Formalización en Lean de problemas de las Olimpiadas Internacionales de Matemáticas (IMO)

Sara Díaz Real

Departamento de Ciencias de la Computación e Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

El presente Trabajo Fin de Máster se ha realizado en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla y ha sido supervisado por José Antonio Alonso Jiménez y María José Hidalgo Doblado.

A mis padres, por su confianza incondicional.

A mi hermano, por ser el mejor ejemplo a seguir que puedo tener.

A José Antonio, por su dedicación y paciencia en este trabajo. Gracias por ser referente desde el día que empecé la carrera.

Índice general

Resumen

El principal objetivo de este trabajo es el estudio detallado de la elaboración de formalizaciones de pruebas con un sistema interactivo como es Lean.

En primer lugar, se han formalizado resultados correspondientes a las asignaturas de Cálculo Infinitesimal y Álgebra Básica del Grado en Matemáticas de la Universidad de Sevilla.

A continuación, se ha llevado a cabo la formalización de una selección variada de problemas pertenecientes a las Olimpiadas Internacionales de Matemáticas (IMO), de los cuales ya se habían planteado formalizaciones previamente.

Finalmente, se concluye el trabajo realizando la formalización de un problema, también correspondiente a las IMO, cuya formalización no había sido propuesta con anterioridad.

Abstract

The main aim of this work is to study in a detailed way how to make formalizations with a proof-assistant software such as Lean.

Firstly, some results belonging to the subjects Cálculo Infinitesimal and Álgebra Básica from the degree in Mathematics at the University of Seville have been formalized.

Later, a selection of problems from the International Mathematical Olympiad (IMO) has been formalized. All of which have been done before.

Finally, the work concluds with a formalization of a problem, taken from the IMO too, that hasn’t been formalized before.

Introducción

Las Olimpiadas Internacionales de Matemáticas (IMO)[1] son probablemente la competición de ejercicios de inteligencia con mayor número de celebraciones en el mundo y es por eso que se encuentra entre los últimos grandes objetivos de la Inteligencia Artificial.

En el proyecto IMO Grand Challenge se plantea construir un sistema de Inteligencia Artificial que sea capaz de ganar la medalla de oro en esta competición.

Con el objetivo de eliminar la ambigüedad que existe a la hora de comprobar si las diversas soluciones propuestas a los ejercicios son correctas o no y qué puntación hay que concederle a cada una, se plantea la variante de las IMO que se conoce como formal–to–formal (F2F), [14]. Básicamente, esta variante consiste en que recibe el enunciado y la solución al problema, ambos formalizados en Lean y comprueba si la solución proporcionada es correcta o no según las reglas estipuladas.

Lean es un demostrador interactivo, mediante el cual se pueden comprobar si los resultados o demostraciones que se proponen a un problema son correctas. Esta herramienta es muy útil pues hay situaciones en las que a los humanos se nos puede olvidar comentar un aspecto trivial de la demostración pero el cual es indispensable para la correcta compresión. Asimismo, hay situaciones en las que por motivos de extensión o complejidad se hace casi imposible poder realizar la comprobación correspondiente. En la actualidad, los demostradores interactivos son un tema muy a la orden del día y que en los últimos tiempos se ha convertido en una herramienta muy importante, como se comenta en [3].

De esta manera, se tiene que el reto que se quiere conseguir consiste en crear un sistema inteligente que sea capaz de obtener la puntuación en F2F necesaria para alcanzar la medalla de oro en la competición, en el caso de competir contra humanos.

[1] https://www.imo-official.org

Como consecuencia de todo esto, actualmente se están desarrollando multitud de trabajos y proyectos que se basan en la formalización de diversos problemas (no sólo de las IMO, también de otras competiciones) con ayuda de diferentes demostradores interactivos como Lean o Isabelle/HOL.

El proyecto MiniF2F, [13], ha comenzado a desarrollarse hace muy poco tiempo y consiste en formalizar problemas correspondientes a diferentes competiciones como las IMO pero también otras como American Mathematics Competitions (AMC). Asimismo, se propone la formalización de problemas de matemáticas de nivel escolar. En este proyecto las formalizaciones que se proponen son a través de varias herramientas, entre ellas Lean.

En el trabajo que aquí se propone, se estudia detalladamente diversas formalizaciones en Lean de problemas matemáticos correspondientes a las IMO. Para llevar a cabo este trabajo nos hemos basado en el artículo [10] en el que estudiantes de la Universidad de Belgrado estudiaron de manera teórica y formalizaron en el programa Isabelle/HOL problemas de las IMO.

Destacar que el trabajo que aquí se expone se encuentra disponible en su totalidad en el siguiente repositorio de GitHub:

Repositorio IMO en Lean

A continuación, se va a detallar la estructura que se ha propuesto en el trabajo:

Primer capítulo

El trabajo se ha comenzando con una **introducción a Lean**, (1), donde se ha descrito brevemente esta herramienta y también su librería matemática **mathlib**. Asimismo, en este capítulo se desarrolla la **formalización de una prueba** de manera muy detallada: explicando y justificando cada paso de la formalización, para que de esta manera se pueda comprender la estructura de las formalizaciones.

Segundo capítulo

A continuación, en el capítulo llamado **elementos de matemáticas en Lean**, (2), se plantean varias formalizaciones sobre lemas o teoremas que

han sido estudiados en asignaturas de formación básica correspondientes al Grado en Matemáticas de la Universidad de Sevilla. En concreto, los resultados son de las asignaturas de Cálculo Infinitesimal y Álgebra Básica. Estas formalizaciones fueron obtenidas con ayuda del tutorial [6] que realicé junto con The Natural Number Game[2].

Tercer capítulo

El siguiente capítulo es el más extenso del trabajo y es en el que se detallan varias **formalizaciones en Lean de problemas de las IMO** que ya han sido formalizados con anterioridad (3). Además de la correspondiente formalización en Lean, en cada problema se plantea su resolución en lenguaje natural para que el lector pueda entender y ver la analogía entre ambas. En concreto se han estudiado los siguientes problemas que se presentan a continuación:

IMO 1959 Q1

Demostrar que la fracción

$$\frac{21 \cdot n + 4}{14 \cdot n + 3}$$

es irreducible para cualquier número natural n.

Probar que una fracción es irreducible es equivalente a demostrar que el numerador y el denominador de dicha fracción son coprimos entre sí, esto es, que el máximo común divisor que estos poseen es 1. Esta es la demostración que se plantea en el trabajo.

La demostración se puede dividir en dos partes principales:

- En primer lugar, se prueba que si existe un número natural k que divide al numerador y que también divide al denominador, entonces dicho número k divide a 1. Esta prueba se formaliza en Lean mediante un lema auxiliar.

[2]The Natural Number Game es un curso en el que se explica la teoría de los números naturales diseñada a modo de juego de manera que a medida que avanzas en él se introducen más técnicas de demostración que se deben ir utilizando.

- En segundo lugar, una vez ya se tiene que el divisor común del numerador y el denominador divide a 1, se puede concluir el resultado deseado muy fácilmente: basta con usar un lema que se encuentra en una de las librerías de mathlib.

Como se verá en la sección 3.1 del trabajo de manera más detallada, el teorema principal mediante el cual se ha formalizado el ejercicio en Lean, es el que se presenta a continuación:

```
theorem imo1959_q1 : ∀ n : ℕ, coprime (21 * n + 4) (14 * n + 3)
```

Asimismo, como se ha comentado, en la formalización de este resultado se ha hecho uso de un lema auxiliar y es el que nos dice que si existe un número k que divida al numerador y al denominador de la fracción, entonces dicho número divide a 1. Su formalización en Lean ha sido necesaria para la conclusión del ejercicio y es la siguiente:

```
lemma Auxiliar
  (n k : ℕ)
  (h1 : k ∣ 21 * n + 4)
  (h2 : k ∣ 14 * n + 3)
  : k ∣ 1
```

IMO 1962 Q4

Resolver la ecuación

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1. \tag{1}$$

Este problema ha sido probablemente el más laborioso del trabajo: ha sido necesario demostrar y formalizar varios resultados y definiciones auxiliares.

En primer lugar se ha definido la propiedad **problema** para referirnos a la ecuación que se quiere resolver, esto es, (1). En Lean esto se ha formalizado de la siguiente manera:

```
def problema (x : R) : Prop :=
  cos x ^ 2 + cos (2 * x) ^ 2 + cos (3 * x) ^ 2 = 1
```

A continuación, se ha definido una función que hemos notado como **funAuxiliar** y que viene dada por la siguiente expresión:

$$\text{funAuxiliar}(x) = \cos(x) \cdot \left(\cos^2(x) - \frac{1}{2} \right) \cdot \cos(3x) \tag{2}$$

Esta definición se ha formalizado en Lean como:

```
def funauxiliar (x : ℝ) : ℝ :=
  cos x * (cos x ^ 2 - 1/2) * cos (3 * x)
```

El siguiente paso en la demostración, ha consistido en probar que resolver la ecuación que nos interesa es equivalente a resolver la expresión obtenida de igualar (2) a cero. En Lean esto lo hemos formalizado con el siguiente lema auxiliar:

```
lemma Equivalencia
  {x : ℝ}
  : problema x ↔ funauxiliar x = 0
```

Luego, se ha resuelto el problema equivalente. Se puede observar que como la **funAuxiliar** está factorizada e igualada a cero, hallar las correspondientes soluciones no es nada complicado. Es más, los dos casos en los que se verificará dicha ecuación será:

- El coseno al cuadrado de x sea igual a un medio.
- El coseno de $3x$ sea nulo.

Esta casuística ha sido formalizada en Lean a través del siguiente lema:

```
lemma CasosSolucion
  {x : R}
  : funauxiliar x = 0 ↔ cos x ^ 2 = 1/2 ∨ cos (3 * x) = 0
```

Una vez ya se ha demostrado en qué dos casos se verifica la expresión para resolver el problema, se ha estudiado cuáles son los números x que verifican dicha expresión. Esto se ha hecho con la formalización de los dos siguientes lemas:

```
lemma SolucionCosenoCuadrado
  {x : R}
  : cos x ^ 2 = 1/2 ↔ ∃ k : Z, x = (2 * k + 1) * π / 4

lemma SolucionCosenoTriple
  {x : R}
  : cos (3 * x) = 0 ↔ ∃ k : Z, x = (2 * k + 1) * π / 6
```

Se puede observar que al estar trabajando con funciones trigonométricas, los valores para los que se verifican las diferentes igualdades son bien conocidos. Por tanto, se tiene ahora que definiendo el conjunto formado por todos los números que verifican una de las dos posibilidades planteadas ya se tendría el conjunto de soluciones del problema. La formalización de dicho conjunto es la siguiente:

```
def Solucion : set R :=
  {x : R | ∃ k : Z, x = (2 * k + 1) * π / 4 ∨ x = (2 * k + 1) * π / 6}
```

Finalmente, ya se tiene la conclusión del ejercicio y es que el conjunto de soluciones del problema se corresponde con el definido justo antes. En Lean se ha formalizado de la siguiente manera:

```
theorem imo1962_q4
  {x : ℝ}
  : problema x ↔ x ∈ Solucion
```

IMO 1977 Q6

Considere la función $f : \mathbb{N}^+ \to \mathbb{N}^+$ satisfaciendo que

$$f(f(n)) < f(n+1)$$

para cualquier número n. Probar que para todo número natural positivo n se verifica que

$$f(n) = n.$$

El correspondiente enunciado en Lean ha sido formalizado de la siguiente manera:

```
theorem imo1977_q6
  (f : ℕ+ → ℕ+)
  (h : ∀ n, f (f n) < f (n + 1))
  : ∀ n, f n = n
```

Para llevar a cabo la demostración de este resultado sólo ha sido necesario el uso de un teorema auxiliar. Este teorema consiste en probar el mismo resultado que se plantea inicialmente y bajo las mismas hipótesis pero considerando que la función f está definida del conjunto de los naturales al conjunto de los naturales (es decir, no tienen porqué ser positivos). En Lean esto se ha formalizado como sigue:

```
theorem Extension
  (f : ℕ → ℕ)
  (h1 : ∀ n, f (f n) < f (n + 1))
  : ∀ n, f n = n
```

Finalmente, para concluir el problema que deseamos basta con aplicar el teorema que acabamos de detallar sobre la siguiente función:

$$f_1(m) = \begin{cases} f(m), & \text{si } m > 0, \\ 0, & \text{si } m = 0. \end{cases} \tag{3}$$

Destacar que para poder aplicar dicho teorema, previamente ha sido necesario probar que la función así definida verifica las hipótesis del teorema, lo cual se comprueba de manera muy sencilla diferenciando por casos.

IMO 2001 Q2

Consideremos a, b y c tres números reales positivos cualesquiera. Demostrar que

$$\frac{a}{\sqrt{a^2+8bc}} + \frac{b}{\sqrt{b^2+8ca}} + \frac{c}{\sqrt{c^2+8ab}} \geq 1. \tag{4}$$

La formalización en Lean del enunciado del teorema es bastante directa:

```
theorem imo2001_q2
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
```

```
  : 1 ≤ a / sqrt (a ^ 2 + 8 * b * c) +
        b / sqrt (b ^ 2 + 8 * c * a) +
        c / sqrt (c ^ 2 + 8 * a * b)
```

Para formalizar este resultado sólo ha sido necesario el uso de un teorema auxiliar. No obstante, mencionar que para demostrar el teorema auxiliar han sido necesarios a su vez otros dos lemas.

El teorema auxiliar mediante el cual se ha podido resolver el último ejercicio del capítulo consiste en probar que bajo las hipótesis del problema se tiene la siguiente desigualdad:

$$1 \leq \frac{a^3}{\sqrt{(a^3)^2 + 8b^3c^3}} + \frac{b^3}{\sqrt{(b^3)^2 + 8c^3a^3}} + \frac{c^3}{\sqrt{(c^3)^2 + 8a^3b^3}}. \tag{5}$$

En Lean este resultado lo hemos formalizado de la siguiente manera:

```
theorem imo2001_q2_aux
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
  : 1 ≤ a ^ 3 / sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) +
        b ^ 3 / sqrt ((b ^ 3) ^ 2 + 8 * c ^ 3 * a ^ 3) +
        c ^ 3 / sqrt ((c ^ 3) ^ 2 + 8 * a ^ 3 * b ^ 3)
```

Si se observan de manera detallada las expresiones (4) y (5), se tiene que considerando los números reales $(a^3)^{-1}$, $(b^3)^{-1}$ y $(c^3)^{-1}$ en (5) se llega directamente a (4).

Cuarto capítulo

Todos estos capítulos que se han desarrollado previamente, han permitido adquirir los conocimientos y la destreza suficiente como para formalizar un problema que no haya sido formalizado con anterioridad. Entonces, en el último capítulo del trabajo se ha llevado a cabo la **formalización en Lean de un nuevo problema que no se había formalizado antes**, (4). En concreto, el problema que se ha formalizado en este capítulo es el Q6 correspondiente al año 2001. El problema que en este capítulo se ha estudiado posee el siguiente enunciado:

Sean a, b, c y d cuatro números enteros tales que $a > b > c > d > 0$. Supongamos que

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Demostrar que $ab + cd$ no es primo.

Asimismo, la formalización que se ha propuesto en el trabajo para resolver el problema que es la siguiente:

```
theorem imo2001q6
  (hd  : 0 < d)
  (hdc : d < c)
  (hcb : c < b)
  (hba : b < a)
  (h : a*c + b*d = (a + b - c + d)*(-a + b + c + d))
  : ¬ prime (a*b + c*d)
```

Para llevar acabo tal formalización en Lean he hecho uso de cinco lemas auxiliares. Expongamos estos cinco resultados:

1. Bajo la hipótesis (h) del problema, se tiene que los cuatro números enteros descritos verifican la siguiente igualdad:

   $$b^2 + bd + d^2 = a^2 - ac + c^2.$$

   Cuyo enunciado en Lean se formaliza como sigue:

   ```
   lemma sumas_equivalentes
     (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
     : b^2 + b*d + d^2 = a^2 - a*c + c^2
   ```

2. Bajos la misma hipótesis (h) del problema, probamos que los cuatro números enteros descritos verifican la siguiente expresión:

   $$(ac + bd)(b^2 + bd + d^2) = (ab + cd)(ad + bc).$$

   Asimismo, la correspondiente formalización en Lean de este resultado es:

   ```
   lemma productos_equivalentes
     (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
     : (a*c + b*d) * (b^2 + b*d + d^2) = (a*b + c*d) * (a*d + b*c)
   ```

3. Bajo todas las hipótesis del problema inicial, se verifica la siguiente desigualdad:

$$ac + bd < ab + cd.$$

Cuya formalización en Lean es:

```
lemma desigualdad_auxiliar1
  (hba : b < a)
  (hcb : c < b)
  (hdc : d < c)
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : a*c + b*d < a*b + c*d:
```

4. Análogamente a la desigualdad anterior, también se demuestra que:

$$ad + bc < ac + bd.$$

Y en este caso la formalización del enunciado en Lean es:

```
lemma desigualdad_auxiliar2
  (hba : b < a)
  (hdc : d < c)
  (h : a*c + b*d = (a + b - c + d)*(-a + b + c + d))
  : a*d + b*c < a*c + b*d:=
  by nlinarith
```

5. El último resultado es inmediato a partir del segundo que hemos enunciado, y es que se tiene que el número $ac + bd$ divide al producto de los números $ab + cd$ y $ad + bc$.

Esto lo hemos formalizado en Lean de la siguiente manera:

```
lemma division
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : a*c + b*d | (a*b+c*d) * (a*d+b*c)
```

Entonces, tras detallar los resultados auxiliares que han sido usados, se tiene que la demostración del teorema principal se hace por contradicción, suponiendo que el número $ab + cd$ sí es primo y llegando entonces a una contradicción.

