 3ª demostración *)

lemma "f ' (s u f -' v) ⊆ f ' s u v"
proof
  fix y
  assume "y ∈ f ' (s u f -' v)"
  then show "y ∈ f ' s u v"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s u f -' v"
    then show "y ∈ f ' s u v"
    proof
      assume "x ∈ s"
      then show "y ∈ f ' s u v" by (simp add: <y = f x>)
    
```

```

next
  assume "x ∈ f ⁻¹ v"
  then show "y ∈ f ⁻¹ s ∪ v" by (simp add: <y = f x>)
qed
qed
qed

(* 4ª demostración *)

lemma "f ⁻¹ (s ∪ f ⁻¹ v) ⊆ f ⁻¹ s ∪ v"
proof
  fix y
  assume "y ∈ f ⁻¹ (s ∪ f ⁻¹ v)"
  then show "y ∈ f ⁻¹ s ∪ v"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ s ∪ f ⁻¹ v"
    then show "y ∈ f ⁻¹ s ∪ v" using <y = f x> by blast
  qed
qed

(* 5ª demostración *)

lemma "f ⁻¹ (s ∪ f ⁻¹ u) ⊆ f ⁻¹ s ∪ u"
  by auto

end

```

3.20.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f ⁻¹ (s ∪ f ⁻¹ v) ⊆ f ⁻¹ s ∪ v
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*}
variable f : α → β

```



```

variable s : set  $\alpha$ 
variable v : set  $\beta$ 

-- 1ª demostración
-- =====

example : f '' (s ∪ f  $^{-1}$  v) ⊆ f '' s ∪ v :=
begin
  intros y hy,
  cases hy with x hx,
  cases hx with hx1 fxy,
  cases hx1 with xs xv,
  { left,
    use x,
    split,
    { exact xs, },
    { exact fxy, }},
  { right,
    rw ← fxy,
    exact xv, },
end

-- 2ª demostración
-- =====

example : f '' (s ∪ f  $^{-1}$  v) ⊆ f '' s ∪ v :=
begin
  rintros y (x, xs | xv, fxy),
  { left,
    use [x, xs, fxy], },
  { right,
    rw ← fxy,
    exact xv, },
end

-- 3ª demostración
-- =====

example : f '' (s ∪ f  $^{-1}$  v) ⊆ f '' s ∪ v :=
begin
  rintros y (x, xs | xv, fxy);
  finish,
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.21. Intersección con la imagen inversa

3.21.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--    $s \cap f^{-1} v \subseteq f^{-1} (f' s \cap v)$ 
----- *)

theory Interseccion_con_la_imagen_inversa
imports Main
begin

(* 1ª demostración *)

lemma "s ∩ f-1 v ⊆ f-1 (f' s ∩ v)"
proof (rule subsetI)
  fix x
  assume "x ∈ s ∩ f-1 v"
  have "f x ∈ f' s"
  proof -
    have "x ∈ s"
      using "x ∈ s ∩ f-1 v" by (rule IntD1)
    then show "f x ∈ f' s"
      by (rule imageI)
  qed
  moreover
  have "f x ∈ v"
  proof -
    have "x ∈ f-1 v"
      using "x ∈ s ∩ f-1 v" by (rule IntD2)
    then show "f x ∈ v"
      by (rule vimageD)
  qed
  ultimately have "f x ∈ f' s ∩ v"
    by (rule IntI)
  then show "x ∈ f-1 (f' s ∩ v)"
    by (rule vimageI2)
qed

(* 2ª demostración *)

lemma "s ∩ f-1 v ⊆ f-1 (f' s ∩ v)"
proof (rule subsetI)
```

```

fix x
assume "x ∈ s ∩ f ⁻¹ v"
have "f x ∈ f ' s"
proof -
  have "x ∈ s" using ⟨x ∈ s ∩ f ⁻¹ v⟩ by simp
  then show "f x ∈ f ' s" by simp
qed
moreover
have "f x ∈ v"
proof -
  have "x ∈ f ⁻¹ v" using ⟨x ∈ s ∩ f ⁻¹ v⟩ by simp
  then show "f x ∈ v" by simp
qed
ultimately have "f x ∈ f ' s ∩ v" by simp
then show "x ∈ f ⁻¹ (f ' s ∩ v)" by simp
qed

(* 3ª demostración *)

lemma "s ∩ f ⁻¹ v ⊆ f ⁻¹ (f ' s ∩ v)"
  by auto

end

```

3.21.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   s ∩ f ⁻¹ v ⊆ f ⁻¹ (f ' s ∩ v)
-- -----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variable s : set α
variable v : set β

-- 1ª demostración
-- =====

```

```

example : s  $\sqcap$  f  $^{-1}$  v  $\subseteq$  f  $^{-1}$  (f '' s  $\sqcap$  v) :=
begin
  intros x hx,
  rw mem_preimage,
  split,
  { apply mem_image_of_mem,
    exact hx.1, },
  { rw  $\leftarrow$  mem_preimage,
    exact hx.2, },
end

-- 2ª demostración
-- =====

example : s  $\sqcap$  f  $^{-1}$  v  $\subseteq$  f  $^{-1}$  (f '' s  $\sqcap$  v) :=
begin
  rintros x (xs, xv),
  split,
  { exact mem_image_of_mem f xs, },
  { exact xv, },
end

-- 3ª demostración
-- =====

example : s  $\sqcap$  f  $^{-1}$  v  $\subseteq$  f  $^{-1}$  (f '' s  $\sqcap$  v) :=
begin
  rintros x (xs, xv),
  exact (mem_image_of_mem f xs, xv),
end

-- 4ª demostración
-- =====

example : s  $\sqcap$  f  $^{-1}$  v  $\subseteq$  f  $^{-1}$  (f '' s  $\sqcap$  v) :=
begin
  rintros x (xs, xv),
  show f x  $\in$  f '' s  $\sqcap$  v,
  split,
  { use [x, xs, rfl] },
  { exact xv },
end

-- 5ª demostración
-- =====

```

```
example : s ∩ f ⁻¹ v ⊆ f ⁻¹ (f ' s ∩ v) :=
inter_preimage_subset s v f
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.22. Unión con la imagen inversa

3.22.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--   s ∪ f ⁻¹ v ⊆ f ⁻¹ (f ' s ∪ v)
-- ----- *)

theory Union_con_la_imagen_inversa
imports Main
begin

(* 1ª demostración *)

lemma "s ∪ f ⁻¹ v ⊆ f ⁻¹ (f ' s ∪ v)"
proof (rule subsetI)
  fix x
  assume "x ∈ s ∪ f ⁻¹ v"
  then have "f x ∈ f ' s ∪ v"
  proof (rule UnE)
    assume "x ∈ s"
    then have "f x ∈ f ' s"
      by (rule imageI)
    then show "f x ∈ f ' s ∪ v"
      by (rule UnI1)
  next
    assume "x ∈ f ⁻¹ v"
    then have "f x ∈ v"
      by (rule vimageD)
    then show "f x ∈ f ' s ∪ v"
      by (rule UnI2)
  qed
  then show "x ∈ f ⁻¹ (f ' s ∪ v)"
    by (rule vimageI2)
qed
```

```
(* 2ª demostración *)

lemma "s ∪ f -' v ⊆ f -' (f ' s ∪ v)"
proof
  fix x
  assume "x ∈ s ∪ f -' v"
  then have "f x ∈ f ' s ∪ v"
  proof
    assume "x ∈ s"
    then have "f x ∈ f ' s" by simp
    then show "f x ∈ f ' s ∪ v" by simp
  next
    assume "x ∈ f -' v"
    then have "f x ∈ v" by simp
    then show "f x ∈ f ' s ∪ v" by simp
  qed
  then show "x ∈ f -' (f ' s ∪ v)" by simp
qed
```

```
(* 3ª demostración *)

lemma "s ∪ f -' v ⊆ f -' (f ' s ∪ v)"
proof
  fix x
  assume "x ∈ s ∪ f -' v"
  then have "f x ∈ f ' s ∪ v"
  proof
    assume "x ∈ s"
    then show "f x ∈ f ' s ∪ v" by simp
  next
    assume "x ∈ f -' v"
    then show "f x ∈ f ' s ∪ v" by simp
  qed
  then show "x ∈ f -' (f ' s ∪ v)" by simp
qed
```

```
(* 4ª demostración *)

lemma "s ∪ f -' v ⊆ f -' (f ' s ∪ v)"
  by auto

end
```

3.22.2. Demostraciones con Lean

```

-- -----
--  Demostrar que
--     $s \cup f^{-1}(v) \subseteq f^{-1}(f(s) \cup v)$ 
-- -----

import data.set.basic

open set

variables {α : Type*} {β : Type*}
variable f : α → β
variable s : set α
variable v : set β

-- 1ª demostración
-- =====

example : s ∪ f ⁻¹ v ⊆ f ⁻¹ (f '' s ∪ v) :=
begin
  intros x hx,
  rw mem_preimage,
  cases hx with xs xv,
  { apply mem_union_left,
    apply mem_image_of_mem,
    exact xs, },
  { apply mem_union_right,
    rw mem_preimage,
    exact xv, },
end

-- 2ª demostración
-- =====

example : s ∪ f ⁻¹ v ⊆ f ⁻¹ (f '' s ∪ v) :=
begin
  intros x hx,
  cases hx with xs xv,
  { apply mem_union_left,
    apply mem_image_of_mem,
    exact xs, },
  { apply mem_union_right,
    exact xv, },
end

```

```

-- 3ª demostración
-- =====

example : s ⊆ f-1 v ⊆ f-1 (f '' s ⊆ v) :=
begin
  rintros x (xs | xv),
  { left,
    exact mem_image_of_mem f xs, },
  { right,
    exact xv, },
end

-- 4ª demostración
-- =====

example : s ⊆ f-1 v ⊆ f-1 (f '' s ⊆ v) :=
begin
  rintros x (xs | xv),
  { exact or.inl (mem_image_of_mem f xs), },
  { exact or.inr xv, },
end

-- 5ª demostración
-- =====

example : s ⊆ f-1 v ⊆ f-1 (f '' s ⊆ v) :=
begin
  intros x h,
  exact or.elim h (λ xs, or.inl (mem_image_of_mem f xs)) or.inr,
end

-- 6ª demostración
-- =====

example : s ⊆ f-1 v ⊆ f-1 (f '' s ⊆ v) :=
λ x h, or.elim h (λ xs, or.inl (mem_image_of_mem f xs)) or.inr

-- 7ª demostración
-- =====

example : s ⊆ f-1 v ⊆ f-1 (f '' s ⊆ v) :=
begin
  rintros x (xs | xv),
  { show f x ∈ f '' s ⊆ v,

```



```

    use [x, xs, rfl] },
  { show f x ∈ f '' s ∪ v,
    right,
    apply xv },
end

-- 8ª demostración
-- =====

example : s ∪ f ⁻¹ v ⊆ f ⁻¹ (f '' s ∪ v) :=
union_preimage_subset s v f

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.23. Imagen de la unión general

3.23.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   f ' (∪ i ∈ I. A i) = (∪ i ∈ I. f ' A i)
-- ----- *)

theory Imagen_de_la_union_general
imports Main
begin

(* 1ª demostración *)

lemma "f ' (∪ i ∈ I. A i) = (∪ i ∈ I. f ' A i)"
proof (rule equalityI)
  show "f ' (∪ i ∈ I. A i) ⊆ (∪ i ∈ I. f ' A i)"
  proof (rule subsetI)
    fix y
    assume "y ∈ f ' (∪ i ∈ I. A i)"
    then show "y ∈ (∪ i ∈ I. f ' A i)"
    proof (rule imageE)
      fix x
      assume "y = f x"
      assume "x ∈ (∪ i ∈ I. A i)"
      then have "f x ∈ (∪ i ∈ I. f ' A i)"
      proof (rule UN_E)

```

```

    fix i
    assume "i ∈ I"
    assume "x ∈ A i"
    then have "f x ∈ f ' A i"
      by (rule imageI)
    with <i ∈ I> show "f x ∈ (⋃ i ∈ I. f ' A i)"
      by (rule UN_I)
  qed
  with <y = f x> show "y ∈ (⋃ i ∈ I. f ' A i)"
    by (rule ssubst)
  qed
qed
next
show "(⋃ i ∈ I. f ' A i) ⊆ f ' (⋃ i ∈ I. A i)"
proof (rule subsetI)
  fix y
  assume "y ∈ (⋃ i ∈ I. f ' A i)"
  then show "y ∈ f ' (⋃ i ∈ I. A i)"
  proof (rule UN_E)
    fix i
    assume "i ∈ I"
    assume "y ∈ f ' A i"
    then show "y ∈ f ' (⋃ i ∈ I. A i)"
    proof (rule imageE)
      fix x
      assume "y = f x"
      assume "x ∈ A i"
      with <i ∈ I> have "x ∈ (⋃ i ∈ I. A i)"
        by (rule UN_I)
      then have "f x ∈ f ' (⋃ i ∈ I. A i)"
        by (rule imageI)
      with <y = f x> show "y ∈ f ' (⋃ i ∈ I. A i)"
        by (rule ssubst)
    qed
  qed
qed
qed
qed

(* 2ª demostración *)

lemma "f ' (⋃ i ∈ I. A i) = (⋃ i ∈ I. f ' A i)"
proof
  show "f ' (⋃ i ∈ I. A i) ⊆ (⋃ i ∈ I. f ' A i)"
  proof
    fix y

```

```

assume "y ∈ f ' (⋃ i ∈ I. A i)"
then show "y ∈ (⋃ i ∈ I. f ' A i)"
proof
  fix x
  assume "y = f x"
  assume "x ∈ (⋃ i ∈ I. A i)"
  then have "f x ∈ (⋃ i ∈ I. f ' A i)"
  proof
    fix i
    assume "i ∈ I"
    assume "x ∈ A i"
    then have "f x ∈ f ' A i" by simp
    with ⟨i ∈ I⟩ show "f x ∈ (⋃ i ∈ I. f ' A i)" by (rule UN_I)
  qed
  with ⟨y = f x⟩ show "y ∈ (⋃ i ∈ I. f ' A i)" by simp
qed
qed
next
show "(⋃ i ∈ I. f ' A i) ⊆ f ' (⋃ i ∈ I. A i)"
proof
  fix y
  assume "y ∈ (⋃ i ∈ I. f ' A i)"
  then show "y ∈ f ' (⋃ i ∈ I. A i)"
  proof
    fix i
    assume "i ∈ I"
    assume "y ∈ f ' A i"
    then show "y ∈ f ' (⋃ i ∈ I. A i)"
    proof
      fix x
      assume "y = f x"
      assume "x ∈ A i"
      with ⟨i ∈ I⟩ have "x ∈ (⋃ i ∈ I. A i)" by (rule UN_I)
      then have "f x ∈ f ' (⋃ i ∈ I. A i)" by simp
      with ⟨y = f x⟩ show "y ∈ f ' (⋃ i ∈ I. A i)" by simp
    qed
  qed
qed
qed
(* 3ª demostración *)

lemma "f ' (⋃ i ∈ I. A i) = (⋃ i ∈ I. f ' A i)"
  by (simp only: image_UN)

```

```
(* 4ª demostración *)

lemma "f ' (⋃ i ∈ I. A i) = (⋃ i ∈ I. f ' A i)"
  by auto

end
```

3.23.2. Demostraciones con Lean

```
-- -----
-- Demostrar que
--   f '' (⋃ i, A i) = ⋃ i, f '' A i
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*} {I : Type*}
variable f : α → β
variables A : ℕ → set α

-- 1ª demostración
-- =====

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
begin
  ext y,
  split,
  { intro hy,
    rw mem_image at hy,
    cases hy with x hx,
    cases hx with xUA fxy,
    rw mem_Union at xUA,
    cases xUA with i xAi,
    rw mem_Union,
    use i,
    rw ← fxy,
    apply mem_image_of_mem,
    exact xAi, },
  { intro hy,
    rw mem_Union at hy,
```

```

cases hy with i yAi,
cases yAi with x hx,
cases hx with xAi fxy,
rw ← fxy,
apply mem_image_of_mem,
rw mem_Union,
use i,
exact xAi, },
end

-- 2ª demostración
-- =====

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
begin
  ext y,
  simp,
  split,
  { rintro ⟨x, ⟨i, xAi⟩, fxy⟩,
    use [i, x, xAi, fxy] },
  { rintro ⟨i, x, xAi, fxy⟩,
    exact ⟨x, ⟨i, xAi⟩, fxy⟩ },
end

-- 3ª demostración
-- =====

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
by tidy

-- 4ª demostración
-- =====

example : f '' (⋃ i, A i) = ⋃ i, f '' A i :=
image_Union

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.24. Imagen de la intersección general

3.24.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--  $f'(\bigcap i, A i) \subseteq \bigcap i, f' A i$ 
-- ----- *)

theory Imagen_de_la_interseccion_general
imports Main
begin

(* 1ª demostración *)

lemma "f' ( $\bigcap i \in I. A i$ )  $\subseteq$  ( $\bigcap i \in I. f' A i$ )"
proof (rule subsetI)
  fix y
  assume "y  $\in$  f' ( $\bigcap i \in I. A i$ )"
  then show "y  $\in$  ( $\bigcap i \in I. f' A i$ )"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume xIA : "x  $\in$  ( $\bigcap i \in I. A i$ )"
    have "f x  $\in$  ( $\bigcap i \in I. f' A i$ )"
    proof (rule INT_I)
      fix i
      assume "i  $\in$  I"
      with xIA have "x  $\in$  A i"
      by (rule INT_D)
      then show "f x  $\in$  f' A i"
      by (rule imageI)
    qed
    with <y = f x> show "y  $\in$  ( $\bigcap i \in I. f' A i$ )"
    by (rule ssubst)
  qed
qed

(* 2ª demostración *)

lemma "f' ( $\bigcap i \in I. A i$ )  $\subseteq$  ( $\bigcap i \in I. f' A i$ )"
proof
  fix y
  assume "y  $\in$  f' ( $\bigcap i \in I. A i$ )"
```

```

then show "y ∈ (⋂ i ∈ I. f ' A i)"
proof
  fix x
  assume "y = f x"
  assume xIA : "x ∈ (⋂ i ∈ I. A i)"
  have "f x ∈ (⋂ i ∈ I. f ' A i)"
  proof
    fix i
    assume "i ∈ I"
    with xIA have "x ∈ A i" by simp
    then show "f x ∈ f ' A i" by simp
  qed
  with <y = f x> show "y ∈ (⋂ i ∈ I. f ' A i)" by simp
qed
qed

(* 3ª demostración *)

lemma "f ' (⋂ i ∈ I. A i) ⊆ (⋂ i ∈ I. f ' A i)"
  by auto

end

```

3.24.2. Demostraciones con Lean

```

-- -----
-- Demostrar que
--   f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i
-- -----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*} {I : Type*}
variable f : α → β
variables A : ℕ → set α

-- 1ª demostración
-- =====

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=

```

```

begin
  intros y h,
  apply mem_Inter_of_mem,
  intro i,
  cases h with x hx,
  cases hx with xIA fxy,
  rw ← fxy,
  apply mem_image_of_mem,
  exact mem_Inter.mp xIA i,
end

-- 2ª demostración
-- =====

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
begin
  intros y h,
  apply mem_Inter_of_mem,
  intro i,
  rcases h with ⟨x, xIA, rfl⟩,
  exact mem_image_of_mem f (mem_Inter.mp xIA i),
end

-- 3ª demostración
-- =====

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
begin
  intro y,
  simp,
  intros x xIA fxy i,
  use [x, xIA i, fxy],
end

-- 4ª demostración
-- =====

example : f '' (⋂ i, A i) ⊆ ⋂ i, f '' A i :=
by tidy

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.25. Imagen de la intersección general mediante inyectiva

3.25.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que si f es inyectiva, entonces
--       $(\bigcap i \in I. f' A i) \subseteq f' (\bigcap i \in I. A i)$ 
-- ----- *)

theory Imagen_de_la_interseccion_general_mediante_inyectiva
imports Main
begin

(* 1ª demostración *)

lemma
  assumes "i ∈ I"
          "inj f"
  shows " $(\bigcap i \in I. f' A i) \subseteq f' (\bigcap i \in I. A i)$ "
proof (rule subsetI)
  fix y
  assume "y ∈ ( $\bigcap i \in I. f' A i$ )"
  then have "y ∈ f' A i"
    using <i ∈ I> by (rule INT_D)
  then show "y ∈ f' ( $\bigcap i \in I. A i$ )"
  proof (rule imageE)
    fix x
    assume "y = f x"
    assume "x ∈ A i"
    have "x ∈ ( $\bigcap i \in I. A i$ )"
    proof (rule INT_I)
      fix j
      assume "j ∈ I"
      show "x ∈ A j"
      proof -
        have "y ∈ f' A j"
          using <y ∈ ( $\bigcap i \in I. f' A i$ )> <j ∈ I> by (rule INT_D)
        then show "x ∈ A j"
          proof (rule imageE)
            fix z
            assume "y = f z"
            assume "z ∈ A j"
```

```

    have "f z = f x"
      using <y = f z> <y = f x> by (rule subst)
    with <inj f> have "z = x"
      by (rule injD)
    then show "x ∈ A j"
      using <z ∈ A j> by (rule subst)
  qed
qed
qed
then have "f x ∈ f ' (⋂ i ∈ I. A i)"
  by (rule imageI)
with <y = f x> show "y ∈ f ' (⋂ i ∈ I. A i)"
  by (rule ssubst)
qed
qed

(* 2ª demostración *)

lemma
  assumes "i ∈ I"
    "inj f"
  shows "(⋂ i ∈ I. f ' A i) ⊆ f ' (⋂ i ∈ I. A i)"
proof
  fix y
  assume "y ∈ (⋂ i ∈ I. f ' A i)"
  then have "y ∈ f ' A i" using <i ∈ I> by simp
  then show "y ∈ f ' (⋂ i ∈ I. A i)"
  proof
    fix x
    assume "y = f x"
    assume "x ∈ A i"
    have "x ∈ (⋂ i ∈ I. A i)"
    proof
      fix j
      assume "j ∈ I"
      show "x ∈ A j"
      proof -
        have "y ∈ f ' A j"
          using <y ∈ (⋂ i ∈ I. f ' A i)> <j ∈ I> by simp
        then show "x ∈ A j"
        proof
          fix z
          assume "y = f z"
          assume "z ∈ A j"
          have "f z = f x" using <y = f z> <y = f x> by simp

```

```

      with <inj f> have "z = x" by (rule injD)
      then show "x ∈ A j" using <z ∈ A j> by simp
    qed
  qed
  then have "f x ∈ f ' (⋂ i ∈ I. A i)" by simp
  with <y = f x> show "y ∈ f ' (⋂ i ∈ I. A i)" by simp
qed
(* 3ª demostración *)

lemma
  assumes "i ∈ I"
    "inj f"
  shows "(⋂ i ∈ I. f ' A i) ⊆ f ' (⋂ i ∈ I. A i)"
  using assms
  by (simp add: image_INT)

end

```

3.25.2. Demostraciones con Lean

```

-- -----
-- Demostrar que si f es inyectiva, entonces
--   (⋂ i, f '' A i) ⊆ f '' (⋂ i, A i)
-- -----

import data.set.basic
import tactic

open set function

variables {α : Type*} {β : Type*} {I : Type*}
variable f : α → β
variables A : I → set α

-- 1ª demostración
-- =====

example
  (i : I)
  (inj f : injective f)

```

```

: (⋀ i, f '' A i) ⊆ f '' (⋀ i, A i) :=
begin
  intros y hy,
  rw mem_Inter at hy,
  rcases hy i with ⟨x, xAi, fxy⟩,
  use x,
  split,
  { apply mem_Inter_of_mem,
    intro j,
    rcases hy j with ⟨z, zAj, fzy⟩,
    convert zAj,
    apply injf,
    rw fxy,
    rw ← fzy, },
  { exact fxy, },
end

-- 2ª demostración
-- =====

example
  (i : I)
  (injf : injective f)
  : (⋀ i, f '' A i) ⊆ f '' (⋀ i, A i) :=
begin
  intro y,
  simp,
  intro h,
  rcases h i with ⟨x, xAi, fxy⟩,
  use x,
  split,
  { intro j,
    rcases h j with ⟨z, zAi, fzy⟩,
    have : f x = f z, by rw [fxy, fzy],
    have : x = z, from injf this,
    rw this,
    exact zAi, },
  { exact fxy, },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.26. Imagen inversa de la unión general

3.26.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que
--  $f^{-1}(\bigcup i \in I. B\ i) = (\bigcup i \in I. f^{-1} B\ i)$ 
-- ----- *)

theory Imagen_inversa_de_la_union_general
imports Main
begin

(* 1ª demostración *)

lemma "f -' (⋃ i ∈ I. B i) = (⋃ i ∈ I. f -' B i)"
proof (rule equalityI)
  show "f -' (⋃ i ∈ I. B i) ⊆ (⋃ i ∈ I. f -' B i)"
  proof (rule subsetI)
    fix x
    assume "x ∈ f -' (⋃ i ∈ I. B i)"
    then have "f x ∈ (⋃ i ∈ I. B i)"
      by (rule vimageD)
    then show "x ∈ (⋃ i ∈ I. f -' B i)"
    proof (rule UN_E)
      fix i
      assume "i ∈ I"
      assume "f x ∈ B i"
      then have "x ∈ f -' B i"
        by (rule vimageI2)
      with <i ∈ I> show "x ∈ (⋃ i ∈ I. f -' B i)"
        by (rule UN_I)
    qed
  qed
qed
next
show "(⋃ i ∈ I. f -' B i) ⊆ f -' (⋃ i ∈ I. B i)"
proof (rule subsetI)
  fix x
  assume "x ∈ (⋃ i ∈ I. f -' B i)"
  then show "x ∈ f -' (⋃ i ∈ I. B i)"
  proof (rule UN_E)
    fix i
    assume "i ∈ I"
    assume "x ∈ f -' B i"
```

```

    then have "f x ∈ B i"
      by (rule vimageD)
    with ⟨i ∈ I⟩ have "f x ∈ (⋃ i ∈ I. B i)"
      by (rule UN_I)
    then show "x ∈ f -' (⋃ i ∈ I. B i)"
      by (rule vimageI2)
  qed
qed
qed

(* 2ª demostración *)

lemma "f -' (⋃ i ∈ I. B i) = (⋃ i ∈ I. f -' B i)"
proof
  show "f -' (⋃ i ∈ I. B i) ⊆ (⋃ i ∈ I. f -' B i)"
  proof
    fix x
    assume "x ∈ f -' (⋃ i ∈ I. B i)"
    then have "f x ∈ (⋃ i ∈ I. B i)" by simp
    then show "x ∈ (⋃ i ∈ I. f -' B i)"
    proof
      fix i
      assume "i ∈ I"
      assume "f x ∈ B i"
      then have "x ∈ f -' B i" by simp
      with ⟨i ∈ I⟩ show "x ∈ (⋃ i ∈ I. f -' B i)" by (rule UN_I)
    qed
  qed
next
  show "(⋃ i ∈ I. f -' B i) ⊆ f -' (⋃ i ∈ I. B i)"
  proof
    fix x
    assume "x ∈ (⋃ i ∈ I. f -' B i)"
    then show "x ∈ f -' (⋃ i ∈ I. B i)"
    proof
      fix i
      assume "i ∈ I"
      assume "x ∈ f -' B i"
      then have "f x ∈ B i" by simp
      with ⟨i ∈ I⟩ have "f x ∈ (⋃ i ∈ I. B i)" by (rule UN_I)
      then show "x ∈ f -' (⋃ i ∈ I. B i)" by simp
    qed
  qed
qed

```

```

(* 3ª demostración *)

lemma "f ⁻¹' (⋃ i ∈ I. B i) = (⋃ i ∈ I. f ⁻¹' B i)"
  by (simp only: vimage_UN)

(* 4ª demostración *)

lemma "f ⁻¹' (⋃ i ∈ I. B i) = (⋃ i ∈ I. f ⁻¹' B i)"
  by auto

end

```

3.26.2. Demostraciones con Lean

```

-----
-- Demostrar que
--   f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i)
-----

import data.set.basic
import tactic

open set

variables {α : Type*} {β : Type*} {I : Type*}
variable f : α → β
variables B : I → set β

-- 1ª demostración
-- =====

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
begin
  ext x,
  split,
  { intro hx,
    rw mem_preimage at hx,
    rw mem_Union at hx,
    cases hx with i fxBi,
    rw mem_Union,
    use i,
    apply mem_preimage.mpr,
    exact fxBi, },

```

```

{ intro hx,
  rw mem_preimage,
  rw mem_Union,
  rw mem_Union at hx,
  cases hx with i xBi,
  use i,
  rw mem_preimage at xBi,
  exact xBi, },
end

-- 2ª demostración
-- =====

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
preimage_Union

-- 3ª demostración
-- =====

example : f ⁻¹' (⋃ i, B i) = ⋃ i, f ⁻¹' (B i) :=
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.27. Imagen inversa de la intersección general

3.27.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que
--   f ⁻' (⋂ i ∈ I. B i) = (⋂ i ∈ I. f ⁻' B i)
-- ----- *)

theory Imagen_inversa_de_la_interseccion_general
imports Main
begin

(* 1ª demostración *)

lemma "f ⁻' (⋂ i ∈ I. B i) = (⋂ i ∈ I. f ⁻' B i)"
proof (rule equalityI)

```



```

show "f -' ( $\bigcap i \in I. B i$ )  $\subseteq$  ( $\bigcap i \in I. f -' B i$ )"
proof (rule subsetI)
  fix x
  assume "x  $\in$  f -' ( $\bigcap i \in I. B i$ )"
  show "x  $\in$  ( $\bigcap i \in I. f -' B i$ )"
  proof (rule INT_I)
    fix i
    assume "i  $\in$  I"
    have "f x  $\in$  ( $\bigcap i \in I. B i$ )"
      using <x  $\in$  f -' ( $\bigcap i \in I. B i$ )> by (rule vimageD)
    then have "f x  $\in$  B i"
      using <i  $\in$  I> by (rule INT_D)
    then show "x  $\in$  f -' B i"
      by (rule vimageI2)
  qed
qed
next
show "( $\bigcap i \in I. f -' B i$ )  $\subseteq$  f -' ( $\bigcap i \in I. B i$ )"
proof (rule subsetI)
  fix x
  assume "x  $\in$  ( $\bigcap i \in I. f -' B i$ )"
  have "f x  $\in$  ( $\bigcap i \in I. B i$ )"
  proof (rule INT_I)
    fix i
    assume "i  $\in$  I"
    with <x  $\in$  ( $\bigcap i \in I. f -' B i$ )> have "x  $\in$  f -' B i"
      by (rule INT_D)
    then show "f x  $\in$  B i"
      by (rule vimageD)
  qed
  then show "x  $\in$  f -' ( $\bigcap i \in I. B i$ )"
    by (rule vimageI2)
qed
qed