Para llegar a tal contradicción, se usa que por el quinto resultado auxiliar se tiene que $ac + bd$ divide al producto de $ab + cd$ con $ad + bc$ y también se tiene la hipótesis de que $ab + cd$ es primo. Entonces, sabemos que se ha de verificar una de las dos siguientes posibilidades:

- Que $ab + cd$ divida a $ac + bd$.
- O bien que $ac + bd$ divida a $ad + bc$.

Estudiando ambas posibilidades se llega a contradicción como consecuencia de las dos desigualdades que se han introducido en los resultados auxiliares.

Apéndice A

Finalmente, en el apéndice A se han explicado todas las tácticas de Lean que se han usado en el trabajo y en el índice de lemas se han indicado los lemas usados.

Capítulo 1

Prueba de teoremas con Lean

En este primer capítulo del trabajo se hace una introducción al uso de Lean[1] y mathlib[2] de manera que sea posible la correcta comprensión del resto del trabajo. Estas herramientas han sido usadas con el objetivo de probar diversos resultados matemáticos; es por eso que, previo al estudio detallado de la estructura de una prueba y cómo realizarla, comenzaremos introduciendo los términos de Lean y mathlib.

1.1. Pilares fundamentales: Lean, IMO y mathlib

Lean es un demostrador interactivo mediante el cual se pueden formalizar multitud de resultados. Existen otros demostradores interactivos muy conocidos como es Isabelle/HOL[3] que ha sido usado en alguna asignatura del Máster en Lógica, Computación e Inteligencia Artificial de la Universidad de Sevilla.

El demostrador interactivo Lean es relativamente joven pues fue desarrollado en 2013 por Leonardo de Moura en Microsoft Research. Asimismo, existen diversas versiones de Lean, en concreto en el trabajo ha sido utilizada la versión 3.28.00.

En este trabajo se ha usado Lean con el objetivo de formalizar problemas correspondientes a las Olimpiadas Internacionales de Matemáticas (IMO).

[1] https://leanprover-community.github.io/
[2] https://leanprover-community.github.io/mathlib_docs/
[3] https://www.cl.cam.ac.uk/research/hvg/Isabelle

Las IMO consisten en una competición de matemáticas que se realiza anualmente para estudiantes preuniversitarios. La competición consta de dos cuestionarios con tres problemas cada uno, formando un total de seis. Los problemas que se proponen suelen ser de diversas áreas de las matemáticas: álgebra, análisis, geometría, teoría de números, ...

Para la formalización de los problemas que se han estudiado en el trabajo, se ha hecho uso de la librería matemática de Lean conocida como mathlib.

La librería mathlib está formada por una gran cantidad de resultados (lemas, teoremas, ...) y definiciones matemáticas ya formalizadas de todas las diferentes áreas de esta ciencia.

El trabajo que se presenta ha sido desarrollado con la siguiente versión de mathlib: 5511275718b4aa55fc66f969cbdf2065160b00bb. No obstante, durante la realización de este trabajo se han producido multitud de problemas con las versiones de mathlib. Estos problemas son bastante comunes cuando se trabaja con sistemas que se encuentran en continuo desarrollo, como es el caso de la librería mathlib.

1.2. Formalización de las demostraciones con Lean

En esta sección del trabajo se va a explicar de manera detallada la estructura de las pruebas que en el resto del trabajo se plantean y el mecanismo mediante el cual se han llevado a cabo. Con el objetivo de que sea más fácil e intuitivo de seguir la explicación, se va a realizar con el desarrollo de un ejemplo muy visual que propuso Scott Morrison en el vídeo [11] en el Congreso Lean for the Curious Mathematician 2020[4]

Dividiremos en tres partes principales el procedimiento mediante el cual se llegará a la formalización final. La primera parte se corresponderá con el planteamiento inicial del problema. La segunda con el desarrollo principal de la prueba y la tercera parte con la finalización o conclusión de la misma.

Asimismo, destacar que tanto en el desarrollo de esta sección como en el desarrollo de todo el trabajo se ha hecho uso de multitud de tácticas, las cuales han sido detalladas en el Apéndice A.

[4] https://leanprover-community.github.io/lftcm2020/

1.2.1. Planteamiento del problema

En primer lugar, se va a enunciar en lenguaje natural el problema que se desea formalizar en Lean. En nuestro caso es el siguiente:

Teorema 1.2.1 (infinito_numeros_primos)**.** Existen infinitos números primos; es decir que para todo número natural n, existe un número primo p mayor o igual que n.

Para llevar a cabo la formalización en Lean de este resultado, es indispensable importar la librería que se necesitan. En estas librerías hay diferentes conceptos, definiciones o lemas auxiliares que ya son conocidos para Lean y que resultan útiles para la prueba.

En el caso del problema que se va a planear, inicialmente sólo es necesario una teoría y es data.nat.prime. En Lean la orden de importación de esta librería se plantea como sigue:

```
import data.nat.prime
```

Nota 1.2.1. Destacar que en el proceso de realización de una prueba no siempre se sabe desde un principio cuáles son las librerías necesarias que habrá que importar. Es por eso que esto se puede hacer en cualquiero momento de la formalización, aunque es necesario escribirlo al comienzo del fichero.

A continuación, se va a habilitar el espacio de nombre de los números naturales en este caso. Esto se hace con el objetivo de simplificar la notación; por ejemplo, escribiendo `prime` en lugar de `nat.prime`. Para hacer esta simplificación en Lean basta con:

```
open nat
```

Ahora sí, ya se tienen las condiciones necesarias para poder enunciar el teorema 1.2.1 y plantear la estructura de la demostración en Lean. Veámoslo:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  sorry,
end
```

En la formalización que se ha planteado se puede observar que hemos usado la estructura de `theorem` donde hemos llamado de la misma forma al

teorema que en 1.2.1. También se puede observar que el enunciado del teorema como tal comienza a partir de los dos puntos y que es totalmente análogo al visto anteriormente. En el enunciado del teorema se ha hecho uso de la función `prime` que se encuentra en la librería que se ha importado.

Definición 1.2.2. Un número natural p es primo si p es mayor o igual que dos y, además, para todo número natural m que divide a p se tiene que o bien m es uno, o bien m es igual a p. Simbólicamente:

$$\text{primo } (p) := p \geq 2 \wedge \forall m(m|p \to m = 1 \vee m = p).$$

La formalización en Lean de esta definición es la siguiente:

```
def prime (p : ℕ) := 2 ≤ p ∧ ∀ m | p, m = 1 ∨ m = p
```

Por otro lado, se tiene que para el planteamiento de todas las formalizaciones que se van a presentar en este trabajo se han usado la estructura de demostración `begin–end` como se puede observar.

Finalmente, para acabar con el planteamiento inicial del problema, mencionar el uso de la táctica sorry. Esta táctica es una herramienta mágica que acepta cualquier resultado. Más adelante iremos viendo la utilidad de esta táctica en el proceso de formalización de cualquier resultado. Asimismo, destacar que cuando en algún teorema o resultado se usa la táctica sorry, Lean nos da un aviso de ello.

1.2.2. Desarrollo de la prueba

Una vez ya se ha hecho el planteamiento del enunciado del teorema, importado las librerías necesarias y definido las variables de trabajo (en nuestro caso trabajamos en el conjunto de los números naturales), se puede proceder al comienzo de la demostración como tal.

El uso de la táctica sorry que hemos mencionado anteriormente, no es que nos pruebe los resultados deseados. El verdadero uso es que durante el proceso de formalización, se va a ir diviendo el problema inicial en subproblemas que formalizaremos al final y de esta forma Lean no nos devuelve error.

Entonces, se tiene que obviando el uso de la táctica sorry, el estado que tenemos actualmente de la formalización en Lean es el siguiente:

```
⊢ ∀ (n : ℕ), ∃ (p : ℕ) (H : p ≥ n), prime p
```

Ahora bien, como la formalización que queremos plantear comienza con un para todo número natural n, lo primero que hacemos es introducir un número natural arbitrario. En Lean esto lo formalizamos como sigue:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,
  sorry,
end
```

Para la introducción del número natural n se ha hecho uso de la táctica intro mediante la cual fijamos dicho número. Al haber introducido el número natural n, se tiene que el estado de la prueba en Lean pasa a ser el siguiente:

```
n: ℕ
⊢ ∃ (p : ℕ) (H : p ≥ n), prime p
```

A continuación, siguiendo con el desarrollo de la prueba, entramos en la parte más matemática de la misma. Para ello, se tiene la idea feliz de considerar los siguientes dos números naturales:

$$m = n! + 1, \tag{1.1}$$

$$p = \text{mínimo factor primo de } m \text{ que sea distinto de 1.} \tag{1.2}$$

En Lean esto se formalizaría como sigue:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  sorry,
end
```

Para introducir los dos números que hemos definido en (1.1) y (1.2) se ha hecho uso de la táctica let que matemáticamente es equivalente a consideremos el número _ definido de la siguiente manera.

Asimismo, a parte del uso de la táctica let, se han utilizado también dos funciones: la función **factorial** de un número y la función que nos devuelve el **mínimo factor primo distinto de uno** de un número. Estas dos funciones están formalizadas en Lean de la siguiente manera:

```
def factorial : nat → nat
  | 0        := 1
  | (succ n) := succ n * factorial n

def min_fac : ℕ → ℕ
  | 0 := 2
  | 1 := 1
  | (n+2) := if 2 ∣ n then 2 else min_fac_aux (n + 2) 3
```

Destacar que para el uso de estas funciones en Lean me ha sido necesario la importación de una nueva teoría, en este caso correspondiente a los factoriales, data.nat.factorial.

Además, se puede observar que al definir la función mínimo factor primo de un número natural se ha hecho uso de una función auxiliar llamada **min_fac_aux** y que se define de la siguiente manera:

Definición 1.2.3. Sea n y k números naturales. Definimos la función **min_fac_aux** con la siguiente casuística:

- Si se verifica que n es menor que el producto de k por sí mismo; entonces la función **min_fac_aux** devuelve el propio número n.
- En el caso de que k divida a n; entonces **min_fac_aux** devuelve el número k.
- En cualquier otro caso, la función **min_fac_aux** aplicada sobre los números n y k es igual a aplicar esta misma función sobre los números n y $k+2$.

Tras todo lo detallado, se tiene que al haber introducido la definición de los números m y p anteriormente descritos en Lean; el estado actual pasa a ser el siguiente:

```
n: ℕ
m: ℕ := n.factorial + 1
p: ℕ := m.min_fac
⊢ ∃ (p : ℕ) (H : p ≥ n), prime p
```

En el estado anterior se observa que, por defecto, Lean usa la notación de puntos, pero se puede cambiar la opción añadiendo

```
set_option pp.structure_projections false
```

con lo que el estado anterior lo escribe con la notación habitual

```
n : ℕ,
m : ℕ := factorial n + 1,
p : ℕ := min_fac m
⊢ ∃ (p : ℕ) (H : p ≥ n), prime p
```

El siguiente paso que queremos realizar en la formalización es decirle a Lean que el número p que estamos buscando es exactamente el que hemos definido justo antes. Esto lo hacemos mediante la táctica use, la cual, como su propio nombre indica, le dice a Lean que use el número que ordenemos como testigo de la fórmula existencial. Veamos como sigue la formalización:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,

  use p,
  sorry,
end
```

Estamos dando un paso muy importante en el que el estado de la formalización pasa a ser:

```
n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
⊢ p ≥ n ∧ prime p
```

Se puede observar que el objetivo a demostrar se ha convertido en una conjunción: por un lado hay que demostrar que p es mayor o igual que n y, por otro lado, que el número p que hemos definido es primo. Ahora nos interesa separar el problema a demostrar en dos subproblemas, esto es muy sencillo

con el uso de la táctica split, la cual nos divide la conjunción. De esta manera la formalización seguiría como:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,

  use p,
  split,
  { sorry, },
  { sorry, },
end
```

Se puede observar que en la formalización se ha pasado de tener un sorry a tener dos; esto se tiene como consecuencia de haber dividido el problema en dos partes. Además, la manera más elegante y cómoda de escribirlo es que como se ha hecho: diferenciando la prueba de cada problema entre llaves. De esta manera, se tiene que, tras la aplicación de la táctica split, el estado pasa a ser:

```
n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
⊢ p ≥ n

n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
⊢ prime p
```

De esta manera, se pueden observar los dos objetivos a demostrar que faltarían para concluir la prueba. Ahora bien, para demostrar el primero de los objetivos, es necesario usar el segundo resultado de los que hay que demostrar. Es por eso, que lo que proponemos es demostrar antes que el número p es primo (que es el resultado que necesitamos para demostrar el primer objetivo).

Para ello introduciremos una hipótesis de la siguiente manera:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m:= factorial n + 1,
  let p:= min_fac m,
  have hp: prime p := sorry,

  use p,
  split,
  {sorry,},
  {sorry,},
end
```

Denotemos la hipótesis de que p sea primo de la siguiente manera:

$$\text{primo}(p) \qquad \text{(hp)}$$

Se puede observar que al haber introducido la hipótesis (hp) tras la definición del propio número p, dicha hipótesis puede ser utilizada a partir de ahí.

Para introducir la hipótesis (hp) se ha hecho uso de la táctica have. En este resultado también hemos añadido la sorry porque la prueba se realizará luego. En el estado del problema el único cambio que ha habido respecto al último presentado es que se ha incluido la hipótesis (hp) en los dos subproblemas:

```
n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
hp : prime p
⊢ p ≥ n

n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
hp : prime p
⊢ prime p
```

Recordemos que el número p se ha definido como el mínimo factor primo de m que sea distinto de uno. Por tanto, se tiene que la prueba de que p es primo será directa y la dejaremos para el final.

Nos centraremos en la prueba de que el número p es mayor o igual que n. En general, muchas de las pruebas de desigualdades en matemáticas se hacen por reducción al absurdo. Es decir, supongo que no es cierto dicha desigualdad y llego a una contradicción en algún momento. Procederemos así para la prueba que a nosotros nos interesa; en Lean esto se formularía con la táctica by_contradiction:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p := sorry,

  use p,
  split,
  { by_contradiction,
    sorry, },
  { sorry, },
end
```

De esta manera también se actualiza el estado y pasamos a tener una nueva hipótesis y tenemos que demostrar falso, así es como Lean nos indica que tenemos que llegar a una contradicción. Esto es:

```
n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac n
hp : prime p
h : ¬p ≥ n
⊢ false
```

Para poder concluir la prueba de este primer problema, se van a introducir las tres siguientes hechos:

$$p|(n!+1), \tag{h1}$$
$$p|n!, \tag{h2}$$
$$p|1. \tag{h3}$$

Introduzcámolas ahora en Lean:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p := sorry,

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := sorry,
    have h2 : p ∣ factorial n := sorry,
    have h3 : p ∣ 1 := sorry,
    sorry, },
  { sorry, },
end
```

En el caso del estado de este subproblema simplemente se han introducido las hipótesis (h1), (h2) y (h3), el objetivo a probar no cambia:

```
n : ℕ
m : ℕ := factorial n + 1
p : ℕ := min_fac m
hp : prime p
h : ¬p ≥ n
h1 : p ∣ factorial n + 1
h2 : p ∣ factorial n
h3 : p ∣ 1
⊢ false
```

A través de las hipótesis añadidas demostrar que se llega a una contradicción no es nada complicado.

Entonces, se ha llegado a lo que es la estructura final de la prueba. Se puede observar que tenemos varios resultados que tenemos pendientes con la táctica sorry, pero los resultados que tenemos son muy sencillos de probar y la siguiente parte de la sección veremos cómo hacerlo con una herramienta muy útil de Lean.

1.2.3. Finalización de la prueba

Una vez ya se ha llegado a la estructura final de la prueba con el uso en una o varias ocasiones de la táctica sorry es muy sencillo llevar a cabo la finalización de la demostración, siempre y cuando estemos seguros de que los resultados que nos falten por demostrar sean ciertos.