(* 2ª demostración *)

lemma "f -' ( $\bigcap i \in I. B i$ ) = ( $\bigcap i \in I. f -' B i$ )"
proof
  show "f -' ( $\bigcap i \in I. B i$ )  $\subseteq$  ( $\bigcap i \in I. f -' B i$ )"
  proof (rule subsetI)
    fix x
    assume hx : "x  $\in$  f -' ( $\bigcap i \in I. B i$ )"
    show "x  $\in$  ( $\bigcap i \in I. f -' B i$ )"
    proof

```

```

    fix i
    assume "i ∈ I"
    have "f x ∈ (⋂ i ∈ I. B i)" using hx by simp
    then have "f x ∈ B i" using <i ∈ I> by simp
    then show "x ∈ f -' B i" by simp
  qed
qed
next
show "(⋂ i ∈ I. f -' B i) ⊆ f -' (⋂ i ∈ I. B i)"
proof
  fix x
  assume "x ∈ (⋂ i ∈ I. f -' B i)"
  have "f x ∈ (⋂ i ∈ I. B i)"
  proof
    fix i
    assume "i ∈ I"
    with <x ∈ (⋂ i ∈ I. f -' B i)> have "x ∈ f -' B i" by simp
    then show "f x ∈ B i" by simp
  qed
  then show "x ∈ f -' (⋂ i ∈ I. B i)" by simp
qed
qed

(* 3 demostración *)

lemma "f -' (⋂ i ∈ I. B i) = (⋂ i ∈ I. f -' B i)"
  by (simp only: vimage_INT)

(* 4ª demostración *)

lemma "f -' (⋂ i ∈ I. B i) = (⋂ i ∈ I. f -' B i)"
  by auto

end

```

3.27.2. Demostraciones con Lean

```

-- Demostrar que
--   f -' (⋂ i, B i) = ⋂ i, f -' (B i)

```

```

import data.set.basic

```

```

import tactic

open set

variables {α : Type*} {β : Type*} {I : Type*}
variable f : α → β
variables B : I → set β

-- 1ª demostración
-- =====

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
begin
  ext x,
  split,
  { intro hx,
    apply mem_Inter_of_mem,
    intro i,
    rw mem_preimage,
    rw mem_preimage at hx,
    rw mem_Inter at hx,
    exact hx i, },
  { intro hx,
    rw mem_preimage,
    rw mem_Inter,
    intro i,
    rw ← mem_preimage,
    rw mem_Inter at hx,
    exact hx i, },
end

-- 2ª demostración
-- =====

example : f ⁻¹' (⋂ i, B i) = ⋂ i, f ⁻¹' (B i) :=
begin
  ext x,
  calc (x ∈ f ⁻¹' ⋂ (i : I), B i)
    ↔ f x ∈ ⋂ (i : I), B i      : mem_preimage
  ... ↔ (∀ i : I, f x ∈ B i)      : mem_Inter
  ... ↔ (∀ i : I, x ∈ f ⁻¹' B i)   : iff_of_eq rfl
  ... ↔ x ∈ ⋂ (i : I), f ⁻¹' B i   : mem_Inter.symm,
end

```

```

-- 3ª demostración
-- =====

example : f -1 (⋂ i, B i) = ⋂ i, f -1 (B i) :=
begin
  ext x,
  simp,
end

-- 4ª demostración
-- =====

example : f -1 (⋂ i, B i) = ⋂ i, f -1 (B i) :=
by { ext, simp }

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.28. Teorema de Cantor

3.28.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar el teorema de Cantor:
--    $\forall f : \alpha \rightarrow \text{set } \alpha, \neg \text{surjective } f$ 
-- ----- *)

theory Teorema_de_Cantor
imports Main
begin

(* 1ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
proof (rule notI)
  assume "surj f"
  let ?S = "{i. i ∉ f i}"
  have "∃ j. ?S = f j"
    using <surj f> by (simp only: surjD)
  then obtain j where "?S = f j"
    by (rule exE)

```

```

show False
proof (cases "j ∈ ?S")
  assume "j ∈ ?S"
  then have "j ∉ f j"
    by (rule CollectD)
  moreover
  have "j ∈ f j"
    using <?S = f j> <j ∈ ?S> by (rule subst)
  ultimately show False
    by (rule notE)
next
  assume "j ∉ ?S"
  with <?S = f j> have "j ∉ f j"
    by (rule subst)
  then have "j ∈ ?S"
    by (rule CollectI)
  with <j ∉ ?S> show False
    by (rule notE)
qed
qed

(* 2ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
proof (rule notI)
  assume "surj f"
  let ?S = "{i. i ∉ f i}"
  have "∃ j. ?S = f j"
    using <surj f> by (simp only: surjD)
  then obtain j where "?S = f j"
    by (rule exE)
  have "j ∉ ?S"
  proof (rule notI)
    assume "j ∈ ?S"
    then have "j ∉ f j"
      by (rule CollectD)
    with <?S = f j> have "j ∉ ?S"
      by (rule ssubst)
    then show False
      using <j ∈ ?S> by (rule notE)
  qed
  moreover
  have "j ∈ ?S"

```

```

proof (rule CollectI)
  show "j ∉ f j"
  proof (rule notI)
    assume "j ∈ f j"
    with <?S = f j> have "j ∈ ?S"
      by (rule ssubst)
    then have "j ∉ f j"
      by (rule CollectD)
    then show False
      using <j ∈ f j> by (rule notE)
  qed
qed
ultimately show False
  by (rule notE)
qed

(* 3ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
proof
  assume "surj f"
  let ?S = "{i. i ∉ f i}"
  have "∃ j. ?S = f j" using <surj f> by (simp only: surjD)
  then obtain j where "?S = f j" by (rule exE)
  have "j ∉ ?S"
  proof
    assume "j ∈ ?S"
    then have "j ∉ f j" by simp
    with <?S = f j> have "j ∉ ?S" by simp
    then show False using <j ∈ ?S> by simp
  qed
  moreover
  have "j ∈ ?S"
  proof
    show "j ∉ f j"
    proof
      assume "j ∈ f j"
      with <?S = f j> have "j ∈ ?S" by simp
      then have "j ∉ f j" by simp
      then show False using <j ∈ f j> by simp
    qed
  qed
qed

```

```

ultimately show False by simp
qed

(* 4ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
proof (rule notI)
  assume "surj f"
  let ?S = "{i. i ∉ f i}"
  have "∃ j. ?S = f j"
    using ⟨surj f⟩ by (simp only: surjD)
  then obtain j where "?S = f j"
    by (rule exE)
  have "j ∈ ?S = (j ∉ f j)"
    by (rule mem_Collect_eq)
  also have "... = (j ∉ ?S)"
    by (simp only: ⟨?S = f j⟩)
  finally show False
    by (simp only: simp_thms(10))
qed

(* 5ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
proof
  assume "surj f"
  let ?S = "{i. i ∉ f i}"
  have "∃ j. ?S = f j" using ⟨surj f⟩ by (simp only: surjD)
  then obtain j where "?S = f j" by (rule exE)
  have "j ∈ ?S = (j ∉ f j)" by simp
  also have "... = (j ∉ ?S)" using ⟨?S = f j⟩ by simp
  finally show False by simp
qed

(* 6ª demostración *)

theorem
  fixes f :: "'α ⇒ 'α set"
  shows "¬ surj f"
  unfolding surj_def
  by best

```

```
end
```

3.28.2. Demostraciones con Lean

```
-----
-- Demostrar el teorema de Cantor:
--    $\forall f : \alpha \rightarrow \text{set } \alpha, \neg \text{surjective } f$ 
-----

import data.set.basic
open function

variables { $\alpha$  : Type}

-- 1ª demostración
-- =====

example :  $\forall f : \alpha \rightarrow \text{set } \alpha, \neg \text{surjective } f :=$ 
begin
  intros f surjf,
  let S := {i | i  $\notin$  f i},
  unfold surjective at surjf,
  cases surjf S with j fjS,
  by_cases j  $\in$  S,
  { apply absurd _ h,
    rw fjS,
    exact h, },
  { apply h,
    rw  $\leftarrow$  fjS at h,
    exact h, },
end

-- 2ª demostración
-- =====

example :  $\forall f : \alpha \rightarrow \text{set } \alpha, \neg \text{surjective } f :=$ 
begin
  intros f surjf,
  let S := {i | i  $\notin$  f i},
  cases surjf S with j fjS,
  by_cases j  $\in$  S,
  { apply absurd _ h,
    rwa fjS, },
```


3.29. En los monoides, los inversos a la izquierda y a la derecha son iguales

```
{ apply h,
  rwa ← f j S at h, },
end

-- 3ª demostración
-- =====

example : ∀ f : α → set α, ¬ surjective f :=
begin
  intros f surjf,
  let S := {i | i ∉ f i},
  cases surjf S with j fjS,
  have h : (j ∈ S) = (j ∉ S), from
    calc (j ∈ S)
      = (j ∉ f j) : set.mem_set_of_eq
      ... = (j ∉ S) : congr_arg not (congr_arg (has_mem.mem j) fjS),
  exact false_of_a_eq_not_a h,
end

-- 4ª demostración
-- =====

example : ∀ f : α → set α, ¬ surjective f :=
cantor_surjective
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.29. En los monoides, los inversos a la izquierda y a la derecha son iguales

3.29.1. Demostraciones con Isabelle/HOL

```
(* En_los_monoides_los_inversos_a_la_izquierda_y_a_la_derecha_son_iguales.thy
-- En los monoides, los inversos a la izquierda y a la derecha son iguales.
-- José A. Alonso Jiménez
-- Sevilla, 29 de junio de 2021
----- *)

(* -----
-- Un [monoid](https://en.wikipedia.org/wiki/Monoid) es un conjunto
-- junto con una operación binaria que es asociativa y tiene elemento
```

```

-- neutro.
--
-- En Lean, está definida la clase de los monoides (como 'monoid') y sus
-- propiedades características son
--   assoc      : (a * b) * c = a * (b * c)
--   left_neutral : 1 * a = a
--   right_neutral : a * 1 = a
--
-- Demostrar que si M es un monide, a ∈ M, b es un inverso de a por la
-- izquierda y c es un inverso de a por la derecha, entonces b = c.
-- ----- *)

theory En_los_monoides_los_inversos_a_la_izquierda_y_a_la_derecha_son_iguales
imports Main
begin

context monoid
begin

(* 1ª demostración *)

lemma
  assumes "b |* a = 1"
         "a |* c = 1"
  shows   "b = c"
proof -
  have    "b = b |* 1"          by (simp only: right_neutral)
  also have "... = b |* (a |* c)" by (simp only: <a |* c = 1>)
  also have "... = (b |* a) |* c" by (simp only: assoc)
  also have "... = 1 |* c"       by (simp only: <b |* a = 1>)
  also have "... = c"           by (simp only: left_neutral)
  finally show "b = c"         by this
qed

(* 2ª demostración *)

lemma
  assumes "b |* a = 1"
         "a |* c = 1"
  shows   "b = c"
proof -
  have    "b = b |* 1"          by simp
  also have "... = b |* (a |* c)" using <a |* c = 1> by simp
  also have "... = (b |* a) |* c" by (simp add: assoc)
  also have "... = 1 |* c"       using <b |* a = 1> by simp

```

3.29. En los monoides, los inversos a la izquierda y a la derecha son iguales

```
also have "... = c"      by simp
finally show "b = c"     by this
qed

(* 3ª demostración *)

lemma
  assumes "b |* a = |1"
         "a |* c = |1"
  shows   "b = c"
  using  assms
  by (metis assoc left_neutral right_neutral)

end
```

3.29.2. Demostraciones con Lean

```
-- En los monoides los inversos a la izquierda y a la derecha son iguales.lean
-- En los monoides, los inversos a la izquierda y a la derecha son iguales.
-- José A. Alonso Jiménez
-- Sevilla, 29 de junio de 2021
-- -----

-- -----
-- Un [monoid](https://en.wikipedia.org/wiki/Monoid) es un conjunto
-- junto con una operación binaria que es asociativa y tiene elemento
-- neutro.
--
-- En Lean, está definida la clase de los monoides (como 'monoid') y sus
-- propiedades características son
--   mul_assoc : (a * b) * c = a * (b * c)
--   one_mul   : 1 * a = a
--   mul_one   : a * 1 = a
--
-- Demostrar que si M es un monide, a ∈ M, b es un inverso de a por la
-- izquierda y c es un inverso de a por la derecha, entonces b = c.
-- -----

import algebra.group.defs

variables {M : Type} [monoid M]
variables {a b c : M}
```

```

-- 1ª demostración
-- =====

example
  (hba : b * a = 1)
  (hac : a * c = 1)
  : b = c :=
begin
  rw ←one_mul c,
  rw ←hba,
  rw mul_assoc,
  rw hac,
  rw mul_one b,
end

-- 2ª demostración
-- =====

example
  (hba : b * a = 1)
  (hac : a * c = 1)
  : b = c :=
by rw [←one_mul c, ←hba, mul_assoc, hac, mul_one b]

-- 3ª demostración
-- =====

example
  (hba : b * a = 1)
  (hac : a * c = 1)
  : b = c :=
calc
  b   = b * 1           : (mul_one b).symm
  ... = b * (a * c)     : congr_arg (λ x, b * x) hac.symm
  ... = (b * a) * c     : (mul_assoc b a c).symm
  ... = 1 * c           : congr_arg (λ x, x * c) hba
  ... = c               : one_mul c

-- 4ª demostración
-- =====

example
  (hba : b * a = 1)
  (hac : a * c = 1)
  : b = c :=
calc
  b   = b * 1           : by finish

```

```

... = b * (a * c) : by finish
... = (b * a) * c : (mul_assoc b a c).symm
... = 1 * c       : by finish
... = c           : by finish

-- 5ª demostración
-- =====

example
  (hba : b * a = 1)
  (hac : a * c = 1)
  : b = c :=
left_inv_eq_right_inv hba hac

```

"7-"Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

3.30. Producto_de_potencias_de_la_misma_base_en_r

3.30.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponente naturales. En Isabelle/HOL la potencia  $x^n$  se
-- caracteriza por los siguientes lemas:
--   power_0    :  $x^0 = 1$ 
--   power_Suc  :  $x^{(Suc\ n)} = x * x^n$ 
--
-- Demostrar que
--    $x^{(m + n)} = x^m * x^n$ 
-- ----- *)

theory Producto_de_potencias_de_la_misma_base_en_monoides
imports Main
begin

context monoid_mult
begin

(* 1ª demostración *)

lemma "x ^ (m + n) = x ^ m * x ^ n"
proof (induct m)

```

```

have "x ^ (0 + n) = x ^ n"                by (simp only: add_0)
also have "... = 1 * x ^ n"                by (simp only: mult_1_left)
also have "... = x ^ 0 * x ^ n"           by (simp only: power_0)
finally show "x ^ (0 + n) = x ^ 0 * x ^ n"
  by this
next
fix m
assume HI : "x ^ (m + n) = x ^ m * x ^ n"
have "x ^ (Suc m + n) = x ^ Suc (m + n)"  by (simp only: add_Suc)
also have "... = x * x ^ (m + n)"          by (simp only: power_Suc)
also have "... = x * (x ^ m * x ^ n)"      by (simp only: HI)
also have "... = (x * x ^ m) * x ^ n"      by (simp only: mult_assoc)
also have "... = x ^ Suc m * x ^ n"        by (simp only: power_Suc)
finally show "x ^ (Suc m + n) = x ^ Suc m * x ^ n"
  by this
qed

```

(* 2^a demostración *)

```

Lemma "x ^ (m + n) = x ^ m * x ^ n"
proof (induct m)
  have "x ^ (0 + n) = x ^ n"                by simp
  also have "... = 1 * x ^ n"                by simp
  also have "... = x ^ 0 * x ^ n"           by simp
  finally show "x ^ (0 + n) = x ^ 0 * x ^ n"
    by this
next
fix m
assume HI : "x ^ (m + n) = x ^ m * x ^ n"
have "x ^ (Suc m + n) = x ^ Suc (m + n)"    by simp
also have "... = x * x ^ (m + n)"           by simp
also have "... = x * (x ^ m * x ^ n)"       using HI by simp
also have "... = (x * x ^ m) * x ^ n"       by (simp add: mult_assoc)
also have "... = x ^ Suc m * x ^ n"         by simp
finally show "x ^ (Suc m + n) = x ^ Suc m * x ^ n"
  by this
qed

```

(* 3^a demostración *)

```

Lemma "x ^ (m + n) = x ^ m * x ^ n"
proof (induct m)
  case 0
  then show ?case
    by simp

```

```

next
  case (Suc m)
  then show ?case
  by (simp add: algebra_simps)
qed

(* 4ª demostración *)

lemma "x ^ (m + n) = x ^ m * x ^ n"
  by (induct m) (simp_all add: algebra_simps)

(* 5ª demostración *)

lemma "x ^ (m + n) = x ^ m * x ^ n"
  by (simp only: power_add)

end

end

```

3.30.2. Demostraciones con Lean

```

-----
-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponentes naturales. En Lean la potencia  $x^n$  se
-- se caracteriza por los siguientes lemas:
--   pow_zero :  $x^0 = 1$ 
--   pow_succ :  $x^{(succ\ n)} = x * x^n$ 
--
-- Demostrar que
--    $x^{(m + n)} = x^m * x^n$ 
-----

import algebra.group_power.basic
open monoid nat

variables {M : Type} [monoid M]
variable x : M
variables (m n : ℕ)

-- Para que no use la notación con puntos
set_option pp.structure_projections false

```

```

-- 1ª demostración
-- =====

example :
  x^(m + n) = x^m * x^n :=
begin
  induction m with m HI,
  { calc x^(0 + n)
    = x^n : congr_arg ((^) x) (nat.zero_add n)
    ... = 1 * x^n : (monoid.one_mul (x^n)).symm
    ... = x^0 * x^n : congr_arg (* (x^n)) (pow_zero x).symm, },
  { calc x^(succ m + n)
    = x^(succ (m + n)) : congr_arg ((^) x) (succ_add m n)
    ... = x * x^(m + n) : pow_succ x (m + n)
    ... = x * (x^m * x^n) : congr_arg ((* x) HI)
    ... = (x * x^m) * x^n : (monoid.mul_assoc x (x^m) (x^n)).symm
    ... = x^(succ m) * x^n : congr_arg (* x^n) (pow_succ x m).symm, },
end

-- 2ª demostración
-- =====

example :
  x^(m + n) = x^m * x^n :=
begin
  induction m with m HI,
  { calc x^(0 + n)
    = x^n : by simp only [nat.zero_add]
    ... = 1 * x^n : by simp only [monoid.one_mul]
    ... = x^0 * x^n : by simp [pow_zero] },
  { calc x^(succ m + n)
    = x^(succ (m + n)) : by simp only [succ_add]
    ... = x * x^(m + n) : by simp only [pow_succ]
    ... = x * (x^m * x^n) : by simp only [HI]
    ... = (x * x^m) * x^n : (monoid.mul_assoc x (x^m) (x^n)).symm
    ... = x^(succ m) * x^n : by simp only [pow_succ], },
end

-- 3ª demostración
-- =====

example :

```



```

x^(m + n) = x^m * x^n :=
begin
  induction m with m HI,
  { calc x^(0 + n)
    = x^n : by simp [nat.zero_add]
    ... = 1 * x^n : by simp
    ... = x^0 * x^n : by simp, },
  { calc x^(succ m + n)
    = x^(succ (m + n)) : by simp [succ_add]
    ... = x * x^(m + n) : by simp [pow_succ]
    ... = x * (x^m * x^n) : by simp [HI]
    ... = (x * x^m) * x^n : (monoid.mul_assoc x (x^m) (x^n)).symm
    ... = x^(succ m) * x^n : by simp [pow_succ], },
end

-- 4ª demostración
-- =====

example :
  x^(m + n) = x^m * x^n :=
begin
  induction m with m HI,
  { show x^(0 + n) = x^0 * x^n,
    by simp [nat.zero_add] },
  { show x^(succ m + n) = x^(succ m) * x^n,
    by finish [succ_add,
               HI,
               monoid.mul_assoc,
               pow_succ], },
end

-- 5ª demostración
-- =====

example :
  x^(m + n) = x^m * x^n :=
pow_add x m n

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

Capítulo 4

Ejercicios de julio de 2021

4.1. Equivalencia de inversos iguales al neutro

4.1.1. Demostraciones con Isabelle/HOL

```
(* -----  
-- Sea M un monoide y  $a, b \in M$  tales que  $a * b = 1$ . Demostrar que  $a = 1$   
-- si y sólo si  $b = 1$ .  
----- *)  
  
theory Equivalencia_de_inversos_iguales_al_neutro  
imports Main  
begin  
  
context monoid  
begin  
  
(* 1ª demostración *)  
  
lemma  
  assumes "a |* b = |1"  
  shows "a = |1  $\leftrightarrow$  b = |1"  
proof (rule iffI)  
  assume "a = |1"  
  have "b = |1 |* b" by (simp only: left_neutral)  
  also have "... = a |* b" by (simp only: <a = |1>)  
  also have "... = |1" by (simp only: <a |* b = |1>)  
  finally show "b = |1" by this  
next  
  assume "b = |1"  
  have "a = a |* |1" by (simp only: right_neutral)
```

```

also have "... = a |* b" by (simp only: <b = 1>)
also have "... = 1"      by (simp only: <a |* b = 1>)
finally show "a = 1"     by this
qed

(* 2ª demostración *)

lemma
  assumes "a |* b = 1"
  shows   "a = 1 ↔ b = 1"
proof
  assume "a = 1"
  have "b = 1 |* b"      by simp
  also have "... = a |* b" using <a = 1> by simp
  also have "... = 1"    using <a |* b = 1> by simp
  finally show "b = 1" .
next
  assume "b = 1"
  have "a = a |* 1"      by simp
  also have "... = a |* b" using <b = 1> by simp
  also have "... = 1"    using <a |* b = 1> by simp
  finally show "a = 1" .
qed

(* 3ª demostración *)

lemma
  assumes "a |* b = 1"
  shows   "a = 1 ↔ b = 1"
  by (metis assms left_neutral right_neutral)

end

end

```

4.1.2. Demostraciones con Lean

```

-- Sea M un monoide y a, b ∈ M tales que a * b = 1. Demostrar que a = 1
-- si y sólo si b = 1.

```

```

import algebra.group.basic

variables {M : Type} [monoid M]
variables {a b : M}

-- 1ª demostración
-- =====

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
begin
  split,
  { intro a1,
    rw a1 at h,
    rw one_mul at h,
    exact h, },
  { intro b1,
    rw b1 at h,
    rw mul_one at h,
    exact h, },
end

-- 2ª demostración
-- =====

example
  (h : a * b = 1)
  : a = 1 ↔ b = 1 :=
begin
  split,
  { intro a1,
    calc b = 1 * b : (one_mul b).symm
      ... = a * b : congr_arg (* b) a1.symm
      ... = 1      : h, },
  { intro b1,
    calc a = a * 1 : (mul_one a).symm
      ... = a * b : congr_arg ((* a) b1.symm
      ... = 1      : h, },
end

-- 3ª demostración
-- =====

example

```

```

(h : a * b = 1)
: a = 1 ↔ b = 1 :=
begin
  split,
  { rintro rfl,
    simpa using h, },
  { rintro rfl,
    simpa using h, },
end

-- 4ª demostración
-- =====

example
(h : a * b = 1)
: a = 1 ↔ b = 1 :=
by split ; { rintro rfl, simpa using h }

-- 5ª demostración
-- =====

example
(h : a * b = 1)
: a = 1 ↔ b = 1 :=
by split ; finish

-- 6ª demostración
-- =====

example
(h : a * b = 1)
: a = 1 ↔ b = 1 :=
by finish [iff_def]

-- 7ª demostración
-- =====

example
(h : a * b = 1)
: a = 1 ↔ b = 1 :=
eq_one_iff_eq_one_of_mul_eq_one h

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.2. Unicidad de inversos en monoides

4.2.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que en los monoides conmutativos, si un elemento tiene un
-- inverso por la derecha, dicho inverso es único.
*----- *)

theory Unicidad_de_inversos_en_monoides
imports Main
begin

context comm_monoid
begin

(* 1ª demostración *)

lemma
  assumes "x |* y = |1"
          "x |* z = |1"
  shows "y = z"
proof -
  have "y = |1 |* y" by (simp only: left_neutral)
  also have "... = (x |* z) |* y" by (simp only: <x |* z = |1>)
  also have "... = (z |* x) |* y" by (simp only: commute)
  also have "... = z |* (x |* y)" by (simp only: assoc)
  also have "... = z |* |1" by (simp only: <x |* y = |1>)
  also have "... = z" by (simp only: right_neutral)
  finally show "y = z" by this
qed

(* 2ª demostración *)

lemma
  assumes "x |* y = |1"
          "x |* z = |1"
  shows "y = z"
proof -
  have "y = |1 |* y" by simp
  also have "... = (x |* z) |* y" using assms(2) by simp
  also have "... = (z |* x) |* y" by simp
  also have "... = z |* (x |* y)" by simp
  also have "... = z |* |1" using assms(1) by simp
```

```

also have "... = z"          by simp
finally show "y = z"         by this
qed

(* 3ª demostración *)

lemma
  assumes "x |* y = 1"
         "x |* z = 1"
  shows "y = z"
  using assms
  by auto

end

end

```

4.2.2. Demostraciones con Lean

```

-- -----
-- Demostrar que en los monoides conmutativos, si un elemento tiene un
-- inverso por la derecha, dicho inverso es único.
-- -----

import algebra.group.basic
import tactic

variables {M : Type} [comm_monoid M]
variables {x y z : M}

-- 1ª demostración
-- =====

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y      : (one_mul y).symm
    ... = (x * z) * y : congr_arg (* y) hz.symm
    ... = (z * x) * y : congr_arg (* y) (mul_comm x z)
    ... = z * (x * y) : mul_assoc z x y
    ... = z * 1       : congr_arg ((* z) hy
    ... = z           : mul_one z

```



```

-- 2ª demostración
-- =====

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y      : by simp only [one_mul]
  ... = (x * z) * y : by simp only [hz]
  ... = (z * x) * y : by simp only [mul_comm]
  ... = z * (x * y) : by simp only [mul_assoc]
  ... = z * 1       : by simp only [hy]
  ... = z           : by simp only [mul_one]

-- 3ª demostración
-- =====

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
calc y = 1 * y      : by simp
  ... = (x * z) * y : by simp [hz]
  ... = (z * x) * y : by simp [mul_comm]
  ... = z * (x * y) : by simp [mul_assoc]
  ... = z * 1       : by simp [hy]
  ... = z           : by simp

-- 4ª demostración
-- =====

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
begin
  apply left_inv_eq_right_inv _ hz,
  rw mul_comm,
  exact hy,
end

-- 5ª demostración
-- =====

```

```

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
left_inv_eq_right_inv (trans (mul_comm _ _) hy) hz

-- 6ª demostración
-- =====

example
  (hy : x * y = 1)
  (hz : x * z = 1)
  : y = z :=
inv_unique hy hz

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.3. Caracterización de producto igual al primer factor

4.3.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Un monoide cancelativo por la izquierda es un monoide
-- https://bit.ly/3h4notA M que cumple la propiedad cancelativa por la
-- izquierda; es decir, para todo  $a, b \in M$ 
--  $a * b = a * c \leftrightarrow b = c$ .
--
-- En Isabelle/HOL la clase de los monoides conmutativos cancelativos
-- por la izquierda es
-- cancel_comm_monoid_add y la propiedad cancelativa por la izquierda es
-- add_left_cancel :  $a + b = a + c \leftrightarrow b = c$ 
--
-- Demostrar que si  $M$  es un monoide cancelativo por la izquierda y
--  $a, b \in M$ , entonces
--  $a + b = a \leftrightarrow b = 0$ 
----- *)

theory Caracterizacion_de_producto_igual_al_primer_factor
imports Main
begin

```

```

context cancel_comm_monoid_add
begin

(* 1ª demostración *)

lemma "a + b = a ↔ b = 0"
proof (rule iffI)
  assume "a + b = a"
  then have "a + b = a + 0" by (simp only: add_0_right)
  then show "b = 0" by (simp only: add_left_cancel)
next
  assume "b = 0"
  have "a + 0 = a" by (simp only: add_0_right)
  with <b = 0> show "a + b = a" by (rule ssubst)
qed

(* 2ª demostración *)

lemma "a + b = a ↔ b = 0"
proof
  assume "a + b = a"
  then have "a + b = a + 0" by simp
  then show "b = 0" by simp
next
  assume "b = 0"
  have "a + 0 = a" by simp
  then show "a + b = a" using <b = 0> by simp
qed

(* 3ª demostración *)

lemma "a + b = a ↔ b = 0"
proof -
  have "(a + b = a) ↔ (a + b = a + 0)" by (simp only: add_0_right)
  also have "... ↔ (b = 0)" by (simp only: add_left_cancel)
  finally show "a + b = a ↔ b = 0" by this
qed

(* 4ª demostración *)

lemma "a + b = a ↔ b = 0"
proof -
  have "(a + b = a) ↔ (a + b = a + 0)" by simp
  also have "... ↔ (b = 0)" by simp
  finally show "a + b = a ↔ b = 0" .