Cuando nos encontramos ante resultados casi inmediatos de probar, Lean posee una táctica muy útil que nos indica qué lema o teorema hay que usar. Esta táctica es library_search. Es por eso que lo primero que intentamos es utilizar dicha táctica en todos los resultados que nos faltan por probar. De esta forma, de los seis resultados que nos faltaban por probar cuatro de ellos se prueban con esta táctica:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p := sorry,

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := by library_search,
    have h2 : p ∣ factorial n := by sorry,
    have h3 : p ∣ 1 := by library_search,
    library_search, },
  { library_search, },
end
```

Para ir simplificando los problemas que nos quedan, a continuación lo que se puede hacer es escribir la demostración que la táctica library_search ha propuesto en los problemas que sí funciona. De esta forma, la formalización en Lean nos queda:

```
theorem infinitos_numeros_primos : ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
```

```
  have hp : prime p := sorry,

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := min_fac_dvd m,
    have h2 : p ∣ factorial n := by sorry,
    have h3 : p ∣ 1 := (nat.dvd_add_right h2).mp h1,
    exact prime.not_dvd_one hp h3,},
  { exact hp, },
end
```

Antes de proseguir con la demostración de los dos resultados que nos faltan por demostrar ((hp) y (h2)), veamos las pruebas que nos han demostrado los cuatro resultados anteriores.

En primer lugar, para el caso de la hipótesis (h1) se ha hecho uso del siguiente teorema:

Teorema 1.2.4. Sea n cualquier número natural, entonces se tiene que el mínimo factor primo de n divide al propio n.

Y su formalización en Lean es la siguiente:

```
theorem min_fac_dvd (n : ℕ) : min_fac n ∣ n
```

En particular, como el número m está definido como el factorial de la suma de n y 1 y p es el mínimo factor primo de m, se tiene de manera inmediata que p divide al factorial de $n+1$.

En segundo lugar, demostrar que (h3) es bastante inmediato a partir de las hipótesis (h1) y (h2). Por la hipótesis (h1) se tiene que p divide a la suma del factorial de n y 1; mientras que por la hipótesis (h2) se sabe que p divide al factorial de n. Entonces aplicando el teorema que se plantea a continuación se tiene de manera inmediata.

Teorema 1.2.5. Sean k, m y n números naturales tales que k divide a m. Entonces se tiene que k divide a la suma de m y n si y solamente si k divide a n. Simbólicamente:

$$\forall (k, m, n \in \mathbb{N}), k|m \longrightarrow (k|m+n \iff k|n).$$

Cuya formalización en Lean es la siguiente:

```
theorem dvd_add_right {k m n : ℕ} (h : k ∣ m) : k ∣ m + n ↔ k ∣ n
```

El tercer resultado que ha demostrado la táctica library_search es la contradicción a la que había llegar en el primer problema de la división. Una vez se tiene la hipótesis (h3) que nos dice que el número p divide a 1 y sabiendo que p es un número primo que se tiene por hipótesis; la contradicción es inmediate por la propia definición de ser primo. Para llegar a dicha contradicción se hace uso de que un número primo no puede dividir al número 1, formalmente este teorema se enunciaría como sigue:

Teorema 1.2.6. Sea p un número natural verificando que es un número primo. Entonces se tiene que p no divide a 1.

Cuya correspondiente formalización en Lean sería:

```
theorem prime.not_dvd_one {p : ℕ} (pp : prime p) : ¬ p ∣ 1
```

Por último, el cuarto resultado es demostrar que p es primo lo cual se tiene de manera inmediata a través de la hipótesis (hp) que nos falta por demostrar. Para ello, Lean nos propone usar la táctica exact.

A continuación, se van a demostrar las dos hipótesis que nos faltan para poder concluir la demostración; es decir, (h2) y (hp). A priori, estos dos resultados no son tan triviales como los anteriores pero veremos cómo resolverlos. Una forma muy útil y cómoda de trabajar cuando la táctica library_search no ha funcionado es escribirlo como un lema independiente. Separemos las dos pruebas que queremos realizar en dos lemas diferentes:

- **Demostración y formalización del hecho h2**

  El hecho (h2) bajo las hipótesis que tenemos puede ser enunciado de la siguiente manera:

  Lema 1.2.7 (h2)**.** Sean m, n y p tres números naturales tales que m es igual a la suma del factorial de n con 1 y p es igual al mínimo factor primo de m. Además, estos números verifican las siguientes tres hipótesis:

  $$\text{primo}(p), \tag{hp}$$
  $$\neg p \geq n, \tag{h}$$
  $$p|(n! + 1). \tag{h1}$$

  Entonces se tiene que p divide al factorial de n.

Para formalizar la demostración de la hipótesis (h2) hemos creado un lema auxiliar con las hipótesis correspondiente. Esta formalización es:

```
lemma h2
  (n : ℕ)
  (m = factorial n + 1)
  (p = min_fac m)
  (hp : prime p)
  (h : ¬p ≥ n)
  (h1 : p ∣ factorial n + 1)
  : p ∣ factorial n :=
  begin
    sorry,
end
```

Se puede observar que hemos vuelto a usar la estructura `begin-end` que usamos para el teorema inicialmente. Esto se debe a que cuando nos encontramos ante resultados que de primeras no somos capaces de demostrar en una línea es más sencillo comenzar la formalización de esta manera.

Ya hemos comprobado que la táctica library_search no es capaz de solucionarnos este lema; sin embargo, existe otra táctica que nos da sugerencias sobre las posibles soluciones. Esta táctica es suggest. En general, cuando se usa esta táctica, Lean propone multitud de soluciones. Es por eso que debemos valorarlas y probar a elegir una. En el caso que a nosotros nos concierne la primera propuesta nos sirve de gran utilidad. Veámosla:

```
lemma h2
  (n : ℕ)
  (m = factorial n + 1)
  (p = min_fac m)
  (hp : prime p)
  (h : ¬p ≥ n)
  (h1 : p ∣ factorial n + 1)
  : p ∣ factorial n:=
begin
  refine (prime.dvd_factorial hp).mpr _,
  sorry,
end
```

La táctica suggest nos ha propuesto el uso del siguiente lema:

Lema 1.2.8. Sean n y p dos números naturales tales que p es primo. Entonces se tiene que p divide al factorial de n si y solamante si p es menor o igual que n

Cuya formalización en Lean es:

```
lemma prime.dvd_factorial : ∀ {n p : ℕ} (hp : prime p), p ∣ n! ↔ p ≤ n
```

Además, en lugar de usar la táctica exact, se ha hecho uso de la táctica refine que es equivalente a la anterior pero se diferencian en que la última podemos dejar que Lean interprete sobre quién se aplica.

Tras el uso de este lema, se tiene que el estado del problema se convierte en el siguiente:

```
n m : ℕ,
H : m = factorial n + 1,
p : ℕ,
H : p = min_fac m,
hp : prime p,
h : ¬p ≥ n,
h1 : p ∣ factorial n + 1
⊢ p ≤ n
```

Una vez se ha conseguido a dar un paso en la demostración, se propone de nuevo el uso de la táctica library_search y es capaz de concluir la prueba de esta hipótesis:

```
lemma h2
  (n : ℕ)
  (m = factorial n + 1)
  (p = min_fac m)
  (hp : prime p)
  (h : ¬p ≥ n)
  (h1 : p ∣ factorial n + 1)
  : p ∣ factorial n:=
begin
  refine (prime.dvd_factorial hp).mpr _,
  library_search,
end
```

En este caso, para culminar la prueba de la hipótesis (h2) ha hecho la siguiente propuesta:

```
lemma h2
  (n : ℕ)
  (m = factorial n + 1)
  (p = min_fac m)
  (hp : prime p)
  (h : ¬p ≥ n)
  (h1 : p ∣ factorial n + 1)
  : p ∣ factorial n:=
begin
  refine (prime.dvd_factorial hp).mpr _,
  exact le_of_not_ge h,
end
```

Simplemente ha hecho uso del lema que se presenta a continuación:

Lema 1.2.9. Sean a y b dos números cualesquiera tales que a no es mayor o igual que b. Entonces se tiene que a es menor o igual que b.

Cuya formalización en Lean es:

```
lemma le_of_not_ge {a b : α} : ¬ a ≥ b → a ≤ b
```

- **Demostración y formalización del hecho hp**

  A continuación, se demostrará el hecho (hp) bajo las hipótesis del problema. Este nuevo lema podría ser enunciado de la siguiente manera:

  Lema 1.2.10. Sean n, m y p números naturales tales que verifican las dos siguiente condiciones:

  $$m = n! + 1 \qquad \text{(h1p)}$$
  $$p = \text{mínimo factor primo de } m \text{ que sea distinto de } 1. \qquad \text{(h2p)}$$

  Entonces se tiene que el número p así definido es primo.

  Y su correspondiente formalización en Lean sería:

```
lemma hp
  (n m p : ℕ)
  (h1p : m = factorial n + 1)
  (h2p : p = min_fac m)
  : prime p :=
begin
  sorry,
end
```

A continuación, procedamos a la demostración de ese resultado:

Demostración: En primer lugar, para que realmente el lema que hemos enunciado se corresponda con el que se tiene en el teorema principal hay que reescribir p como nos dice la hipótesis (h2p). En Lean esto se haría de la siguiente forma:

```
lemma hp
  (n m p : ℕ)
  (h1p : m = factorial n + 1)
  (h2p : p = min_fac m)
  : prime p :=
begin
  rw h2p,
  sorry,
end
```

En este caso para introducir la hipótesis (h2p) en el objetivo a probar se ha hecho uso de la táctica rw. De esta forma, el objetivo a demostrar se convierte en:

```
nmp: ℕ
h1p: m = factorial n + 1
h2p: p = min_fac m
⊢ prime (min_fac m)
```

Al intentar aplicar la táctica library_search, seguimos sin tener una propuesta de solución. Es por eso que intentamos con la táctica suggest y se obtienen varias prupuestas, de las que se ha elegido la siguiente:

```
lemma hp
  (n m p : ℕ)
  (h1p : m = factorial n + 1)
  (h2p : p = min_fac m)
  : prime p :=
begin
  rw h2p,
  refine min_fac_prime _,
  sorry,
end
```

Lo que la táctica suggest nos ha propuesto es hacer uso del teorema que se va a presentar a continuación junto con la táctica refine.

Teorema 1.2.11. Sea n un número natural distinto de 1. Entonces se tiene que el mínimo factor primo de n es primo.

La formalización en Lean de este teorema sería:

```
theorem min_fac_prime {n : ℕ} (n1 : n ≠ 1) : prime (min_fac n)
```

Tras la aplicación de este teorema el objetivo a demostrar pasa a ser el siguiente:

```
nmp: ℕ
h1p: m = factorial n + 1
h2p: p = min_fac m
⊢ m ≠ 1
```

De esta manera, si observamos las hipótesis que tenemos, el objetivo a probar es bastante sencillo. En primer lugar, se va a introducir la hipótesis de que el factorial del número n es mayor que cero, esto es:

$$0 < n! \tag{1.3}$$

Este resultado se tiene de manera inmediata con el teorema siguiente:

Teorema 1.2.12. Sea n un número cualquiera, entonces se tiene que el factorial de dicho número es mayor estrictamente que cero.

Y en Lean nos quedaría:

```
theorem factorial_pos : ∀ n, 0 < n!
```

La introducción en Lean de esta hipótesis se haría a través de la taćtica have y se plantearía como sigue:

```
lemma hp
  (n m p : ℕ)
  (h1p : m = factorial n + 1)
  (h2p : p = min_fac m)
  : prime p :=
begin
  rw h2p,
  refine min_fac_prime _,
  have : 0 < factorial n := factorial_pos n,
  sorry,
end
```

De esta forma, el objetivo a demostrar no cambia, simplemente se añade la hipótesis definida al conjunto de las hipótesis.

A continuación, si observamos las hipótesis que tenemos, se puede comprobar que haciendo uso de la desigualdad planteada y la definición del número m se puede concluir la formalización. Como se trata de un paso en el que hay que trabajar con desigualdades tiene sentido probar con la táctica linarith. De esta manera, se concluiría la prueba:

```
lemma hp
  (n m p : ℕ)
  (h1p : m = factorial n + 1)
  (h2p : p = min_fac m)
  : prime p :=
begin
  rw h2p,
  refine min_fac_prime _,
  have : 0 < factorial n := factorial_pos n,
  linarith,
end
```

Ya se tendría demostrada entonces la hipótesis (hp).

□

Al haber demostrado y formalizado las hipótesis (h2) y (hp) ya se tendría completada la prueba de que existen infinitos números primos naturales. Bastaría con incluir las demostraciones realizadas en la formalización principal:

```
theorem infinitos_numeros_primos:
∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p ,
  { refine min_fac_prime _,
    have : 0 < factorial n := factorial_pos n,
    linarith,},

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := min_fac_dvd m,
```

```
    have h2 : p ∣ factorial n,
    { refine (prime.dvd_factorial hp).mpr _,
      exact le_of_not_ge h,},
    have h3 : p ∣ 1 := (nat.dvd_add_right h2).mp h1,
    exact prime.not_dvd_one hp h3,},
  { exact hp, },
  end
```

Para finalizar de manera completa la formalización en Lean de este resultado faltaría un último paso. Este paso consiste en tratar de reducir las demostraciones de algunos hechos; en concreto, los hechos (hp) y (h2) son los que podrían ser simplificados. Para ello, se hace uso de la táctica show_term de la siguiente manera:

```
theorem infinitos_numeros_primos:
  ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p ,
  show_term{ refine min_fac_prime _,
    have : 0 < factorial n := factorial_pos n,
    linarith,},

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := min_fac_dvd m,
    have h2 : p ∣ factorial n,
    show_term{ refine (prime.dvd_factorial hp).mpr _,
      exact le_of_not_ge h,},
    have h3 : p ∣ 1 := (nat.dvd_add_right h2).mp h1,
    exact prime.not_dvd_one hp h3,},
  { exact hp, },
end
```

En el caso del hecho (hp) la propuesta que nos hace Lean no es óptima y por eso mismo la desestimamos; mientras la que hace para el hecho (h2) sí lo es y la introducimos en nuestra formalización. Entonces, se tiene que la formalización del teorema final sería:

```
theorem infinitos_numeros_primos:
  ∀ n, ∃ p ≥ n, prime p:=
begin
  intro n,

  let m := factorial n + 1,
  let p := min_fac m,
  have hp : prime p ,
  { refine min_fac_prime _,
    have : 0 < factorial n := factorial_pos n,
    linarith,},

  use p,
  split,
  { by_contradiction,
    have h1 : p ∣ factorial n + 1 := min_fac_dvd m,
    have h2 : p ∣ factorial n,
    exact iff.mpr (prime.dvd_factorial hp) (le_of_not_ge h),
    have h3 : p ∣ 1 := (nat.dvd_add_right h2).mp h1,
    exact prime.not_dvd_one hp h3,},
  { exact hp, },
end
```

Capítulo 2

Elementos de matemáticas en Lean

En este capítulo, para continuar con la introducción al uso de Lean, se presentan algunas demostraciones de las que se estudian en diversas asignaturas del Grado de Matemáticas formalizadas en Lean.

2.1. Cálculo Infinitesimal

En esta primera sección veremos diferentes resultados de análisis correspondientes a la asignatura de formación básica Cálculo Infinitesimal del primer año del Grado.

Para llevar a cabo el desarrollo de tales resultados hemos hecho uso de [1] y [2].

Estudiaremos un total de seis resultados. Los tres primeros son relativos a la convergencia de sucesiones, mientras que los tres últimos son sobre la paridad de las funciones.

2.1.1. Unicidad del límite

Definición 2.1.1. Una sucesión $\{u_n\}$ se dice que converge a un número real c si, dado cualquier número real $\varepsilon > 0$, existe un número natural N tal que si n es cualquier número natural mayor o igual que N se cumple que $|u_n - c| \leq \varepsilon$.

Simbólicamente,

$$\forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall n \geq N, |u_n - c| \leq \varepsilon$$

La formalización de la definición anterior en Lean es

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

donde la notación para el valor absoluto se ha definido por

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

Teorema 2.1.2. Cada sucesión tiene como máximo un límite.

Demostración: Comenzaremos viendo la demostración en lenguaje natural. Para ello, procederemos por reducción al absurdo: supondremos que la sucesión $\{u_n\}$ posee dos límites distintos que denotaremos por l y l'. Tenemos que demostrar que estos límites son iguales; es decir, que $l = l'$.

En primer lugar, por la definición de límite anterior, 2.1.1, se tiene que para el límite l podemos escribir que:

$$\forall \varepsilon > 0, \exists N_1 \in \mathbb{N}, \forall n \geq N_1 |u_n - l| \leq \frac{\varepsilon}{2}. \tag{2.1}$$

Análogamente, para l' se tiene que:

$$\forall \varepsilon > 0, \exists N_2 \in \mathbb{N}, \forall n \geq N_2 |u_n - l'| \leq \frac{\varepsilon}{2}. \tag{2.2}$$

A continuación, definamos $N_0 = \text{máx}(N_1, N_2)$. De manera que, considerando un número arbitrario n que sea mayor o igual que N_0 y teniendo en cuenta lo visto anteriormente, podemos escribir:

$$\begin{array}{rcll} |l - l'| & = & |l - u_{N_0} + u_{N_0} - l'| & \\ & \leq & |u_{N_0} - l| + |u_{N_0} - l'| & [(*)] \\ & \leq & \frac{\varepsilon}{2} + \frac{\varepsilon}{2} & [(**)] \\ & = & \varepsilon & \end{array}$$

donde en $(*)$ se ha hecho uso de la desigualdad triangular y la definición de valor absoluto y en $(**)$ se ha usado las desigualdades (2.1) y (2.2).

Por tanto, se ha demostrado que para todo ε mayor que cero, se verifica que

$$|l - l'| \leq \varepsilon. \tag{2.3}$$

Finalmente, se concluye la demostración sin más que aplicar sobre (2.3) el siguiente resultado:

$$\forall x, y \in \mathbb{R}, (\forall \varepsilon > 0, |x - y| \leq \varepsilon) \Longrightarrow x = y. \tag{2.4}$$

□

La formalización en Lean del teorema anterior es

```
import data.real.basic

variables (u : ℕ → ℝ)
variables (a b x y : ℝ)

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

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

lemma eq_of_abs_sub_le_all
  : (∀ ε > 0, |x - y| ≤ ε) → x = y :=
begin
  intro h,
  apply eq_of_abs_sub_nonpos,
  by_contradiction H,
  push_neg at H,
  specialize h (|x-y|/2) (by linarith),
  linarith,
end

example
  (ha : limite u a)
  (hb : limite u b)
  : a = b :=
begin
  apply eq_of_abs_sub_le_all,
  intros eps eps_pos,
  cases ha (eps/2) (by linarith) with N1 hN1,
  cases hb (eps/2) (by linarith) with N2 hN2,
  let N0:=max N1 N2,
  calc  |a-b|
      = |(a-u N0)+(u N0-b)| : by ring_nf
  ... ≤ |a-u N0| + |u N0-b| : by apply abs_add
  ... = |u N0-a| + |u N0-b| : by rw abs_sub
  ... ≤ eps                 : by linarith [hN1 N0 (le_max_left N1 N2),
```