```

```

qed

(* 5ª demostración *)

lemma "a + b = a ↔ b = 0"
  by (simp only: add_cancel_left_right)

(* 6ª demostración *)

lemma "a + b = a ↔ b = 0"
  by auto

end

end

```

4.3.2. Demostraciones con Lean

```

-----
-- Un monoide cancelativo por la izquierda es un monoide
-- https://bit.ly/3h4notA M que cumple la propiedad cancelativa por la
-- izquierda; es decir, para todo  $a, b \in M$ 
--  $a * b = a * c \leftrightarrow b = c$ .
--
-- En Lean la clase de los monoides cancelativos por la izquierda es
-- left_cancel_monoid y la propiedad cancelativa por la izquierda es
-- mul_left_cancel_iff : a * b = a * c ↔ b = c
--
-- Demostrar que si  $M$  es un monoide cancelativo por la izquierda y
--  $a, b \in M$ , entonces
--  $a * b = a \leftrightarrow b = 1$ 
-----

import algebra.group.basic

universe u
variables {M : Type u} [left_cancel_monoid M]
variables {a b : M}

-- ?ª demostración
-- =====

example : a * b = a ↔ b = 1 :=

```

```

begin
  split,
  { intro h,
    rw ← @mul_left_cancel_iff _ _ a b 1,
    rw mul_one,
    exact h, },
  { intro h,
    rw h,
    exact mul_one a, },
end

-- ?ª demostración
-- =====

example : a * b = a ↔ b = 1 :=
calc a * b = a ↔ a * b = a * 1 : by rw mul_one
    ... ↔ b = 1          : mul_left_cancel_iff

-- ?ª demostración
-- =====

example : a * b = a ↔ b = 1 :=
mul_right_eq_self

-- ?ª demostración
-- =====

example : a * b = a ↔ b = 1 :=
by finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.4. Unicidad del elemento neutro en los grupos

4.4.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que en los monoides conmutativos, si un elemento tiene un
-- inverso por la derecha, dicho inverso es único.
-- ----- *)

```

```

theory Unicidad_de_inversos_en_monoides
imports Main
begin

context comm_monoid
begin

(* 1ª demostración *)

lemma
  assumes "x |* y = |1"
          "x |* z = |1"
  shows "y = z"
proof -
  have "y = |1 |* y" by (simp only: left_neutral)
  also have "... = (x |* z) |* y" by (simp only: <x |* z = |1>)
  also have "... = (z |* x) |* y" by (simp only: commute)
  also have "... = z |* (x |* y)" by (simp only: assoc)
  also have "... = z |* |1" by (simp only: <x |* y = |1>)
  also have "... = z" by (simp only: right_neutral)
  finally show "y = z" by this
qed

(* 2ª demostración *)

lemma
  assumes "x |* y = |1"
          "x |* z = |1"
  shows "y = z"
proof -
  have "y = |1 |* y" by simp
  also have "... = (x |* z) |* y" using assms(2) by simp
  also have "... = (z |* x) |* y" by simp
  also have "... = z |* (x |* y)" by simp
  also have "... = z |* |1" using assms(1) by simp
  also have "... = z" by simp
  finally show "y = z" by this
qed

(* 3ª demostración *)

lemma
  assumes "x |* y = |1"
          "x |* z = |1"
  shows "y = z"

```

```

using assms
by auto

end

end

```

4.4.2. Demostraciones con Lean

```

-----
-- Demostrar que un grupo sólo posee un elemento neutro.
-----

import algebra.group.basic

universe u
variables {G : Type u} [group G]

-- 1ª demostración
-- =====

example
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
calc e = 1 * e : (one_mul e).symm
    ... = 1      : h 1

-- 2ª demostración
-- =====

example
  (e : G)
  (h : ∀ x, x * e = x)
  : e = 1 :=
self_eq_mul_left.mp (congr_arg _ (congr_arg _ (eq.symm (h e))))

-- 3ª demostración
-- =====

example
  (e : G)
  (h : ∀ x, x * e = x)

```

```

: e = 1 :=
by finish

-- Referencia
-- =====
-- Propiedad 3.17 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.5. Unicidad de los inversos en los grupos

4.5.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que si a es un elemento de un grupo G, entonces a tiene un
-- único inverso; es decir, si b es un elemento de G tal que a * b = 1,
-- entonces b es inverso de a.
----- *)

theory Unicidad_de_los_inversos_en_los_grupos
imports Main
begin

context group
begin

(* 1ª demostración *)

lemma
  assumes "a |* b = 1"
  shows "inverse a = b"
proof -
  have "inverse a = inverse a |* 1"      by (simp only: right_neutral)
  also have "... = inverse a |* (a |* b)" by (simp only: assms(1))
  also have "... = (inverse a |* a) |* b" by (simp only: assoc [symmetric])
  also have "... = 1 |* b"                by (simp only: left_inverse)
  also have "... = b"                    by (simp only: left_neutral)
  finally show "inverse a = b"          by this
qed

```



```

(* 2ª demostración *)

Lemma
  assumes "a |* b = |1"
  shows "inverse a = b"
proof -
  have "inverse a = inverse a |* |1"      by simp
  also have "... = inverse a |* (a |* b)" using assms by simp
  also have "... = (inverse a |* a) |* b" by (simp add: assoc [symmetric])
  also have "... = |1 |* b"              by simp
  also have "... = b"                    by simp
  finally show "inverse a = b"           .
qed

(* 3ª demostración *)

Lemma
  assumes "a |* b = |1"
  shows "inverse a = b"
proof -
  from assms have "inverse a |* (a |* b) = inverse a"
    by simp
  then show "inverse a = b"
    by (simp add: assoc [symmetric])
qed

(* 4ª demostración *)

Lemma
  assumes "a |* b = |1"
  shows "inverse a = b"
  using assms
  by (simp only: inverse_unique)

end

end

(*
-- Referencia
-- =====

-- Propiedad 3.18 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.

```

```
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49
*)
```

4.5.2. Demostraciones con Lean

```
-- Demostrar que si  $a$  es un elemento de un grupo  $G$ , entonces  $a$  tiene un
-- único inverso; es decir, si  $b$  es un elemento de  $G$  tal que  $a * b = 1$ ,
-- entonces  $a^{-1} = b$ .
```

```
import algebra.group.basic
```

```
universe u
variables {G : Type u} [group G]
variables {a b : G}
```

```
-- 1ª demostración
-- =====
```

```
example
```

```
(h : a * b = 1)
: a-1 = b :=
calc a-1 = a-1 * 1      : (mul_one a-1).symm
... = a-1 * (a * b) : congr_arg ((* a-1) h).symm
... = (a-1 * a) * b : (mul_assoc a-1 a b).symm
... = 1 * b         : congr_arg (* b) (inv_mul_self a)
... = b             : one_mul b
```

```
-- 2ª demostración
-- =====
```

```
example
```

```
(h : a * b = 1)
: a-1 = b :=
calc a-1 = a-1 * 1      : by simp only [mul_one]
... = a-1 * (a * b) : by simp only [h]
... = (a-1 * a) * b : by simp only [mul_assoc]
... = 1 * b         : by simp only [inv_mul_self]
... = b             : by simp only [one_mul]
```

```
-- 3ª demostración
-- =====
```

```

example
  (h : a * b = 1)
  : a-1 = b :=
calc a-1 = a-1 * 1      : by simp
    ... = a-1 * (a * b) : by simp [h]
    ... = (a-1 * a) * b : by simp
    ... = 1 * b         : by simp
    ... = b             : by simp

-- 4ª demostración
-- =====

example
  (h : a * b = 1)
  : a-1 = b :=
calc a-1 = a-1 * (a * b) : by simp [h]
    ... = b              : by simp

-- 5ª demostración
-- =====

example
  (h : b * a = 1)
  : b = a-1 :=
eq_inv_of_mul_eq_one h

-- Referencia
-- =====

-- Propiedad 3.18 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.6. Inverso del producto

4.6.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea  $G$  un grupo y  $a, b \in G$ . Entonces,
--  $(a * b)^{-1} = b^{-1} * a^{-1}$ 

```

```

----- *)

theory Inverso_del_producto
imports Main
begin

context group
begin

(* 1ª demostración *)

lemma "inverse (a |* b) = inverse b |* inverse a"
proof (rule inverse_unique)
  have "(a |* b) |* (inverse b |* inverse a) =
    ((a |* b) |* inverse b) |* inverse a"
    by (simp only: assoc)
  also have "... = (a |* (b |* inverse b)) |* inverse a"
    by (simp only: assoc)
  also have "... = (a |* |1) |* inverse a"
    by (simp only: right_inverse)
  also have "... = a |* inverse a"
    by (simp only: right_neutral)
  also have "... = |1"
    by (simp only: right_inverse)
  finally show "a |* b |* (inverse b |* inverse a) = |1"
    by this
qed

(* 2ª demostración *)

lemma "inverse (a |* b) = inverse b |* inverse a"
proof (rule inverse_unique)
  have "(a |* b) |* (inverse b |* inverse a) =
    ((a |* b) |* inverse b) |* inverse a"
    by (simp only: assoc)
  also have "... = (a |* (b |* inverse b)) |* inverse a"
    by (simp only: assoc)
  also have "... = (a |* |1) |* inverse a"
    by simp
  also have "... = a |* inverse a"
    by simp
  also have "... = |1"
    by simp
  finally show "a |* b |* (inverse b |* inverse a) = |1"
    .

```

```

qed

(* 3ª demostración *)

lemma "inverse (a |* b) = inverse b |* inverse a"
proof (rule inverse_unique)
  have "a |* b |* (inverse b |* inverse a) =
        a |* (b |* inverse b) |* inverse a"
    by (simp only: assoc)
  also have "... = |1"
    by simp
  finally show "a |* b |* (inverse b |* inverse a) = |1" .
qed

(* 4ª demostración *)

lemma "inverse (a |* b) = inverse b |* inverse a"
  by (simp only: inverse_distrib_swap)

end

end

(*
-- Referencia
-- =====

-- Propiedad 3.19 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49
*)

```

4.6.2. Demostraciones con Lean

```

-----
-- Sea  $G$  un grupo y  $a, b \in G$ . Entonces,
--  $(a * b)^{-1} = b^{-1} * a^{-1}$ 
-----

```

```
import algebra.group.basic
```

```
universe u
```

```

variables {G : Type u} [group G]
variables {a b : G}

-- 1ª demostración
-- =====

example : (a * b)-1 = b-1 * a-1 :=
begin
  apply inv_eq_of_mul_eq_one,
  calc a * b * (b-1 * a-1)
    = ((a * b) * b-1) * a-1 : (mul_assoc _ _ _).symm
  ... = (a * (b * b-1)) * a-1 : congr_arg (* a-1) (mul_assoc a _ _)
  ... = (a * 1) * a-1 : congr_arg2 _ (congr_arg _ (mul_inv_self b)) rfl
  ... = a * a-1 : congr_arg (* a-1) (mul_one a)
  ... = 1 : mul_inv_self a
end

-- 2ª demostración
-- =====

example : (a * b)-1 = b-1 * a-1 :=
begin
  apply inv_eq_of_mul_eq_one,
  calc a * b * (b-1 * a-1)
    = ((a * b) * b-1) * a-1 : by simp only [mul_assoc]
  ... = (a * (b * b-1)) * a-1 : by simp only [mul_assoc]
  ... = (a * 1) * a-1 : by simp only [mul_inv_self]
  ... = a * a-1 : by simp only [mul_one]
  ... = 1 : by simp only [mul_inv_self]
end

-- 3ª demostración
-- =====

example : (a * b)-1 = b-1 * a-1 :=
begin
  apply inv_eq_of_mul_eq_one,
  calc a * b * (b-1 * a-1)
    = ((a * b) * b-1) * a-1 : by simp [mul_assoc]
  ... = (a * (b * b-1)) * a-1 : by simp
  ... = (a * 1) * a-1 : by simp
  ... = a * a-1 : by simp
  ... = 1 : by simp,
end

```

```

-- 4ª demostración
-- =====

example : (a * b)-1 = b-1 * a-1 :=
mul_inv_rev a b

-- 5ª demostración
-- =====

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

-- Referencia
-- =====

-- Propiedad 3.19 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.7. Inverso del inverso en grupos

4.7.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea G un grupo y a ∈ G. Demostrar que
-- (a-1)-1 = a
-- ----- *)

theory Inverso_del_inverso_en_grupos
imports Main
begin

context group
begin

(* 1ª demostración *)

lemma "inverse (inverse a) = a"
proof -
  have "inverse (inverse a) =

```

```

      (inverse (inverse a)) |* |1"
    by (simp only: right_neutral)
  also have "... = inverse (inverse a) |* (inverse a |* a)"
    by (simp only: left_inverse)
  also have "... = (inverse (inverse a) |* inverse a) |* a"
    by (simp only: assoc)
  also have "... = |1 |* a"
    by (simp only: left_inverse)
  also have "... = a"
    by (simp only: left_neutral)
  finally show "inverse (inverse a) = a"
    by this
qed

(* 2ª demostración *)

lemma "inverse (inverse a) = a"
proof -
  have "inverse (inverse a) =
    (inverse (inverse a)) |* |1"
    by simp
  also have "... = inverse (inverse a) |* (inverse a |* a)" by simp
  also have "... = (inverse (inverse a) |* inverse a) |* a" by simp
  also have "... = |1 |* a" by simp
  finally show "inverse (inverse a) = a" by simp
qed

(* 3ª demostración *)

lemma "inverse (inverse a) = a"
proof (rule inverse_unique)
  show "inverse a |* a = |1"
    by (simp only: left_inverse)
qed

(* 4ª demostración *)

lemma "inverse (inverse a) = a"
proof (rule inverse_unique)
  show "inverse a |* a = |1" by simp
qed

(* 5ª demostración *)

lemma "inverse (inverse a) = a"
by (rule inverse_unique) simp

```



```

(* 6ª demostración *)

lemma "inverse (inverse a) = a"
  by (simp only: inverse_inverse)

(* 7ª demostración *)

lemma "inverse (inverse a) = a"
  by simp

end

end

(*
-- Referencia
-- =====

-- Propiedad 3.20 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49
*)

```

4.7.2. Demostraciones con Lean

```

-----
-- Sea  $G$  un grupo y  $a \in G$ . Demostrar que
--  $(a^{-1})^{-1} = a$ 
-----

import algebra.group.basic

universe u
variables {G : Type u} [group G]
variables {a b : G}

-- 1ª demostración
-- =====

example :  $(a^{-1})^{-1} = a$  :=
calc  $(a^{-1})^{-1}$ 
    =  $(a^{-1})^{-1} * 1$           : (mul_one  $(a^{-1})^{-1}$ ).symm

```

```

... = (a-1)-1 * (a-1 * a) : congr_arg ((*) (a-1)-1) (inv_mul_self a).symm
... = ((a-1)-1 * a-1) * a : (mul_assoc _ _ _).symm
... = 1 * a                    : congr_arg (*) (inv_mul_self a-1)
... = a                        : one_mul a

-- 2ª demostración
-- =====

example : (a-1)-1 = a :=
calc (a-1)-1
    = (a-1)-1 * 1          : by simp only [mul_one]
... = (a-1)-1 * (a-1 * a) : by simp only [inv_mul_self]
... = ((a-1)-1 * a-1) * a : by simp only [mul_assoc]
... = 1 * a                : by simp only [inv_mul_self]
... = a                    : by simp only [one_mul]

-- 3ª demostración
-- =====

example : (a-1)-1 = a :=
calc (a-1)-1
    = (a-1)-1 * 1          : by simp
... = (a-1)-1 * (a-1 * a) : by simp
... = ((a-1)-1 * a-1) * a : by simp
... = 1 * a                : by simp
... = a                    : by simp

-- 4ª demostración
-- =====

example : (a-1)-1 = a :=
begin
  apply inv_eq_of_mul_eq_one,
  exact mul_left_inv a,
end

-- 5ª demostración
-- =====

example : (a-1)-1 = a :=
inv_eq_of_mul_eq_one (mul_left_inv a)

-- 6ª demostración
-- =====

```

```

example : (a-1)-1 = a :=
inv_inv a

-- 7ª demostración
-- =====

example : (a-1)-1 = a :=
by simp

-- Referencia
-- =====

-- Propiedad 3.20 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf#page=49

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.8. Propiedad cancelativa en grupos

4.8.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea G un grupo y a,b,c ∈ G. Demostrar que si a * b = a * c, entonces
-- b = c.
----- *)

theory Propiedad_cancelativa_en_grupos
imports Main
begin

context group
begin

(* 1ª demostración *)

lemma
  assumes "a |* b = a |* c"
  shows   "b = c"
proof -
  have "b = |1 |* b"
    by (simp only: left_neutral)

```

```

also have "... = (inverse a |* a) |* b" by (simp only: left_inverse)
also have "... = inverse a |* (a |* b)" by (simp only: assoc)
also have "... = inverse a |* (a |* c)" by (simp only: <a |* b = a |* c>)
also have "... = (inverse a |* a) |* c" by (simp only: assoc)
also have "... = |1 |* c" by (simp only: left_inverse)
also have "... = c" by (simp only: left_neutral)
finally show "b = c" by this

```

qed

(* 2a demostración *)

Lemma

assumes "a |* b = a |* c"

shows "b = c"

proof -

```

have "b = |1 |* b" by simp
also have "... = (inverse a |* a) |* b" by simp
also have "... = inverse a |* (a |* b)" by (simp only: assoc)
also have "... = inverse a |* (a |* c)" using <a |* b = a |* c> by simp
also have "... = (inverse a |* a) |* c" by (simp only: assoc)
also have "... = |1 |* c" by simp
finally show "b = c" by simp

```

qed

(* 3a demostración *)

Lemma

assumes "a |* b = a |* c"

shows "b = c"

proof -

```

have "b = (inverse a |* a) |* b" by simp
also have "... = inverse a |* (a |* b)" by (simp only: assoc)
also have "... = inverse a |* (a |* c)" using <a |* b = a |* c> by simp
also have "... = (inverse a |* a) |* c" by (simp only: assoc)
finally show "b = c" by simp

```

qed

(* 4a demostración *)

Lemma

assumes "a |* b = a |* c"

shows "b = c"

proof -

```

have "inverse a |* (a |* b) = inverse a |* (a |* c)"
  by (simp only: <a |* b = a |* c>)

```

```

then have "(inverse a |* a) |* b = (inverse a |* a) |* c"
  by (simp only: assoc)
then have "|1 |* b = |1 |* c"
  by (simp only: left_inverse)
then show "b = c"
  by (simp only: left_neutral)
qed

```

(* 5ª demostración *)

Lemma

```

assumes "a |* b = a |* c"
shows   "b = c"

```

proof -

```

have "inverse a |* (a |* b) = inverse a |* (a |* c)"
  by (simp only: <a |* b = a |* c>)
then have "(inverse a |* a) |* b = (inverse a |* a) |* c"
  by (simp only: assoc)
then have "|1 |* b = |1 |* c"
  by (simp only: left_inverse)
then show "b = c"
  by (simp only: left_neutral)

```

qed

(* 6ª demostración *)

Lemma

```

assumes "a |* b = a |* c"
shows   "b = c"

```

proof -

```

have "inverse a |* (a |* b) = inverse a |* (a |* c)"
  using <a |* b = a |* c> by simp
then have "(inverse a |* a) |* b = (inverse a |* a) |* c"
  by (simp only: assoc)
then have "|1 |* b = |1 |* c"
  by simp
then show "b = c"
  by simp

```

qed

(* 7ª demostración *)

Lemma

```

assumes "a |* b = a |* c"
shows   "b = c"

```

```

using assms
by (simp only: left_cancel)

end

end

(*
-- Referencias
-- =====

-- Propiedad 3.22 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf
*)

```

4.8.2. Demostraciones con Lean

```

-- Sea G un grupo y  $a, b, c \in G$ . Demostrar que si  $a * b = a * c$ , entonces
--  $b = c$ .
-- -----

```

```

import algebra.group.basic

universe u
variables {G : Type u} [group G]
variables {a b c : G}

-- 1ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
calc b = 1 * b          : (one_mul b).symm
    ... = (a-1 * a) * b : congr_arg (* b) (inv_mul_self a).symm
    ... = a-1 * (a * b) : mul_assoc a-1 a b
    ... = a-1 * (a * c) : congr_arg ((* a-1) h)
    ... = (a-1 * a) * c : (mul_assoc a-1 a c).symm
    ... = 1 * c          : congr_arg (* c) (inv_mul_self a)
    ... = c              : one_mul c

```

```

-- 2ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
calc b = 1 * b          : by rw one_mul
    ... = (a⁻¹ * a) * b : by rw inv_mul_self
    ... = a⁻¹ * (a * b) : by rw mul_assoc
    ... = a⁻¹ * (a * c) : by rw h
    ... = (a⁻¹ * a) * c : by rw mul_assoc
    ... = 1 * c         : by rw inv_mul_self
    ... = c             : by rw one_mul

-- 3ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
calc b = 1 * b          : by simp
    ... = (a⁻¹ * a) * b : by simp
    ... = a⁻¹ * (a * b) : by simp
    ... = a⁻¹ * (a * c) : by simp [h]
    ... = (a⁻¹ * a) * c : by simp
    ... = 1 * c         : by simp
    ... = c             : by simp

-- 4ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
calc b = a⁻¹ * (a * b) : by simp
    ... = a⁻¹ * (a * c) : by simp [h]
    ... = c             : by simp

-- 4ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
begin

```

```

have h1 : a-1 * (a * b) = a-1 * (a * c),
  { by finish [h] },
have h2 : (a-1 * a) * b = (a-1 * a) * c,
  { by finish },
have h3 : 1 * b = 1 * c,
  { by finish },
have h3 : b = c,
  { by finish },
exact h3,
end

-- 4ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
begin
  have : a-1 * (a * b) = a-1 * (a * c),
    { by finish [h] },
  have h2 : (a-1 * a) * b = (a-1 * a) * c,
    { by finish },
  have h3 : 1 * b = 1 * c,
    { by finish },
  have h3 : b = c,
    { by finish },
  exact h3,
end

-- 4ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
begin
  have h1 : a-1 * (a * b) = a-1 * (a * c),
    { congr, exact h, },
  have h2 : (a-1 * a) * b = (a-1 * a) * c,
    { simp only [h1, mul_assoc], },
  have h3 : 1 * b = 1 * c,
    { simp only [h2, (inv_mul_self a).symm], },
  rw one_mul at h3,
  rw one_mul at h3,
  exact h3,

```



```

end

-- 5ª demostración
-- =====

example
  (h: a * b = a * c)
  : b = c :=
mul_left_cancel h

-- 6ª demostración
-- =====

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

-- Referencias
-- =====

-- Propiedad 3.22 del libro "Abstract algebra: Theory and applications"
-- de Thomas W. Judson.
-- http://abstract.ups.edu/download/aata-20200730.pdf

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.9. Potencias de potencias en monoides

4.9.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponentes naturales. En Lean la potencia  $x^n$  se
-- se caracteriza por los siguientes lemas:
--   power_0    :  $x^0 = 1$ 
--   power_Suc  :  $x^{(Suc\ n)} = x * x^n$ 
--
-- Demostrar que si  $M$ ,  $a \in M$  y  $m, n \in \mathbb{N}$ , entonces
--    $a^{(m * n)} = (a^m)^n$ 
--
-- Indicación: Se puede usar el lema

```

```

--      power_add : a^(m + n) = a^m * a^n
----- *)

theory Potencias_de_potencias_en_monoides
imports Main
begin

context monoid_mult
begin

(* 1ª demostración *)

lemma "a^(m * n) = (a^m)^n"
proof (induct n)
  have "a ^ (m * 0) = a ^ 0"
    by (simp only: mult_0_right)
  also have "... = 1"
    by (simp only: power_0)
  also have "... = (a ^ m) ^ 0"
    by (simp only: power_0)
  finally show "a ^ (m * 0) = (a ^ m) ^ 0"
    by this
next
  fix n
  assume HI : "a ^ (m * n) = (a ^ m) ^ n"
  have "a ^ (m * Suc n) = a ^ (m + m * n)"
    by (simp only: mult_Suc_right)
  also have "... = a ^ m * a ^ (m * n)"
    by (simp only: power_add)
  also have "... = a ^ m * (a ^ m) ^ n"
    by (simp only: HI)
  also have "... = (a ^ m) ^ Suc n"
    by (simp only: power_Suc)
  finally show "a ^ (m * Suc n) = (a ^ m) ^ Suc n"
    by this
qed

(* 2ª demostración *)

lemma "a^(m * n) = (a^m)^n"
proof (induct n)
  have "a ^ (m * 0) = a ^ 0"
    by simp
  also have "... = 1"
    by simp
  also have "... = (a ^ m) ^ 0"
    by simp
  finally show "a ^ (m * 0) = (a ^ m) ^ 0" .

```

```

next
  fix n
  assume HI : "a ^ (m * n) = (a ^ m) ^ n"
  have "a ^ (m * Suc n) = a ^ (m + m * n)" by simp
  also have "... = a ^ m * a ^ (m * n)" by (simp add: power_add)
  also have "... = a ^ m * (a ^ m) ^ n" using HI by simp
  also have "... = (a ^ m) ^ Suc n" by simp
  finally show "a ^ (m * Suc n) = (a ^ m) ^ Suc n" .

qed

(* 3ª demostración *)

lemma "a^(m * n) = (a^m)^n"
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case by (simp add: power_add)
qed

(* 4ª demostración *)

lemma "a^(m * n) = (a^m)^n"
  by (induct n) (simp_all add: power_add)

(* 5ª demostración *)

lemma "a^(m * n) = (a^m)^n"
  by (simp only: power_mult)

end

end

```

4.9.2. Demostraciones con Lean

```

-- En los [monoides](https://en.wikipedia.org/wiki/Monoid) se define la
-- potencia con exponentes naturales. En Lean la potencia  $x^n$  se
-- se caracteriza por los siguientes lemas:
--   pow_zero :  $x^0 = 1$ 

```

```

--      pow_succ' : x^(succ n) = x * x^n
--
-- Demostrar que si  $M$ ,  $a \in M$  y  $m, n \in \mathbb{N}$ , entonces
--       $a^{(m * n)} = (a^m)^n$ 
--
-- Indicación: Se puede usar el lema
--      pow_add :  $a^{(m + n)} = a^m * a^n$ 
-----

import algebra.group_power.basic
open monoid nat

variables {M : Type} [monoid M]
variable   a : M
variables (m n : ℕ)

-- Para que no use la notación con puntos
set_option pp.structure_projections false

-- 1ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
begin
  induction n with n HI,
  { calc a^(m * 0)
        = a^0                : congr_arg ((^) a) (nat.mul_zero m)
        ... = 1               : pow_zero a
        ... = (a^m)^0         : (pow_zero (a^m)).symm },
  { calc a^(m * succ n)
        = a^(m * n + m)      : congr_arg ((^) a) (nat.mul_succ m n)
        ... = a^(m * n) * a^m : pow_add a (m * n) m
        ... = (a^m)^n * a^m   : congr_arg (* a^m) HI
        ... = (a^m)^(succ n) : (pow_succ' (a^m) n).symm },
end

-- 2ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
begin
  induction n with n HI,
  { calc a^(m * 0)
        = a^0                : by simp only [nat.mul_zero]

```

```

... = 1                                : by simp only [pow_zero]
... = (a^m)^0                          : by simp only [pow_zero] },
{ calc a^(m * succ n)
  = a^(m * n + m) : by simp only [nat.mul_succ]
... = a^(m * n) * a^m : by simp only [pow_add]
... = (a^m)^n * a^m : by simp only [HI]
... = (a^m)^(succ n) : by simp only [pow_succ'] },
end

-- 3ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
begin
  induction n with n HI,
  { calc a^(m * 0)
    = a^0 : by simp [nat.mul_zero]
    ... = 1 : by simp
    ... = (a^m)^0 : by simp },
  { calc a^(m * succ n)
    = a^(m * n + m) : by simp [nat.mul_succ]
    ... = a^(m * n) * a^m : by simp [pow_add]
    ... = (a^m)^n * a^m : by simp [HI]
    ... = (a^m)^(succ n) : by simp [pow_succ'] },
end

-- 4ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
begin
  induction n with n HI,
  { by simp [nat.mul_zero] },
  { by simp [nat.mul_succ,
    pow_add,
    HI,
    pow_succ'] },
end

-- 5ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=

```

```

begin
  induction n with n HI,
  { rw nat.mul_zero,
    rw pow_zero,
    rw pow_zero, },
  { rw nat.mul_succ,
    rw pow_add,
    rw HI,
    rw pow_succ', }
end

-- 6ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
begin
  induction n with n HI,
  { rw [nat.mul_zero, pow_zero, pow_zero] },
  { rw [nat.mul_succ, pow_add, HI, pow_succ'] }
end

-- 7ª demostración
-- =====

example : a^(m * n) = (a^m)^n :=
pow_mul a m n

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.10. Los monoides booleanos son conmutativos

4.10.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Un monoide M es booleano si
--    $\forall x \in M, x * x = 1$ 
-- y es conmutativo si
--    $\forall x y \in M, x * y = y * x$ 
--
-- Demostrar que los monoides booleanos son conmutativos.
-- ----- *)

```

```

theory Los_monoides_booleanos_son_conmutativos
imports Main
begin

context monoid
begin

(* 1ª demostración *)

lemma
  assumes "∀ x. x |* x = |1"
  shows   "∀ x y. x |* y = y |* x"
proof (rule allI)+
  fix a b
  have "a |* b = (a |* b) |* |1"
    by (simp only: right_neutral)
  also have "... = (a |* b) |* (a |* a)"
    by (simp only: assms)
  also have "... = ((a |* b) |* a) |* a"
    by (simp only: assoc)
  also have "... = (a |* (b |* a)) |* a"
    by (simp only: assoc)
  also have "... = (|1 |* (a |* (b |* a))) |* a"
    by (simp only: left_neutral)
  also have "... = ((b |* b) |* (a |* (b |* a))) |* a"
    by (simp only: assms)
  also have "... = (b |* (b |* (a |* (b |* a)))) |* a"
    by (simp only: assoc)
  also have "... = (b |* ((b |* a) |* (b |* a))) |* a"
    by (simp only: assoc)
  also have "... = (b |* |1) |* a"
    by (simp only: assms)
  also have "... = b |* a"
    by (simp only: right_neutral)
  finally show "a |* b = b |* a"
    by this
qed

(* 2ª demostración *)

lemma
  assumes "∀ x. x |* x = |1"
  shows   "∀ x y. x |* y = y |* x"
proof (rule allI)+

```

```

fix a b
have "a |* b = (a |* b) |* |1" by simp
also have "... = (a |* b) |* (a |* a)" by (simp add: assms)
also have "... = ((a |* b) |* a) |* a" by (simp add: assoc)
also have "... = (a |* (b |* a)) |* a" by (simp add: assoc)
also have "... = (|1 |* (a |* (b |* a))) |* a" by simp
also have "... = ((b |* b) |* (a |* (b |* a))) |* a" by (simp add: assms)
also have "... = (b |* (b |* (a |* (b |* a)))) |* a" by (simp add: assoc)
also have "... = (b |* ((b |* a) |* (b |* a))) |* a" by (simp add: assoc)
also have "... = (b |* |1) |* a" by (simp add: assms)
also have "... = b |* a" by simp
finally show "a |* b = b |* a" by this
qed

(* 3a demostración *)

Lemma
  assumes "∀ x. x |* x = |1"
  shows "∀ x y. x |* y = y |* x"
proof (rule allI)+
  fix a b
  have "a |* b = (a |* b) |* (a |* a)" by (simp add: assms)
  also have "... = (a |* (b |* a)) |* a" by (simp add: assoc)
  also have "... = ((b |* b) |* (a |* (b |* a))) |* a" by (simp add: assms)
  also have "... = (b |* ((b |* a) |* (b |* a))) |* a" by (simp add: assoc)
  also have "... = (b |* |1) |* a" by (simp add: assms)
  finally show "a |* b = b |* a" by simp
qed

(* 4a demostración *)

Lemma
  assumes "∀ x. x |* x = |1"
  shows "∀ x y. x |* y = y |* x"
  by (metis assms assoc right_neutral)

end

end

```


4.10.2. Demostraciones con Lean

```

-- -----
-- Un monoide es un conjunto junto con una operación binaria que es
-- asociativa y tiene elemento neutro.
--
-- Un monoide M es booleano si
--    $\forall x \in M, x * x = 1$ 
-- y es conmutativo si
--    $\forall x y \in M, x * y = y * x$ 
--
-- En Lean, está definida la clase de los monoides (como 'monoid') y sus
-- propiedades características son
--   mul_assoc : (a * b) * c = a * (b * c)
--   one_mul   : 1 * a = a
--   mul_one   : a * 1 = a
--
-- Demostrar que los monoides booleanos son conmutativos.
-- -----

```

```
import algebra.group.basic
```

```
example
```

```

{M : Type} [monoid M]
(h :  $\forall x : M, x * x = 1$ )
:  $\forall x y : M, x * y = y * x$  :=

```

```
begin
```

```

  intros a b,
  calc a * b
    = (a * b) * 1
    : (mul_one (a * b)).symm
  ... = (a * b) * (a * a)
    : congr_arg ((*) (a*b)) (h a).symm
  ... = ((a * b) * a) * a
    : (mul_assoc (a*b) a a).symm
  ... = (a * (b * a)) * a
    : congr_arg (* a) (mul_assoc a b a)
  ... = (1 * (a * (b * a))) * a
    : congr_arg (* a) (one_mul (a*(b*a))).symm
  ... = ((b * b) * (a * (b * a))) * a
    : congr_arg (* a) (congr_arg (* (a*(b*a))) (h b).symm)
  ... = (b * (b * (a * (b * a)))) * a
    : congr_arg (* a) (mul_assoc b b (a*(b*a)))
  ... = (b * ((b * a) * (b * a))) * a
    : congr_arg (* a) (congr_arg ((*) b) (mul_assoc b a (b*a)).symm)

```

```

... = (b * 1) * a
      : congr_arg (* a) (congr_arg ((*) b) (h (b*a)))
... = b * a
      : congr_arg (* a) (mul_one b),
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.11. Límite de sucesiones constantes

4.11.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \epsilon$ )"
--
-- Demostrar que el límite de la sucesión constante  $c$  es  $c$ .
----- *)

theory Limite_de_sucesiones_constantes
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \epsilon$ )"

(* 1ª demostración *)

lemma "limite ( $\lambda n. c$ ) c"
proof (unfold limite_def)
  show " $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |c - c| < \epsilon$ "
proof (intro allI impI)
  fix  $\epsilon :: real$ 
  assume "0 <  $\epsilon$ "
  have " $\forall n \geq 0 :: nat. |c - c| < \epsilon$ "
  proof (intro allI impI)
    fix n :: nat
    assume "0  $\leq n$ "
    have "c - c = 0"

```

```

    by (simp only: diff_self)
  then have " $|c - c| = 0$ "
    by (simp only: abs_eq_0_iff)
  also have " $\dots < \varepsilon$ "
    by (simp only:  $\langle 0 < \varepsilon \rangle$ )
  finally show " $|c - c| < \varepsilon$ "
    by this
qed
then show " $\exists k :: \text{nat}. \forall n \geq k. |c - c| < \varepsilon$ "
  by (rule exI)
qed
qed

(* 2ª demostración *)

lemma "limite ( $\lambda n. c$ ) c"
proof (unfold limite_def)
  show " $\forall \varepsilon > 0. \exists k :: \text{nat}. \forall n \geq k. |c - c| < \varepsilon$ "
  proof (intro allI impI)
    fix  $\varepsilon :: \text{real}$ 
    assume " $0 < \varepsilon$ "
    have " $\forall n \geq 0 :: \text{nat}. |c - c| < \varepsilon$ " by (simp add:  $\langle 0 < \varepsilon \rangle$ )
    then show " $\exists k :: \text{nat}. \forall n \geq k. |c - c| < \varepsilon$ " by (rule exI)
  qed
qed

(* 3ª demostración *)

lemma "limite ( $\lambda n. c$ ) c"
  unfolding limite_def
  by simp

(* 4ª demostración *)

lemma "limite ( $\lambda n. c$ ) c"
  by (simp add: limite_def)

end

```

4.11.2. Demostraciones con Lean

```

-----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--    $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '||x||' := abs x
--
-- Demostrar que el límite de la sucesión constante  $c$  es  $c$ .
-----

import data.real.basic

variable (u :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variable (c :  $\mathbb{R}$ )

notation '||x||' := abs x

def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
 $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 

-- 1ª demostración
-- =====

example :
  limite ( $\lambda n, c$ ) c :=
begin
  unfold limite,
  intros  $\varepsilon h\varepsilon$ ,
  use 0,
  intros n hn,
  dsimp,
  simp,
  exact hε,
end

-- 2ª demostración
-- =====

example :
  limite ( $\lambda n, c$ ) c :=
begin
  intros  $\varepsilon h\varepsilon$ ,

```

```

use 0,
rintro n -,
norm_num,
assumption,
end

-- 3ª demostración
-- =====

example :
  limite (λ n, c) c :=
begin
  intros ε hε,
  use 0,
  intros n hn,
  calc | (λ n, c) n - c |
      = | c - c |      : rfl
  ... = 0              : by simp
  ... < ε              : hε
end

-- 4ª demostración
-- =====

example :
  limite (λ n, c) c :=
begin
  intros ε hε,
  by finish,
end

-- 5ª demostración
-- =====

example :
  limite (λ n, c) c :=
λ ε hε, by finish

-- 6ª demostración
-- =====

example :
  limite (λ n, c) c :=
assume ε,
assume hε : ε > 0,

```

```

exists.intro 0
(
  assume n,
  assume hn : n ≥ 0,
  show | (λ n, c) n - c | < ε, from
    calc | (λ n, c) n - c |
      = | c - c | : rfl
      ... = 0 : by simp
      ... < ε : hε)

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.12. Unicidad del límite de las sucesiones convergentes

4.12.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat ⇒ real) ⇒ real ⇒ bool"
--   where "limite u c ↔ (∀ε>0. ∃k::nat. ∀n≥k. |u n - c| < ε)"
--
-- Demostrar que cada sucesión tiene como máximo un límite.
----- *)

theory Unicidad_del_limite_de_las_sucesiones_convergentes
imports Main HOL.Real
begin

definition limite :: "(nat ⇒ real) ⇒ real ⇒ bool"
  where "limite u c ↔ (∀ε>0. ∃k::nat. ∀n≥k. |u n - c| < ε)"

lemma aux :
  assumes "limite u a"
         "limite u b"
  shows   "b ≤ a"
proof (rule ccontr)
  assume "¬ b ≤ a"
  let ?ε = "b - a"
  have "0 < ?ε/2"

```

```

using <| b ≤ a |> by auto
obtain A where hA : "∀n ≥ A. |u n - a| < ?ε/2"
using assms(1) limite_def <0 < ?ε/2> by blast
obtain B where hB : "∀n ≥ B. |u n - b| < ?ε/2"
using assms(2) limite_def <0 < ?ε/2> by blast
let ?C = "max A B"
have hCa : "∀n ≥ ?C. |u n - a| < ?ε/2"
using hA by simp
have hCb : "∀n ≥ ?C. |u n - b| < ?ε/2"
using hB by simp
have "∀n ≥ ?C. |a - b| < ?ε"
proof (intro allI impI)
  fix n assume "n ≥ ?C"
  have "|a - b| = |(a - u n) + (u n - b)|" by simp
  also have "... ≤ |u n - a| + |u n - b|" by simp
  finally show "|a - b| < b - a"
    using hCa hCb <n ≥ ?C> by fastforce
qed
then show False by fastforce
qed

theorem
  assumes "limite u a"
    "limite u b"
  shows "a = b"
proof (rule antisym)
  show "a ≤ b" using assms(2) assms(1) by (rule aux)
next
  show "b ≤ a" using assms(1) assms(2) by (rule aux)
qed
end

```

4.12.2. Demostraciones con Lean

```

-- -----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--      $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 

```

```

--      notation '|x|' := abs x
--
-- Demostrar que cada sucesión tiene como máximo un límite.
-----

import data.real.basic

variables {u : ℕ → ℝ}
variables {a b : ℝ}

notation '|x|' := abs x

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

-- 1ª demostración
-- =====

lemma aux
  (ha : limite u a)
  (hb : limite u b)
  : b ≤ a :=
begin
  by_contra h,
  set ε := b - a with hε,
  cases ha (ε/2) (by linarith) with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,
  linarith,
end

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
le_antisymm (aux hb ha) (aux ha hb)

-- 2ª demostración
-- =====

```



```

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contra h,
  wlog hab : a < b,
  { have : a < b ∨ a = b ∨ b < a := lt_trichotomy a b,
    tauto },
  set ε := b - a with hε,
  specialize ha (ε/2),
  have hε2 : ε/2 > 0 := by linarith,
  specialize ha hε2,
  cases ha with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,
  linarith,
end

```

```

-- 3ª demostración
-- =====

```

```

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  by_contra h,
  wlog hab : a < b,
  { have : a < b ∨ a = b ∨ b < a := lt_trichotomy a b,
    tauto },
  set ε := b - a with hε,
  cases ha (ε/2) (by linarith) with A hA,
  cases hb (ε/2) (by linarith) with B hB,
  set N := max A B with hN,
  have hAN : A ≤ N := le_max_left A B,
  have hBN : B ≤ N := le_max_right A B,
  specialize hA N hAN,
  specialize hB N hBN,
  rw abs_lt at hA hB,

```

```
linarith,
end
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.13. Límite cuando se suma una constante

4.13.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \epsilon$ )"
--
-- Demostrar que si el límite de la sucesión  $u(i)$  es  $a$  y  $c \in \mathbb{R}$ ,
-- entonces el límite de  $u(i)+c$  es  $a+c$ .
-- ----- *)

theory Limite_cuando_se_suma_una_constante
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \epsilon$ )"

(* 1ª demostración *)

lemma
  assumes "limite u a"
  shows   "limite ( $\lambda i. u\ i + c$ ) (a + c)"
proof (unfold limite_def)
  show " $\forall \epsilon > 0. \exists k. \forall n \geq k. |u\ n + c - (a + c)| < \epsilon$ "
proof (intro allI impI)
  fix  $\epsilon :: real$ 
  assume " $0 < \epsilon$ "
  then have " $\exists k. \forall n \geq k. |u\ n - a| < \epsilon$ "
    using assms limite_def by simp
  then obtain k where " $\forall n \geq k. |u\ n - a| < \epsilon$ "
    by (rule exE)
  then have " $\forall n \geq k. |u\ n + c - (a + c)| < \epsilon$ "

```

```

    by simp
  then show "∃k. ∀n≥k. |u n + c - (a + c)| < ε"
    by (rule exI)
qed
qed

(* 2ª demostración *)

lemma
  assumes "límite u a"
  shows   "límite (λ i. u i + c) (a + c)"
  using  assms limite_def
  by     simp
end

```

4.13.2. Demostraciones con Lean

```

-----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--   λ u a, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - a| < ε
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '||x||' := abs x
--
-- Demostrar que si el límite de la sucesión  $u(i)$  es  $a$  y  $c \in \mathbb{R}$ , entonces
-- el límite de  $u(i)+c$  es  $a+c$ .
-----

import data.real.basic
import tactic

variables {u :  $\mathbb{N} \rightarrow \mathbb{R}$ }
variables {a c :  $\mathbb{R}$ }

notation '||x||' := abs x

def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
λ u c, ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - c| < ε

```

```

-- 1ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ i, u i + c) (a + c) :=
begin
  intros ε hε,
  dsimp,
  cases h ε hε with k hk,
  use k,
  intros n hn,
  calc |u n + c - (a + c)|
      = |u n - a|           : by norm_num
      ... < ε               : hk n hn,
end

-- 2ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ i, u i + c) (a + c) :=
begin
  intros ε hε,
  dsimp,
  cases h ε hε with k hk,
  use k,
  intros n hn,
  convert hk n hn using 2,
  ring,
end

-- 3ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ i, u i + c) (a + c) :=
begin
  intros ε hε,
  convert h ε hε,
  by norm_num,
end

```

```
-- 4ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ i, u i + c) (a + c) :=
λ ε hε, (by convert h ε hε; norm_num)
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.14. Límite de la suma de sucesiones convergentes

4.14.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \epsilon$ )"
--
-- Demostrar que el límite de la suma de dos sucesiones convergentes es
-- la suma de los límites de dichas sucesiones.
-- ----- *)

theory Limite_de_la_suma_de_sucesiones_convergentes
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \epsilon$ )"

(* 1ª demostración *)

lemma
  assumes "limite u a"
        "limite v b"
  shows  "limite (λ n. u n + v n) (a + b)"
proof (unfold limite_def)
  show " $\forall \epsilon > 0$ .  $\exists k$ .  $\forall n \geq k$ .  $|(u\ n + v\ n) - (a + b)| < \epsilon$ "
```

```

proof (intro allI impI)
  fix  $\varepsilon :: \text{real}$ 
  assume " $0 < \varepsilon$ "
  then have " $0 < \varepsilon/2$ "
    by simp
  then have " $\exists k. \forall n \geq k. |u\ n - a| < \varepsilon/2$ "
    using assms(1) limite_def by blast
  then obtain Nu where hNu : " $\forall n \geq \text{Nu}. |u\ n - a| < \varepsilon/2$ "
    by (rule exE)
  then have " $\exists k. \forall n \geq k. |v\ n - b| < \varepsilon/2$ "
    using  $\langle 0 < \varepsilon/2 \rangle$  assms(2) limite_def by blast
  then obtain Nv where hNv : " $\forall n \geq \text{Nv}. |v\ n - b| < \varepsilon/2$ "
    by (rule exE)
  have " $\forall n \geq \max\ \text{Nu}\ \text{Nv}. |(u\ n + v\ n) - (a + b)| < \varepsilon$ "
  proof (intro allI impI)
    fix n :: nat
    assume " $n \geq \max\ \text{Nu}\ \text{Nv}$ "
    have " $|(u\ n + v\ n) - (a + b)| = |(u\ n - a) + (v\ n - b)|$ "
      by simp
    also have " $\dots \leq |u\ n - a| + |v\ n - b|$ "
      by simp
    also have " $\dots < \varepsilon/2 + \varepsilon/2$ "
      using hNu hNv  $\langle \max\ \text{Nu}\ \text{Nv} \leq n \rangle$  by fastforce
    finally show " $|(u\ n + v\ n) - (a + b)| < \varepsilon$ "
      by simp
  qed
  then show " $\exists k. \forall n \geq k. |u\ n + v\ n - (a + b)| < \varepsilon$ "
    by (rule exI)
qed
qed

(* 2ª demostración *)

```

Lemma

```

assumes "limite u a"
        "limite v b"
shows   "limite ( $\lambda\ n. u\ n + v\ n$ ) (a + b)"
proof (unfold limite_def)
  show " $\forall \varepsilon > 0. \exists k. \forall n \geq k. |(u\ n + v\ n) - (a + b)| < \varepsilon$ "
  proof (intro allI impI)
    fix  $\varepsilon :: \text{real}$ 
    assume " $0 < \varepsilon$ "
    then have " $0 < \varepsilon/2$ " by simp
    obtain Nu where hNu : " $\forall n \geq \text{Nu}. |u\ n - a| < \varepsilon/2$ "
      using  $\langle 0 < \varepsilon/2 \rangle$  assms(1) limite_def by blast

```

```

obtain Nv where hNv : "∀n ≥ Nv. |v n - b| < ε/2"
using <0 < ε/2> assms(2) limite_def by blast
have "∀n ≥ max Nu Nv. |(u n + v n) - (a + b)| < ε"
using hNu hNv
by (smt (verit, ccfv_threshold) field_sum_of_halves max.boundedE)
then show "∃k. ∀n ≥ k. |u n + v n - (a + b)| < ε"
by blast
qed
qed
end

```

4.14.2. Demostraciones con Lean

```

-----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--    $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '||x||' := abs x
--
-- Demostrar que el límite de la suma de dos sucesiones convergentes es
-- la suma de los límites de dichas sucesiones.
-----

import data.real.basic

variables (u v :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variables (a b :  $\mathbb{R}$ )

notation '||x||' := abs x

def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
 $\lambda u c, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - c| < \varepsilon$ 

-- 1ª demostración
-- =====

example
  (hu : limite u a)

```

```

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

-- 2ª demostración
-- =====

example
(hu : limite u a)
(hv : limite v b)
: limite (u + v) (a + b) :=
begin
  intros ε hε,
  cases hu (ε/2) (by linarith) with Nu hNu,
  cases hv (ε/2) (by linarith) with Nv hNv,
  use max Nu Nv,
  intros n hn,
  have hn1 : n ≥ Nu := le_of_max_le_left hn,
  specialize hNu n hn1,
  have hn2 : n ≥ Nv := le_of_max_le_right hn,
  specialize hNv n hn2,
  calc |(u + v) n - (a + b)|
    = |(u n + v n) - (a + b)| : by refl

```



```

... = |(u n - a) + (v n - b)| : by {congr, ring}
... ≤ |u n - a| + |v n - b|   : by apply abs_add
... < ε / 2 + ε / 2          : by linarith
... = ε                       : by apply add_halves,
end

-- 3ª demostración
-- =====

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

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

-- 4ª demostración
-- =====

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε hε,

```

```

cases hu (ε/2) (by linarith) with Nu hNu,
cases hv (ε/2) (by linarith) with Nv hNv,
use max Nu Nv,
intros n hn,
cases max_ge_iff.mp hn with hn₁ hn₂,
calc |(u + v) n - (a + b)|
    = |u n + v n - (a + b)| : by refl
... = |(u n - a) + (v n - b)| : by { congr, ring }
... ≤ |u n - a| + |v n - b| : by apply abs_add
... < ε/2 + ε/2           : add_lt_add (hNu n hn₁) (hNv n hn₂)
... = ε                   : by simp
end

-- 5ª demostración
-- =====

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

-- 6ª demostración
-- =====

example
  (hu : limite u a)
  (hv : limite v b)
  : limite (u + v) (a + b) :=
begin
  intros ε Hε,
  cases hu (ε/2) (by linarith) with L HL,
  cases hv (ε/2) (by linarith) with M HM,

```

```

set N := max L M with hN,
use N,
have HLN : N ≥ L := le_max_left _ _,
have HMN : N ≥ M := le_max_right _ _,
intros n Hn,
have H3 : |u n - a| < ε/2 := HL n (by linarith),
have H4 : |v n - b| < ε/2 := HM n (by linarith),
calc |(u + v) n - (a + b)|
  = |(u n + v n) - (a + b)| : by refl
... = |(u n - a) + (v n - b)| : by { congr, ring }
... ≤ |(u n - a)| + |(v n - b)| : by apply abs_add
... < ε/2 + ε/2 : by linarith
... = ε : by ring
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.15. Límite multiplicado por una constante

4.15.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u n - c| < \epsilon$ )"
--
-- Demostrar que que si el límite de  $u(i)$  es  $a$ , entonces el de
--  $c \cdot u(i)$  es  $c \cdot a$ .
-- ----- *)

theory Limite_multiplicado_por_una_constante
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u n - c| < \epsilon$ )"

lemma
  assumes "limite u a"
  shows   "limite ( $\lambda n. c * u n$ ) (c * a)"

```

```

proof (unfold limite_def)
  show "∀ε>0. ∃k. ∀n≥k. |c * u n - c * a| < ε"
proof (intro allI impI)
  fix ε :: real
  assume "0 < ε"
  show "∃k. ∀n≥k. |c * u n - c * a| < ε"
proof (cases "c = 0")
  assume "c = 0"
  then show "∃k. ∀n≥k. |c * u n - c * a| < ε"
    by (simp add: <0 < ε>)
next
  assume "c ≠ 0"
  then have "0 < |c|"
    by simp
  then have "0 < ε/|c|"
    by (simp add: <0 < ε>)
  then obtain N where hN : "∀n≥N. |u n - a| < ε/|c|"
    using assms limite_def
    by auto
  have "∀n≥N. |c * u n - c * a| < ε"
  proof (intro allI impI)
    fix n
    assume "n ≥ N"
    have "|c * u n - c * a| = |c * (u n - a)|"
      by argo
    also have "... = |c| * |u n - a|"
      by (simp only: abs_mult)
    also have "... < |c| * (ε/|c|)"
      using hN <n ≥ N> <0 < |c|>
      by (simp only: mult_strict_left_mono)
    finally show "|c * u n - c * a| < ε"
      using <0 < |c|>
      by auto
  qed
  then show "∃k. ∀n≥k. |c * u n - c * a| < ε"
    by (rule exI)
qed
qed
qed
end

```

4.15.2. Demostraciones con Lean

```

-- -----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--    $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u\ n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '||x||' := abs x
--
-- Demostrar que que si el límite de  $u(i)$  es  $a$ , entonces el de
--  $c \cdot u(i)$  es  $c \cdot a$ .
-- -----

```

```

import data.real.basic
import tactic

variables (u v :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variables (a c :  $\mathbb{R}$ )

notation '||x||' := abs x

def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
 $\lambda u c, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u\ n - c| < \varepsilon$ 

-- 1ª demostración
-- =====

example
  (h : limite u a)
  : limite ( $\lambda n, c * (u\ n)$ ) (c * a) :=
begin
  by_cases hc : c = 0,
  { subst hc,
    intros  $\varepsilon$  h $\varepsilon$ ,
    by finish, },
  { intros  $\varepsilon$  h $\varepsilon$ ,
    have hc' :  $0 < |c| := \text{abs\_pos.mpr hc}$ ,
    have h $\varepsilon c$  :  $0 < \varepsilon / |c| := \text{div\_pos h}\varepsilon\ hc'$ ,
    specialize h ( $\varepsilon / |c|$ ) h $\varepsilon c$ ,
    cases h with N hN,
    use N,
    intros n hn,

```

```

    specialize hN n hn,
    dsimp only,
    rw ← mul_sub,
    rw abs_mul,
    rw ← lt_div_iff' hc',
    exact hN, }
end

-- 2ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ n, c * (u n)) (c * a) :=
begin
  by_cases hc : c = 0,
  { subst hc,
    intros ε hε,
    by finish, },
  { intros ε hε,
    have hc' : 0 < |c| := abs_pos.mpr hc,
    have hεc : 0 < ε / |c| := div_pos hε hc',
    specialize h (ε/|c|) hεc,
    cases h with N hN,
    use N,
    intros n hn,
    specialize hN n hn,
    dsimp only,
    calc |c * u n - c * a|
      = |c * (u n - a)| : congr_arg abs (mul_sub c (u n) a).symm
      ... = |c| * |u n - a| : abs_mul c (u n - a)
      ... < |c| * (ε / |c|) : (mul_lt_mul_left hc').mpr hN
      ... = ε : mul_div_cancel' ε (ne_of_gt hc') }
end

-- 3ª demostración
-- =====

example
  (h : limite u a)
  : limite (λ n, c * (u n)) (c * a) :=
begin
  by_cases hc : c = 0,
  { subst hc,
    intros ε hε,

```

```

    by finish, },
  { intros  $\varepsilon$  h $\varepsilon$ ,
    have hc' :  $0 < |c|$  := by finish,
    have h $\varepsilon$ c :  $0 < \varepsilon / |c|$  := div_pos h $\varepsilon$  hc',
    cases h ( $\varepsilon / |c|$ ) h $\varepsilon$ c with N hN,
    use N,
    intros n hn,
    specialize hN n hn,
    dsimp only,
    rw [← mul_sub, abs_mul, ← lt_div_iff' hc'],
    exact hN, }
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.16. El límite de u es a syss el de $u-a$ es 0

4.16.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función ( $u : \mathbb{N} \rightarrow \mathbb{R}$ ) de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \varepsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \varepsilon$ )"
--
-- Demostrar que el límite de  $u(i)$  es  $a$  si y solo si el de  $u(i)-a$  es
-- 0.
----- *)

theory "El_límite_de_u_es_a_syss_el_de_u-a_es_0"
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \varepsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \varepsilon$ )"

(* 1ª demostración *)

lemma
  "limite u a  $\leftrightarrow$  limite ( $\lambda i. u\ i - a$ ) 0"
proof -

```

```

have "limite u a  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - a| < \varepsilon$ )"
  by (rule limite_def)
also have "...  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|(u\ n - a) - 0| < \varepsilon$ )"
  by simp
also have "...  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
  by (rule limite_def[symmetric])
finally show "limite u a  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
  by this
qed

(* 2ª demostración *)

lemma
  "limite u a  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
proof -
  have "limite u a  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - a| < \varepsilon$ )"
    by (simp only: limite_def)
  also have "...  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|(u\ n - a) - 0| < \varepsilon$ )"
    by simp
  also have "...  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
    by (simp only: limite_def)
  finally show "limite u a  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
    by this
qed

(* 3ª demostración *)

lemma
  "limite u a  $\leftrightarrow$  limite ( $\lambda i$ .  $u\ i - a$ ) 0"
  using limite_def
  by simp
end

```

4.16.2. Demostraciones con Lean

```

-- -----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función ( $u : \mathbb{N} \rightarrow \mathbb{R}$ ) de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite : ( $\mathbb{N} \rightarrow \mathbb{R}$ )  $\rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--      $\lambda u\ a$ ,  $\forall \varepsilon > 0$ ,  $\exists N$ ,  $\forall n \geq N$ ,  $|u\ n - a| < \varepsilon$ 

```



```
-- donde se usa la notación |x| para el valor absoluto de x
-- notation '||x||' := abs x
--
-- Demostrar que el límite de  $u(i)$  es  $a$  si y solo si el de  $u(i)$ - $a$  es
-- 0.
```

```
import data.real.basic
import tactic
```

```
variable {u :  $\mathbb{N} \rightarrow \mathbb{R}$ }
variables {a c x :  $\mathbb{R}$ }
```

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

```
def limite : ( $\mathbb{N} \rightarrow \mathbb{R}$ )  $\rightarrow \mathbb{R} \rightarrow \mathbf{Prop} :=$ 
 $\lambda$  u c,  $\forall \varepsilon > 0, \exists N, \forall n \geq N, |u\ n - c| < \varepsilon$ 
```

```
-- 1ª demostración
-- =====
```

```
example
  : limite u a  $\leftrightarrow$  limite ( $\lambda$  i, u i - a) 0 :=
begin
  rw iff_eq_eq,
  calc limite u a
    =  $\forall \varepsilon > 0, \exists N, \forall n \geq N, |u\ n - a| < \varepsilon$  : rfl
  ... =  $\forall \varepsilon > 0, \exists N, \forall n \geq N, |(u\ n - a) - 0| < \varepsilon$  : by simp
  ... = limite ( $\lambda$  i, u i - a) 0 : rfl,
end
```

```
-- 2ª demostración
-- =====
```

```
example
  : limite u a  $\leftrightarrow$  limite ( $\lambda$  i, u i - a) 0 :=
begin
  split,
  { intros h  $\varepsilon$  h $\varepsilon$ ,
    convert h  $\varepsilon$  h $\varepsilon$ ,
    norm_num, },
  { intros h  $\varepsilon$  h $\varepsilon$ ,
    convert h  $\varepsilon$  h $\varepsilon$ ,
    norm_num, },
end
```

```

-- 3ª demostración
-- =====

example
  : limite u a ↔ limite (λ i, u i - a) 0 :=
begin
  split;
  { intros h ε hε,
    convert h ε hε,
    norm_num, },
end

-- 4ª demostración
-- =====

lemma limite_con_suma
  (c : ℝ)
  (h : limite u a)
  : limite (λ i, u i + c) (a + c) :=
λ ε hε, (by convert h ε hε; norm_num)

lemma CNS_limite_con_suma
  (c : ℝ)
  : limite u a ↔ limite (λ i, u i + c) (a + c) :=
begin
  split,
  { apply limite_con_suma },
  { intro h,
    convert limite_con_suma (-c) h; simp, },
end

example
  (u : ℕ → ℝ)
  (a : ℝ)
  : limite u a ↔ limite (λ i, u i - a) 0 :=
begin
  convert CNS_limite_con_suma (-a),
  simp,
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.17. Producto de sucesiones convergentes a cero

4.17.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \varepsilon$ )"
--
-- Demostrar que si las sucesiones  $u(n)$  y  $v(n)$  convergen a cero,
-- entonces  $u(n) \cdot v(n)$  también converge a cero.
----- *)

theory Producto_de_sucesiones_convergentes_a_cero
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \varepsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \varepsilon$ )"

lemma
  assumes "limite u 0"
    "limite v 0"
  shows "limite ( $\lambda n$ . u n * v n) 0"
proof (unfold limite_def; intro allI impI)
  fix  $\varepsilon :: \text{real}$ 
  assume h $\varepsilon$  : "0 <  $\varepsilon$ "
  then obtain U where hU : " $\forall n \geq U$ .  $|u\ n - 0| < \varepsilon$ "
    using assms(1) limite_def
    by auto
  obtain V where hV : " $\forall n \geq V$ .  $|v\ n - 0| < 1$ "
    using h $\varepsilon$  assms(2) limite_def
    by fastforce
  have " $\forall n \geq \max U\ V$ .  $|u\ n * v\ n - 0| < \varepsilon$ "
  proof (intro allI impI)
    fix n
    assume hn : " $\max U\ V \leq n$ "
    then have "U  $\leq n$ "
      by simp

```

```

then have "⌊u n - 0⌋ < ε"
  using hU by blast
have hnV : "V ≤ n"
  using hn by simp
then have "⌊v n - 0⌋ < 1"
  using hV by blast
have "⌊u n * v n - 0⌋ = ⌊(u n - 0) * (v n - 0)⌋"
  by simp
also have "... = ⌊u n - 0⌋ * ⌊v n - 0⌋"
  by (simp add: abs_mult)
also have "... < ε * 1"
  using <⌊u n - 0⌋ < ε> <⌊v n - 0⌋ < 1>
  by (rule abs_mult_less)
also have "... = ε"
  by simp
finally show "⌊u n * v n - 0⌋ < ε"
  by this
qed
then show "∃k. ∀n ≥ k. ⌊u n * v n - 0⌋ < ε"
  by (rule exI)
qed
end