```
                                                hN2 N0 (le_max_right N1 N2)],
end
```

Para poder trabajar con el conjunto de los números reales ha sido necesario importar la librería data.real.basic.

En la formalización de este teorema en Lean han sido necesarios el uso de los siguiente lemas y teoremas:

- `eq_of_abs_sub_nonpos : (a - b) ≤ 0 → a = b`
- `abs_add : abs (a + b) ≤ abs a + abs b`
- `abs_sub : abs (a - b) = abs (b - a)`
- `le_max_left : a ≤ max a b`
- `le_max_right : b ≤ max a b`

Además, también se han usado las tácticas apply, by_contradiction, push_neg, specialize, calc, intro, intros, cases y linarith.

2.1.2. Las sucesiones convergentes son sucesiones de Cauchy

Previo al enunciado y demostración del teorema, veamos en qué consiste una sucesión de Cauchy.

Definición 2.1.3. Una sucesión $\{u_n\}$ se dice que es de Cauchy si, dado cualquier número real $\varepsilon > 0$, existe un número natural N tal que si p y q son cualesquiera dos números naturales mayores o iguales que N se cumple que $|u_p - u_q| \leq \varepsilon$. Simbólicamente:

$$\forall \varepsilon > 0, N \in \mathbb{N}, \forall p, q \geq N, |u_p - u_q| \leq \varepsilon. \tag{2.5}$$

En Lean esta definición se formaliza como:

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

Teorema 2.1.4. Toda sucesión convergente es una sucesión de Cauchy.

Demostración: Sea $\{u_n\}$ una sucesión convergente; es decir, por la definición 2.1.1, se tiene que si denotamos l como el límite de la sucesión en cuestión se verifica que

$$\forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall n \geq N, |u_n - l| \leq \frac{\varepsilon}{2}. \tag{2.6}$$

A continuación, consideremos $p, q \in \mathbb{N}$ tales que estos dos números verifican que son mayores o iguales que N. Entonces tenemos que se puede escribir:

$$\begin{array}{rcll} |u_p - u_q| & = & |u_p - l + l - u_q| & \\ & \leq & |u_p - l| + |u_q - l| & [(*)] \\ & \leq & \frac{\varepsilon}{2} + \frac{\varepsilon}{2} & [(**)] \\ & = & \varepsilon & \end{array}$$

donde $(*)$ se ha hecho uso de la desigualdad triangular y de la propia definición del valor absoluto; mientras que en $(**)$ se ha usado lo visto en (2.6) para los casos de p y q.

De esta forma, se ha demostrado que para cualquier número real $\varepsilon > 0$ se verifica que $|u_p - u_q| \leq 0$; es decir, hemos demostrado que $\{u_n\}$ es una sucesión de Cauchy.

□

En Lean esto se formaliza como sigue:

```
import data.real.basic

variables {u : ℕ → ℝ} {a l : ℝ}

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

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

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

```
example : (∃ l, seq_limit u l) → cauchy_sequence u :=
begin
  intros h eps eps_pos,
  cases h with l hl,
  rw seq_limit at hl,
  cases hl (eps/2) (by linarith) with N hN,
  use N,
    intros p q hp hq,
  calc  |u p - u q|
      = |(u p - l)+(l - u q)|  : by ring
  ... ≤ |u p - l| + |l - u q|  : by apply abs_add
  ... = |u p - l| + |u q - l|  : by rw abs_sub l (u q)
  ... ≤ eps/2 + eps/2          : by linarith [hN p hp, hN q hq]
  ... = eps                    : by ring,
end
```

Al igual que en el resultado anterior, para la formalización que acabamos de plantear, ha sido necesario importar la librería data.real.basic para poder trabajar con los números reales y se han usado los dos siguientes lemas auxiliares:

- `abs_add : abs (a + b) ≤ abs a + abs b`
- `abs_sub : abs (a - b) = abs (b - a)`

Además, se han usado las tácticas rw. use, apply, ring, intros, cases y linarith.

2.1.3. Suma de sucesiones convergentes

Teorema 2.1.5. Sean $\{u_n\}$ y $\{v_n\}$ dos sucesiones convergentes, entonces se tiene que la suma de estas dos sucesiones converge a la suma de sus respectivos límites. Simbólicamente,

$$lim(u_n + v_n) = lim(u_n) + lim(v_n). \tag{2.7}$$

Demostración: Como consecuencia de que las sucesiones $\{u_n\}$ y $\{v_n\}$ sean convergentes se tiene que:

$$\forall \varepsilon > 0, \exists N_1 \in \mathbb{N}, \forall n \geq N_1 |u_n - l| \leq \frac{\varepsilon}{2} \tag{2.8}$$

$$\forall \varepsilon > 0, \exists N_2 \in \mathbb{N}, \forall n \geq N_1 |v_n - l'| \leq \frac{\varepsilon}{2}, \tag{2.9}$$

donde hemos denotado l y l' a los límites de $\{u_n\}$ y $\{v_n\}$, respectivamente.

A continuación, definimos $N_0 := \text{máx}(N_1, N_2)$ y consideremos un número natural n que sea mayor o igual que N_0, de manera que:

$$\begin{aligned} |(u_n + v_n) - (l + l')| &= |u_p - l + l - u_q| \\ &\leq |u_n - l| + |v_n - l'| \quad [(*)] \\ &\leq \tfrac{\varepsilon}{2} + \tfrac{\varepsilon}{2} \quad [(**)] \\ &= \varepsilon \end{aligned}$$

donde en $(*)$ se usado la desigualdad triangular y en $(**)$ las desigualdades vistas en (2.8) y (2.9).

De manera que finalmente se ha demostrado que para cualquier número real $\varepsilon > 0$, existe un número natural N_0 tal que para cualquier número n mayor o igual que N_0, se verifica que $|(u_n + v_n) - (l + l')| \leq \varepsilon$, o lo que es lo mismo que la sucesión $\{u_n + v_n\}$ converge a $(l + l')$.

□

En Lean esto se formaliza como:

```
import data.real.basic

variables {u v: ℕ → ℝ} {l l' : ℝ}

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

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

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

example (hu : seq_limit u l) (hv : seq_limit v l') :
seq_limit (u + v) (l + l') :=
begin
  intros eps eps_pos,
  cases hu (eps/2) (by linarith) with N1 hN1,
  cases hv (eps/2) (by linarith) with N2 hN2,
  let N0:= max N1 N2,
  use N0,
  intros n hn,
  rw ge_max_iff at hn,
  calc  |(u + v) n - (l + l')|
      = |u n + v n - (l + l')|   : rfl
```

```
  ... = |(u n - l) + (v n - l')| : by congr' 1 ; ring
  ... ≤ |u n - l| + |v n - l'|   : by apply abs_add
  ... ≤  eps                     : by linarith [hN1 n (by linarith),
                                                hN2 n (by linarith)],
end
```

En el caso de este resultado, se ha importado librería data.real.basic, para trabajar con el conjunto de los números reales y sólo se ha usado el siguiente lema auxiliar:

- `abs_add : abs (a + b) ≤ abs a + abs b`

Además del lema auxiliar, se han usado las tácticas rw, use, let, apply, ring, congr', intros, cases y linarith.

2.1.4. Paridad de la suma de funciones

Como ya se adelantó, los tres resultados que vamos a estudiar a continuación son relativos a la paridad de las funciones. Es por eso, que previo al desarrollo de los resultados veamos las dos siguientes definiciones:

Definición 2.1.6. Sea $f : \mathbb{R} \to \mathbb{R}$, se dirá que f es una **función par** si para cualquier número real x se verifica que $f(-x) = f(x)$.

Definición 2.1.7. Sea $f : \mathbb{R} \to \mathbb{R}$, se dirá que f es una **función impar** si para cualquier número real x se verifica que $f(-x) = -f(x)$.

Estas dos definiciones se formalizan en Lean como sigue:

```
def even_fun (f : ℝ → ℝ) := ∀ x, f (-x) = f x
def odd_fun  (f : ℝ → ℝ) := ∀ x, f (-x) = -f x
```

A continuación, ya estamos preparados para ver la demostración del resultado deseado.

Teorema 2.1.8. Sean f y g dos funciones pares, entonces se tiene que la suma de ambas, $(f + g)$, es también una función par.

Demostración: Por la definición 2.1.6 de una función par, se tiene que se verifica lo siguiente:

$$\forall x \in \mathbb{R}, f(-x) = f(x) \tag{2.10}$$
$$\forall x \in \mathbb{R}, g(-x) = g(x). \tag{2.11}$$

Como queremos demostrar que la suma de las funciones f y g es una función par, se tendría que probar que

$$\forall x \in \mathbb{R}, (f+g)(-x) = (f+g)(x). \tag{2.12}$$

Para probarlo, comencemos considerando un número real x arbitrario, entonces estudiemos:

$$\begin{array}{rcll} (f+g)(-x) & = & f(-x)+g(-x) & [(1)] \\ & \leq & f(x)+g(x) & [(2)] \\ & \leq & (f+g)(x), & [(3)] \end{array}$$

donde tanto en el paso (1) como en el (3) hemos usado la propia definición de la suma de funciones, mientras que en (2) se han usado las hipótesis de paridad (2.10) y (2.11). Por tanto, ya tendríamos el resultado deseado.

□

Esta demostración se formaliza en Lean de la siguiente manera:

```
import data.real.basic

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

example
  (f g : ℝ → ℝ)
  : even_fun f → even_fun g →  even_fun (f + g) :=
begin
  intros hf hg,
  intros x,
  calc (f + g) (-x)
      = f (-x) + g (-x) : rfl
  ... = f x + g (-x)    : by rw hf
  ... = f x + g x       : by rw hg
  ... = (f + g) x       : rfl
end
```

Como en el resto de ejemplos anteriores, para formalizar el resultado ha sido necesario importar la librería data.real.basic para trabajar con el conjunto de los números reales.

Para llevar a cabo la formalización en Lean de este resultado no ha sido necesario utilizar ningún lema ni teorema auxiliar. No obstante, sí ha sido necesario usar las tácticas intros, intro, rw, calc y rfl.

2.1.5. Paridad de la composición de funciones

Teorema 2.1.9. Sea f una función par y sea g una función arbitraria, entonces se tiene que la composición de g con f; es decir, $(g \circ f)$ es también par.

Demostración: Al igual que en la demostración anterior, por la definición 2.1.6, se tiene que para la función f se verifica que

$$\forall x \in \mathbb{R}, f(-x) = f(x). \tag{2.13}$$

Consideremos ahora un número real x arbitrario y estudiemos la composición de una función arbitraria g con f:

$$\begin{array}{rcll} (g \circ f)(-x) & = & g(f(-x)) & [(1)] \\ & \leq & g(f(x)) & [(2)] \\ & \leq & (g \circ f)(x), & [(3)] \end{array}$$

donde en (1) y (3) se ha usado la definición de la composición; mientras que en (2) se ha hecho uso de la hipótesis de paridad de f, (2.13).

□

En Lean esto se formalizaría como sigue:

```
import data.real.basic

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

example (f g : ℝ → ℝ) : even_fun f → even_fun (g ∘ f) :=
begin
  intros hf x,
  calc (g ∘ f) (-x)
      = g (f (-x)) : rfl
  ... = g (f x)    : by rw hf,
end
```

En el ejemplo se ha importado la librería data.real.basic para trabajar con el conjunto de los números reales.

Al igual que en el caso anterior, para formalziar este resultado no ha sido necesario el uso de ningún lema o teorema auxiliar; pero sí se han usado las tácticas intros, rw, calc y rfl.

2.1.6. Imparidad de la composición de funciones impares

Teorema 2.1.10. Sean f y g dos funciones impares, entonces se tiene que la composición de ambas, $(g \circ f)$, es también una función impar.

Demostración: Por la definición 2.1.7 de una función impar vista anteriormente, se tiene que se verifica lo siguiente:

$$\forall x \in \mathbb{R}, f(-x) = -f(x) \tag{2.14}$$
$$\forall x \in \mathbb{R}, g(-x) = -g(x). \tag{2.15}$$

A continuación, vamos a estudiar la composición de estas dos funciones para un número real x arbitrario:

$$\begin{aligned} (g \circ f)(-x) &= g(f(-x)) && [(1)] \\ &\leq g(-f(x)) && [(2)] \\ &\leq -(g \circ f)(x), && [(3)] \end{aligned}$$

donde en (2) y en (3) se han usado las hipótesis de funciones impares, (2.14) y (2.15), respectivamente.

De esta forma ya tendríamos la demostración deseada.

□

Prosigamos con la formalización en Lean de esta demostración:

```
import data.real.basic

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

example
  (f g : ℝ → ℝ)
  : odd_fun f → odd_fun g →  odd_fun (g ∘ f) :=
begin
  intros hf hg x,
  calc (g ∘ f) (-x)
      =  g (f (-x)) : rfl
  ... =  g (-f x)   : by rw hf
```

```
  ... = -g (f x)     : by rw hg
  ... =  -(g ∘ f) x : rfl,
end
```

Para llevar a cabo la formalización de este resultado ha sido necesario utilizar las tácticas intros, rw, calc y rfl.

Asimismo, ha sido necesario importar la librería data.real.basic para trabajar con el conjunto de los números reales.

2.2. Álgebra básica

En esta segunda sección se estudiarán diversos resultados correspondientes a la asignatura de "Álgebra Básica". Al igual que Cálculo Infinitesimal, también es del primer año del grado en Matemáticas.

Se van a estudiar dos resultados para los cuales es necesario introducir previamente el siguiente concepto:

Definición 2.2.1. Denotaremos la divisibilidad en el conjunto de los números enteros por el símbolo "|". Se dirá que el número entero a es un **divisor de** b (o que a **divide a** b) si y solamente si existe otro número entero k tal que $b = a \cdot k$. Simbólicamente,

$$a|b \iff \exists k \in \mathbb{Z}, b = a \cdot k. \tag{2.16}$$

A continuación, una vez hemos introducido el concepto, ya estamos listos para ver los resultados en cuestión.

2.2.1. Transitividad de la división

Teorema 2.2.2. Sean a, b y c tres número enteros tales que verifican que a divide a b y b divide a c, entonces se tiene que a divide a c.

Demostración: Por la definición 2.2.1 de división en el conjunto de los números enteros vista anteriormente, como consecuencia de que a divide a b, se tiene que existe un $k_1 \in \mathbb{Z}$ tal que

$$b = a \cdot k_1. \tag{2.17}$$

Asimismo, como consecuencia de que b divide a c se verifica que existe un $k_2 \in \mathbb{Z}$ tal que

$$c = b \cdot k_2. \tag{2.18}$$

De manera que si introducimos la descomposición de b descrita en (2.17) en la de c vista en (2.18), se tiene que:

$$\begin{aligned} c &= (a \cdot k_1) \cdot k_2 \\ &= a \cdot (k_1 \cdot k_2) \\ &= a \cdot k_3, \end{aligned}$$

donde $k_3 = k_1 \cdot k_2$. Y ya tendríamos el resultado deseado.

□

Veamos ahora la formalización en Lean:

```
import data.int.parity

variables (a b c : ℤ)

example (h₁ : a ∣ b) (h₂ : b ∣ c) : a ∣ c :=
begin
  cases h₁ with k1 hk1,
  cases h₂ with k2 hk2,
  rw hk1 at hk2,
  use k1*k2,
  rw ← mul_assoc,
exact hk2,
end
```

Para la formalización del resultado se ha importado la librería data.int.basic que nos permite trabajar con el conjunto de los números enteros.

En la formalización de este resultado ha sido necesario usar el siguiente lema auxiliar:

- `mul_assoc : a * b * c = a * (b * c)`

Aparte del lema auxiliar, se han usado las tácticas cases, rw, use y exact,

2.2.2. Aditividad de la división

Teorema 2.2.3. Sean a, b y c tres números enteros tales que verifican que a divide a b y a c, entonces se tiene que a divide a la suma de b y c. Simbólica-

mente:

$$a|b,\ a|c \implies a|(b+c). \tag{2.19}$$

Demostración: Análogamente a la demostración anterior, por la definición 2.2.1, se tiene que existen $k_1, k_2 \in \mathbb{Z}$ tales que

$$b = a \cdot k_1 \tag{2.20}$$
$$c = a \cdot k_2. \tag{2.21}$$

De manera que sumando las dos expresiones descritas en las expresiones (2.20) y (2.21), se llega a que:

$$b + c = a \cdot (k_1 + k_2).$$

Teniendo en cuenta la definición 2.2.1, ya se tendría el resultado.

□

La formalización de esta demostración en Lean sería:

```
import data.int.parity
import tactic

variables (a b c : ℤ)

example
  (h1 : a ∣ b)
  (h2 : a ∣ c)
  : a ∣ b+c :=
begin
  cases h1 with k1 hk1,
  rw hk1,
  cases h2 with k2 hk2,
  rw hk2,
  use k1+k2,
  ring,
end
```

En la formalización de este resultado ha sido necesario importar dos librerías:

- data.int.basic, para trabajar con el conjunto de los números enteros.
- tactic, para poder hacer uso de algunas tácticas como ring.

Asimismo, aparte de la táctica ring, ha sido necesario el uso de las tácticas cases, rw y use.

Capítulo 3

Problemas de las IMO en Lean

En este capítulo se presentan soluciones de problemas de las Olimpiadas Internacionales de Matemáticas (IMO) formalizadas en Lean.

Se van a estudiar las soluciones a diversos problemas que ya han sido formalizados en Lean previamente y que vienen propuestas en [12]. Además, también detallaremos la solución de estos problemas en lenguaje natural. Para desarrollar este capítulo, se ha seguido el estilo propuesto en el artículo [10], donde se formalizan las soluciones a problemas de las Olimpiadas Internacionales de Matemáticas en Isabelle/HOL.

3.1. IMO 1959 Q1

En esta sección se va a detallar la solución al problema Q1 correspondiente al año 1959. En primer lugar, se presentarán el enunciado y su solución en lenguaje natural, de manera que esta última se asemeje a la solución que posteriormente se formalizará en Lean. Luego, se analizará de manera detallada dicha formalización del problema en Lean.

Destacar que la formalización original del problema se ha obtenido de [8] y fue propuesta por Kevin Lacker.

Problema 1 (1959–Q1). Demostrar que la fracción

$$\frac{21 \cdot n + 4}{14 \cdot n + 3}$$

es irreducible para cualquier número natural n.

Para ver que esta fracción es irreducible para cualquier número natural n se demostrará que el numerador y el denominador de ésta son dos números coprimos independientemente del valor de n.