```

4.17.2. Demostraciones con Lean

```

-- -----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--      $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '⌊x⌋' := abs x
--
-- Demostrar que si las sucesiones  $u(n)$  y  $v(n)$  convergen a cero,
-- entonces  $u(n) \cdot v(n)$  también converge a cero.
-- -----

import data.real.basic
import tactic

```

```

variables {u v :  $\mathbb{N} \rightarrow \mathbb{R}$ }
variables {c :  $\mathbb{R}$ }

notation '|x|' := abs x

def limite : ( $\mathbb{N} \rightarrow \mathbb{R}$ ) →  $\mathbb{R} \rightarrow \mathbf{Prop} :=
\lambda u c, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - c| < \varepsilon

-- 1ª demostración
-- =====

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
begin
  intros  $\varepsilon$  h $\varepsilon$ ,
  cases hu  $\varepsilon$  h $\varepsilon$  with U hU,
  cases hv 1 zero_lt_one with V hV,
  set N := max U V with hN,
  use N,
  intros n hn,
  specialize hU n (le_of_max_le_left hn),
  specialize hV n (le_of_max_le_right hn),
  rw sub_zero at *,
  calc |(u * v) n|
    = |u n * v n| : rfl
    ... = |u n| * |v n| : abs_mul (u n) (v n)
    ... <  $\varepsilon$  * 1 : mul_lt_mul'' hU hV (abs_nonneg (u n)) (abs_nonneg (v n))
    ... =  $\varepsilon$  : mul_one  $\varepsilon$ ,
end

-- 2ª demostración
-- =====

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
begin
  intros  $\varepsilon$  h $\varepsilon$ ,
  cases hu  $\varepsilon$  h $\varepsilon$  with U hU,
  cases hv 1 (by linarith) with V hV,
  set N := max U V with hN,
  use N,$ 
```

```

intros n hn,
specialize hU n (le_of_max_le_left hn),
specialize hV n (le_of_max_le_right hn),
rw sub_zero at *,
calc |(u * v) n|
  = |u n * v n| : rfl
  ... = |u n| * |v n| : abs_mul (u n) (v n)
  ... < ε * 1 : by { apply mul_lt_mul'' hU hV ; simp [abs_nonneg] }
  ... = ε : mul_one ε,
end

-- 3ª demostración
-- =====

example
  (hu : limite u 0)
  (hv : limite v 0)
  : limite (u * v) 0 :=
begin
  intros ε hε,
  cases hu ε hε with U hU,
  cases hv 1 (by linarith) with V hV,
  set N := max U V with hN,
  use N,
  intros n hn,
  have hUN : U ≤ N := le_max_left U V,
  have hVN : V ≤ N := le_max_right U V,
  specialize hU n (by linarith),
  specialize hV n (by linarith),
  rw sub_zero at F hU hV,
  rw pi.mul_apply,
  rw abs_mul,
  convert mul_lt_mul'' hU hV _ _ , simp,
  all_goals {apply abs_nonneg},
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.18. Teorema del emparedado

4.18.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--   where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \epsilon$ )"
--
-- Demostrar que si para todo  $n$ ,  $u(n) \leq v(n) \leq w(n)$  y  $u(n)$  tiene el
-- mismo límite que, entonces  $v(n)$  también tiene dicho límite.
----- *)

theory Teorema_del_emparedado
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \epsilon$ )"

lemma
  assumes "limite u a"
    "limite w a"
    " $\forall n$ .  $u\ n \leq v\ n$ "
    " $\forall n$ .  $v\ n \leq w\ n$ "
  shows "limite v a"
proof (unfold limite_def; intro allI impI)
  fix  $\epsilon :: \text{real}$ 
  assume h $\epsilon$  : " $0 < \epsilon$ "
  obtain N where hN : " $\forall n \geq N$ .  $|u\ n - a| < \epsilon$ "
    using assms(1) h $\epsilon$  limite_def
    by auto
  obtain N' where hN' : " $\forall n \geq N'$ .  $|w\ n - a| < \epsilon$ "
    using assms(2) h $\epsilon$  limite_def
    by auto
  have " $\forall n \geq \max\ N\ N'$ .  $|v\ n - a| < \epsilon$ "
  proof (intro allI impI)
    fix n
    assume hn : " $n \geq \max\ N\ N'$ "
    have " $v\ n - a < \epsilon$ "
    proof -

```

```

have "v n - a ≤ w n - a"
  using assms(4) by simp
also have "... ≤ |w n - a|"
  by simp
also have "... < ε"
  using hN' hn by auto
finally show "v n - a < ε" .
qed
moreover
have "-(v n - a) < ε"
proof -
  have "-(v n - a) ≤ -(u n - a)"
    using assms(3) by auto
  also have "... ≤ |u n - a|"
    by simp
  also have "... < ε"
    using hN hn by auto
  finally show "-(v n - a) < ε" .
qed
ultimately show "|v n - a| < ε"
  by (simp only: abs_less_iff)
qed
then show "∃k. ∀n≥k. |v n - a| < ε"
  by (rule exI)
qed
end

```

4.18.2. Demostraciones con Lean

```

-- -----
-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--      $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '!'x'|' := abs x
--
-- Demostrar que si para todo  $n$ ,  $u(n) \leq v(n) \leq w(n)$  y  $u(n)$  tiene el
-- mismo límite que  $w(n)$ , entonces  $v(n)$  también tiene dicho límite.
-- -----

```



```

import data.real.basic

variables (u v w :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variable (a :  $\mathbb{R}$ )

notation '||x||' := abs x

def limite : ( $\mathbb{N} \rightarrow \mathbb{R}$ ) →  $\mathbb{R} \rightarrow \mathbf{Prop} :=
\lambda u c, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - c| \leq \varepsilon

-- Nota. En la demostración se usará el siguiente lema:
lemma max_ge_iff
  {p q r :  $\mathbb{N}$ }
  :  $r \geq \max p q \leftrightarrow r \geq p \wedge r \geq q :=
max_le_iff

-- 1ª demostración
-- =====

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

```

    ... ≤ ε      : hN'.2, },
end

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

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

```

```

    rw abs_le at *,
    split ; linarith,
end

-- 4ª demostración
example
  (hu : limite u a)
  (hw : limite w a)
  (h :  $\forall n, u\ n \leq v\ n$ )
  (h' :  $\forall n, v\ n \leq w\ n$ ) :
  limite v a :=
assume  $\varepsilon$ ,
assume h $\varepsilon$  :  $\varepsilon > 0$ ,
exists.elim (hu  $\varepsilon$  h $\varepsilon$ )
  ( assume N,
    assume hN :  $\forall (n : \mathbb{N}), n \geq N \rightarrow |u\ n - a| \leq \varepsilon$ ,
    exists.elim (hw  $\varepsilon$  h $\varepsilon$ )
      ( assume N',
        assume hN' :  $\forall (n : \mathbb{N}), n \geq N' \rightarrow |w\ n - a| \leq \varepsilon$ ,
        show  $\exists N, \forall n, n \geq N \rightarrow |v\ n - a| \leq \varepsilon$ , from
          exists.intro (max N N')
          ( assume n,
            assume hn :  $n \geq \max N\ N'$ ,
            have h1 :  $n \geq N \wedge n \geq N'$ ,
            from max_ge_iff.mp hn,
            have h2 :  $-\varepsilon \leq v\ n - a$ ,
            { have h2a :  $|u\ n - a| \leq \varepsilon$ ,
              from hN n h1.1,
              calc - $\varepsilon$ 
                 $\leq u\ n - a$  : and.left (abs_le.mp h2a)
                 $\dots \leq v\ n - a$  : by linarith [h n], },
            have h3 :  $v\ n - a \leq \varepsilon$ ,
            { have h3a :  $|w\ n - a| \leq \varepsilon$ ,
              from hN' n h1.2,
              calc v\ n - a
                 $\leq w\ n - a$  : by linarith [h' n]
                 $\dots \leq \varepsilon$  : and.right (abs_le.mp h3a), },
            show  $|v\ n - a| \leq \varepsilon$ ,
            from abs_le.mpr (and.intro h2 h3))))

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.19. La composición de crecientes es creciente

4.19.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Se dice que una función f de  $\mathbb{R}$  en  $\mathbb{R}$  es creciente https://bit.ly/2USggL
-- si para todo x e y tales que  $x \leq y$  se tiene que  $f(x) \leq f(y)$ .
--
-- En Isabelle/HOL que f sea creciente se representa por 'mono f'.
--
-- Demostrar que la composición de dos funciones crecientes es una
-- función creciente.
----- *)

theory La_composicion_de_crecientes_es_creciente
imports Main HOL.Real
begin

(* 1ª demostración *)
lemma
  fixes f g :: "real  $\Rightarrow$  real"
  assumes "mono f"
          "mono g"
  shows   "mono (g  $\circ$  f)"
proof (rule monoI)
  fix x y :: real
  assume "x  $\leq$  y"
  have "(g  $\circ$  f) x = g (f x)"
    by (simp only: o_apply)
  also have "...  $\leq$  g (f y)"
    using assms <x  $\leq$  y>
    by (simp only: monoD)
  also have "... = (g  $\circ$  f) y"
    by (simp only: o_apply)
  finally show "(g  $\circ$  f) x  $\leq$  (g  $\circ$  f) y"
    by this
qed

(* 2ª demostración *)
lemma
  fixes f g :: "real  $\Rightarrow$  real"
  assumes "mono f"
```

```

      "mono g"
shows   "mono (g ∘ f)"
proof (rule monoI)
  fix x y :: real
  assume "x ≤ y"
  have "(g ∘ f) x = g (f x)"      by simp
  also have "... ≤ g (f y)"        by (simp add: <x ≤ y> assms monoD)
  also have "... = (g ∘ f) y"      by simp
  finally show "(g ∘ f) x ≤ (g ∘ f) y" .
qed

(* 3ª demostración *)
lemma
  assumes "mono f"
        "mono g"
  shows   "mono (g ∘ f)"
  by (metis assms comp_def mono_def)
end

```

4.19.2. Demostraciones con Lean

```

-----
-- Se dice que una función  $f$  de  $\mathbb{R}$  en  $\mathbb{R}$  es creciente https://bit.ly/2USggL
-- si para todo  $x$  e  $y$  tales que  $x \leq y$  se tiene que  $f(x) \leq f(y)$ .
--
-- En Lean que  $f$  sea creciente se representa por 'monotone f'.
--
-- Demostrar que la composición de dos funciones crecientes es una
-- función creciente.
-----

import data.real.basic

variables (f g : ℝ → ℝ)

-- 1ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
begin
  intros x y hxy,

```

```

calc (g  $\circ$  f) x
      = g (f x)      : rfl
... ≤ g (f y)      : hg (hf hxy)
... = (g  $\circ$  f) y : rfl,
end

-- 2ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g  $\circ$  f) :=
begin
  unfold monotone at *,
  intros x y h,
  unfold function.comp,
  apply hg,
  apply hf,
  exact h,
end

-- 3ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g  $\circ$  f) :=
begin
  intros x y h,
  apply hg,
  apply hf,
  exact h,
end

-- 4ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g  $\circ$  f) :=
begin
  intros x xy h,
  apply hg,
  exact hf h,
end

-- 5ª demostración
example

```

```

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

-- 6ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
λ x y h, hg (hf h)

-- 7ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
begin
  intros x y h,
  specialize hf h,
  exact hg hf,
end

-- 8ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
assume x y,
assume h1 : x ≤ y,
have h2 : f x ≤ f y,
  from hf h1,
show (g ∘ f) x ≤ (g ∘ f) y, from
  calc (g ∘ f) x
    = g (f x)      : rfl
    ... ≤ g (f y)   : hg h2
    ... = (g ∘ f) y : by refl

-- 9ª demostración
example
  (hf : monotone f)
  (hg : monotone g)

```

```

: monotone (g ∘ f) :=
-- by hint
by tauto

-- 10ª demostración
example
  (hf : monotone f)
  (hg : monotone g)
  : monotone (g ∘ f) :=
-- by library_search
monotone.comp hg hf

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

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

4.20.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea una función  $f$  de  $\mathbb{R}$  en  $\mathbb{R}$ . Se dice que  $f$  es creciente
-- https://bit.ly/2UShggL si para todo  $x$  e  $y$  tales que  $x \leq y$  se tiene
-- que  $f(x) \leq f(y)$ . Se dice que  $f$  es decreciente si para todo  $x$  e  $y$ 
-- tales que  $x \leq y$  se tiene que  $f(x) \geq f(y)$ .
--
-- En Isabelle/HOL que  $f$  sea creciente se representa por ' $\text{mono } f$ ' y que
-- sea decreciente por ' $\text{antimono } f$ '.
--
-- Demostrar que si  $f$  es creciente y  $g$  es decreciente, entonces  $(g \circ f)$ 
-- es decreciente.
----- *)

theory La_composicion_de_una_funcion_creciente_y_una_decreciente_es_decreciente
imports Main HOL.Real
begin

(* 1ª demostración *)
lemma
  fixes f g :: "real  $\Rightarrow$  real"
  assumes "mono f"
           "antimono g"
  shows  "antimono (g ∘ f)"

```



```

proof (rule antimonoI)
  fix x y :: real
  assume "x ≤ y"
  have "(g ∘ f) y = g (f y)"
    by (simp only: o_apply)
  also have "... ≤ g (f x)"
    using assms <x ≤ y>
    by (meson antimonoE monoE)
  also have "... = (g ∘ f) x"
    by (simp only: o_apply)
  finally show "(g ∘ f) x ≥ (g ∘ f) y"
    by this
qed

(* 2ª demostración *)
lemma
  fixes f g :: "real ⇒ real"
  assumes "mono f"
         "antimono g"
  shows "antimono (g ∘ f)"
proof (rule antimonoI)
  fix x y :: real
  assume "x ≤ y"
  have "(g ∘ f) y = g (f y)"      by simp
  also have "... ≤ g (f x)"        by (meson <x ≤ y> assms antimonoE monoE)
  also have "... = (g ∘ f) x"      by simp
  finally show "(g ∘ f) x ≥ (g ∘ f) y" .
qed

(* 3ª demostración *)
lemma
  assumes "mono f"
         "antimono g"
  shows "antimono (g ∘ f)"
  by (metis assms mono_def antimono_def comp_apply)
end

```

4.20.2. Demostraciones con Lean

```

-- -----
-- Sea una función  $f$  de  $\mathbb{R}$  en  $\mathbb{R}$ . Se dice que  $f$  es creciente
-- https://bit.ly/2UShggL si para todo  $x$  e  $y$  tales que  $x \leq y$  se tiene

```

```
-- que  $f(x) \leq f(y)$ . Se dice que  $f$  es decreciente si para todo  $x$  e  $y$ 
-- tales que  $x \leq y$  se tiene que  $f(x) \geq f(y)$ .
--
-- Demostrar que si  $f$  es creciente y  $g$  es decreciente, entonces  $(g \circ f)$ 
-- es decreciente.
-- -----
```

```
import data.real.basic
```

```
variables (f g :  $\mathbb{R} \rightarrow \mathbb{R}$ )
```

```
def creciente (f :  $\mathbb{R} \rightarrow \mathbb{R}$ ) : Prop :=
   $\forall \{x y\}, x \leq y \rightarrow f\ x \leq f\ y$ 
```

```
def decreciente (f :  $\mathbb{R} \rightarrow \mathbb{R}$ ) : Prop :=
   $\forall \{x y\}, x \leq y \rightarrow f\ x \geq f\ y$ 
```

```
-- 1ª demostración
```

```
example
```

```
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g  $\circ$  f) :=
```

```
begin
```

```
  intros x y hxy,
  calc (g  $\circ$  f) x
    = g (f x)      : rfl
  ...  $\geq$  g (f y)    : hg (hf hxy)
  ... = (g  $\circ$  f) y : rfl,
```

```
end
```

```
-- 2ª demostración
```

```
example
```

```
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g  $\circ$  f) :=
```

```
begin
```

```
  unfold creciente decreciente at *,
  intros x y h,
  unfold function.comp,
  apply hg,
  apply hf,
  exact h,
```

```
end
```

```
-- 3ª demostración
```

```

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

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

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

-- 6ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ∘ f) :=
λ x y h, hg (hf h)

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

```

```

assume h : x ≤ y,
have h1 : f x ≤ f y,
  from hf h,
show (g ◦ f) x ≥ (g ◦ f) y,
  from hg h1

-- 8ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ◦ f) :=
assume x y,
assume h : x ≤ y,
show (g ◦ f) x ≥ (g ◦ f) y,
  from hg (hf h)

-- 9ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ◦ f) :=
λ x y h, hg (hf h)

-- 10ª demostración
example
  (hf : creciente f)
  (hg : decreciente g)
  : decreciente (g ◦ f) :=
-- by hint
by tauto

```

4.21. Una función creciente e involutiva es la identidad

4.21.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea una función f de  $\mathbb{R}$  en  $\mathbb{R}$ .
-- + Se dice que f es creciente si para todo x e y tales que  $x \leq y$  se
--   tiene que  $f(x) \leq f(y)$ .
-- + Se dice que f es involutiva si para todo x se tiene que  $f(f(x)) = x$ .

```

```

--
-- En Isabelle/HOL que  $f$  sea creciente se representa por 'mono  $f$ '.
--
-- Demostrar que si  $f$  es creciente e involutiva, entonces  $f$  es la
-- identidad.
-- ----- *)

theory Una_funcion_creciente_e_involutiva_es_la_identidad
imports Main HOL.Real
begin

definition involutiva :: "(real  $\Rightarrow$  real)  $\Rightarrow$  bool"
  where "involutiva  $f \leftrightarrow (\forall x. f (f x) = x)"$ 

(* 1ª demostración *)
lemma
  fixes  $f :: "real \Rightarrow real"$ 
  assumes "mono  $f$ "
        "involutiva  $f$ "
  shows "f = id"
proof (unfold fun_eq_iff; intro allI)
  fix x
  have " $x \leq f x \vee f x \leq x$ "
    by (rule linear)
  then have " $f x = x$ "
  proof (rule disjE)
    assume " $x \leq f x$ "
    then have " $f x \leq f (f x)$ "
      using assms(1) by (simp only: monoD)
    also have " $\dots = x$ "
      using assms(2) by (simp only: involutiva_def)
    finally have " $f x \leq x$ "
      by this
    show " $f x = x$ "
      using <math>f x \leq x</math> <math>x \leq f x</math> by (simp only: antisym)
  next
    assume " $f x \leq x$ "
    have " $x = f (f x)$ "
      using assms(2) by (simp only: involutiva_def)
    also have " $\dots \leq f x$ "
      using <math>f x \leq x</math> assms(1) by (simp only: monoD)
    finally have " $x \leq f x$ "
      by this
    show " $f x = x$ "
      using <math>f x \leq x</math> <math>x \leq f x</math> by (simp only: monoD)
  end
end

```

```

qed
then show "f x = id x"
  by (simp only: id_apply)
qed

(* 2ª demostración *)
lemma
  fixes f :: "real ⇒ real"
  assumes "mono f"
        "involutiva f"
  shows   "f = id"
proof
  fix x
  have "x ≤ f x ∨ f x ≤ x"
    by (rule linear)
  then have "f x = x"
  proof
    assume "x ≤ f x"
    then have "f x ≤ f (f x)"
      using assms(1) by (simp only: monoD)
    also have "... = x"
      using assms(2) by (simp only: involutiva_def)
    finally have "f x ≤ x"
      by this
    show "f x = x"
      using ⟨f x ≤ x⟩ ⟨x ≤ f x⟩ by auto
  next
    assume "f x ≤ x"
    have "x = f (f x)"
      using assms(2) by (simp only: involutiva_def)
    also have "... ≤ f x"
      by (simp add: ⟨f x ≤ x⟩ assms(1) monoD)
    finally have "x ≤ f x"
      by this
    show "f x = x"
      using ⟨f x ≤ x⟩ ⟨x ≤ f x⟩ by auto
  qed
  then show "f x = id x"
    by simp
qed

(* 3ª demostración *)
lemma
  fixes f :: "real ⇒ real"
  assumes "mono f"

```

```

      "involutiva f"
shows   "f = id"
proof
  fix x
  have "x ≤ f x ∨ f x ≤ x"
    by (rule linear)
  then have "f x = x"
  proof
    assume "x ≤ f x"
    then have "f x ≤ x"
      by (metis assms involutiva_def mono_def)
    then show "f x = x"
      using <x ≤ f x> by auto
  next
    assume "f x ≤ x"
    then have "x ≤ f x"
      by (metis assms involutiva_def mono_def)
    then show "f x = x"
      using <f x ≤ x> by auto
  qed
  then show "f x = id x"
    by simp
qed
end

```

4.21.2. Demostraciones con Lean

```

-----
-- Sea una función  $f$  de  $\mathbb{R}$  en  $\mathbb{R}$ .
-- + Se dice que  $f$  es creciente si para todo  $x$  e  $y$  tales que  $x \leq y$  se
--   tiene que  $f(x) \leq f(y)$ .
-- + Se dice que  $f$  es involutiva si para todo  $x$  se tiene que  $f(f(x)) = x$ .
--
-- En Lean que  $f$  sea creciente se representa por 'monotone  $f$ ' y que sea
-- involutiva por 'involutive  $f$ '
--
-- Demostrar que si  $f$  es creciente e involutiva, entonces  $f$  es la
-- identidad.
-----

import data.real.basic
open function

```

```

variable (f : ℝ → ℝ)

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

-- 2ª demostración
example
  (hc : monotone f)
  (hi : involutive f)
  : f = id :=
begin
  funext,
  cases (le_total (f x) x) with h1 h2,
  { apply antisymm h1,
    have h3 : f (f x) ≤ f x := hc h1,
    rwa hi at h3, },
  { apply antisymm _ h2,
    have h4 : f x ≤ f (f x) := hc h2,
    rwa hi at h4, },
end

-- 3ª demostración
example
  (hc : monotone f)
  (hi : involutive f)

```



```

: f = id :=
begin
  funext,
  cases (le_total (f x) x) with h1 h2,
  { apply antisymm h1,
    calc x
      = f (f x) : (h1 x).symm
    ... ≤ f x   : hc h1 },
  { apply antisymm _ h2,
    calc f x
      ≤ f (f x) : hc h2
    ... = x     : h1 x },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.22. Si ' $f x \leq f y \rightarrow x \leq y$ ', entonces f es inyectiva

4.22.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea f una función de  $\mathbb{R}$  en  $\mathbb{R}$  tal que
--  $\forall x y, f(x) \leq f(y) \rightarrow x \leq y$ 
-- Demostrar que f es inyectiva.
-- ----- *)

theory "Si_f(x)_leq_f(y)_to_x_leq_y,_entonces_f_es_inyectiva"
imports Main HOL.Real
begin

(* 1ª demostración *)
lemma
  fixes f :: "real  $\Rightarrow$  real"
  assumes " $\forall x y. f x \leq f y \rightarrow x \leq y$ "
  shows "inj f"
proof (rule injI)
  fix x y
  assume "f x = f y"
  show "x = y"
proof (rule antisym)
  show "x  $\leq$  y"

```

```

    by (simp only: assms <f x = f y>)
  next
    show "y ≤ x"
    by (simp only: assms <f x = f y>)
  qed
qed

(* 2ª demostración *)
lemma
  fixes f :: "real ⇒ real"
  assumes "∀ x y. f x ≤ f y → x ≤ y"
  shows "inj f"
proof (rule injI)
  fix x y
  assume "f x = f y"
  then show "x = y"
    using assms
    by (simp add: eq_iff)
qed

(* 3ª demostración *)
lemma
  fixes f :: "real ⇒ real"
  assumes "∀ x y. f x ≤ f y → x ≤ y"
  shows "inj f"
  by (smt (verit, ccfv_threshold) assms inj_on_def)
end

```

4.22.2. Demostraciones con Lean

```

-- Sea f una función de ℝ en ℝ tal que
--   ∀ x y, f(x) ≤ f(y) → x ≤ y
-- Demostrar que f es inyectiva.

```

```

import data.real.basic
open function

variable (f : ℝ → ℝ)

-- 1ª demostración

```

```

example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : injective f :=
begin
  intros x y hxy,
  apply le_antisymm,
  { apply h,
    exact le_of_eq hxy, },
  { apply h,
    exact ge_of_eq hxy, },
end

-- 2ª demostración
example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : injective f :=
begin
  intros x y hxy,
  apply le_antisymm,
  { exact h (le_of_eq hxy), },
  { exact h (ge_of_eq hxy), },
end

-- 3ª demostración
example
  (h : ∀ {x y}, f x ≤ f y → x ≤ y)
  : injective f :=
λ x y hxy, le_antisymm (h hxy.le) (h hxy.ge)

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.23. Los supremos de las sucesiones crecientes son sus límites

4.23.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sea u una sucesión creciente. Demostrar que si M es un supremo de u,
-- entonces el límite de u es M.
-- ----- *)

theory Los_supremos_de_las_sucesiones_crecientes_son_sus_limites

```

```

imports Main HOL.Real
begin

(* (limite u c) expresa que el límite de u es c. *)
definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool" where
  "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k$ .  $\forall n \geq k$ .  $|u\ n - c| \leq \epsilon$ )"

(* (supremo u M) expresa que el supremo de u es M. *)
definition supremo :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool" where
  "supremo u M  $\leftrightarrow$  (( $\forall n$ .  $u\ n \leq M$ )  $\wedge$  ( $\forall \epsilon > 0$ .  $\exists k$ .  $\forall n \geq k$ .  $u\ n \geq M - \epsilon$ ))"

(* 1ª demostración *)
lemma
  assumes "mono u"
        "supremo u M"
  shows  "limite u M"
proof (unfold limite_def; intro allI impI)
  fix  $\epsilon ::$  real
  assume " $0 < \epsilon$ "
  have hM : "(( $\forall n$ .  $u\ n \leq M$ )  $\wedge$  ( $\forall \epsilon > 0$ .  $\exists k$ .  $\forall n \geq k$ .  $u\ n \geq M - \epsilon$ ))"
    using assms(2)
    by (simp add: supremo_def)
  then have " $\forall \epsilon > 0$ .  $\exists k$ .  $\forall n \geq k$ .  $u\ n \geq M - \epsilon$ "
    by (rule conjunct2)
  then have " $\exists k$ .  $\forall n \geq k$ .  $u\ n \geq M - \epsilon$ "
    by (simp only:  $0 < \epsilon$ )
  then obtain n0 where " $\forall n \geq n0$ .  $u\ n \geq M - \epsilon$ "
    by (rule exE)
  have " $\forall n \geq n0$ .  $|u\ n - M| \leq \epsilon$ "
  proof (intro allI impI)
    fix n
    assume " $n \geq n0$ "
    show " $|u\ n - M| \leq \epsilon$ "
    proof (rule abs_leI)
      have " $\forall n$ .  $u\ n \leq M$ "
        using hM by (rule conjunct1)
      then have " $u\ n - M \leq M - M$ "
        by simp
      also have " $\dots = 0$ "
        by (simp only: diff_self)
      also have " $\dots \leq \epsilon$ "
        using  $0 < \epsilon$  by (simp only: less_imp_le)
      finally show " $u\ n - M \leq \epsilon$ "
        by this
    next

```

```

    have "-ε = (M - ε) - M"
    by simp
  also have "... ≤ u n - M"
    using <∀n≥n0. M - ε ≤ u n> <n0 ≤ n> by auto
  finally have "-ε ≤ u n - M"
    by this
  then show "- (u n - M) ≤ ε"
    by simp
qed
qed
then show "∃k. ∀n≥k. |u n - M| ≤ ε"
  by (rule exI)
qed

(* 2ª demostración *)
lemma
  assumes "mono u"
        "supremo u M"
  shows   "limite u M"
proof (unfold limite_def; intro allI impI)
  fix ε :: real
  assume "0 < ε"
  have hM : "((∀n. u n ≤ M) ∧ (∀ε>0. ∃k. ∀n≥k. u n ≥ M - ε))"
    using assms(2)
    by (simp add: supremo_def)
  then have "∃k. ∀n≥k. u n ≥ M - ε"
    using <0 < ε> by presburger
  then obtain n0 where "∀n≥n0. u n ≥ M - ε"
    by (rule exE)
  then have "∀n≥n0. |u n - M| ≤ ε"
    using hM by auto
  then show "∃k. ∀n≥k. |u n - M| ≤ ε"
    by (rule exI)
qed
end

```

4.23.2. Demostraciones con Lean

```

-- -----
-- Sea u una sucesión creciente. Demostrar que si M es un supremo de u,
-- entonces el límite de u es M.
-- -----

```

```

import data.real.basic

variable (u :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variable (M :  $\mathbb{R}$ )

notation '||x||' := abs x

-- (limite u c) expresa que el límite de u es c.
def limite (u :  $\mathbb{N} \rightarrow \mathbb{R}$ ) (c :  $\mathbb{R}$ ) :=
   $\forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - c| \leq \varepsilon$ 

-- (supremo u M) expresa que el supremo de u es M.
def supremo (u :  $\mathbb{N} \rightarrow \mathbb{R}$ ) (M :  $\mathbb{R}$ ) :=
  ( $\forall n, u n \leq M$ )  $\wedge \forall \varepsilon > 0, \exists n_0, u n_0 \geq M - \varepsilon$ 

-- 1ª demostración
example
  (hu : monotone u)
  (hM : supremo u M)
  : limite u M :=
begin
  -- unfold limite,
  intros  $\varepsilon$  h $\varepsilon$ ,
  -- unfold supremo at h,
  cases hM with hM1 hM2,
  cases hM2  $\varepsilon$  h $\varepsilon$  with n0 hn0,
  use n0,
  intros n hn,
  rw abs_le,
  split,
  { -- unfold monotone at h',
    specialize hu hn,
    calc - $\varepsilon$ 
      = (M -  $\varepsilon$ ) - M : by ring
      ...  $\leq u n_0$  - M : sub_le_sub_right hn0 M
      ...  $\leq u n$  - M : sub_le_sub_right hu M },
  { calc u n - M
       $\leq M$  - M : sub_le_sub_right (hM1 n) M
      ... = 0 : sub_self M
      ...  $\leq \varepsilon$  : le_of_lt h $\varepsilon$ , },
end

-- 2ª demostración
example

```

```

(hu : monotone u)
(hM : supremo u M)
: limite u M :=
begin
  intros ε hε,
  cases hM with hM1 hM2,
  cases hM2 ε hε with n0 hn0,
  use n0,
  intros n hn,
  rw abs_le,
  split,
  { linarith [hu hn] },
  { linarith [hM1 n] },
end

-- 3ª demostración
example
(hu : monotone u)
(hM : supremo u M)
: limite u M :=
begin
  intros ε hε,
  cases hM with hM1 hM2,
  cases hM2 ε hε with n0 hn0,
  use n0,
  intros n hn,
  rw abs_le,
  split ; linarith [hu hn, hM1 n],
end

-- 4ª demostración
example
(hu : monotone u)
(hM : supremo u M)
: limite u M :=
assume ε,
assume hε : ε > 0,
have hM1 : ∀ (n : ℕ), u n ≤ M,
  from hM.left,
have hM2 : ∀ (ε : ℝ), ε > 0 → (∃ (n0 : ℕ), u n0 ≥ M - ε),
  from hM.right,
exists.elim (hM2 ε hε)
( assume n0,
  assume hn0 : u n0 ≥ M - ε,
  have h1 : ∀ n, n ≥ n0 → |u n - M| ≤ ε,

```

```

{ assume n,
  assume hn : n ≥ n₀,
  have h2 : -ε ≤ u n - M,
    { have h3 : u n₀ ≤ u n,
      from hu hn,
      calc -ε
        = (M - ε) - M : by ring
        ... ≤ u n₀ - M : sub_le_sub_right hn₀ M
        ... ≤ u n - M : sub_le_sub_right h3 M },
  have h4 : u n - M ≤ ε,
    { calc u n - M
      ≤ M - M : sub_le_sub_right (hM₁ n) M
      ... = 0 : sub_self M
      ... ≤ ε : le_of_lt hε },
  show |u n - M| ≤ ε,
    from abs_le.mpr (and.intro h2 h4) },
show ∃ N, ∀ n, n ≥ N → |u n - M| ≤ ε,
from exists.intro n₀ h1)

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.24. Un número es par syss lo es su cuadrado

4.24.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que un número es par si y solo si lo es su cuadrado.
----- *)

theory Un_numero_es_par_syss_lo_es_su_cuadrado
imports Main
begin

(* 1ª demostración *)
lemma
  fixes n :: int
  shows "even (n2) ↔ even n"
proof (rule iffI)
  assume "even (n2)"
  show "even n"
proof (rule ccontr)
  assume "odd n"
  then obtain k where "n = 2*k+1"

```



```

    by (rule oddE)
  then have "n↑2 = 2*(2*k*(k+1))+1"
  proof -
    have "n↑2 = (2*k+1)↑2"
      by (simp add: <n = 2 * k + 1>)
    also have "... = 4*k↑2+4*k+1"
      by algebra
    also have "... = 2*(2*k*(k+1))+1"
      by algebra
    finally show "n↑2 = 2*(2*k*(k+1))+1" .
  qed
  then have "∃k'. n↑2 = 2*k'+1"
    by (rule exI)
  then have "odd (n↑2)"
    by fastforce
  then show False
    using <even (n↑2)> by blast
  qed
next
  assume "even n"
  then obtain k where "n = 2*k"
    by (rule evenE)
  then have "n↑2 = 2*(2*k↑2)"
    by simp
  then show "even (n↑2)"
    by simp
  qed

(* 2ª demostración *)
lemma
  fixes n :: int
  shows "even (n↑2) ↔ even n"
proof
  assume "even (n↑2)"
  show "even n"
  proof (rule ccontr)
    assume "odd n"
    then obtain k where "n = 2*k+1"
      by (rule oddE)
    then have "n↑2 = 2*(2*k*(k+1))+1"
      by algebra
    then have "odd (n↑2)"
      by simp
    then show False
      using <even (n↑2)> by blast
  qed

```

```

qed
next
  assume "even n"
  then obtain k where "n = 2*k"
    by (rule evenE)
  then have "n↑2 = 2*(2*k↑2)"
    by simp
  then show "even (n↑2)"
    by simp
qed

(* 3ª demostración *)
lemma
  fixes n :: int
  shows "even (n↑2) ↔ even n"
proof -
  have "even (n↑2) = (even n ∧ (0::nat) < 2)"
    by (simp only: even_power)
  also have "... = (even n ∧ True)"
    by (simp only: less_numeral_simps)
  also have "... = even n"
    by (simp only: HOL.simp_thms(21))
  finally show "even (n↑2) ↔ even n"
    by this
qed

(* 4ª demostración *)
lemma
  fixes n :: int
  shows "even (n↑2) ↔ even n"
proof -
  have "even (n↑2) = (even n ∧ (0::nat) < 2)"
    by (simp only: even_power)
  also have "... = even n"
    by simp
  finally show "even (n↑2) ↔ even n" .
qed

(* 5ª demostración *)
lemma
  fixes n :: int
  shows "even (n↑2) ↔ even n"
  by simp
end

```

4.24.2. Demostraciones con Lean

```
-- Demostrar que un número es par si y solo si lo es su cuadrado.
```

```
import data.int.parity
```

```
import tactic
```

```
open int
```

```
variable (n : ℤ)
```

```
-- 1ª demostración
```

```
example :
```

```
  even (n2) ↔ even n :=
```

```
begin
```

```
  split,
```

```
  { contrapose,
```

```
    rw ← odd_iff_not_even,
```

```
    rw ← odd_iff_not_even,
```

```
    unfold odd,
```

```
    intro h,
```

```
    cases h with k hk,
```

```
    use 2*k*(k+1),
```

```
    rw hk,
```

```
    ring, },
```

```
  { unfold even,
```

```
    intro h,
```

```
    cases h with k hk,
```

```
    use 2*k2,
```

```
    rw hk,
```

```
    ring, },
```

```
end
```

```
-- 2ª demostración
```

```
example :
```

```
  even (n2) ↔ even n :=
```

```
begin
```

```
  split,
```

```
  { contrapose,
```

```
    rw ← odd_iff_not_even,
```

```
    rw ← odd_iff_not_even,
```

```
    rintro (k, rfl),
```

```
    use 2*k*(k+1),
```

```
    ring, },
```

```

{ rintro (k, rfl),
  use 2*k^2,
  ring, },
end

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

-- 4ª demostración
example :
  even (n^2) ↔ even n :=
calc even (n^2)

```

```

    ↔ even (n * n)      : iff_of_eq (congr_arg even (sq n))
... ↔ (even n ∨ even n) : int.even_mul
... ↔ even n           : or_self (even n)

-- 5ª demostración
example :
  even (n^2) ↔ even n :=
calc even (n^2)
  ↔ even (n * n)      : by ring_nf
... ↔ (even n ∨ even n) : int.even_mul
... ↔ even n          : by simp

-- 6ª demostración
example :
  even (n^2) ↔ even n :=
begin
  split,
  { contrapose,
    intro h,
    rw ← odd_iff_not_even at *,
    cases h with k hk,
    use 2*k*(k+1),
    calc n^2
      = (2*k+1)^2      : by rw hk
    ... = 4*k^2+4*k+1   : by ring
    ... = 2*(2*k*(k+1))+1 : by ring, },
  { intro h,
    cases h with k hk,
    use 2*k^2,
    calc n^2
      = (2*k)^2      : by rw hk
    ... = 2*(2*k^2) : by ring, },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.25. Acotación de sucesiones convergente

4.25.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que si u es una sucesión convergente, entonces está
-- acotada; es decir,
--       $\exists k b. \forall n \geq k. |u\ n| \leq b$ 
----- *)
```

```
theory Acotacion_de_convergentes
```

```
imports Main HOL.Real
```

```
begin
```

```
(* (limite u c) expresa que el límite de u es c. *)
```

```
definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool" where
  "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k. \forall n \geq k. |u\ n - c| \leq \epsilon$ )"
```

```
(* (convergente u) expresa que u es convergente. *)
```

```
definition convergente :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  bool" where
  "convergente u  $\leftrightarrow$  ( $\exists a. \text{limite } u\ a$ )"
```

```
(* 1ª demostración *)
```

```
lemma
```

```
  assumes "convergente u"
```

```
  shows   " $\exists k b. \forall n \geq k. |u\ n| \leq b$ "
```

```
proof -
```

```
  obtain a where "limite u a"
```

```
    using assms convergente_def by blast
```

```
  then obtain k where hk : " $\forall n \geq k. |u\ n - a| \leq 1$ "
```

```
    using limite_def zero_less_one by blast
```

```
  have " $\forall n \geq k. |u\ n| \leq 1 + |a|$ "
```

```
  proof (intro allI impI)
```

```
    fix n
```

```
    assume hn : " $n \geq k$ "
```

```
    have " $|u\ n| = |u\ n - a + a|$ " by simp
```

```
    also have " $\dots \leq |u\ n - a| + |a|$ " by simp
```

```
    also have " $\dots \leq 1 + |a|$ " by (simp add: hk hn)
```

```
    finally show " $|u\ n| \leq 1 + |a|$ " .
```

```
  qed
```

```
  then show " $\exists k b. \forall n \geq k. |u\ n| \leq b$ "
```

```
    by (intro exI)
```

```
qed
```

```
(* 2ª demostración *)
```

```
lemma
```

```
  assumes "convergente u"
```

```
  shows   " $\exists k b. \forall n \geq k. |u\ n| \leq b$ "
```

```
proof -
```

```

obtain a where "limite u a"
  using assms convergente_def by blast
then obtain k where hk : "∀n≥k. |u n - a| ≤ 1"
  using limite_def zero_less_one by blast
have "∀n≥k. |u n| ≤ 1 + |a|"
  using hk by fastforce
then show "∃ k b. ∀n≥k. |u n| ≤ b"
  by auto
qed
end

```

4.25.2. Demostraciones con Lean

```

-----
-- Demostrar que si u es una sucesión convergente, entonces está
-- acotada; es decir,
--   ∃ k b. ∀n≥k. |u n| ≤ b
-----

```

```
import data.real.basic
```

```
variable {u : ℕ → ℝ}
```

```
variable {a : ℝ}
```

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

```
-- (limite u c) expresa que el límite de u es c.
```

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

```
-- (convergente u) expresa que u es convergente.
```

```
def convergente (u : ℕ → ℝ) :=
  ∃ a, limite u a
```

```
-- 1ª demostración
```

```
example
```

```
(h : convergente u)
```

```
: ∃ k b, ∀ n, n ≥ k → |u n| ≤ b :=
```

```
begin
```

```
cases h with a ua,
```

```
cases ua 1 zero_lt_one with k h,
```

```
use [k, 1 + |a|],
```

```

intros n hn,
specialize h n hn,
calc |u n|
  = |u n - a + a|      : congr_arg abs (eq_add_of_sub_eq rfl)
  ... ≤ |u n - a| + |a| : abs_add (u n - a) a
  ... ≤ 1 + |a|        : add_le_add_right h _
end

-- 2ª demostración
example
  (h : convergente u)
  : ∃ k b, ∀ n, n ≥ k → |u n| ≤ b :=
begin
  cases h with a ua,
  cases ua 1 zero_lt_one with k h,
  use [k, 1 + |a|],
  intros n hn,
  specialize h n hn,
  calc |u n|
    = |u n - a + a|      : by ring_nf
    ... ≤ |u n - a| + |a| : abs_add (u n - a) a
    ... ≤ 1 + |a|        : by linarith,
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.26. La paradoja del barbero

4.26.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar la paradoja del barbero https://bit.ly/3eWYvVw es decir,
-- que no existe un hombre que afeite a todos los que no se afeitan a sí
-- mismo y sólo a los que no se afeitan a sí mismo.
-- ----- *)

theory La_paradoja_del_barbero
imports Main
begin

(* 1ª demostración *)
lemma
  "¬(∃ x::'H. ∀ y::'H. afeita x y ↔ ¬ afeita y y)"

```



```

proof (rule notI)
  assume "∃ x. ∀ y. afeitado x y ↔ ¬ afeitado y y"
  then obtain b where "∀ y. afeitado b y ↔ ¬ afeitado y y"
    by (rule exE)
  then have h : "afeitado b b ↔ ¬ afeitado b b"
    by (rule allE)
  show False
proof (cases "afeitado b b")
  assume "afeitado b b"
  then have "¬ afeitado b b"
    using h by (rule rev_iffD1)
  then show False
    using <afeitado b b> by (rule notE)
next
  assume "¬ afeitado b b"
  then have "afeitado b b"
    using h by (rule rev_iffD2)
  with <¬ afeitado b b> show False
    by (rule notE)
qed
qed

(* 2ª demostración *)
lemma
  "¬(∃ x::'H. ∀ y::'H. afeitado x y ↔ ¬ afeitado y y)"
proof
  assume "∃ x. ∀ y. afeitado x y ↔ ¬ afeitado y y"
  then obtain b where "∀ y. afeitado b y ↔ ¬ afeitado y y"
    by (rule exE)
  then have h : "afeitado b b ↔ ¬ afeitado b b"
    by (rule allE)
  then show False
    by simp
qed

(* 3ª demostración *)
lemma
  "¬(∃ x::'H. ∀ y::'H. afeitado x y ↔ ¬ afeitado y y)"
  by auto

end

```

4.26.2. Demostraciones con Lean

```

-----
-- Demostrar la paradoja del barbero https://bit.ly/3eWYvVw es decir,
-- que no existe un hombre que afeite a todos los que no se afeitan a sí
-- mismo y sólo a los que no se afeitan a sí mismo.
-----

```

```
import tactic
```

```
variable (Hombre : Type)
```

```
variable (afeita : Hombre → Hombre → Prop)
```

```
-- 1ª demostración
```

```
example :
```

```
¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
```

```
begin
```

```
  intro h,
```

```
  cases h with b hb,
```

```
  specialize hb b,
```

```
  by_cases (afeita b b),
```

```
  { apply absurd h,
```

```
    exact hb.mp h, },
```

```
  { apply h,
```

```
    exact hb.mpr h, },
```

```
end
```

```
-- 2ª demostración
```

```
example :
```

```
¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
```

```
begin
```

```
  intro h,
```

```
  cases h with b hb,
```

```
  specialize hb b,
```

```
  by_cases (afeita b b),
```

```
  { exact (hb.mp h) h, },
```

```
  { exact h (hb.mpr h), },
```

```
end
```

```
-- 3ª demostración
```

```
example :
```

```
¬(∃ x : Hombre, ∀ y : Hombre, afeita x y ↔ ¬ afeita y y) :=
```

```
begin
```

```
  intro h,
```

```
  cases h with b hb,
```

```

    specialize hb b,
  by itauto,
end

-- 4ª demostración
example :
  ¬ (∃ x : Hombre, ∀ y : Hombre, afeitado x y ↔ ¬ afeitado y y) :=
begin
  rintro (b, hb),
  exact (iff_not_self (afeitado b b)).mp (hb b),
end

-- 5ª demostración
example :
  ¬ (∃ x : Hombre, ∀ y : Hombre, afeitado x y ↔ ¬ afeitado y y) :=
λ (b, hb), (iff_not_self (afeitado b b)).mp (hb b)

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.27. Propiedad de la densidad de los reales

4.27.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Sean x, y números reales tales que
--   ∀ z, y < z → x ≤ z
-- Demostrar que x ≤ y.
-- ----- *)

theory Propiedad_de_la_densidad_de_los_reales
imports Main HOL.Real
begin

(* 1ª demostración *)
lemma
  fixes x y :: real
  assumes "∀ z. y < z → x ≤ z"
  shows "x ≤ y"
proof (rule linorder_class.leI; intro notI)
  assume "y < x"
  then have "∃ z. y < z ∧ z < x"
    by (rule dense)

```

```

then obtain a where ha : "y < a ∧ a < x"
  by (rule exE)
have "¬ a < a"
  by (rule order.irrefl)
moreover
have "a < a"
proof -
  have "y < a → x ≤ a"
    using assms by (rule allE)
  moreover
  have "y < a"
    using ha by (rule conjunct1)
  ultimately have "x ≤ a"
    by (rule mp)
  moreover
  have "a < x"
    using ha by (rule conjunct2)
  ultimately show "a < a"
    by (simp only: less_le_trans)
qed
ultimately show False
  by (rule notE)
qed

(* 2ª demostración *)
lemma
  fixes x y :: real
  assumes "∃ z. y < z ⇒ x ≤ z"
  shows "x ≤ y"
proof (rule linorder_class.leI; intro notI)
  assume "y < x"
  then have "∃ z. y < z ∧ z < x"
    by (rule dense)
  then obtain a where hya : "y < a" and hax : "a < x"
    by auto
  have "¬ a < a"
    by (rule order.irrefl)
  moreover
  have "a < a"
  proof -
    have "a < x"
      using hax .
    also have "... ≤ a"
      using assms[OF hya] .
    finally show "a < a" .
  qed

```

```

qed
ultimately show False
  by (rule notE)
qed

(* 3ª demostración *)
lemma
  fixes x y :: real
  assumes "⋀ z. y < z ⟹ x ≤ z"
  shows "x ≤ y"
proof (rule linorder_class.leI; intro notI)
  assume "y < x"
  then have "∃z. y < z ∧ z < x"
    by (rule dense)
  then obtain a where hya : "y < a" and hax : "a < x"
    by auto
  have "¬ a < a"
    by (rule order.irrefl)
  moreover
  have "a < a"
    using hax assms[OF hya] by (rule less_le_trans)
  ultimately show False
    by (rule notE)
qed

(* 4ª demostración *)
lemma
  fixes x y :: real
  assumes "⋀ z. y < z ⟹ x ≤ z"
  shows "x ≤ y"
by (meson assms dense not_less)

(* 5ª demostración *)
lemma
  fixes x y :: real
  assumes "⋀ z. y < z ⟹ x ≤ z"
  shows "x ≤ y"
using assms by (rule dense_ge)

(* 6ª demostración *)
lemma
  fixes x y :: real
  assumes "∀ z. y < z → x ≤ z"
  shows "x ≤ y"
using assms by (simp only: dense_ge)

```

end

4.27.2. Demostraciones con Lean

```
-- Sean x, y números reales tales que
--    $\forall z, y < z \rightarrow x \leq z$ 
-- Demostrar que  $x \leq y$ .
```

```
import data.real.basic
```

```
variables {x y : ℝ}
```

```
-- 1ª demostración
```

```
example
```

```
(h :  $\forall z, y < z \rightarrow x \leq z$ ) :
```

```
x ≤ y :=
```

```
begin
```

```
  apply le_of_not_gt,
```

```
  intro hxy,
```

```
  cases (exists_between hxy) with a ha,
```

```
  apply (lt_irrefl a),
```

```
  calc a
```

```
    < x : ha.2
```

```
  ... ≤ a : h a ha.1,
```

```
end
```

```
-- 2ª demostración
```

```
example
```

```
(h :  $\forall z, y < z \rightarrow x \leq z$ ) :
```

```
x ≤ y :=
```

```
begin
```

```
  apply le_of_not_gt,
```

```
  intro hxy,
```

```
  cases (exists_between hxy) with a ha,
```

```
  apply (lt_irrefl a),
```

```
  exact lt_of_lt_of_le ha.2 (h a ha.1),
```

```
end
```

```
-- 3ª demostración
```

```
example
```

```
(h :  $\forall z, y < z \rightarrow x \leq z$ ) :
```

```

x ≤ y :=
begin
  apply le_of_not_gt,
  intro hxy,
  cases (exists_between hxy) with a ha,
  exact (lt_irrefl a) (lt_of_lt_of_le ha.2 (h a ha.1)),
end

-- 3ª demostración
example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
begin
  apply le_of_not_gt,
  intro hxy,
  rcases (exists_between hxy) with (a, ha),
  exact (lt_irrefl a) (lt_of_lt_of_le ha.2 (h a ha.1)),
end

-- 4ª demostración
example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
begin
  apply le_of_not_gt,
  intro hxy,
  rcases (exists_between hxy) with (a, haya, hax),
  exact (lt_irrefl a) (lt_of_lt_of_le hax (h a haya)),
end

-- 5ª demostración
example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
le_of_not_gt (λ hxy,
  let (a, haya, hax) := exists_between hxy in
  lt_irrefl a (lt_of_lt_of_le hax (h a haya)))

-- 6ª demostración
example
  (h : ∀ z, y < z → x ≤ z) :
  x ≤ y :=
le_of_forall_le_of_dense h

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.28. Propiedad cancelativa del producto de números naturales

4.28.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Sean  $k, m, n$  números naturales. Demostrar que
--  $k * m = k * n \leftrightarrow m = n \vee k = 0$ 
*----- *)

theory Propiedad_cancelativa_del_producto_de_numeros_naturales
imports Main
begin

(* 1ª demostración *)
Lemma
  fixes  $k\ m\ n :: \text{nat}$ 
  shows " $k * m = k * n \leftrightarrow m = n \vee k = 0$ "
proof -
  have " $k \neq 0 \implies k * m = k * n \implies m = n$ "
  proof (induct n arbitrary: m)
    fix m
    assume " $k \neq 0$ " and " $k * m = k * 0$ "
    show " $m = 0$ "
    using  $\langle k * m = k * 0 \rangle$ 
    by (simp only: mult_left_cancel[OF  $\langle k \neq 0 \rangle$ ])
  next
    fix n m
    assume HI : " $\square m. \llbracket k \neq 0; k * m = k * n \rrbracket \implies m = n$ "
    and hk : " $k \neq 0$ "
    and " $k * m = k * \text{Suc } n$ "
    then show " $m = \text{Suc } n$ "
    proof (cases m)
      assume " $m = 0$ "
      then show " $m = \text{Suc } n$ "
      using  $\langle k * m = k * \text{Suc } n \rangle$ 
      by (simp only: mult_left_cancel[OF  $\langle k \neq 0 \rangle$ ])
    next
      fix m'
      assume " $m = \text{Suc } m'$ "
      then have " $k * \text{Suc } m' = k * \text{Suc } n$ "
      using  $\langle k * m = k * \text{Suc } n \rangle$  by (rule subst)
      then have " $k * m' + k = k * n + k$ "

```



```

    by (simp only: mult_Suc_right)
  then have "k * m' = k * n"
    by (simp only: add_right_imp_eq)
  then have "m' = n"
    by (simp only: HI[OF hk])
  then show "m = Suc n"
    by (simp only: <m = Suc m'>)
qed
qed
then show "k * m = k * n ↔ m = n ∨ k = 0"
  by auto
qed

(* 2ª demostración *)
lemma
  fixes k m n :: nat
  shows "k * m = k * n ↔ m = n ∨ k = 0"
proof -
  have "k ≠ 0 ⇒ k * m = k * n ⇒ m = n"
  proof (induct n arbitrary: m)
    fix m
    assume "k ≠ 0" and "k * m = k * 0"
    then show "m = 0" by simp
  next
    fix n m
    assume "⌊m. ⌊k ≠ 0; k * m = k * n⌋ ⇒ m = n"
      and "k ≠ 0"
      and "k * m = k * Suc n"
    then show "m = Suc n"
    proof (cases m)
      assume "m = 0"
      then show "m = Suc n"
        using <k * m = k * Suc n> <k ≠ 0> by auto
    next
      fix m'
      assume "m = Suc m'"
      then show "m = Suc n"
        using <k * m = k * Suc n> <k ≠ 0> by force
    qed
  qed
  then show "k * m = k * n ↔ m = n ∨ k = 0" by auto
qed

(* 3ª demostración *)
lemma

```

```

fixes k m n :: nat
shows "k * m = k * n ↔ m = n ∨ k = 0"
proof -
  have "k ≠ 0 ⇒ k * m = k * n ⇒ m = n"
  proof (induct n arbitrary: m)
    case 0
    then show ?case
      by simp
  next
    case (Suc n)
    then show ?case
    proof (cases m)
      case 0
      then show ?thesis
        using Suc.prem by auto
    next
      case (Suc nat)
      then show ?thesis
        using Suc.prem by auto
    qed
  qed
  then show ?thesis
    by auto
qed

(* 4ª demostración *)
lemma
  fixes k m n :: nat
  shows "k * m = k * n ↔ m = n ∨ k = 0"
proof -
  have "k ≠ 0 ⇒ k * m = k * n ⇒ m = n"
  proof (induct n arbitrary: m)
    case 0
    then show "m = 0" by simp
  next
    case (Suc n)
    then show "m = Suc n"
      by (cases m) (simp_all add: eq_commute [of 0])
  qed
  then show ?thesis by auto
qed

(* 5ª demostración *)
lemma
  fixes k m n :: nat

```

```

shows "k * m = k * n ↔ m = n ∨ k = 0"
by (simp only: mult_cancel1)

(* 6ª demostración *)
lemma
  fixes k m n :: nat
  shows "k * m = k * n ↔ m = n ∨ k = 0"
by simp

end

```

4.28.2. Demostraciones con Lean

```

-----
-- Sean k, m, n números naturales. Demostrar que
--   k * m = k * n ↔ m = n ∨ k = 0
-----

import data.nat.basic
open nat

variables {k m n : ℕ}

-- Para que no use la notación con puntos
set_option pp.structure_projections false

-- 1ª demostración
example :
  k * m = k * n ↔ m = n ∨ k = 0 :=
begin
  have h1: k ≠ 0 → k * m = k * n → m = n,
  { induction n with n HI generalizing m,
    { by finish, },
    { cases m,
      { by finish, },
      { intros hk hS,
        congr,
        apply HI hk,
        rw mul_succ at hS,
        rw mul_succ at hS,
        exact add_right_cancel hS, }}}},
  by finish,
end

```

```
-- 2ª demostración
example :
  k * m = k * n ↔ m = n ∨ k = 0 :=
begin
  have h1: k ≠ 0 → k * m = k * n → m = n,
    { induction n with n HI generalizing m,
      { by finish, },
      { cases m,
        { by finish, },
        { intros hk hS,
          congr,
          apply HI hk,
          simp only [mul_succ] at hS,
          exact add_right_cancel hS, }}}},
    by finish,
end
```

```
-- 3ª demostración
example :
  k * m = k * n ↔ m = n ∨ k = 0 :=
begin
  have h1: k ≠ 0 → k * m = k * n → m = n,
    { induction n with n HI generalizing m,
      { by finish, },
      { cases m,
        { by finish, },
        { by finish, }}}},
    by finish,
end
```

```
-- 4ª demostración
example :
  k * m = k * n ↔ m = n ∨ k = 0 :=
begin
  have h1: k ≠ 0 → k * m = k * n → m = n,
    { induction n with n HI generalizing m,
      { by finish, },
      { cases m; by finish }},
    by finish,
end
```

```
-- 5ª demostración
example :
  k * m = k * n ↔ m = n ∨ k = 0 :=
```

```

begin
  have h1:  $k \neq 0 \rightarrow k * m = k * n \rightarrow m = n$ ,
    { induction n with n HI generalizing m ; by finish },
  by finish,
end

-- 5ª demostración
example :
   $k * m = k * n \leftrightarrow m = n \vee k = 0 :=$ 
begin
  by_cases hk :  $k = 0$ ,
  { by simp, },
  { rw mul_right_inj' hk,
    by tauto, },
end

-- 6ª demostración
example :
   $k * m = k * n \leftrightarrow m = n \vee k = 0 :=$ 
mul_eq_mul_left_iff

-- 7ª demostración
example :
   $k * m = k * n \leftrightarrow m = n \vee k = 0 :=$ 
by simp

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.29. Límite de sucesión menor que otra sucesión

4.29.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar
-- mediante una función ( $u : \mathbb{N} \rightarrow \mathbb{R}$ ) de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
--     where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0. \exists k :: nat. \forall n \geq k. |u\ n - c| < \epsilon$ )"
--
-- Demostrar que si  $u(n) \rightarrow a, v(n) \rightarrow c$  y  $u(n) \leq v(n)$  para todo  $n$ ,

```

```

-- entonces  $a \leq c$ .
----- *)

theory Limite_de_sucesion_menor_que_otra_sucesion
imports Main HOL.Real
begin

definition limite :: "(nat  $\Rightarrow$  real)  $\Rightarrow$  real  $\Rightarrow$  bool"
  where "limite u c  $\leftrightarrow$  ( $\forall \epsilon > 0$ .  $\exists k :: \text{nat}$ .  $\forall n \geq k$ .  $|u\ n - c| < \epsilon$ )"

(* 1ª demostración *)
lemma
  assumes "limite u a"
        "limite v c"
        " $\forall n$ .  $u\ n \leq v\ n$ "
  shows "a  $\leq$  c"
proof (rule leI ; intro notI)
  assume "c < a"
  let ? $\epsilon$  = "(a - c) / 2"
  have "0 < ? $\epsilon$ "
    using <c < a> by simp
  obtain Nu where HNu : " $\forall n \geq Nu$ .  $|u\ n - a| < ?\epsilon$ "
    using assms(1) limite_def <0 < ? $\epsilon$ > by blast
  obtain Nv where HNV : " $\forall n \geq Nv$ .  $|v\ n - c| < ?\epsilon$ "
    using assms(2) limite_def <0 < ? $\epsilon$ > by blast
  let ?N = "max Nu Nv"
  have "?N  $\geq$  Nu"
    by simp
  then have Ha : " $|u\ ?N - a| < ?\epsilon$ "
    using HNu by simp
  have "?N  $\geq$  Nv"
    by simp
  then have Hc : " $|v\ ?N - c| < ?\epsilon$ "
    using HNV by simp
  have "a - c < a - c"
  proof -
    have "a - c = (a - u ?N) + (u ?N - c)"
      by simp
    also have "...  $\leq$  (a - u ?N) + (v ?N - c)"
      using assms(3) by auto
    also have "...  $\leq$  |(a - u ?N) + (v ?N - c)|"
      by (rule abs_ge_self)
    also have "...  $\leq$  |a - u ?N| + |v ?N - c|"
      by (rule abs_triangle_ineq)
    also have "... = |u ?N - a| + |v ?N - c|"

```

```

    by (simp only: abs_minus_commute)
  also have "... < ?ε + ?ε"
    using Ha Hc by (simp only: add_strict_mono)
  also have "... = a - c"
    by (rule field_sum_of_halves)
  finally show "a - c < a - c"
    by this
qed
have "¬ a - c < a - c"
  by (rule less_irrefl)
then show False
  using <a - c < a - c> by (rule notE)
qed

(* 2ª demostración *)
lemma
  assumes "limite u a"
    "limite v c"
    "∀n. u n ≤ v n"
  shows "a ≤ c"
proof (rule leI ; intro notI)
  assume "c < a"
  let ?ε = "(a - c) / 2"
  have "0 < ?ε"
    using <c < a> by simp
  obtain Nu where HNu : "∀n ≥ Nu. |u n - a| < ?ε"
    using assms(1) limite_def <0 < ?ε> by blast
  obtain Nv where HNV : "∀n ≥ Nv. |v n - c| < ?ε"
    using assms(2) limite_def <0 < ?ε> by blast
  let ?N = "max Nu Nv"
  have "?N ≥ Nu"
    by simp
  then have Ha : "|u ?N - a| < ?ε"
    using HNu by simp
  then have Ha' : "u ?N - a < ?ε ∧ -(u ?N - a) < ?ε"
    by argo
  have "?N ≥ Nv"
    by simp
  then have Hc : "|v ?N - c| < ?ε"
    using HNV by simp
  then have Hc' : "v ?N - c < ?ε ∧ -(v ?N - c) < ?ε"
    by argo
  have "a - c < a - c"
    using assms(3) Ha' Hc'
    by (smt (verit, best) field_sum_of_halves)

```

```

have "¬ a - c < a - c"
  by simp
then show False
  using <a - c < a - c> by simp
qed

(* 3ª demostración *)
lemma
  assumes "limite u a"
    "limite v c"
    "∀n. u n ≤ v n"
  shows "a ≤ c"
proof (rule leI ; intro notI)
  assume "c < a"
  let ?ε = "(a - c) / 2"
  have "0 < ?ε"
    using <c < a> by simp
  obtain Nu where HNu : "∀n ≥ Nu. |u n - a| < ?ε"
    using assms(1) limite_def <0 < ?ε> by blast
  obtain Nv where HNV : "∀n ≥ Nv. |v n - c| < ?ε"
    using assms(2) limite_def <0 < ?ε> by blast
  let ?N = "max Nu Nv"
  have "?N ≥ Nu"
    by simp
  then have Ha : "|u ?N - a| < ?ε"
    using HNu by simp
  then have Ha' : "u ?N - a < ?ε ∧ -(u ?N - a) < ?ε"
    by argo
  have "?N ≥ Nv"
    by simp
  then have Hc : "|v ?N - c| < ?ε"
    using HNV by simp
  then have Hc' : "v ?N - c < ?ε ∧ -(v ?N - c) < ?ε"
    by argo
  show False
    using assms(3) Ha' Hc'
    by (smt (verit, best) field_sum_of_halves)
qed

end

```


4.29.2. Demostraciones con Lean

```

-- En Lean, una sucesión  $u_0, u_1, u_2, \dots$  se puede representar mediante
-- una función  $(u : \mathbb{N} \rightarrow \mathbb{R})$  de forma que  $u(n)$  es  $u_n$ .
--
-- Se define que  $a$  es el límite de la sucesión  $u$ , por
--   def limite :  $(\mathbb{N} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \text{Prop} :=$ 
--    $\lambda u a, \forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - a| < \varepsilon$ 
-- donde se usa la notación  $|x|$  para el valor absoluto de  $x$ 
--   notation '||x||' := abs x
--
-- Demostrar que si  $u_n \rightarrow a$ ,  $v_n \rightarrow c$  y  $u_n \leq v_n$  para todo  $n$ , entonces
--  $a \leq c$ .