Antes de proseguir, introduzcamos el concepto de que dos números sean coprimos:

Definición 3.1.1. Se dirá que dos números naturales a y b son **coprimos** o **primos relativos** si no tienen ningún factor primo en común, o equivalentemente, si no poseen otro divisor común distinto de 1.

La correspondiente formalización en Lean de esta definición es:

```
def coprime (a b : ℕ) : Prop := gcd a b = 1
```

A continuación, se va a introducir un lema auxiliar, el cual nos dirá que si existe un número natural k que divide al numerador y al denominador de la fracción anterior; entonces, dicho número k dividide a 1.

Lema 3.1.2 (Auxiliar)**.** Sean k y n dos números naturales tales que k divide al numerador y al denominador de la fracción anterior; es decir,

$$k \mid (21 \cdot n + 4) \tag{h1}$$
$$k \mid (14 \cdot n + 3). \tag{h2}$$

Entonces se tiene que k divide a 1, para cualquier número natural n.

Demostración: Como consecuencia de que k divida al numerador de la fracción, (h1), se tiene que también divide a un múltiplo cualquiera de éste; es decir,

$$k \mid 2 \cdot (21 \cdot n + 4), \tag{h3}$$

donde en particular hemos considerado el múltiplo resultante de multiplicar por 2.

Análogamente para el caso del denominador, como se verifica (h2), se puede escribir que

$$k \mid 3 \cdot (14 \cdot n + 3), \tag{h4}$$

donde en este caso se ha considerado el múltiplo resultante de multiplicar por 3.

Por otro lado, se tiene que desarrollando los productos de (h3) y (h4) como sigue:

$$2 \cdot (21 \cdot n + 4) = 42 \cdot n + 8 \tag{3.1}$$
$$3 \cdot (14 \cdot n + 3) = 42 \cdot n + 9, \tag{3.2}$$

se puede observar que para todo número natural n, se verifica la siguiente relación entre el múltiplo del numerador y el del denominador:

$$3 \cdot (14 \cdot n + 3) = 2 \cdot (21 \cdot n + 4) + 1. \tag{h5}$$

Ahora bien, como consecuencia de que k sea divisor de $3 \cdot (14 \cdot n + 4)$, (h4), y la igualdad vista en (h5), se llega a que:

$$k \mid 2 \cdot (21 \cdot n + 4) + 1 \tag{3.3}$$

Para concluir la demostración de este lema, basta con usar la siguiente propiedad de la división en los números naturales:

Proposición 3.1.3. Sean k, m y n tres números naturales tales que k divide a m. Entonces se tiene que k divide a la suma de m y n si y solamente si k divide a n. Simbólicamente,

$$\forall \{k, m, n \in \mathbb{N}\}, k \mid m \to (k \mid m + n \leftrightarrow k \mid n) \tag{3.4}$$

De esta forma, aplicando la propiedad 3.1.3 sobre (3.3), se llega a que k divide a 1; que era lo que queríamos demostrar.

□

A continuación vamos a ver la formalización en Lean de este Lema:

```
import tactic.ring
import data.nat.prime

open nat

lemma Auxiliar
  (n k : ℕ)
  (h1 : k ∣ 21 * n + 4)
  (h2 : k ∣ 14 * n + 3)
  : k ∣ 1 :=
begin
  have h3 : k ∣ 2 * (21 * n + 4),
```

```
    from dvd_mul_of_dvd_right h1 2,
  have h4 : k ∣ 3 * (14 * n + 3),
    from dvd_mul_of_dvd_right h2 3,
  have h5 : 3 * (14 * n + 3) = 2 * (21 * n + 4) + 1,
    { ring, },
  rw h5 at h4,
  rw ← nat.dvd_add_right h3,
  exact h4,
end
```

En la prueba del lema se han usado los siguientes lemas:

- `dvd_mul_of_dvd_right: a ∣ b → ∀ c, a ∣ c * b`
- `nat.dvd_add_right: ∀ {k m n : ℕ}, k ∣ m → (k ∣ m + n ↔ k ∣ n)`

y las tácticas from, have, ring y rw.

Finalmente, aplicando el Lema Auxiliar 3.1.2 se puede concluir que para cualquier número natural n se verifica que los números $21 \cdot n + 4$ y $14 \cdot n + 3$ son coprimos. O, equivalentemente, que la fracción

$$\frac{21 \cdot n + 4}{14 \cdot n + 3}$$

es irreducible. Esta afirmación se tiene como consecuencia de la definición de dos números coprimos, vista en 3.1.1.

A continuación, vamos a introducir la formalización en Lean propuesta:

```
theorem imo1959_q1 : ∀ n : ℕ, coprime (21 * n + 4) (14 * n + 3) :=
begin
  assume n,
  apply coprime_of_dvd',
  intros k hk h1 h2,
  exact Auxiliar n k h1 h2,
end
```

En la demostración del teorema se ha usado el siguiente lema:

- `coprime_of_dvd' : (∀ k, prime k → k ∣ m → k ∣ n → k ∣ 1) → coprime m n`

y las tácticas apply, assume, exact e intros.

3.2. IMO 1962 Q4

En esta sección se va a detallar la solución al problema Q4 correspondiente al año 1962. Realizaremos la demostración en lenguaje natural del problema y a su vez se presentará la correspondiente formalización en Lean.

La formalización que aquí se presenta ha sido inspirada en las que se proponen en [9] realizadas por Kevin Lacker y Healther Macbeth.

Problema 2 (1962–Q4). Resolver la ecuación

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1$$

.

Para llevar a cabo la formalización en Lean de este problema se va a importar la teoría analysis.special_functions.trigonometric (sobre funciones trigonométicas) y habilitar el espacio de nombre de los números reales (para simplificar la notación, por ejemplo escribiendo `cos` en lugar de `real.cos`). Además, se indica con `noncomputable theory` que se van a usar funciones no computables (es decir, funciones para las que Lean no generará código evaluable).

Todo lo anterior se expresa en Lean por

```
import analysis.special_functions.trigonometric

open real
open_locale real
noncomputable theory
```

Previo a la resolución como tal del problema, se van a introducir una serie de definiciones que nos servirán para llevar una notación más compacta:

Definición 3.2.1. De aquí en adelante usaremos la etiqueta **problema** para referirnos a la expresión que queremos resolver, es decir,

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1. \tag{3.5}$$

De manera que la formalización en Lean de la definición 3.2.1 es:

```
def problema (x : ℝ) : Prop :=
cos x ^ 2 + cos (2 * x) ^ 2 + cos (3 * x) ^ 2 = 1
```

Definición 3.2.2. De aquí en adelante usaremos la etiqueta **funAuxiliar** para referirnos a la siguiente expresión:

$$\cos(x) \cdot \left(\cos^2(x) - \frac{1}{2}\right) \cdot \cos(3x).$$

La formalización en Lean en este caso sería:

```
def funauxiliar (x : ℝ) : ℝ :=
cos x * (cos x ^ 2 - 1/2) * cos (3 * x)
```

3.2.1. Equivalencia de problemas

Una vez introducido estas dos definiciones, se va a proceder a demostrar que resolver problema es equivalente a resolver la expresión obtenida al igualar a cero la funAuxiliar.

Esta demostración se detallará a continuación, pero previo a ella se necesita de la demostración de un lema auxiliar:

Lema 3.2.3 (Igualdad)**.** Para cualquier x perteneciente al conjunto de los números reales, se verifica la siguiente igualdad:

$$\frac{\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1}{4} = \text{funAuxiliar}(x). \tag{3.6}$$

Demostración: En primer lugar, según la definición vista en 3.2.2, se tendría que la expresión (3.6) se convierte en:

$$\frac{\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1}{4} = \cos(x) \cdot \left(\cos^2(x) - \frac{1}{2}\right) \cdot \cos(3x). \tag{3.7}$$

Ahora bien, para llevar a cabo la demostración comenzaremos introduciendo las conocidas definiciones del coseno del ángulo doble y del coseno del ángulo triple, que son:

$$\cos(2x) = 2\cos^2(x) - 1 \tag{3.8}$$

$$\cos(3x) = 4\cos^3(x) - 3\cos(x), \tag{3.9}$$

donde estas dos propiedades en Lean se formalizan con los dos siguientes lemas:

- `cos_two_mul : cos (2 * x) = 2 * cos x ^ 2 - 1`
- `cos_three_mul : cos (3 * x) = 4 * cos x ^ 3 - 3 * cos x`

A continuación, se van a desarrollar los dos términos de la igualdad (3.7). Comencemos por el primer término de los dos:

$$\frac{\cos^2(x)+\cos^2(2x)+\cos^2(3x)-1}{4} = \frac{\cos^2(x)+(2\cos^2(x)-1)^2}{4} \tag{3.10}$$

$$+\frac{(4\cos^3(x)-3\cos(x))^2-1}{4}, \tag{3.11}$$

donde simplemente se han introducido las definiciones (3.8) y (3.9).

Desarrollando los cuadrados que aparecen en (3.10) y simplificando los términos, se acaba obteniendo que:

$$\frac{\cos^2(x)+\cos^2(2x)+\cos^2(3x)-1}{4} = \cos(x)\left(\frac{3}{2}\cos(x)+4\cos^5(x)-5\cos^3(x)\right) \tag{3.12}$$

Por otro lado, desarrollando de manera totalmente análoga el segundo término de la igualdad (3.7), se tiene que:

$$\cos(x)\cdot\left(\cos^2(x)-\frac{1}{2}\right)\cdot\cos(3x) \overset{(*)}{=} \cos(x)\cdot\left(\cos^2(x)-\frac{1}{2}\right)\cdot(4\cos^3(x)-3\cos(x)) \tag{3.13}$$

$$= \cos(x)\cdot\left(4\cos^5(x)-5\cos^3(x)+\frac{3}{2}\cos(x)\right), \tag{3.14}$$

donde en (3.13) se ha hecho uso de las definiciones (3.8) y (3.9).

De manera que observando las expresiones (3.12) y (3.14), se puede concluir que la igualdad planteada en (3.7) es cierta. De esta forma, ya se tendría el lema demostrado.

□

Veamos la formalización en Lean de este lema:

```
lemma Igualdad {x : ℝ} :
  (cos x ^ 2 + cos (2 * x) ^ 2 + cos (3 * x) ^ 2 - 1) / 4 = funauxiliar x :=
begin
  rw funauxiliar,
```

```
  rw real.cos_two_mul,
  rw cos_three_mul,
  ring_nf,
end
```

En la prueba del lema se han usado los siguientes lemas:

- `cos_three_mul : cos (3 * x) = 4 * cos x ^ 3 - 3 * cos x`
- `cos_two_mul : cos (2 * x) = 2 * cos x ^ 2 - 1`

y las tácticas ring y rw.

Una vez se ha introducido este lema, se puede proceder a la demostración del lema que ya se adelantó y que consiste en probar que resolver la expresión **problema**, (3.5), es equivalente a resolver la expresión obtenida de igualar a cero **funauxiliar**, 3.2.2. Veámoslo:

Lema 3.2.4 (Equivalencia)**.** Resolver la expresión (3.5) es equivalente a resolver la expresión

$$\text{funAuxiliar}(x) = 0. \tag{3.15}$$

Demostración: Esta demostración se trata de una doble implicación, por ello, separaremos las implicaciones:

$\longrightarrow$ En primer lugar, comenzamos considerando el problema planteado por la siguiente expresión:

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1. \tag{h1}$$

Queremos demostrar que este problema planteado es equivalente a considerar el problema planteado en (3.15).

Aplicando el Lema 3.2.3 sobre (3.15), se tiene que el objetivo a demostrar pasa a ser que el problema (h1) es equivalente al problema:

$$\frac{\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1}{4} = 0. \tag{3.16}$$

Observando esta expresión, donde tenemos una fracción igualada a cero, se concluye que el numerador ha de ser nulo. Entonces, se tiene que demostrar (3.16) es equivalente a demostrar que

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1 = 0. \tag{3.17}$$

De esta forma ya se ha teminado esta implicación pues es exactamente la hipótesis (h1), basta con pasar el término unidad al lado derecho de la igualdad.

⟵ A continuación, supongamos que se tiene el problema

$$\text{funAuxiliar}(x) = 0. \tag{h2}$$

Queremos demostrar que resolver este problema es equivalente a resolver el problema planteado en (3.5).

En primer lugar, aplicamos el Lema 3.2.3 sobre la hipótesis (h2). De manera que se obtiene que la hipótesis se transforma en la siguiente:

$$\frac{\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1}{4} = 0. \tag{3.18}$$

Como se tiene una división igualada a cero, el numerador ha de ser cero. De esta forma, se verifica que:

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) - 1 = 0 \tag{3.19}$$

Para concluir esta implicación basta con pasar el término uno hacia el otro lado de la igualdad. De esta forma se obtiene que

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1, \tag{3.20}$$

que era el resultado que deseábamos.

□

La formalización en Lean de este Lema es la siguiente:

```
lemma Equivalencia
  {x : ℝ}
  : problema x ↔ funauxiliar x = 0 :=
begin
  split,
  { intro h1,
    rw problema at h1,
    rw ← Igualdad,
    rw div_eq_zero_iff,
    norm_num,
    rw sub_eq_zero,
    exact h1, },
  { intro h2,
```

```
    rw problema,
    rw ← Igualdad at h2,
    rw div_eq_zero_iff at h2,
    norm_num at h2,
    rw sub_eq_zero at h2,
    exact h2, },
end
```

En la prueba del lema se han usado los siguientes lemas:

- `div_eq_zero_iff : a / b = 0 ↔ a = 0 ∨ b = 0`
- `sub_eq_zero : a - b = 0 ↔ a = b`

y las tácticas exact, norm_num y rw.

En el caso de la prueba de nuestro lema, la táctica norm_num ha sido usada para demostrar que cuatro es distinto de cero. Esta táctica prueba de manera directa este tipo de igualdades y desigualdades.

3.2.2. Resolución del problema equivalente

Una vez ya se ha demostrado que el problema (3.15) es equivalente al problema que inicialmente queríamos demostrar, se va a proceder a encontrar los ceros de (3.15). Para ello, se introducirán tres lemas auxiliares. El primero de ellos consistirá en ver en qué dos casos se verifica el problema (3.15); mientras que en los dos últimos lemas se detallará la forma de la solución para los dos casos que se obtendrán en el primer lema.

Lema 3.2.5 (CasosSolucion)**.** El problema (3.15) se verifica si y solamente si o bien el coseno al cuadrado de x es igual a un medio, o bien, el coseno del triple de x es nulo. Simbólicamente:

$$\text{funAuxiliar}(x) = 0 \iff \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{3.21}$$

Demostración: En primer lugar, previo a la demostración en sí del problema (3.21), se va a proceder a la reescritura del mismo. Para ello, lo que hacemos es escribir la definición de funAuxiliar vista en la definición 3.2.2, es decir,

$$\text{funAuxiliar}(x) = \cos(x) \cdot \left(\cos^2(x) - \frac{1}{2}\right) \cdot \cos(3x). \tag{3.22}$$

Aplicando ahora la propiedad asociativa de la multiplicaión, se tiene que (3.22) es equivalente a

$$\cos(x) \cdot \left(\left(\cos^2(x) - \frac{1}{2} \right) \cdot \cos(3x) \right). \tag{3.23}$$

De esta forma como en el problema inicial se tiene que funAuxiliar está igualada a cero; se sabe que cuando un producto está igualado a cero uno de los dos términos ha de ser nulo. En el caso que a nosotros nos incumbe, esto se reescribe de la siguiente manera:

$$\cos(x) \cdot \left(\left(\cos^2(x) - \frac{1}{2} \right) \cdot \cos(3x) \right) = 0 \iff \cos(x) = 0 \vee \left(\cos^2(x) - \frac{1}{2} \right) \cdot \cos(3x) = 0 \tag{3.24}$$

$$\iff \cos(x) = 0 \vee \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0, \tag{3.25}$$

donde hemos aplicado dos veces el resultado de que el producto de dos términos esté igualado a cero.

De esta forma, tenemos que el problema inicial, 3.2.1, es equivalente al que se plantea a continuación:

$$\cos(x) = 0 \vee \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0 \iff \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{3.26}$$

Ahora ya sí procederemos a la demostración de (3.26); como se trata de un si y solamente si lo haremos por doble implicación.

$\boxed{\longrightarrow}$ Suponemos que se verifica que

$$\cos(x) = 0 \vee \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{h1}$$

Entonces tenemos que demostrar que bajo la suposición (h1), se verifica que

$$\cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{3.27}$$

La hipótesis (h1) significa que alguna de las disyunciones ha de ser cierta. Por ello, considerar dicha hipótesis es equivalente a considerar dos posibles casos, que serían los siguientes:

$$\cos(x) = 0 \tag{h11}$$

$$\cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{h12}$$

Esto supone que la demostración pasa a ser dividida en dos subproblemas: el primero de ellos consiste en demostrar (3.27) suponiendo cierta la hipótesis (h11) y el segundo de ellos es también demostrar (3.27) pero ahora bajo la hipótesis (h12).

- **Subproblema 1**

  Con el objetivo de demostrar (3.27) suponiendo cierta la hipótesis (h11), basta con demostrar una de las dos disyunciones. En este caso se demostrará la segunda de ellas, es decir, se demostrará que el coseno del triple de x es nulo.

  Para demostrar esto, basta con hacer uso de la definición del ángulo triple. De esta forma se tiene que:

  $$\cos(3x) = 4\cos^3(x) - 3\cos(x) \overset{(*)}{=} 0, \tag{3.28}$$

  donde en $(*)$ se ha hecho uso de la hipótesis (h11). Y ya se tendría el resultado.

- **Subproblema 2**

  En este caso suponemos cierta la hipótesis (h12) y tenemos que demostrar (3.27), lo cual se tiene de manera directa pues son la misma afirmación.