```

```

import data.real.basic
import tactic

variables (u v :  $\mathbb{N} \rightarrow \mathbb{R}$ )
variables (a c :  $\mathbb{R}$ )

notation '||x||' := abs x

def limite (u :  $\mathbb{N} \rightarrow \mathbb{R}$ ) (c :  $\mathbb{R}$ ) :=
 $\forall \varepsilon > 0, \exists N, \forall n \geq N, |u n - c| < \varepsilon$ 

-- 1ª demostración
example
  (hu : limite u a)
  (hv : limite v c)
  (hle :  $\forall n, u n \leq v n$ )
  :  $a \leq c$  :=
begin
  apply le_of_not_lt,
  intro hlt,
  set  $\varepsilon := (a - c) / 2$  with hεac,
  have hε :  $0 < \varepsilon$  :=
    half_pos (sub_pos.mpr hlt),
  cases hu  $\varepsilon$  hε with Nu HNu,
  cases hv  $\varepsilon$  hε with Nv HNv,
  let N := max Nu Nv,
  have HNu' :  $Nu \leq N$  := le_max_left Nu Nv,
  have HNv' :  $Nv \leq N$  := le_max_right Nu Nv,
  have Ha :  $|u N - a| < \varepsilon$  := HNu N HNu',

```

```

have Hc :  $|v\ N - c| < \varepsilon := \text{HNv}\ N\ \text{HNv}'$ ,
have HN :  $u\ N \leq v\ N := \text{hle}\ N$ ,
apply lt_irrefl (a - c),
calc a - c
      = (a - u N) + (u N - c) : by ring
    ...  $\leq (a - u\ N) + (v\ N - c)$  : by simp [HN]
    ...  $\leq |(a - u\ N) + (v\ N - c)|$  : le_abs_self ((a - u N) + (v N - c))
    ...  $\leq |a - u\ N| + |v\ N - c|$  : abs_add (a - u N) (v N - c)
    ... =  $|u\ N - a| + |v\ N - c|$  : by simp only [abs_sub]
    ...  $< \varepsilon + \varepsilon$  : add_lt_add Ha Hc
    ... = a - c : add_halves (a - c),
end

-- 2ª demostración
example
  (hu : limite u a)
  (hv : limite v c)
  (hle :  $\forall\ n, u\ n \leq v\ n$ )
  : a  $\leq$  c :=
begin
  apply le_of_not_lt,
  intro hlt,
  set  $\varepsilon := (a - c) / 2$  with h $\varepsilon$ ,
  cases hu  $\varepsilon$  (by linarith) with Nu HNu,
  cases hv  $\varepsilon$  (by linarith) with Nv HNV,
  let N := max Nu Nv,
  have Ha :  $|u\ N - a| < \varepsilon :=$ 
    HNu N (le_max_left Nu Nv),
  have Hc :  $|v\ N - c| < \varepsilon :=$ 
    HNV N (le_max_right Nu Nv),
  have HN :  $u\ N \leq v\ N := \text{hle}\ N$ ,
  apply lt_irrefl (a - c),
  calc a - c
      = (a - u N) + (u N - c) : by ring
    ...  $\leq (a - u\ N) + (v\ N - c)$  : by simp [HN]
    ...  $\leq |(a - u\ N) + (v\ N - c)|$  : le_abs_self ((a - u N) + (v N - c))
    ...  $\leq |a - u\ N| + |v\ N - c|$  : abs_add (a - u N) (v N - c)
    ... =  $|u\ N - a| + |v\ N - c|$  : by simp only [abs_sub]
    ...  $< \varepsilon + \varepsilon$  : add_lt_add Ha Hc
    ... = a - c : add_halves (a - c),
end

-- 3ª demostración
example
  (hu : limite u a)

```

```

(hv : limite v c)
(hle :  $\forall n, u\ n \leq v\ n$ )
: a  $\leq$  c :=
begin
  apply le_of_not_lt,
  intro hlt,
  set  $\varepsilon := (a - c) / 2$  with h $\varepsilon$ ,
  cases hu  $\varepsilon$  (by linarith) with Nu HNu,
  cases hv  $\varepsilon$  (by linarith) with Nv HNv,
  let N := max Nu Nv,
  have Ha :  $|u\ N - a| < \varepsilon :=$ 
    HNu N (le_max_left Nu Nv),
  have Hc :  $|v\ N - c| < \varepsilon :=$ 
    HNv N (le_max_right Nu Nv),
  have HN :  $u\ N \leq v\ N := hle\ N$ ,
  apply lt_irrefl (a - c),
  calc a - c
    = (a - u N) + (u N - c) : by ring
    ...  $\leq (a - u N) + (v N - c)$  : by simp [HN]
    ...  $\leq |(a - u N) + (v N - c)|$  : by simp [le_abs_self]
    ...  $\leq |a - u N| + |v N - c|$  : by simp [abs_add]
    ... =  $|u N - a| + |v N - c|$  : by simp [abs_sub]
    ...  $< \varepsilon + \varepsilon$  : add_lt_add Ha Hc
    ... = a - c : by simp,
end

-- 4ª demostración
example
(hu : limite u a)
(hv : limite v c)
(hle :  $\forall n, u\ n \leq v\ n$ )
: a  $\leq$  c :=
begin
  apply le_of_not_lt,
  intro hlt,
  set  $\varepsilon := (a - c) / 2$  with h $\varepsilon$ ,
  cases hu  $\varepsilon$  (by linarith) with Nu HNu,
  cases hv  $\varepsilon$  (by linarith) with Nv HNv,
  let N := max Nu Nv,
  have Ha :  $|u\ N - a| < \varepsilon :=$ 
    HNu N (le_max_left Nu Nv),
  have Hc :  $|v\ N - c| < \varepsilon :=$ 
    HNv N (le_max_right Nu Nv),
  have HN :  $u\ N \leq v\ N := hle\ N$ ,
  apply lt_irrefl (a - c),

```

```
rw abs_lt at Ha Hc,
linarith,
end
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.30. Las sucesiones acotadas por cero son nulas

4.30.1. Demostraciones con Isabelle/HOL

```
(* -----
-- Demostrar que las sucesiones acotadas por cero son nulas.
-- ----- *)

theory Las_sucesiones_acotadas_por_cero_son_nulas
imports Main HOL.Real
begin

(* 1ª demostración *)
lemma
  fixes a :: "nat ⇒ real"
  assumes "∀n. |a n| ≤ 0"
  shows "∀n. a n = 0"
proof (rule allI)
  fix n
  have "|a n| = 0"
  proof (rule antisym)
    show "|a n| ≤ 0"
      using assms by (rule allE)
  next
    show "0 ≤ |a n|"
      by (rule abs_ge_zero)
  qed
  then show "a n = 0"
    by (simp only: abs_eq_0_iff)
qed

(* 2ª demostración *)
lemma
  fixes a :: "nat ⇒ real"
  assumes "∀n. |a n| ≤ 0"
```

```

shows    "∀n. a n = 0"
proof (rule allI)
  fix n
  have "|a n| = 0"
  proof (rule antisym)
    show "|a n| ≤ 0" try
      using assms by (rule allE)
  next
    show "0 ≤ |a n|"
      by simp
  qed
  then show "a n = 0"
    by simp
qed

(* 3ª demostración *)
lemma
  fixes a :: "nat ⇒ real"
  assumes "∀n. |a n| ≤ 0"
  shows   "∀n. a n = 0"
proof (rule allI)
  fix n
  have "|a n| = 0"
    using assms by auto
  then show "a n = 0"
    by simp
qed

(* 4ª demostración *)
lemma
  fixes a :: "nat ⇒ real"
  assumes "∀n. |a n| ≤ 0"
  shows   "∀n. a n = 0"
using assms by auto
end

```

4.30.2. Demostraciones con Lean

```

-- -----
-- Demostrar que las sucesiones acotadas por cero son nulas.
-- -----

```

```

import data.real.basic
import tactic

variable (u :  $\mathbb{N} \rightarrow \mathbb{R}$ )

notation '||x||' := abs x

-- 1ª demostración
example
  (h :  $\forall n, ||u n|| \leq 0$ )
  :  $\forall n, u n = 0 :=$ 
begin
  intro n,
  rw ← abs_eq_zero,
  specialize h n,
  apply le_antisymm,
  { exact h, },
  { exact abs_nonneg (u n), },
end

-- 2ª demostración
example
  (h :  $\forall n, ||u n|| \leq 0$ )
  :  $\forall n, u n = 0 :=$ 
begin
  intro n,
  rw ← abs_eq_zero,
  specialize h n,
  exact le_antisymm h (abs_nonneg (u n)),
end

-- 3ª demostración
example
  (h :  $\forall n, ||u n|| \leq 0$ )
  :  $\forall n, u n = 0 :=$ 
begin
  intro n,
  rw ← abs_eq_zero,
  exact le_antisymm (h n) (abs_nonneg (u n)),
end

-- 4ª demostración
example
  (h :  $\forall n, ||u n|| \leq 0$ )
  :  $\forall n, u n = 0 :=$ 

```

```

begin
  intro n,
  exact abs_eq_zero.mp (le_antisymm (h n) (abs_nonneg (u n))),
end

-- 5ª demostración
example
  (h : ∀ n, |u n| ≤ 0)
  : ∀ n, u n = 0 :=
λ n, abs_eq_zero.mp (le_antisymm (h n) (abs_nonneg (u n)))

-- 6ª demostración
example
  (h : ∀ n, |u n| ≤ 0)
  : ∀ n, u n = 0 :=
by finish

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

4.31. Producto de una sucesión acotada por otra convergente a cero

4.31.1. Demostraciones con Isabelle/HOL

```

(* -----
-- Demostrar que el producto de una sucesión acotada por una convergente
-- a 0 también converge a 0.
----- *)

theory Producto_de_una_sucesion_acotada_por_otra_convergente_a_cero
imports Main HOL.Real
begin

definition limite :: "(nat ⇒ real) ⇒ real ⇒ bool"
  where "limite u c ↔ (∀ε>0. ∃k::nat. ∀n≥k. |u n - c| < ε)"

definition acotada :: "(nat ⇒ real) ⇒ bool"
  where "acotada u ↔ (∃B. ∀n. |u n| ≤ B)"

lemma
  assumes "acotada u"
          "limite v 0"

```

```

shows "limite (λn. u n * v n) 0"
proof -
  obtain B where hB : "∀n. |u n| ≤ B"
    using assms(1) acotada_def by auto
  then have hBnoneg : "0 ≤ B" by auto
  show "limite (λn. u n * v n) 0"
  proof (cases "B = 0")
    assume "B = 0"
    show "limite (λn. u n * v n) 0"
    proof (unfold limite_def; intro allI impI)
      fix ε :: real
      assume "0 < ε"
      have "∀n ≥ 0. |u n * v n - 0| < ε"
      proof (intro allI impI)
        fix n :: nat
        assume "n ≥ 0"
        show "|u n * v n - 0| < ε"
          using <0 < ε> <B = 0> hB by auto
      qed
      then show "∃k. ∀n ≥ k. |u n * v n - 0| < ε"
        by (rule exI)
    qed
  next
    assume "B ≠ 0"
    then have hBpos : "0 < B"
      using hBnoneg by auto
    show "limite (λn. u n * v n) 0"
    proof (unfold limite_def; intro allI impI)
      fix ε :: real
      assume "0 < ε"
      then have "0 < ε/B"
        by (simp add: hBpos)
      then obtain N where hN : "∀n ≥ N. |v n - 0| < ε/B"
        using assms(2) limite_def by auto
      have "∀n ≥ N. |u n * v n - 0| < ε"
      proof (intro allI impI)
        fix n :: nat
        assume "n ≥ N"
        have "|v n| < ε/B"
          using <N ≤ n> hN by auto
        have "|u n * v n - 0| = |u n| * |v n|"
          by (simp add: abs_mult)
        also have "... ≤ B * |v n|"
          by (simp add: hB mult_right_mono)
        also have "... < B * (ε/B)"

```



```

    using <|v n| < ε/B> hBpos
    by (simp only: mult_strict_left_mono)
  also have "... = ε"
    using <B ≠ 0> by simp
  finally show "|u n * v n - 0| < ε"
    by this
qed
then show "∃k. ∀n≥k. |u n * v n - 0| < ε"
  by (rule exI)
qed
qed
qed
end

```

4.31.2. Demostraciones con Lean

```

-- -----
-- Demostrar que el producto de una sucesión acotada por una convergente
-- a 0 también converge a 0.
-- -----

import data.real.basic
import tactic

variables (u v : ℕ → ℝ)
variable (a : ℝ)

notation '||x||' := abs x

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

def acotada (a : ℕ → ℝ) :=
  ∃ B, ∀ n, |a n| ≤ B

-- 1ª demostración
example
  (hU : acotada u)
  (hV : limite v 0)
  : limite (u*v) 0 :=
begin
  cases hU with B hB,

```

```

have hBnoneg : 0 ≤ B,
  calc 0 ≤ |u 0| : abs_nonneg (u 0)
    ... ≤ B      : hB 0,
by_cases hB0 : B = 0,
{ subst hB0,
  intros ε hε,
  use 0,
  intros n hn,
  simp_rw [sub_zero] at *,
  calc |(u * v) n|
    = |u n * v n| : congr_arg abs (pi.mul_apply u v n)
    ... = |u n| * |v n| : abs_mul (u n) (v n)
    ... ≤ 0 * |v n| : mul_le_mul_of_nonneg_right (hB n) (abs_nonneg (v n))
    ... = 0 : zero_mul (|v n|)
    ... < ε : hε, },
{ change B ≠ 0 at hB0,
  have hBpos : 0 < B := (ne.le_iff_lt hB0.symm).mp hBnoneg,
  intros ε hε,
  cases hV (ε/B) (div_pos hε hBpos) with N hN,
  use N,
  intros n hn,
  simp_rw [sub_zero] at *,
  calc |(u * v) n|
    = |u n * v n| : congr_arg abs (pi.mul_apply u v n)
    ... = |u n| * |v n| : abs_mul (u n) (v n)
    ... ≤ B * |v n| : mul_le_mul_of_nonneg_right (hB n) (abs_nonneg _)
    ... < B * (ε/B) : mul_lt_mul_of_pos_left (hN n hn) hBpos
    ... = ε : mul_div_cancel' ε hB0 },
end

-- 2ª demostración
example
  (hU : acotada u)
  (hV : limite v 0)
  : limite (u*v) 0 :=
begin
  cases hU with B hB,
  have hBnoneg : 0 ≤ B,
    calc 0 ≤ |u 0| : abs_nonneg (u 0)
      ... ≤ B      : hB 0,
  by_cases hB0 : B = 0,
  { subst hB0,
    intros ε hε,
    use 0,
    intros n hn,

```

```

simp_rw [sub_zero] at *,
calc |(u * v) n|
    = |u n| * |v n| : by finish [abs_mul]
... ≤ 0 * |v n|      : mul_le_mul_of_nonneg_right (hB n) (abs_nonneg (v n))
... = 0              : by ring
... < ε              : hε, },
{ change B ≠ 0 at hB0,
  have hBpos : 0 < B := (ne.le_iff_lt hB0.symm).mp hBnoneg,
  intros ε hε,
  cases hV (ε/B) (div_pos hε hBpos) with N hN,
  use N,
  intros n hn,
  simp_rw [sub_zero] at *,
  calc |(u * v) n|
      = |u n| * |v n| : by finish [abs_mul]
... ≤ B * |v n|      : mul_le_mul_of_nonneg_right (hB n) (abs_nonneg _)
... < B * (ε/B)      : by finish
... = ε              : mul_div_cancel' ε hB0 },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

Capítulo 5

Ejercicios de agosto de 2021

5.1. La congruencia módulo 2 es una relación de equivalencia

5.1.1. Demostraciones con Isabelle/HOL

```
(* -----  
-- Se define la relación R entre los números enteros de forma que x está  
-- relacionado con y si x-y es divisible por 2. Demostrar que R es una  
-- relación de equivalencia.  
----- *)  
  
theory La_congruencia_modulo_2_es_una_relacion_de_equivalencia  
imports Main  
begin  
  
definition R :: "(int × int) set" where  
  "R = {(m, n). even (m - n)}"  
  
lemma R_iff [simp]:  
  "((x, y) ∈ R) = even (x - y)"  
by (simp add: R_def)  
  
(* 1ª demostración *)  
lemma "equiv UNIV R"  
proof (rule equivI)  
  show "refl R"  
  proof (unfold refl_on_def; intro conjI)  
    show "R ⊆ UNIV × UNIV"  
    proof -
```

```

    have "R ⊆ UNIV"
      by (rule top.extremum)
    also have "... = UNIV × UNIV"
      by (rule Product_Type.UNIV_Times_UNIV[symmetric])
    finally show "R ⊆ UNIV × UNIV"
      by this
  qed
next
  show "∀x∈UNIV. (x, x) ∈ R"
  proof
    fix x :: int
    assume "x ∈ UNIV"
    have "even 0" by (rule even_zero)
    then have "even (x - x)" by (simp only: diff_self)
    then show "(x, x) ∈ R"
      by (simp only: R_iff)
  qed
qed
next
  show "sym R"
  proof (unfold sym_def; intro allI impI)
    fix x y :: int
    assume "(x, y) ∈ R"
    then have "even (x - y)"
      by (simp only: R_iff)
    then show "(y, x) ∈ R"
    proof (rule evenE)
      fix a :: int
      assume ha : "x - y = 2 * a"
      have "y - x = -(x - y)"
        by (rule minus_diff_eq[symmetric])
      also have "... = -(2 * a)"
        by (simp only: ha)
      also have "... = 2 * (-a)"
        by (rule mult_minus_right[symmetric])
      finally have "y - x = 2 * (-a)"
        by this
      then have "even (y - x)"
        by (rule dvdI)
      then show "(y, x) ∈ R"
        by (simp only: R_iff)
    qed
  qed
qed
next
  show "trans R"

```

```

proof (unfold trans_def; intro allI impI)
  fix x y z
  assume hxy : "(x, y) ∈ R" and hyz : "(y, z) ∈ R"
  have "even (x - y)"
    using hxy by (simp only: R_iff)
  then obtain a where ha : "x - y = 2 * a"
    by (rule dvdE)
  have "even (y - z)"
    using hyz by (simp only: R_iff)
  then obtain b where hb : "y - z = 2 * b"
    by (rule dvdE)
  have "x - z = (x - y) + (y - z)"
    by simp
  also have "... = (2 * a) + (2 * b)"
    by (simp only: ha hb)
  also have "... = 2 * (a + b)"
    by (simp only: distrib_left)
  finally have "x - z = 2 * (a + b)"
    by this
  then have "even (x - z)"
    by (rule dvdI)
  then show "(x, z) ∈ R"
    by (simp only: R_iff)
qed
qed

(* 2ª demostración *)
lemma "equiv UNIV R"
proof (rule equivI)
  show "refl R"
  proof (unfold refl_on_def; intro conjI)
    show "R ⊆ UNIV × UNIV" by simp
  next
    show "∀x∈UNIV. (x, x) ∈ R"
    proof
      fix x :: int
      assume "x ∈ UNIV"
      have "x - x = 2 * 0"
        by simp
      then show "(x, x) ∈ R"
        by simp
    qed
  qed
next
  show "sym R"

```

```

proof (unfold sym_def; intro allI impI)
  fix x y :: int
  assume "(x, y) ∈ R"
  then have "even (x - y)"
    by simp
  then obtain a where ha : "x - y = 2 * a"
    by blast
  then have "y - x = 2 * (-a)"
    by simp
  then show "(y, x) ∈ R"
    by simp
qed
next
show "trans R"
proof (unfold trans_def; intro allI impI)
  fix x y z
  assume hxy : "(x, y) ∈ R" and hyz : "(y, z) ∈ R"
  have "even (x - y)"
    using hxy by simp
  then obtain a where ha : "x - y = 2 * a"
    by blast
  have "even (y - z)"
    using hyz by simp
  then obtain b where hb : "y - z = 2 * b"
    by blast
  have "x - z = 2 * (a + b)"
    using ha hb by auto
  then show "(x, z) ∈ R"
    by simp
qed
qed

(* 3ª demostración *)
lemma "equiv UNIV R"
proof (rule equivI)
  show "refl R"
  proof (unfold refl_on_def; intro conjI)
    show "R ⊆ UNIV × UNIV"
    by simp
  next
    show "∀x∈UNIV. (x, x) ∈ R"
    by simp
  qed
next
show "sym R"

```



```

proof (unfold sym_def; intro allI impI)
  fix x y
  assume "(x, y) ∈ R"
  then show "(y, x) ∈ R"
    by simp
qed
next
show "trans R"
proof (unfold trans_def; intro allI impI)
  fix x y z
  assume "(x, y) ∈ R" and "(y, z) ∈ R"
  then show "(x, z) ∈ R"
    by simp
qed
qed

(* 4ª demostración *)
lemma "equiv UNIV R"
proof (rule equivI)
  show "refl R"
    unfolding refl_on_def by simp
next
  show "sym R"
    unfolding sym_def by simp
next
  show "trans R"
    unfolding trans_def by simp
qed

(* 5ª demostración *)
lemma "equiv UNIV R"
  unfolding equiv_def refl_on_def sym_def trans_def
  by simp

(* 6ª demostración *)
lemma "equiv UNIV R"
  by (simp add: equiv_def refl_on_def sym_def trans_def)

end

```

5.1.2. Demostraciones con Lean

 -- Se define la relación R entre los números enteros de forma que x está
 -- relacionado con y si $x-y$ es divisible por 2. Demostrar que R es una
 -- relación de equivalencia.

```
import data.int.basic
```

```
import tactic
```

```
def R (m n :  $\mathbb{Z}$ ) := 2  $\mid$  (m - n)
```

-- 1ª demostración

```
example : equivalence R :=
```

```
begin
```

```
  repeat {split},
```

```
  { intro x,
```

```
    unfold R,
```

```
    rw sub_self,
```

```
    exact dvd_zero 2, },
```

```
  { intros x y hxy,
```

```
    unfold R,
```

```
    cases hxy with a ha,
```

```
    use -a,
```

```
    calc y - x
```

```
      = -(x - y) : (neg_sub x y).symm
```

```
    ... = -(2 * a) : by rw ha
```

```
    ... = 2 * -a : neg_mul_eq_mul_neg 2 a, },
```

```
  { intros x y z hxy hyz,
```

```
    cases hxy with a ha,
```

```
    cases hyz with b hb,
```

```
    use a + b,
```

```
    calc x - z
```

```
      = (x - y) + (y - z) : (sub_add_sub_cancel x y z).symm
```

```
    ... = 2 * a + 2 * b : congr_arg2 ((+)) ha hb
```

```
    ... = 2 * (a + b) : (mul_add 2 a b).symm , },
```

```
end
```

-- 2ª demostración

```
example : equivalence R :=
```

```
begin
```

```
  repeat {split},
```

```
  { intro x,
```

```
    simp [R], },
```

```

{ rintros x y ⟨a, ha⟩,
  use -a,
  linarith, },
{ rintros x y z ⟨a, ha⟩ ⟨b, hb⟩,
  use a + b,
  linarith, },
end

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.2. Las funciones con inversa por la izquierda son inyectivas

5.2.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- izquierda por
--   definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa_izq f ⇔ (∃g. ∀x. g (f x) = x)"
-- Además, que f es inyectiva sobre un conjunto está definido por
--   definition inj_on :: "('a ⇒ 'b) ⇒ 'a set ⇒ bool"
--     where "inj_on f A ⇔ (∀x∈A. ∀y∈A. f x = f y → x = y)"
-- y que f es inyectiva por
--   abbreviation inj :: "('a ⇒ 'b) ⇒ bool"
--     where "inj f ≡ inj_on f UNIV"
--
-- Demostrar que si f tiene inversa por la izquierda, entonces f es
-- inyectiva.
-- ----- *)

theory Las_funciones_con_inversa_por_la_izquierda_son_inyectivas
imports Main
begin

definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_izq f ⇔ (∃g. ∀x. g (f x) = x)"

(* 1ª demostración *)
lemma
  assumes "tiene_inversa_izq f"
  shows   "inj f"

```

```

proof (unfold inj_def; intro allI impI)
  fix x y
  assume "f x = f y"
  obtain g where hg : "∀x. g (f x) = x"
    using assms tiene_inversa_izq_def by auto
  have "x = g (f x)"
    by (simp only: hg)
  also have "... = g (f y)"
    by (simp only: <f x = f y>)
  also have "... = y"
    by (simp only: hg)
  finally show "x = y" .
qed

(* 2ª demostración *)
lemma
  assumes "tiene_inversa_izq f"
  shows "inj f"
  by (metis assms inj_def tiene_inversa_izq_def)

end

```

5.2.2. Demostraciones con Lean

```

-----
-- En Lean, que g es una inversa por la izquierda de f está definido por
--   left_inverse (g : β → α) (f : α → β) : Prop :=
--     ∀ x, g (f x) = x
-- y que f tenga inversa por la izquierda está definido por
--   has_left_inverse (f : α → β) : Prop :=
--     ∃ finv : β → α, left_inverse finv f
-- Finalmente, que f es inyectiva está definido por
--   injective (f : α → β) : Prop :=
--     ∀ [x y], f x = f y → x = y
--
-- Demostrar que si f tiene inversa por la izquierda, entonces f es
-- inyectiva.
-----

import tactic
open function

universes u v

```

```

variables {α : Type u}
variable {β : Type v}
variable {f : α → β}

-- 1ª demostración
example
  (hf : has_left_inverse f)
  : injective f :=
begin
  intros x y hxy,
  unfold has_left_inverse at hf,
  unfold left_inverse at hf,
  cases hf with g hg,
  calc x = g (f x) : (hg x).symm
      ... = g (f y) : congr_arg g hxy
      ... = y      : hg y
end

-- 2ª demostración
example
  (hf : has_left_inverse f)
  : injective f :=
begin
  intros x y hxy,
  cases hf with g hg,
  calc x = g (f x) : (hg x).symm
      ... = g (f y) : congr_arg g hxy
      ... = y      : hg y
end

-- 3ª demostración
example
  (hf : has_left_inverse f)
  : injective f :=
exists.elim hf (λ finv inv, inv.injective)

-- 4ª demostración
example
  (hf : has_left_inverse f)
  : injective f :=
has_left_inverse.injective hf

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.3. Las funciones inyectivas tienen inversa por la izquierda

5.3.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- izquierda por
--   definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa_izq f ⇔ (∃g. ∀x. g (f x) = x)"
-- Además, que f es inyectiva sobre un conjunto está definido por
--   definition inj_on :: "('a ⇒ 'b) ⇒ 'a set ⇒ bool"
--     where "inj_on f A ⇔ (∀x∈A. ∀y∈A. f x = f y → x = y)"
-- y que f es inyectiva por
--   abbreviation inj :: "('a ⇒ 'b) ⇒ bool"
--     where "inj f ≡ inj_on f UNIV"
--
-- Demostrar que si f es una función inyectiva, entonces f tiene
-- inversa por la izquierda.
-- ----- *)
```

```
theory Las_funciones_inyectivas_tienen_inversa_por_la_izquierda
imports Main
```

```
begin
```

```
definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_izq f ⇔ (∃g. ∀x. g (f x) = x)"
```