$\boxed{\longleftarrow}$ A continuación se tiene que demostrar la implicación contraria. Esto es, suponiendo cierta la hipótesis

$$\cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0, \tag{h2}$$

hay que demostrar que es equivalente a

$$\cos(x) = 0 \vee \cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0. \tag{3.29}$$

En este caso, la implicación es directa puesto que basta con quedarse con las dos disyunciones de la derecha en (3.29) que se corresponden con la propia hipótesis (h2).

□

A continuación, se plantea la formalización en Lean de este lema:

```
lemma CasosSolucion
  {x : ℝ}
  : funauxiliar x = 0 ↔ cos x ^ 2 = 1/2 ∨ cos (3 * x) = 0 :=
begin
  rw funauxiliar,
  rw mul_assoc,
  rw mul_eq_zero,
  rw mul_eq_zero,
  rw sub_eq_zero,
  split,
  { intro h1,
    cases h1 with h11 h12,
    right,
    rw cos_three_mul,
    rw h11,
    ring,
    exact h12,},
  { intro h2,
    right,
    exact h2,},
end
```

En la prueba del lema se han usado los siguientes lemas:

- `cos_three_mul : cos (3 * x) = 4 * cos x ^ 3 - 3 * cos x`
- `mul_assoc : (a * b) * c = a * (b * c)`
- `mul_eq_zero : a * b = 0 ↔ a = 0 ∨ b = 0`
- `sub_eq_zero : a - b = 0 ↔ a = b`

y las tácticas exact, left, right, ring, rw y split.

Lema 3.2.6 (SolucionCosenoCuadrado)**.** La expresión coseno al cuadrado de x es igual a un medio si y solamente si existe un k perteneciente al conjunto de los números enteros tal que x es igual al producto de $2k$ más uno por π cuarto. Simbólicamente:

$$\cos^2(x) = \frac{1}{2} \iff \exists k \in \mathbb{Z}, x = (2k+1) \cdot \frac{\pi}{4}. \tag{3.30}$$

Demostración: Se puede observar que se trata de una demostración de doble implicación. No obstante, previo a la demostración de las correspondientes implicaciones, reescribiremos el problema (3.30). Para ello, comenzaremos usando la siguiente relación

$$\cos^2(x) = \frac{1}{2} + \frac{\cos(2x)}{2}. \tag{3.31}$$

De esta forma, el problema (3.30), se puede reescribir como sigue:

$$\frac{1}{2} + \frac{\cos(2x)}{2} = \frac{1}{2} \iff \exists k \in \mathbb{Z}, x = (2k+1) \cdot \frac{\pi}{4}. \tag{3.32}$$

Se puede observar que en la primera parte de la equivalencia, la igualdad tiene el mismo término a derecha que a izquierda, de manera que se puede eliminar. Obtenemos que

$$\frac{\cos(2x)}{2} = 0 \iff \exists k \in \mathbb{Z}, x = (2k+1) \cdot \frac{\pi}{4}. \tag{3.33}$$

Por último, en (3.33), se tiene una fracción igualada a 0, esto significa que el numerador de dicha fracción ha de ser nulo. Por tanto, el problema que inicialmente planteamos, (3.30), quedaría reescrito de la siguiente manera:

$$\cos(2x) = 0 \iff \exists k \in \mathbb{Z}, x = (2k+1) \cdot \frac{\pi}{4}. \tag{3.34}$$

Procedamos ahora a la demostración por doble implicación del problema (3.34).

$\boxed{\longrightarrow}$ Supongamos que se verifica que

$$\cos(2x) = 0 \tag{h1}$$

Entonces, se quiere demostrar que

$$\exists k \in \mathbb{Z}, x = (2k+1) \cdot \frac{\pi}{4}. \tag{3.35}$$

Para ello, comenzaremos usando la caracterización de la solución del coseno de un ángulo igualado a cero. Esto es:

Proposición 3.2.7. Sea θ un número perteneciente al conjunto de los números naturales. Entonces se tiene que se verifica la expresión coseno de θ igual a cero si y solamente si existe un número k perteneciente al conjunto de los números enteros tal que θ es igual al producto del doble de k más uno por π medio. Simbólicamente:

$$\cos(\theta) = 0 \iff \exists k \in \mathbb{Z}, \theta = \frac{(2k+1)\pi}{2}. \tag{3.36}$$

La formalización en Lean de esta proposición es la planteada a continuación:

```
theorem cos_eq_zero_iff
  {θ : ℝ}
  : cos θ = 0 ↔ ∃ k : ℤ, θ = (2 * k + 1) * π / 2
```

Entonces, usando la propiedad 3.2.7 sobre nuestra hipótesis (h1), se tiene que

$$\exists k \in \mathbb{Z}, 2x = \frac{(2k+1)\pi}{2}. \tag{3.37}$$

A partir de la hipótesis (3.37), se sabe que existe un número entero verificando dicha hipótesis. Sin pérdida de generalidad, se puede considerar que dicho número entero es un determinado número que denotaremos por k_1 y de esta forma la hipótesis (3.37) se convierte en:

$$2x = \frac{(2k_1+1)\pi}{2}. \tag{hk1}$$

Por tanto, para demostrar (3.35) basta con usar el mismo número entero, k_1, y la hipótesis (hk1). Pasando el dos que se encuentra multiplicando en la parte izquierda de la igualdad ya se tendría lo que queremos demostrar.

De manera que ya hemos terminado la primera implicación de la demostración.

$\boxed{\longleftarrow}$ Supongamos ahora que existe k perteneciente al conjunto de los números enteros, a partir del cual x se puede escribir como el producto del doble de k más uno, por π cuarto. Simbólicamente,

$$\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{4}. \tag{h2}$$

Sin pérdida de generalidad, la hipótesis (h2) se puede reescribir considerarando que el número entero que la verifica es un determinado número entero que denotaremos por k_2. Entonces la hiótesis (h2) pasa a ser:

$$x = \frac{(2k_2+1)\pi}{4}. \tag{hk2}$$

Entonces se quiere demostrar que se verifica la siguiente expresión:

$$\cos(2x) = 0. \tag{3.38}$$

Análogamente a como hicimos en la implicación anterior, sustituiremos la expresión del coseno igualada a cero por la descrita en la Proposición 3.2.7. Entonces, si se lo aplicamos a (3.38), se obtiene que el objetivo a probar pasa a ser que exista un número entero k tal que el ángulo $2x$ se pueda escribir como se describió en (3.36). Simbólicamente:

$$\exists k \in \mathbb{Z}, 2x = \frac{(2k+1)\pi}{2}. \tag{3.39}$$

Finalmante, se puede concluir la demostración haciendo uso del número entero k_2 y de la hipótesis (hk2).

□

La formalización en Lean del lema anteriormente desarrollado es la que se presenta a continuación:

```
lemma SolucionCosenoCuadrado
  {x : ℝ}
  : cos x ^ 2 = 1/2 ↔ ∃ k : ℤ, x = (2 * k + 1) * π / 4 :=
begin
  rw cos_sq,
  rw add_right_eq_self,
  rw div_eq_zero_iff,
  norm_num,
  split,
  { intro h1,
    rw cos_eq_zero_iff at h1,
    cases h1 with k1 hk1,
    use k1,
    linarith, },
  { intro h2,
    cases h2 with k2 hk2,
    rw cos_eq_zero_iff,
    use k2,
    linarith,},
end
```

Se puede observar que en la demostración en Lean aparece el número entero k representado por ↑k. Esto es debido a lo que en Lean se conoce como coerción (o inmersión), en nuestro caso, entre el conjunto de los números reales y el conjunto de los números enteros. Realmente, esto no es más que todo número entero se puede ver como un número real en el caso que sea necesario.

Seguidamente, veamos los lemas auxiliares que se han utilizado para realizar la formalización de este lema son los siguiente:

- `cos_sq : cos x ^ 2 = 1 / 2 + cos (2 * x) / 2`
- `add_right_eq_self : a + b = a ↔ a = 0`
- `div_eq_zero_iff : a / b = 0 ↔ a = 0 ∨ b = 0`

Se han usado las tácticas cases, exact, intro, right, ring, linarith, rw y split.

Lema 3.2.8 (SolucionCosenoTriple)**.** La expresión coseno del triple de x es igual a cero si y solamente si existe un k perteneciente al conjunto de los números enteros tal que x es igual al producto de $2k$ más uno por π sexto. Simbólicamente:

$$\cos(3x) = 0 \iff \exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}. \tag{3.40}$$

Demostración: Antes de comenzar con la demostración en sí del lema, vamos a reescribir el problema (3.40) haciendo uso de la Proposición 3.2.7. Entonces, nos quedaría el siguiente problema:

$$\exists k \in \mathbb{Z}, 3x = \frac{(2k+1)\pi}{2} \iff \exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}. \tag{3.41}$$

Ahora sí, se procederá a la demostración de las correspondientes implicaciones por separado.

$\boxed{\longrightarrow}$ Supongamos que se verifica la siguiente hipótesis:

$$\exists k \in \mathbb{Z}, 3x = \frac{(2k+1)\pi}{2}. \tag{h1}$$

Se quiere demostrar, que entonces se verifica también la siguiente afirmación:

$$\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}. \tag{3.42}$$

Al considerar la hipótesis (h1) sabemos que existe al menos un número entero verificando esa expresión. Sin pérdida de generalidad podemos considerar que el número entero que verifica (h1) es k_1. De manera que la hipótesis pasa a ser:

$$3x = \frac{(2k_1+1)\pi}{2}. \tag{hk1}$$

Con todo esto, la demostración de (3.42) se tiene de manera directa mediante el uso del número entero k_1 y la hipótesis (hk1).

⟵ Supongamos que se verifica ahora la siguiente hipótesis:

$$\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}. \tag{h2}$$

Se quiere demostrar que entonces, se verifica la siguiente afirmación:

$$\exists k \in \mathbb{Z}, 3x = \frac{(2k+1)\pi}{2}. \tag{3.43}$$

De manera totalmente análoga a como se ha procedido en la implicación anterior; sin pérdida de generalidad, en la hipótesis (h2), como sabemos que existe dicho número entero, se puede denotar como k_2 y la hipótesis pasa a ser:

$$x = \frac{(2k_2+1)\pi}{6}. \tag{hk2}$$

Entonces, ya se puede concluir la demostración, pues basta con usar el mismo número entero, k_2, en (3.43) y usar la hipótesis (hk2).

□

A continuación, se presenta la correspondiente formalización en Lean de la demostración del lema que acabamos de detallar:

```
lemma SolucionCosenoTriple
  {x : ℝ}
  : cos (3 * x) = 0 ↔ ∃ k : ℤ, x = (2 * k + 1) * π / 6 :=
begin
  rw cos_eq_zero_iff,
  split,
  { intro h1,
    cases h1 with k1 hk1,
    use k1,
    linarith,},
  { intro h2,
    cases h2 with k2 hk2,
    use k2,
    linarith,},
end
```

En la formalización de este lema en Lean sólo ha sido necesario el uso de un teorema y es el siguiente:

- `cos_eq_zero_iff : cos θ = 0 ↔ ∃ k : ℤ, θ = (2 * k + 1) * π / 2`

Además,se han usado las tácticas cases, intro, linarith, rw, split y use.

3.2.3. Conclusión

Finalmente, una vez se ha demostrado tanto la equivalencia entre el problema original que queríamos resolver y el problema auxiliar, como las soluciones del problema auxiliar; se puede concluir el ejercicio.

En primer lugar, por lo que hemos visto en la subsección 3.2.2, se sabe que el problema equivalente (y por tanto del problema original) tiene solución en los dos siguientes casos:

$$\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{4} \tag{3.44}$$

$$\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}. \tag{3.45}$$

Esto nos permite definir el conjunto de soluciones del problema como el conjunto de los números reales, x, tales que o bien verifiquen (3.44), o bien (3.45). De manera formal, podemos introducir el siguiente conjunto correspondiente a las soluciones del problema:

$$\text{Solucion} = \left\{ x \in \mathbb{R} \mid \exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{4} \vee x = \frac{(2k+1)\pi}{6} \right\}. \tag{3.46}$$

La formalización en Lean de esta definición es la que se detalla a continuación:

```
def Solucion : set ℝ :=
{x : ℝ | ∃ k : ℤ, x = (2 * k + 1) * π / 4 ∨ x = (2 * k + 1) * π / 6}
```

Por tanto, para concluir el problema basta con introducir un teorema que hará uso de los lemas auxiliares que se han demostrado a lo largo de la sección.

Teorema 3.2.9. Sea x perteneciente al conjunto de los números reales, entonces x es solución la ecuación

$$\cos^2(x) + \cos^2(2x) + \cos^2(3x) = 1, \tag{3.47}$$

si y solamente si x pertenece al conjunto descrito en (3.46).

Demostración: En primer lugar, a través del Lema 3.2.4 se tiene que resolver el problema (3.47) es equivalente a resolver la expresión:

$$\text{funAuxiliar}(x) = 0 \iff x \in \text{Solucion}. \tag{3.48}$$

y, por 3.2.5, a

$$\cos^2(x) = \frac{1}{2} \vee \cos(3x) = 0 \iff x \in \text{Solucion}. \tag{3.49}$$

Entonces, sin más que hacer uso de los dos lemas auxiliares 3.2.6 y 3.2.8 que nos dan las condiciones necesarias y suficientes para que el coseno al cuadrado de x sea un medio (en el caso del primer lema) y para que el coseno del triple de x sea nulo (en el caso del segundo). Se tiene que (3.49) pasa a ser:

$$\left(\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{4}\right) \vee \left(\exists k \in \mathbb{Z}, x = \frac{(2k+1)\pi}{6}\right) \iff x \in \text{Solucion}. \tag{3.50}$$

A continuación, para concluir la demostración del teorema, basta con usar la propiedad distributiva del existencial sobre la disyunción, cuya formalización en Lean es

```
theorem exists_or_distrib :
(∃ x, p x ∨ q x) ↔ (∃ x, p x) ∨ (∃ x, q x)
```

Observando la definición del conjunto (3.46) y haciendo uso de la propiedad anterior se conluye el ejercicio.

□

Finalmente, la formalización en Lean de este lema es:

```
theorem imo1962_q4
  {x : ℝ}
  : problema x ↔ x ∈ Solucion :=
begin
  rw Equivalencia,
  rw CasosSolucion,
  rw SolucionCosenoTriple,
  rw SolucionCosenoCuadrado,
  rw Solucion,
  exact exists_or_distrib.symm,
end
```

Se han usado las tácticas rw y exact.

3.3. IMO 1977 Q6

En esta sección se va a detallar la solución al problema Q6 que se propuso en el año 1977. Comenzaremos presentando el enunciado de dicho problema y su correspondiente solución en lenguaje natural. Posteriormente se presentará la formalización en Lean de dicho problema y el análisis de la misma.

Destacar que la formalización original del problema se ha obtenido de [4] y fue propuesta por Tian Chen.

Problema 3 (1977–Q6). Consideremos la función $f : \mathbb{N}^+ \to \mathbb{N}^+$ satisfaciendo que

$$f(f(n)) < f(n+1)$$

para cualquier número n. Probar que para todo número natural positivo n se verifica que

$$f(n) = n.$$

Nota 3.3.1. En el enunciado del problema anterior se observa que la función f está definida del conjunto $\mathbb{N}^+$ en sí mismo; dicho conjunto se define como el conjunto de los números naturales positivos.

Para llevar a cabo la formalización en Lean del problema que se ha planteado, se va a importar la teoría data.pnat.basic sobre el conjunto formado por los números naturales que son positivos.

En Lean esto lo expresamos como sigue:

```
import data.pnat.basic
```

Para llevar a cabo la resolución del problema que se nos plantea, se va introducir un teorema auxiliar, el cual consistirá en demostrar el mismo resultado que se nos plantea pero para una función que está definida en todo el conjunto de los números naturales. Enunciemos dicho teorema:

Teorema 3.3.1 (Extension)**.** Sea $f : \mathbb{N} \to \mathbb{N}$ una función satisfaciendo que

$$f(f(n)) < f(n+1) \tag{3.51}$$

para cualquier número natural n. Entonces se tiene que para todo número natural n se verifica que

$$f(n) = n. \tag{3.52}$$

Demostración: Suponiendo que se tiene una función f definida entre el conjunto de los números naturales; es decir, $f : \mathbb{N} \to \mathbb{N}$, verificando que

$$f(f(n)) < f(n+1); \tag{h1}$$

se tiene que demostrar que para cualquier número natural se cumple que $f(n)$ es igual a n. Simbólicamente:

$$\forall n \in \mathbb{N}, f(n) = n. \tag{3.53}$$

Previo a la demostración en sí de este resultado, se van a probar una serie de resultados auxiliares que nos serán necesarios para la conclusión del primero. Destacar que estos resultados auxiliares que se van a demostrar se encuentran bajo las mismas condiciones que el problema principal.

Hipótesis h2

Comenzaremos demostrando que dados dos números naturales cualesquiera k, n tales que k es menor o igual que n. Entonces se tiene que k es menor o igual que $f(n)$. Simbólicamente:

$$\forall k, n \in \mathbb{N}, k \leq n \longrightarrow k \leq f(n). \tag{h2}$$

Para demostrar (h2), aplicaremos el método de inducción en el número natural k:

Probar por inducción en k el resultado consiste en demostrarlo para el caso en el que k es nulo y que suponiendo que se verifica para k, entonces se verifica para k más uno. Distingamos entonces los casos:

- **Caso $k = 0$.**

  En este caso, hay que demostrar que

  $$\forall n, 0 \leq f(n), \tag{3.54}$$

  bajo las hipótesis (h1) y que para todo número natural n se verifica que

  $$0 \leq n. \tag{3.55}$$

  Para demostrar (3.54) basta con aplicar que por definición todo número natural es mayor o igual que cero. Entonces, como consecuencia de que la función f está definida del conjunto de los números naturales en sí mismo, se tiene que $f(n)$ siempre va a ser mayor o igual que cero para cualquier n natural.