```
(* 1ª demostración *)
```

```
lemma
```

```
  assumes "inj f"
```

```
  shows   "tiene_inversa_izq f"
```

```
proof (unfold tiene_inversa_izq_def)
```

```
  let ?g = "(λy. SOME x. f x = y)"
```

```
  have "∀x. ?g (f x) = x"
```

```
  proof (rule allI)
```

```
    fix a
```

```
    have "∃x. f x = f a"
```

```
      by auto
```

```
    then have "f (?g (f a)) = f a"
```

```
      by (rule someI_ex)
```

```
    then show "?g (f a) = a"
```

```
      using assms
```

```

    by (simp only: injD)
  qed
  then show "( $\exists g. \forall x. g (f x) = x$ )"
    by (simp only: exI)
  qed

(* 2ª demostración *)
lemma
  assumes "inj f"
  shows   "tiene_inversa_izq f"
proof (unfold tiene_inversa_izq_def)
  have " $\forall x. inv f (f x) = x$ "
  proof (rule allI)
    fix x
    show " $inv f (f x) = x$ "
      using assms by (simp only: inv_f_f)
  qed
  then show "( $\exists g. \forall x. g (f x) = x$ )"
    by (simp only: exI)
  qed

(* 3ª demostración *)
lemma
  assumes "inj f"
  shows   "tiene_inversa_izq f"
proof (unfold tiene_inversa_izq_def)
  have " $\forall x. inv f (f x) = x$ "
    by (simp add: assms)
  then show "( $\exists g. \forall x. g (f x) = x$ )"
    by (simp only: exI)
  qed
end

```

5.3.2. Demostraciones con Lean

```

-- -----
-- En Lean, que  $g$  es una inversa por la izquierda de  $f$  está definido por
--   left_inverse (g :  $\beta \rightarrow \alpha$ ) (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--      $\forall x, g (f x) = x$ 
-- y que  $f$  tenga inversa por la izquierda está definido por
--   has_left_inverse (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--      $\exists finv : \beta \rightarrow \alpha, left\_inverse finv f$ 

```

```

-- Finalmente, que f es inyectiva está definido por
--   injective (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--        $\forall [x\ y],\ f\ x = f\ y \rightarrow x = y$ 
--
-- Demostrar que si f es una función inyectiva con dominio no vacío,
-- entonces f tiene inversa por la izquierda.
-----

import tactic
open function classical

variables { $\alpha\ \beta$ : Type*}
variable {f :  $\alpha \rightarrow \beta$ }

-- 1ª demostración
example
  [h $\alpha$  : nonempty  $\alpha$ ]
  (hf : injective f)
  : has_left_inverse f :=
begin
  classical,
  unfold has_left_inverse,
  let g :=  $\lambda\ y,$  if h :  $\exists\ x,$  f x = y then some h else choice h $\alpha$ ,
  use g,
  unfold left_inverse,
  intro a,
  have h1 :  $\exists\ x : \alpha,$  f x = f a := Exists.intro a rfl,
  dsimp at *,
  dsimp [g],
  rw dif_pos h1,
  apply hf,
  exact some_spec h1,
end

-- 2ª demostración
example
  [h $\alpha$  : nonempty  $\alpha$ ]
  (hf : injective f)
  : has_left_inverse f :=
begin
  classical,
  let g :=  $\lambda\ y,$  if h :  $\exists\ x,$  f x = y then some h else choice h $\alpha$ ,
  use g,
  intro a,
  have h1 :  $\exists\ x : \alpha,$  f x = f a := Exists.intro a rfl,

```



```
dsimp [g],
rw dif_pos h1,
exact hf (some_spec h1),
end

-- 3ª demostración
example
  [hα : nonempty α]
  (hf : injective f)
  : has_left_inverse f :=
begin
  unfold has_left_inverse,
  use inv_fun f,
  unfold left_inverse,
  intro x,
  apply hf,
  apply inv_fun_eq,
  use x,
end

-- 4ª demostración
example
  [hα : nonempty α]
  (hf : injective f)
  : has_left_inverse f :=
begin
  use inv_fun f,
  intro x,
  apply hf,
  apply inv_fun_eq,
  use x,
end

-- 5ª demostración
example
  [hα : nonempty α]
  (hf : injective f)
  : has_left_inverse f :=
(inv_fun f, left_inverse_inv_fun hf)

-- 6ª demostración
example
  [hα : nonempty α]
  (hf : injective f)
  : has_left_inverse f :=
```

```
injective.has_left_inverse hf
```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.4. Una función tiene inversa por la izquierda si y solo si es inyectiva

5.4.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- izquierda por
--   definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa_izq f ↔ (∃g. ∀x. g (f x) = x)"
-- Además, que f es inyectiva sobre un conjunto está definido por
--   definition inj_on :: "('a ⇒ 'b) ⇒ 'a set ⇒ bool"
--     where "inj_on f A ↔ (∀x∈A. ∀y∈A. f x = f y → x = y)"
-- y que f es inyectiva por
--   abbreviation inj :: "('a ⇒ 'b) ⇒ bool"
--     where "inj f ≡ inj_on f UNIV"
--
-- Demostrar que una función f, con dominio no vacío, tiene inversa por
-- la izquierda si y solo si es inyectiva.
-- ----- *)

theory Una_funcion_tiene_inversa_por_la_izquierda_si_y_solo_si_es_inyectiva
imports Main
begin

definition tiene_inversa_izq :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_izq f ↔ (∃g. ∀x. g (f x) = x)"

(* 1ª demostración *)
lemma
  "tiene_inversa_izq f ↔ inj f"
proof (rule iffI)
  assume "tiene_inversa_izq f"
  show "inj f"
proof (unfold inj_def; intro allI impI)
  fix x y
  assume "f x = f y"
  obtain g where hg : "∀x. g (f x) = x"

```

```

    using <tiene_inversa_izq f> tiene_inversa_izq_def
    by auto
  have "x = g (f x)"
    by (simp only: hg)
  also have "... = g (f y)"
    by (simp only: <f x = f y>)
  also have "... = y"
    by (simp only: hg)
  finally show "x = y" .
qed
next
  assume "inj f"
  show "tiene_inversa_izq f"
  proof (unfold tiene_inversa_izq_def)
    have "∀x. inv f (f x) = x"
    proof (rule allI)
      fix x
      show "inv f (f x) = x"
        using <inj f> by (simp only: inv_f_f)
    qed
    then show "(∃g. ∀x. g (f x) = x)"
      by (simp only: exI)
    qed
  qed

(* 2ª demostración *)
lemma
  "tiene_inversa_izq f ↔ inj f"
proof (rule iffI)
  assume "tiene_inversa_izq f"
  then show "inj f"
    by (metis inj_def tiene_inversa_izq_def)
next
  assume "inj f"
  then show "tiene_inversa_izq f"
    by (metis the_inv_f_f tiene_inversa_izq_def)
qed

(* 3ª demostración *)
lemma
  "tiene_inversa_izq f ↔ inj f"
by (metis tiene_inversa_izq_def inj_def the_inv_f_f)

end

```

5.4.2. Demostraciones con Lean

```

-----
-- En Lean, que  $g$  es una inversa por la izquierda de  $f$  está definido por
--    $\text{left\_inverse } (g : \beta \rightarrow \alpha) (f : \alpha \rightarrow \beta) : \text{Prop} :=$ 
--    $\forall x, g (f x) = x$ 
-- y que  $f$  tenga inversa por la izquierda está definido por
--    $\text{has\_left\_inverse } (f : \alpha \rightarrow \beta) : \text{Prop} :=$ 
--    $\exists \text{ finv} : \beta \rightarrow \alpha, \text{left\_inverse finv } f$ 
-- Finalmente, que  $f$  es inyectiva está definido por
--    $\text{injective } (f : \alpha \rightarrow \beta) : \text{Prop} :=$ 
--    $\forall \square x y \square, f x = f y \rightarrow x = y$ 
--
-- Demostrar que una función  $f$ , con dominio no vacío, tiene inversa por
-- la izquierda si y solo si es inyectiva.
-----

```

```

import tactic
open function

variables {α : Type*} [nonempty α]
variable {β : Type*}
variable {f : α → β}

-- 1ª demostración
example : has_left_inverse f ↔ injective f :=
begin
  split,
  { intro hf,
    intros x y hxy,
    cases hf with g hg,
    calc x = g (f x) : (hg x).symm
      ... = g (f y) : congr_arg g hxy
      ... = y       : hg y, },
  { intro hf,
    use inv_fun f,
    intro x,
    apply hf,
    apply inv_fun_eq,
    use x, },
end

-- 2ª demostración
example : has_left_inverse f ↔ injective f :=
begin

```

```

split,
{ intro hf,
  exact has_left_inverse.injective hf },
{ intro hf,
  exact injective.has_left_inverse hf },
end

-- 3ª demostración
example : has_left_inverse f ↔ injective f :=
  (has_left_inverse.injective, injective.has_left_inverse)

-- 4ª demostración
example : has_left_inverse f ↔ injective f :=
  injective_iff_has_left_inverse.symm

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.5. Las funciones con inversa por la derecha son suprayectivas

5.5.1. Demostraciones con Isabelle/HOL

```

(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- derecha por
--   definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa_dcha f ↔ (∃g. ∀y. f (g y) = y)"
--
-- Demostrar que si f es una función suprayectiva, entonces f tiene
-- inversa por la derecha.
-- ----- *)

theory Las_funciones_con_inversa_por_la_derecha_son_suprayectivas
imports Main
begin

definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_dcha f ↔ (∃g. ∀y. f (g y) = y)"

(* 1ª demostración *)
lemma
  assumes "tiene_inversa_dcha f"

```

```

shows "surj f"
proof (unfold surj_def; intro allI)
  fix y
  obtain g where "∀y. f (g y) = y"
    using assms tiene_inversa_dcha_def by auto
  then have "f (g y) = y"
    by (rule allE)
  then have "y = f (g y)"
    by (rule sym)
  then show "∃x. y = f x"
    by (rule exI)
qed

(* 2ª demostración *)
lemma
  assumes "tiene_inversa_dcha f"
  shows "surj f"
proof (unfold surj_def; intro allI)
  fix y
  obtain g where "∀y. f (g y) = y"
    using assms tiene_inversa_dcha_def by auto
  then have "y = f (g y)"
    by simp
  then show "∃x. y = f x"
    by (rule exI)
qed

(* 3ª demostración *)
lemma
  assumes "tiene_inversa_dcha f"
  shows "surj f"
proof (unfold surj_def; intro allI)
  fix y
  obtain g where "∀y. f (g y) = y"
    using assms tiene_inversa_dcha_def by auto
  then show "∃x. y = f x"
    by metis
qed

(* 4ª demostración *)
lemma
  assumes "tiene_inversa_dcha f"
  shows "surj f"
proof (unfold surj_def; intro allI)
  fix y

```

```

show "∃x. y = f x"
  using assms tiene_inversa_dcha_def
  by metis
qed

(* 5ª demostración *)
lemma
  assumes "tiene_inversa_dcha f"
  shows "surj f"
using assms tiene_inversa_dcha_def surj_def
by metis

end

```

5.5.2. Demostraciones con Lean

```

-----
-- En Lean, que g es una inversa por la izquierda de f está definido por
--   left_inverse (g : β → α) (f : α → β) : Prop :=
--     ∀ x, g (f x) = x
-- que g es una inversa por la derecha de f está definido por
--   right_inverse (g : β → α) (f : α → β) : Prop :=
--     left_inverse f g
-- y que f tenga inversa por la derecha está definido por
--   has_right_inverse (f : α → β) : Prop :=
--     ∃ g : β → α, right_inverse g f
-- Finalmente, que f es suprayectiva está definido por
--   def surjective (f : α → β) : Prop :=
--     ∀ b, ∃ a, f a = b
--
-- Demostrar que si la función f tiene inversa por la derecha, entonces
-- f es suprayectiva.
-----

import tactic
open function

variables {α β: Type*}
variable {f : α → β}

-- 1ª demostración
example
  (hf : has_right_inverse f)

```

```

: surjective f :=
begin
  unfold surjective,
  unfold has_right_inverse at hf,
  cases hf with g hg,
  intro b,
  use g b,
  exact hg b,
end

-- 2ª demostración
example
  (hf : has_right_inverse f)
  : surjective f :=
begin
  intro b,
  cases hf with g hg,
  use g b,
  exact hg b,
end

-- 3ª demostración
example
  (hf : has_right_inverse f)
  : surjective f :=
begin
  intro b,
  cases hf with g hg,
  use [g b, hg b],
end

-- 4ª demostración
example
  (hf : has_right_inverse f)
  : surjective f :=
has_right_inverse.surjective hf

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.6. Las funciones suprayectivas tienen inversa por la derecha

5.6.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- derecha por
-- definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
-- "tiene_inversa_dcha f ⇔ (∃g. ∀y. f (g y) = y)"
--
-- Demostrar que si f es una función suprayectiva, entonces f tiene
-- inversa por la derecha.
-- ----- *)

theory Las_funciones_suprayectivas_tienen_inversa_por_la_derecha
imports Main
begin

definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_dcha f ⇔ (∃g. ∀y. f (g y) = y)"

(* 1ª demostración *)
lemma
  assumes "surj f"
  shows "tiene_inversa_dcha f"
proof (unfold tiene_inversa_dcha_def)
  let ?g = "λy. SOME x. f x = y"
  have "∀y. f (?g y) = y"
  proof (rule allI)
    fix y
    have "∃x. y = f x"
      using assms by (rule surjD)
    then have "∃x. f x = y"
      by auto
    then show "f (?g y) = y"
      by (rule someI_ex)
  qed
  then show "∃g. ∀y. f (g y) = y"
    by auto
qed

(* 2ª demostración *)
```

```

lemma
  assumes "surj f"
  shows "tiene_inversa_dcha f"
proof (unfold tiene_inversa_dcha_def)
  let ?g = "λy. SOME x. f x = y"
  have "∀y. f (?g y) = y"
  proof (rule allI)
    fix y
    have "∃x. f x = y"
      by (metis assms surjD)
    then show "f (?g y) = y"
      by (rule someI_ex)
  qed
  then show "∃g. ∀y. f (g y) = y"
    by auto
qed

(* 3ª demostración *)
lemma
  assumes "surj f"
  shows "tiene_inversa_dcha f"
proof (unfold tiene_inversa_dcha_def)
  have "∀y. f (inv f y) = y"
    by (simp add: assms surj_f_inv_f)
  then show "∃g. ∀y. f (g y) = y"
    by auto
qed

(* 4ª demostración *)
lemma
  assumes "surj f"
  shows "tiene_inversa_dcha f"
  by (metis assms surjD tiene_inversa_dcha_def)

end

```

5.6.2. Demostraciones con Lean

```

-- -----
-- En Lean, que g es una inversa por la izquierda de f está definido por
--   left_inverse (g : β → α) (f : α → β) : Prop :=
--     ∀ x, g (f x) = x
-- que g es una inversa por la derecha de f está definido por

```

```

--      right_inverse (g :  $\beta \rightarrow \alpha$ ) (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--      left_inverse f g
-- y que f tenga inversa por la derecha está definido por
--      has_right_inverse (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--       $\exists$  g :  $\beta \rightarrow \alpha$ , right_inverse g f
-- Finalmente, que f es suprayectiva está definido por
--      def surjective (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--       $\forall$  b,  $\exists$  a, f a = b
--
-- Demostrar que si f es una función suprayectiva, entonces f tiene
-- inversa por la derecha.

```

```

import tactic
open function classical

variables { $\alpha$   $\beta$ : Type*}
variable {f :  $\alpha \rightarrow \beta$ }

-- 1ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
begin
  unfold has_right_inverse,
  let g :=  $\lambda$  y, some (hf y),
  use g,
  unfold function.right_inverse,
  unfold function.left_inverse,
  intro b,
  apply some_spec (hf b),
end

-- 2ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
begin
  let g :=  $\lambda$  y, some (hf y),
  use g,
  intro b,
  apply some_spec (hf b),
end

-- 3ª demostración

```

```

example
  (hf : surjective f)
  : has_right_inverse f :=
begin
  use surj_inv hf,
  intro b,
  exact surj_inv_eq hf b,
end

-- 4ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
begin
  use surj_inv hf,
  exact surj_inv_eq hf,
end

-- 5ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
begin
  use [surj_inv hf, surj_inv_eq hf],
end

-- 6ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
(surj_inv hf, surj_inv_eq hf)

-- 7ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
(⟦_, surj_inv_eq hf⟧)

-- 8ª demostración
example
  (hf : surjective f)
  : has_right_inverse f :=
surjective.has_right_inverse hf

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.7. Una función tiene inversa por la derecha si y solo si es suprayectiva

5.7.1. Demostraciones con Isabelle/HOL

```
(* -----
-- En Isabelle/HOL, se puede definir que f tenga inversa por la
-- derecha por
--   definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa_dcha f ⇔ (∃g. ∀y. f (g y) = y)"
--
-- Demostrar que la función f tiene inversa por la derecha si y solo si
-- es suprayectiva.
----- *)

theory Una_funcion_tiene_inversa_por_la_derecha_si_y_solo_si_es_suprayectiva
imports Main
begin

definition tiene_inversa_dcha :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa_dcha f ⇔ (∃g. ∀y. f (g y) = y)"

(* 1ª demostración *)
lemma
  "tiene_inversa_dcha f ⇔ surj f"
proof (rule iffI)
  assume hf : "tiene_inversa_dcha f"
  show "surj f"
  proof (unfold surj_def; intro allI)
    fix y
    obtain g where "∀y. f (g y) = y"
      using hf tiene_inversa_dcha_def by auto
    then have "f (g y) = y"
      by (rule allE)
    then have "y = f (g y)"
      by (rule sym)
    then show "∃x. y = f x"
      by (rule exI)
  qed
next
  assume hf : "surj f"
  show "tiene_inversa_dcha f"
  proof (unfold tiene_inversa_dcha_def)
```

```

let ?g = "λy. SOME x. f x = y"
have "∀y. f (?g y) = y"
proof (rule allI)
  fix y
  have "∃x. f x = y"
  by (metis hf surjD)
  then show "f (?g y) = y"
  by (rule someI_ex)
qed
then show "∃g. ∀y. f (g y) = y"
by auto
qed
qed

(* 2ª demostración *)
lemma
  "tiene_inversa_dcha f ↔ surj f"
proof (rule iffI)
  assume "tiene_inversa_dcha f"
  then show "surj f"
  using tiene_inversa_dcha_def surj_def
  by metis
next
  assume "surj f"
  then show "tiene_inversa_dcha f"
  by (metis surjD tiene_inversa_dcha_def)
qed
end

```

5.7.2. Demostraciones con Lean

```

-- -----
-- En Lean, que g es una inversa por la izquierda de f está definido por
--   left_inverse (g : β → α) (f : α → β) : Prop :=
--     ∀ x, g (f x) = x
-- que g es una inversa por la derecha de f está definido por
--   right_inverse (g : β → α) (f : α → β) : Prop :=
--     left_inverse f g
-- y que f tenga inversa por la derecha está definido por
--   has_right_inverse (f : α → β) : Prop :=
--     ∃ g : β → α, right_inverse g f
-- Finalmente, que f es suprayectiva está definido por

```

```

--      def surjective (f :  $\alpha \rightarrow \beta$ ) : Prop :=
--           $\forall b, \exists a, f\ a = b$ 
--
--      Demostrar que la función f tiene inversa por la derecha si y solo si
--      es suprayectiva.
--      -----

import tactic
open function classical

variables { $\alpha\ \beta$ : Type*}
variable {f :  $\alpha \rightarrow \beta$ }

-- 1ª demostración
example : has_right_inverse f  $\leftrightarrow$  surjective f :=
begin
  split,
  { intros hf b,
    cases hf with g hg,
    use g b,
    exact hg b, },
  { intro hf,
    let g :=  $\lambda\ y, \text{some } (hf\ y)$ ,
    use g,
    intro b,
    apply some_spec (hf b), },
end

-- 2ª demostración
example : has_right_inverse f  $\leftrightarrow$  surjective f :=
surjective_iff_has_right_inverse.symm

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

5.8. Las funciones con inversa son biyectivas

5.8.1. Demostraciones con Isabelle/HOL

```

(* -----
--      En Isabelle se puede definir que g es una inversa de f por
--      definition inversa :: "( $'a \Rightarrow 'b$ )  $\Rightarrow$  ( $'b \Rightarrow 'a$ )  $\Rightarrow$  bool" where
--          "inversa f g  $\leftrightarrow$  ( $\forall x. (g \circ f)\ x = x$ )  $\wedge$  ( $\forall y. (f \circ g)\ y = y$ )"

```

```

-- y que f tiene inversa por
--   definition tiene_inversa :: "('a ⇒ 'b) ⇒ bool" where
--     "tiene_inversa f ⇔ (∃ g. inversa f g)"
--
-- Demostrar que si la función f tiene inversa, entonces f es biyectiva.
-- ----- *)

theory Las_funciones_con_inversa_son_biyectivas
imports Main
begin

definition inversa :: "('a ⇒ 'b) ⇒ ('b ⇒ 'a) ⇒ bool" where
  "inversa f g ⇔ (∀ x. (g ∘ f) x = x) ∧ (∀ y. (f ∘ g) y = y)"

definition tiene_inversa :: "('a ⇒ 'b) ⇒ bool" where
  "tiene_inversa f ⇔ (∃ g. inversa f g)"

(* 1a demostración *)
lemma
  fixes f :: "'a ⇒ 'b"
  assumes "tiene_inversa f"
  shows   "bij f"
proof -
  obtain g where h1 : "∀ x. (g ∘ f) x = x" and
                h2 : "∀ y. (f ∘ g) y = y"
  by (meson assms inversa_def tiene_inversa_def)
  show "bij f"
proof (rule bijI)
  show "inj f"
proof (rule injI)
    fix x y
    assume "f x = f y"
    then have "g (f x) = g (f y)"
      by simp
    then show "x = y"
      using h1 by simp
  qed
next
  show "surj f"
proof (rule surjI)
    fix y
    show "f (g y) = y"
      using h2 by simp
  qed
qed

```



```

qed

(* 2ª demostración *)
lemma
  fixes f :: "'a ⇒ 'b"
  assumes "tiene_inversa f"
  shows   "bij f"
proof -
  obtain g where h1 : "∀ x. (g ∘ f) x = x" and
                h2 : "∀ y. (f ∘ g) y = y"
  by (meson assms inversa_def tiene_inversa_def)
  show "bij f"
proof (rule bijI)
  show "inj f"
proof (rule injI)
  fix x y
  assume "f x = f y"
  then have "g (f x) = g (f y)"
    by simp
  then show "x = y"
    using h1 by simp
qed
next
  show "surj f"
proof (rule surjI)
  fix y
  show "f (g y) = y"
    using h2 by simp
qed
qed
qed
end

```

5.8.2. Demostraciones con Lean

```

-- -----
-- En Lean se puede definir que g es una inversa de f por
--   def inversa (f : X → Y) (g : Y → X) :=
--     (∀ x, (g ∘ f) x = x) ∧ (∀ y, (f ∘ g) y = y)
-- y que f tiene inversa por
--   def tiene_inversa (f : X → Y) :=
--     ∃ g, inversa g f

```

```

--
-- Demostrar que si la función f tiene inversa, entonces f es biyectiva.
-----

import tactic
open function

variables {X Y : Type*}
variable (f : X → Y)

def inversa (f : X → Y) (g : Y → X) :=
  (∀ x, (g ∘ f) x = x) ∧ (∀ y, (f ∘ g) y = y)

def tiene_inversa (f : X → Y) :=
  ∃ g, inversa g f

-- 1ª demostración
example
  (hf : tiene_inversa f)
  : bijective f :=
begin
  rcases hf with ⟨g, ⟨h1, h2⟩⟩,
  split,
  { intros a b hab,
    calc a = g (f a) : (h2 a).symm
      ... = g (f b) : congr_arg g hab
      ... = b       : h2 b, },
  { intro y,
    use g y,
    exact h1 y, },
end

-- 2ª demostración
example
  (hf : tiene_inversa f)
  : bijective f :=
begin
  rcases hf with ⟨g, ⟨h1, h2⟩⟩,
  split,
  { intros a b hab,
    calc a = g (f a) : (h2 a).symm
      ... = g (f b) : congr_arg g hab
      ... = b       : h2 b, },
  { intro y,
    use [g y, h1 y], },

```

```

end

-- 3ª demostración
example
  (hf : tiene_inversa f)
  : bijective f :=
begin
  rcases hf with ⟨g, ⟨h1, h2⟩⟩,
  split,
  { exact left_inverse.injective h2, },
  { exact right_inverse.surjective h1, },
end

-- 4ª demostración
example
  (hf : tiene_inversa f)
  : bijective f :=
begin
  rcases hf with ⟨g, ⟨h1, h2⟩⟩,
  exact ⟨left_inverse.injective h2,
        right_inverse.surjective h1⟩,
end

-- 5ª demostración
example :
  tiene_inversa f → bijective f :=
begin
  rintros ⟨g, ⟨h1, h2⟩⟩,
  exact ⟨left_inverse.injective h2,
        right_inverse.surjective h1⟩,
end

-- 6ª demostración
example :
  tiene_inversa f → bijective f :=
λ ⟨g, ⟨h1, h2⟩⟩, ⟨left_inverse.injective h2,
                  right_inverse.surjective h1⟩

```

Se puede interactuar con las pruebas anteriores en [esta sesión con Lean](#).

Índice alfabético

Conjuntos

Diferencia, 17, 24, 42, 47, 128, 132
Intersección, 9, 13, 21, 29, 35, 39, 47, 59, 61, 65, 77, 120, 124, 135, 146

Intersección general, 65, 70, 158, 161, 168

Unión, 13, 21, 24, 35, 39, 42, 47, 56, 70, 83, 114, 142, 149

Unión general, 61, 153, 165

Funciones

biyectivas, 343

Imagen, 83, 91, 107, 120, 124, 128, 135, 142, 146, 149, 153, 158, 161

Imagen inversa, 77, 91, 94, 97, 100, 104, 110, 114, 132, 135, 142, 146, 149, 165, 168

inversa, 343

inversa por la derecha, 333, 337, 341

inversa por la izquierda, 323, 326, 330

inyectivas, 97, 124, 161, 323, 326, 330

suprayectivas, 104, 172, 333, 337, 341

Funciones reales

crecientes, 260, 264, 268

decrecientes, 264

involutivas, 268

inyectivas, 273

Grupos

cancelativa, 211

inversos, 200, 203, 207

neutro, 197

Lógica

de primer orden, 288

Monoides

cancelativos, 194

inversos, 177, 187, 191

neutro, 187

potencias, 181, 217

Números enteros

congruencia modular, 317

Paridad, 280

Números naturales

Paridad, 56, 59

Primos, 59

Números reales

densidad, 291

inducción, 296

orden, 291

Relaciones

de equivalencia, 317

Sucesiones

acotadas, 285, 308

convergentes, 285

crecientes, 275

límite, 226, 230, 234, 237, 243, 247, 251, 255, 275, 301, 311

supremo, 275

Teorema de Cantor, [172](#)

Teorema del emparedado, [255](#)

Bibliografía

- [1] J. A. Alonso. [Introducción al razonamiento automático con OTTER](#) ¹, 2006.
- [2] J. A. Alonso. [Introducción a la demostración asistida por ordenador \(con Isabelle/Isar\)](#) ², 2008.
- [3] J. A. Alonso. [Demostración asistida por ordenador con Isabelle/HOL \(curso 2013-14\)](#) ³, 2013.
- [4] J. A. Alonso. [Demostración asistida por ordenador con Coq](#) ⁴, 2018.
- [5] J. A. Alonso. [Demostración asistida por ordenador con Isabelle/HOL \(curso 2018-19\)](#) ⁵, 2018.
- [6] J. A. Alonso. [Lógica con Lean](#) ⁶, 2020.
- [7] J. A. Alonso. [Lean para matemáticos](#) ⁷, 2021.
- [8] J. A. Alonso. [Matemáticas en Lean](#) ⁸, 2021.
- [9] J. A. Alonso. [DAO \(Demostración Asistida por Ordenador\) con Lean](#) ⁹, 2021.
- [10] J. Avigad, K. Buzzard, R. Y. Lewis, and P. Massot. [Mathematics in Lean](#) ¹⁰, 2020.

¹<https://www.cs.us.es/~jalonso/publicaciones/2006-int-raz-aut-otter.pdf>

²<https://www.cs.us.es/~jalonso/publicaciones/2008-IDA0.pdf>

³https://www.cs.us.es/~jalonso/publicaciones/2013-Introduccion_a_la_demostracion_asistida_por_ordenador_con_IsabelleHOL.pdf

⁴<https://raw.githubusercontent.com/jaalonso/DAOconCoq/master/texto/DAOconCoq.pdf>

⁵<https://www.cs.us.es/~jalonso/publicaciones/2018-DAOconIsabelleHOL.pdf>

⁶https://raw.githubusercontent.com/jaalonso/Logica_con_Lean/master/Logica_con_Lean.pdf

⁷https://github.com/jaalonso/Lean_para_matematicos

⁸https://github.com/jaalonso/Matematicas_en_Lean

⁹https://raw.githubusercontent.com/jaalonso/DAO_con_Lean/master/DAO_con_Lean.pdf

¹⁰https://leanprover-community.github.io/mathematics_in_lean/

- [11] J. Avigad, L. de Moura, and S. Kong. [Theorem Proving in Lean](#) ¹¹, 2021.
- [12] J. Avigad, R. Y. Lewis, and F. van Doorn. [Logic and proof](#) ¹², 2020.
- [13] A. Baanen, A. Bentkamp, J. Blanchette, J. Hölzl, and J. Limperg. [The Hitchhiker's Guide to Logical Verification](#) ¹³, 2020.
- [14] K. Buzzard. [Course on formalising mathematics](#) ¹⁴, 2021.
- [15] K. Buzzard and M. Pedramfar. [The Natural Number Game, version 1.3.3](#) ¹⁵.
- [16] P. Massot. [Introduction aux mathématiques formalisées](#) ¹⁶.
- [17] Varios. [LFTCM 2020: Lean for the Curious Mathematician 2020](#) ¹⁷.
- [18] Varios. [Isabelle/HOL: Higher-Order Logic](#) ¹⁸, 2021.

¹¹https://leanprover.github.io/theorem_proving_in_lean/theorem_proving_in_lean.pdf

¹²https://leanprover.github.io/logic_and_proof

¹³https://raw.githubusercontent.com/blanchette/logical_verification_2020/master/hitchhikers_guide.pdf

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

¹⁵https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/

¹⁶<https://www.imo.universite-paris-saclay.fr/~pmassot/enseignement/math114/>

¹⁷<https://leanprover-community.github.io/lftcm2020/schedule.html>

¹⁸<https://isabelle.in.tum.de/dist/library/HOL/HOL/document.pdf>