- **Caso $k+1$.**

  En este caso, asumiendo como cierta la hipótesis de inducción, es decir, suponiendo que se verifica que

  $$\forall n \in \mathbb{N}, k \leq n \longrightarrow k \leq f(n), \tag{h_ind}$$

  hay que demostrar que para todo número natural n que sea mayor o igual que $k+1$, se verifica que

  $$k+1 \leq f(n). \tag{3.56}$$

  Una vez ya se ha introducido la hipótesis de inducción, se va a denotar la hipótesis de que para cualquier número natural n sea mayor igual que $k+1$ como:

  $$k+1 \leq n. \tag{hk}$$

  Ahora bien, demostrar (3.56) es equivalente a demostrar que k sea menor estricto que $f(n)$, esto es:

  $$k < f(n). \tag{3.57}$$

  A continuación, se van a deducir una serie de hipótesis que se tienen de manera casi directa.

  - A partir de la hipótesis (hk), se tiene de manera directa que pasando restando el uno de una parte de la desigualdad a otra, se tiene que

    $$k \leq n-1. \tag{hk1}$$

  - Haciendo uso ahora de la hipótesis de inducción, (h_ind), y de la hipótesis anteriormente deducida, (hk1), se puede deducir que

    $$k \leq f(n-1). \tag{hk2}$$

  - De manera totalmente análoga en como se ha deducido la hipótesis (hk2), se puede deducir que

    $$k \leq f(f(n-1)), \tag{hk3}$$

    sin más que aplicar la hipótesis de inducción, (h_ind), y el resultado anteriormente deducido, (hk2).

- La última hipótesis que introduciremos en este apartado, será haciendo uso de la hipótesis principal del teorema, es decir, (h1). Se particularizará para el caso en el que se considera $n-1$, de esta forma se tendría:

$$f(f(n-1)) < f(n-1+1). \tag{3.58}$$

Ahora bien, la expresión (3.58) se puede reescribir como

$$f(f(n-1)) < f(n). \tag{h11}$$

Esta reescritura se ha llevado a cabo mediante el uso de la cancelación en la suma de los números naturales, esto es:

Proposición 3.3.2. Sean m y n dos números pertenecientes al conjunto de los números naturales tales que m es menor o igual que n. Entonces se tiene que si al número n le restamos y sumamos el número m el resultado que se obtiene es n. Simbólicamente:

$$\forall n, m \in \mathbb{N}, m \leq n \longrightarrow n - m + m = n. \tag{3.59}$$

La formalización en Lean de esta proposición es la que se presenta a continuación:

```
theorem sub_add_cancel {n m : ℕ} (h : m ≤ n) : n - m + m = n
```

De esta forma, al haber hecho uso de la proposición 3.3.2, el objetivo a demostrar que antes era (3.57), ahora se le añade otro nuevo objetivo y es demostrar que además se cumple que el número natural n es mayor o igual que 1, esto es:

$$n \geq 1. \tag{3.60}$$

Por claridad en la notación, estos dos objetivos mencionados se van a dividir en dos problemas diferentes:

1. **Problema 1**

   Demostrar (3.57) es bastante sencillo con ayuda de las hipótesis deducidas anteoriormente. Veámoslo:

$$k \overset{(*)}{\leq} f(f(n-1)) \overset{(**)}{<} f(n), \tag{3.61}$$

   donde en $(*)$ se ha hecho uso de la hipótesis (hk3) y en $(**)$ se ha usado la hipótesis (h11).

2. **Problema 2**

   Ahora se tiene que demostrar (3.60) también bajo todas las hipótesis que se han deducido posteriormente. Aunque realmente para probar este resultado no son necesarias, sólo se necesitan los dos siguientes resultados generales de los números naturales:

   Lema 3.3.3. Sea n un número natural cualquiera, entonces dicho número n es mayor o igual que cero.

   Cuya formalización en Lean es:

```
lemma zero_le : ∀ (n : ℕ), 0 ≤ n
```

   Lema 3.3.4. Sean n y m dos números naturales tales que n es menor o igual que m. Entonces se tiene que el número natural sucesor de n (es decir, $n+1$) es menor o igual que el sucesor de m. Simbólicamente:

$$\forall n, m \in \mathbb{N}, n \leq m \longrightarrow n+1 \leq m+1. \tag{3.62}$$

   Y la formalización en Lean de este lema es la siguiente:

```
lemma succ_le_succ {n m : ℕ} : n ≤ m → succ n ≤ succ m
```

   De esta forma se tendría que si consideramos un número natural k, haciendo uso del lema 3.3.3, se tiene que

$$k \geq 0. \tag{3.63}$$

   Si además, ahora se hace uso del lema 3.3.4 considerando los números naturales k y 0 y la hipótesis (3.63), se puede concluir que

$$1 \leq k+1 \tag{hk0}$$

   Ahora bien, para finalizar la prueba de este problema basta con aplicar la transitividad de las relaciones menor o igual que y menor que sobre las hipótesis (hk) y (hk0).

Una vez ya se han demostrado los objetivos (3.57) y (3.60), se tiene que ya se ha probado el caso $k+1$ del método de inducción.

De esta forma, tras haber terminado la prueba de la hipótesis (h2) por el método de inducción, se procederá a la continuación de la prueba.

Hipótesis hf

Proseguiremos con la demostración de que para cualquier número natural n se verifica que dicho número n es menor o igual que $f(n)$. Simbólicamente:

$$\forall n \in \mathbb{N}, n \leq f(n). \tag{hf}$$

Fijando un número natural n cualesquiera, se tiene que le podemos aplicar la hipótesis (h2) considerando que los dos números naturales que aparecen en el enunciado de esa hipótesis son el mismo n.

Como consecuencia de que sabemos que cualquier número es menor o igual que sí mismo, se tiene que la condición de (h2) se verifica y por tanto se concluye que

$$n \leq f(n), \tag{3.64}$$

que era lo que queríamos demostrar.

Hipótesis mon

El enunciado de la hipótesis que queremos demostrar que se verifica en nuestro problema es que para cualquier número natural n se verifica que $f(n)$ es menor estrictamente que $f(n+1)$. Simbólicamente:

$$\forall n \in \mathbb{N}, f(n) < f(n+1). \tag{mon}$$

Para demostrar que (mon) es cierto bajo las hipótesis de nuestro problema, comenzamos fijando un número natural n.

Ahora bien, por un lado se tiene que aplicando la hipótesis (h1) a dicho número natural n resulta que

$$f(f(n)) < f(n+1). \tag{3.65}$$

Por otro lado, se sabe que la imagen de aplicar la función f a un número natural n es un número natural; es decir, $f(n)$ pertenece al conjunto de los número naturales. Entonces, si aplicamos la hipótesis (hf) a $f(n)$ se tiene que

$$f(n) \leq f(f(n)). \tag{3.66}$$

De manera que utilizando la transitividad de las relaciones menor o igual que y menor que en los resultados obtenidos en (3.65) y (3.66) ya se tendría el resultado deseado, (mon).

Hipótesis f_mon

Este va a ser el último resultado que vamos a demostrar previo a la conclusión en sí del teorema y consiste en demostrar que la función f que se está considerando es estrictamente monótona.

Definición 3.3.5. Se dice que una función f es estrictamente monótona si para cualesquiera a y b tales que a es menor estrictamente que b, se tiene que $f(a)$ es también menor estrictamente que $f(b)$. Simbólicamente:

$$\forall a, b, a < b \longrightarrow f(a) < f(b). \tag{3.67}$$

Esta definición en Lean tiene la siguiente formalización:

```
def strict_mono [has_lt α] [has_lt β] (f : α → β) : Prop :=
∀ ⦃a b⦄, a < b → f a < f b
```

En el caso de nuestra función, que se encuentra definida en el conjunto de los números naturales, existe un lema el cual nos da las condiciones para que la función sea estrictamente monótona.

Lema 3.3.6. Sea f una función que tiene como conjunto de salida el conjunto de los números naturales y sea el conjunto de llegada un conjunto cualquiera que sea preorden, es decir, $f : \mathbb{N} \to \beta$ donde se ha denotado β como el preorden. Se tiene que si para cualquier número natural n se verifica que $f(n)$ es menor estrictamente que $f(n+1)$, entonces f es una función estrictamente monótona.

Y su correspondiente formalización en Lean sería:

```
lemma nat {β} [preorder β] {f : ℕ → β} (h : ∀n, f n < f (n+1)) :
strict_mono f
```

De esta forma, como nuestra función f verifica la hipótesis (hf), al hacer uso del lema 3.3.6 se tendría ya que la función de nuestro problema es estrictamente monótona.

Una vez ya se han introducido todos los resultados auxiliares que eran necesarios, recordemos que el objetivo a demostrar era que dada la función f definida entre el conjunto de los números naturales tal que se verificaba la hipótesis (h1). Había que demostrar que para cualquier número natural n se verifica que $f(n)$ es igual a n. Simbólicamente era:

$$\forall n, f(n) = n. \tag{3.68}$$

Para demostrar (3.68), fijamos un número natural n, entonces se tiene que probar que se verifica la igualdad

$$f(n) = n. \tag{3.69}$$

Haciendo uso de la hipótesis (hf) para el número natural n que estamos considerando se tiene entonces que

$$n \leq f(n). \tag{3.70}$$

Por tanto, haciendo uso de esto, el objetivo (3.69) pasa a convertirse en

$$f(n) < n + 1. \tag{3.71}$$

Para concluir la demostración basta con probar (3.71); esto se puede hacer de manera directa con la hipótesis principal del problema, (h1), y la monotonía de la función f.

□

La formalización en Lean de este teorema sería la siguiente:

```
theorem Extension
  (f : ℕ → ℕ)
  (h1 : ∀ n, f (f n) < f (n + 1))
  : ∀ n, f n = n :=
begin
  have h2: ∀ (k n : ℕ), k ≤ n → k ≤ f n,
  { intro k,
     induction k with k h_ind,
     { intros n hn,
       exact nat.zero_le (f n), },
     { intros n hk,
       apply nat.succ_le_of_lt,
       rw nat.succ_eq_add_one at hk,
       have hk1: k ≤ n-1 := nat.le_sub_right_of_add_le hk,
       have hk2: k ≤ f (n-1):= h_ind (n-1) hk1,
       have hk3: k ≤ f(f(n-1)) := h_ind (f(n-1)) hk2,
       have h11: f (f (n-1)) < f(n-1+1):= h1 (n-1),
       rw nat.sub_add_cancel at h11,
       { calc k ≤ f(f(n-1)) : hk3
            ... < f(n)      : h11,},
       have hk0: 1 ≤ k+1 := nat.succ_le_succ (nat.zero_le k),
       exact (le_trans hk0 hk), }},
  have hf: ∀ n, n ≤ f n,
    { intro n,
      apply h2 n n,
      exact le_rfl, },
  have mon: ∀ n, f n < f(n+1),
    { intro n,
      exact lt_of_le_of_lt (hf (f n)) (h1 n), },
```

```
  have f_mon: strict_mono f := strict_mono.nat mon,
  intro n,
  apply nat.eq_of_le_of_lt_succ (hf n),
  exact (f_mon.lt_iff_lt.mp (h1 n)),
end
```

En la formalización de este teorema Lean han sido necesarios el uso de los siguiente lemas y teoremas:

- `nat.zero_le : ∀ (n: ℕ), 0≤n`
- `succ_le_of_lt {a b : ℕ} (h : a < b) : succ a ≤ b`
- `succ_eq_add_one (n : ℕ) : succ n = n + 1`
- `le_sub_right_of_add_le (h : m + k ≤ n) : m ≤ n - k`
- `sub_add_cancel {n m : ℕ} (h : m ≤ n) : n - m + m = n`
- `succ_le_succ {n m : ℕ} : n ≤ m → succ n ≤ succ m`
- `le_trans : ∀ {a b c : α}, a ≤ b → b ≤ c → a ≤ c`
- `le_rfl [preorder α] {x : α} : x ≤ x`
- `lt_of_le_of_lt : ∀ {a b c : α}, a ≤ b → b < c → a < c`
- `nat {β} [preorder β] {f : ℕ → β} (h : ∀n, f n < f (n+1)) :  strict_mono f`
- `eq_of_le_of_lt_succ {n m : ℕ} (h₁ : n ≤ m) (h₂ : m < n + 1) : m = n`

Además,se han usado las tácticas apply, exact, have, induction, intro y rw.

Una vez ya se ha demostrado el teorema 3.3.1, se va a introducir el teorema con el cual el problema estaría resuelto. El teorema que se va a presentar a continuación no es más que un caso particular del teorema 3.3.1, ya probado.

La diferencia entre estos dos teoremas es que mientras que en 3.3.1 se consideraba que la función f estaba definida del conjunto de los números naturales en sí mismo; en el siguiente teorema, dicha función estará definida del conjunto de los números naturales positivos en sí mismo. Veámoslo:

Teorema 3.3.7 (imo1977_q6)**.** Sea $f : \mathbb{N}^+ \to \mathbb{N}^+$ una función satisfaciendo que

$$f(f(n)) < f(n+1) \tag{3.72}$$

para cualquier número natural positivo n. Entonces se tiene que para todo número natural positivo n se verifica que

$$f(n) = n. \tag{3.73}$$

Demostración: Considerando la función f definida del conjunto de los números naturales positivos en sí mismo verificando que

$$\forall n \in \mathbb{N}^+, f(f(n)) < f(n+1). \tag{h}$$

Entonces, se quiere probar que para cualquier número natural positivo n se cumple (3.73).

En primer lugar, fijemos un número natural positivo arbitrario n. A continuación, se hace uso del teorema 3.3.1, el cual está definido sobre el conjunto de los números naturales. Aplicamos este teorema sobre una función que está definida sobre el conjunto de los número naturales, de manera que si consideramos un número m natural se distinguen dos casos:

1. Que el número natural m sea positivo. En esta situación, la función se define como $f(m)$, donde f es la definida en el enunciado de este teorema.
2. Que el número natural m sea nulo. En este caso la aplicación definida nos lleva este número al cero.

Si denotamos esta función por f_1, simbólicamente se tendría:

$$f_1(m) = \begin{cases} f(m), & \text{si } m > 0, \\ 0, & \text{si } m = 0. \end{cases} \tag{3.74}$$

Ahora bien, para poder aplicar el teorema 3.3.1 y de esta forma poder concluir la demostración, faltaría probar que se verifica la hipótesis del problema. Esto es:

$$\forall x \in \mathbb{N}, f_1(f_1(x)) < f_1(x+1), \tag{3.75}$$

Para ello, comenzaremos fijando un número natural x arbitrario. De esta forma, se tiene que probar que

$$f_1(f_1(x)) < f_1(x+1), \tag{3.76}$$

Como consecuencia de como se ha definido la función f_1, (3.74), para demostrar (3.76) vamos a dividir la prueba en dos casos distintos:

- **Caso** $x = 0$

  En este caso la expresión a demostrar (3.76), se convierte en

  $$f_1(f_1(0)) < f_1(0+1). \tag{3.77}$$

  Sin más que aplicar la definición de la función, (3.74), se tiene que demostrar (3.77) se convierte en

  $$0 < f(1). \tag{3.78}$$

  Este resultado se tiene de manera inmediata pues el conjunto imagen de la función f es el conjunto de los números naturales positivos.

- **Caso** $x > 0$

  En este caso, se tiene que por la definición de f_1, probar (3.76) se convierte en

  $$f(f(x)) < f(x+1). \tag{3.79}$$

  Pero este resultado es inmediato haciendo uso de la hipótesis (h) y de esta forma se tendría también demostrado este caso.

Concluyendo, tras la demostración de estos dos casos, ya se tendría la prueba del teorema que buscábamos.

□

Finalmente, la formalización en Lean de este teorema sería:

```
theorem imo1977_q6
  (f : ℕ+ → ℕ+)
  (h : ∀ n, f (f n) < f (n + 1))
  : ∀ n, f n = n :=
begin
  intro n,
  simpa using Extension (λ m, if 0 < m then f m.to_pnat' else 0) _ n,
  intro x,
  cases x,
  { simp },
  { simpa using h _},
end
```

En la formalización en Lean de este teorema sólo ha sido necesario el uso de la siguiente aplicación:

```
def to_pnat' (n : ℕ) : ℕ+ := succ_pnat (pred n)
```

Además,se han usado las tácticas cases, intro, simpa y simp.

3.4. IMO 2001 Q2

En esta nueva sección, se va a detallar la resolución del problema Q2 propuesto en el año 2001. Como en el resto de planteamientos anteoriores que se han detallado en el capítulo, se comenzará presentado la demostración en lenguaje natural del problema y luego su correspondiente formalización en Lean.

La formalización en Lean de este problema se ha basado en la que recientemente propuso Tian Chen y que se puede encontrar en [5].

Problema 4 (2001–Q2). Consideremos a, b y c tres números reales y positivos cualesquiera. Demostrar que

$$\frac{a}{\sqrt{a^2+8bc}}+\frac{b}{\sqrt{b^2+8ca}}+\frac{c}{\sqrt{c^2+8ab}}\geq 1.$$

Para llevar a cabo la formalización en Lean de este problema, es necesario la importación de la teoría analysis.special_functions.pow (sobre las potencias) y también habilitar el espacio de nombre de los números reales (para la simplificación de notación de algunas funciones).

En Lean todo lo detallado se formaliza como sigue:

```
import analysis.special_functions.pow

open real
```

Además, se han de introducir las variables a, b y c que en este problema pertenecen al conjunto de los números reales (la condición de que son positivos se introducirá como hipótesis). En Lean esto se formaliza de la siguiente forma:

```
variables {a b c : ℝ}
```

3.4.1. Resultados auxiliares

Para llevar a cabo la resolución de este problema, se van a utilizar una serie de lemas y teoremas auxiliares.

El primer resultado auxiliar es el que se presenta a continuación:

Lema 3.4.1 (suma_pos)**.** Sean a, b y c tres números reales y positivos, esto es:

$$0 < a, \tag{ha}$$
$$0 < b, \tag{hb}$$
$$0 < c. \tag{hc}$$

Entonces, se tiene que para estos tres números se verifica la siguiente desigualdad:

$$0 < a^4 + b^4 + c^4. \tag{3.80}$$

Demostración: La demostración de este lema es bastante sencilla, se comienza usando el resultado de que la potencia de cualquier número positivo es también positiva. Utilizando esto sobre las hipótesis (ha), (hb) y (hc), se tienen las tres siguientes consecuencias:

$$0 < a^4, \tag{ha1}$$
$$0 < b^4, \tag{hb1}$$
$$0 < c^4. \tag{hc1}$$

Para finalizar, basta con usar dos veces el resultado que nos dice que la suma de dos números positivos es positiva. Esto supone que, si usamos este resultado sobre las hipótesis (ha1) y (hb1) se tiene que

$$0 < a^4 + b^4. \tag{sum1}$$

Reiteramos este procedimiento ahora usando el mismo resultado sobre las hipótesis (sum1) y (hc1) y se obtiene:

$$0 < a^4 + b^4 + c^4. \tag{3.81}$$

De esta forma, ya se tendría el resultado deseado.

□

La formalización en Lean de este lema es la siguiente:

```
lemma suma_pos
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
  : 0 < a ^ 4 + b ^ 4 + c ^ 4 :=
begin
  have ha1 : 0 < a ^ 4          := pow_pos ha 4,
  have hb1 : 0 < b ^ 4          := pow_pos hb 4,
  have hc1 : 0 < c ^ 4          := pow_pos hc 4,
  have sum1 : 0 < a ^ 4 + b ^ 4 := add_pos ha1 hb1,
  exact (add_pos sum1 hc1),
end
```

Para la formalización en Lean de este lema se han utilizado los siguientes lemas auxiliares:

- `pow_pos {a : R} (H : 0 < a) : ∀ (n : ℕ), 0 < a ^ n`
- `add_pos {a b : α} (h: 0 < a) (h': 0 < b), 0 < a + b`

Además, las únicas dos tácticas que se han usado son exact y have.

Para proseguir con la resolución del problema se va a introducir el segundo resultado auxiliar. Éste consiste en demostrar que para cualesquiera tres números reales positivos a, b y c, se verifica una cota que nos resultará de gran utilidad más adelante.

Lema 3.4.2 (cota)**.** Sean a, b y c tres números reales y positivos, esto es:

$$0 < a, \tag{ha}$$
$$0 < b, \tag{hb}$$
$$0 < c. \tag{hc}$$

Entonces, se tiene que para estos números a, b y c se verifica la siguiente desigualdad:

$$\frac{a^4}{a^4 + b^4 + c^4} \leq \frac{a^3}{\sqrt{(a^3)^2 + 8b^3c^3}}. \tag{3.82}$$

Demostración: La demostración consiste en que bajo las hipótesis de que los tres números que estamos considerando son positivos, es decir, (ha), (hb) y (hc); se tiene que probar la expresión (3.82).

Para llevar a cabo tal prueba, se comenzará introduciendo una serie de hipótesis que se pueden deducir fácilmente:

- Por la hipótesis (ha) se tiene que a es un número positivo, entonces cualquier potencia de él lo es y en particular:

$$0 < a^3. \tag{ha3}$$

- Se tiene que cualquier número elevado a la segunda potencia es siempre mayor o igual que cero. El caso particular que nos interesa es:

$$0 \leq (a^3)^2. \tag{ha32}$$

- Análogamente a como se ha procedido en la hipótesis (ha3), se tiene que al multiplicar la potencia de un número positivo por cualquier número positivo, el resultado también es positivo. En nuestro caso considerando la tercera potencia de b y el número ocho:

$$0 < 8b^3. \tag{hb8}$$

- Ahora bien, se sabe que al multiplicar dos números positivos, el resultado es también un número positivo. Por tanto, si aplicamos dicho resultado a la hipótesis anterior, (hb8), y a la tercera potencia de c (que también es positivo por analogía con la hipótesis (ha3)), se tiene que:

$$0 < 8b^3c^3. \tag{hbc}$$

- Haciendo uso del resultado que nos dice que la suma de dos números positivos es positiva sobre las hipótesis (ha32) y (hbc), se obtiene el siguiente resultado:

$$0 < (a^3)^2 + 8b^3c^3 \tag{hdenom1}$$

- Para la deducción de esta hipótesis se hará uso del siguiente teorema:

  Teorema 3.4.3. Sea x un número cualquiera. Entonces se tiene que la raíz cuadrada de x es positiva si y solamente si x es positivo. Simbólicamente:

$$\sqrt{x} > 0 \iff x > 0. \tag{3.83}$$

  Cuya formalización en Lean es:

```
theorem sqrt_pos : 0 < sqrt x ↔ 0 < x
```

Entonces se tiene que aplicando dicho resultado sobre (hbc), se obtiene la hipótesis buscada:

$$0 < \sqrt{(a^3)^2 + 8b^3c^3}. \tag{hsqrt}$$

- Usando el lema 3.4.1 sobre los números a, b y c, puesto que se satisfacen las hipótesis (ha), (hb) y (hc); se tiene que:

$$0 < a^4 + b^4 + c^4. \tag{hdenom2}$$

- Por último, de manera inmediata se tiene que si un número es positivo, entonces dicho número es mayor o igual que cero. Aplicando esto sobre el número (hdenom2), se obtiene que:

$$0 \leq a^4 + b^4 + c^4. \tag{hdenom3}$$

De esta manera, una vez se han introducido estas hipótesis, recordemos que el objetivo a demostrar era (3.82). A continuación, como consecuencia de que los dos denominadores de dicha desigualdad son positivos (probado en (hdenom2) y (hsqrt)), se tiene que podemos pasar multiplicando los denominadores sin que se cambie de sentido la desigualdad. Esto es:

$$a^4\sqrt{(a^3)^2 + 8b^3c^3} \leq a^3(a^4 + b^4 + c^4). \tag{3.84}$$

La expresión (3.84) puede ser reescrita haciendo uso de la propiedad asociativa de la multiplicación y separando la potencia a la cuarta de a como el producto de la potencia tercera de a por a. De esta forma se obtendría:

$$a^3(a\sqrt{(a^3)^2 + 8b^3c^3}) \leq a^3(a^4 + b^4 + c^4). \tag{3.85}$$

Aplicando ahora que como la potencia al cubo de a es un número mayor o igual que cero (porque ya se ha visto en la hipótesis (ha3) que es un número positivo), se puede simplificar la expresión (3.85) puesto que ambos lados de la igualdad están multiplicados por a al cubo. De manera que el objetivo a demostrar se convierte en:

$$a\sqrt{(a^3)^2 + 8b^3c^3} \leq a^4 + b^4 + c^4. \tag{3.86}$$

Proseguimos trabajando con la expresión (3.86); procedemos a aplicar que cuando se tiene una desigualdad cuyos términos son mayores o iguales

que cero, como es nuestro caso, se mantiene la desigualdad al elevar ambos términos a la misma potencia (siempre y cuando el número de la potencia sea mayor estricto que cero). Entonces, aplicando esto sobre la expresión (3.86) y elevando al cuadrado, se tiene que:

$$(a\sqrt{(a^3)^2 + 8b^3c^3})^2 \leq (a^4 + b^4 + c^4)^2. \tag{3.87}$$

Para finalizar esta parte de la demostración, aplicaremos los siguientes resultados:

1. El producto de dos números elevado a cualquier potencia es el producto del primer término a dicha potencia por el segundo también a la misma potencia.
2. El resultado de la potencia al cuadrado de la raiz cuadrada de un número es dicho número.

Además de estos dos resultados, pasaremos restando el primer término de la desigualdad al segundo. De manera que aplicando los tres resultados de manera conjunta sobre (3.87) se tendría la siguiente expresión:

$$0 \leq (a^4 + b^4 + c^4)^2 - a^2((a^3)^2 + 8b^3c^3). \tag{3.88}$$

A continuación, para poder seguir con el desarrollo de la expresión a demostrar, es decir, (3.88), se van a introducir otra serie de hipótesis al igual que hicimos al principio de la demostración.

- **Hipótesis desarrollo**

  Esta hipótesis consiste en que para cualesquiera números reales a, b ó c, se tiene que:

$$\begin{aligned}&(a^4 + b^4 + c^4)^2 - a^2((a^3)^2 + 8b^3c^3) \qquad \text{(desarrollo)}\\ &= 2(a^2(b^2 - c^2))^2 + (b^4 - c^4)^2 + (2(a^2bc - b^2c^2))^2.\end{aligned}$$

  Para demostrar esta hipótesis desarrollaremos los dos términos de la igualdad y veremos que se obtiene lo mismo.

  Término 1:

$$\begin{aligned}&(a^4 + b^4 + c^4)^2 - a^2((a^3)^2 + 8b^3c^3)\\ &= (a^4 + b^4)^2 + (c^4)^2 + 2(a^4 + b^4)c^4 - a^8 - 8a^2b^3c^3\\ &= a^8 + b^8 + 2a^4b^4 + c^8 + 2a^4c^4 + 2b^4c^4 - a^8 - 8a^2b^3c^3\\ &= b^8 + c^8 + 2a^4b^4 + 2a^4c^4 + 2b^4c^4 - 8a^2b^3c^3.\end{aligned}$$

Término 2:

$$\begin{aligned}
&2(a^2(b^2-c^2))^2+(b^4-c^4)^2+(2(a^2bc-b^2c^2))^2\\
&=2a^4(b^4+c^4-2b^2c^2)+b^8+c^8-2b^4c^4+4(a^4b^2c^2-b^4c^4-2a^2b^3c^3)\\
&=2a^4b^4+2a^4c^4-4a^4b^2c^2+b^8+c^8-2b^4c^4+4a^4b^2c^2+4b^4c^4-8a^2b^3c^3\\
&=b^8+c^8+2a^4b^4+2a^4c^4+2b^4c^4-8a^2b^3c^3.
\end{aligned}$$

De esta forma, ya se tendría demostrada la igualdad (desarrollo).

- **Hipótesis h1, h2 y h3:**

  Como ya usamos en la hipótesis (ha32), cualquier número elevado al cuadrado es siempre mayor o igual que cero. En nuestro caso aplicaremos este resultado para tres números diferentes obteniendo así las tres siguientes hipótesis:

$$0 \leq (a^2(b^2-c^2))^2, \tag{h1}$$
$$0 \leq (b^4-c^4)^2, \tag{h2}$$
$$0 \leq (2(a^2bc-b^2c^2))^2. \tag{h3}$$

- **Hipótesis h1':**

  Además, en el caso de la hipótesis (h1), nos interesa tener que dicho término multiplicado por dos es también mayor o igual que cero. Lo cual se tiene de manera inmediata porque ambos términos son mayores o iguales que cero. Entonces:

$$0 \leq 2(a^2(b^2-c^2))^2. \tag{h1'}$$

De esta forma, se tiene que ya se puede terminar la demostración del problema. Para ello, apliquemos el resultado de que la suma de dos números no negativos es también mayor igual que cero sobre las hipótesis (h2) y (h3). Obteniendo así:

$$0 \leq (b^4-c^4)^2+(2(a^2bc-b^2c^2))^2. \tag{aux1}$$

Aplicando de nuevo este mismo resultado sobre (h1') y el que se acaba de obtener, (aux1), se tiene que:

$$0 \leq 2(a^2(b^2-c^2))^2+(b^4-c^4)^2+(2(a^2bc-b^2c^2))^2. \tag{tesis}$$

Entonces, finalmente volviendo a la expresión (3.88), que era el objetivo a demostrar, y usando la hipótesis (desarrollo), se tiene que el objetivo a demostrar se convierte en:

$$0 \leq 2(a^2(b^2-c^2))^2+((b^4-c^4)^2+(2(a^2bc-b^2c^2))^2). \tag{3.89}$$

Usando la propiedad asociativa de la suma, se tendría que el objetivo a demostrar es (tesis), lo cual ya nos probaría el lema.

□

Veamos la formalización en Lean de este lema:

```
lemma cota
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
  : a ^ 4 / (a ^ 4 + b ^ 4 + c ^ 4) ≤
    a ^ 3 / sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) :=
begin
  have ha3 : 0 < a ^ 3 :=
    pow_pos ha 3,
  have ha32 : 0 ≤ (a ^ 3) ^ 2 :=
    pow_two_nonneg (a ^ 3),
  have hb8 : 0 < 8 * b ^ 3 :=
    mul_pos (by norm_num) (pow_pos hb 3),
  have hbc : 0 < 8 * b ^ 3 * c ^ 3 :=
    mul_pos hb8 (pow_pos hc 3),
  have hdenom1 : 0 < (a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3 :=
    add_pos_of_nonneg_of_pos ha32 hbc,
  have hsqrt : 0 < sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) :=
    sqrt_pos.mpr hdenom1,
  have hdenom2 : 0 < a ^ 4 + b ^ 4 + c ^ 4 :=
    suma_pos ha hb hc,
  have hdenom3 : 0 ≤ a ^ 4 + b ^ 4 + c ^ 4 :=
    le_of_lt hdenom2,
  rw div_le_div_iff hdenom2 hsqrt,
  rw pow_succ',
  rw mul_assoc,
  apply mul_le_mul_of_nonneg_left _ (le_of_lt ha3),
  rw ← pow_succ',
  apply le_of_pow_le_pow _ hdenom3 zero_lt_two,
  rw mul_pow,
  rw sq_sqrt (le_of_lt hdenom1),
  rw ← sub_nonneg,
  have desarrollo :
    (a ^ 4 + b ^ 4 + c ^ 4) ^ 2 -
       a ^ 2 * ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3)
    = 2 * (a ^ 2 * (b ^ 2 - c ^ 2)) ^ 2 + (b ^ 4 - c ^ 4) ^ 2 +
      (2 * (a ^ 2 * b * c - b ^ 2 * c ^ 2)) ^ 2 :=
    by ring,
  have h1 : 0 ≤ (a ^ 2 * (b ^ 2 - c ^ 2)) ^ 2 :=
```

```
    pow_two_nonneg _,
  have h1' : 0 ≤ 2 * (a ^ 2 * (b ^ 2 - c ^ 2)) ^ 2:=
    mul_nonneg zero_le_two h1,
  have h2 : 0 ≤ (b ^ 4 - c ^ 4) ^ 2:=
    pow_two_nonneg _,
  have h3 : 0 ≤ (2 * (a ^ 2 * b * c - b ^ 2 * c ^ 2)) ^ 2 :=
    pow_two_nonneg _,
  have aux1 : 0 ≤ (b ^ 4 - c ^ 4) ^ 2 +
                  (2 * (a ^ 2 * b * c - b ^ 2 * c ^ 2)) ^ 2 :=
    add_nonneg h2 h3,
  have tesis : 0 ≤ 2 * (a ^ 2 * (b ^ 2 - c ^ 2)) ^ 2 +
                   ((b ^ 4 - c ^ 4) ^ 2 +
                    (2 * (a ^ 2 * b * c - b ^ 2 * c ^ 2)) ^ 2) :=
    add_nonneg h1' aux1,
  rw desarrollo,
  rw add_assoc,
  exact tesis,
end
```

En la formalización de este teorema Lean han sido necesarios el uso de los siguiente lemas y teoremas:

- `pow_pos : 0 < a → ∀ (n : ℕ), 0 < a ^ n`
- `pow_two_nonneg : (∀ a : R), 0 ≤ a ^ 2`
- `mul_pos : 0 < a → 0 < b → 0 < a * b`
- `add_pos_of_nonneg_of_pos : 0 ≤ a → 0 < b → 0 < a + b`
- `sqrt_pos : 0 < sqrt x ↔ 0 < x`
- `div_le_div_iff 0 < b → 0 < d → (a / b ≤ c / d ↔ a * d ≤ c * b)`
- `pow_succ' (a : M) (n : ℕ) : a^(n+1) = a^n * a`
- `mul_assoc : a * b * c = a * (b * c)`
- `mul_le_mul_of_nonneg_left : a ≤ b → 0 ≤ c → c * a ≤ c * b`

- `le_of_pow_le_pow : ∀ (n : ℕ), 0 ≤ b → 0 < n → a ^ n ≤ b ^ n → a ≤ b`
- `mul_pow (a b : M) (n : ℕ) : (a * b)^n = a^n * b^n`
- `sq_sqrt : 0 ≤ x → sqrt x ^ 2 = x`
- `sub_nonneg : 0 ≤ a - b ↔ b ≤ a`
- `mul_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a * b`
- `add_nonneg : 0 ≤ a → 0 ≤ b → 0 ≤ a + b`
- `add_assoc : a + b + c = a + (b + c)`

Además,se han usado las tácticas apply, exact, have y rw.

Por último, el tercer resultado auxiliar para resolver el ejercicio se trata de un teorema que será la herramienta principal para resolver el problema en sí y que hará uso de los dos lemas auxiliares que se han presentado anteriormente. El teorema es el siguiente:

Teorema 3.4.4. Sean a, b y c tres números reales y positivos, esto es:

$$0 < a, \tag{ha}$$
$$0 < b, \tag{hb}$$
$$0 < c. \tag{hc}$$

Entonces, se tiene que para estos números a, b y c se verifica la siguiente desigualdad:

$$1 \leq \frac{a^3}{\sqrt{(a^3)^2 + 8b^3c^3}} + \frac{b^3}{\sqrt{(b^3)^2 + 8c^3a^3}} + \frac{c^3}{\sqrt{(c^3)^2 + 8a^3b^3}}. \tag{3.90}$$

Demostración: Bajo las hipótesis (ha), (hb) y (hc), se va a demostrar que se tiene la desigualdad (3.90).

Para ello, se van a demostrar cinco hipótesis mediante las cuales se tendrá de manera inmediata el resultado:

- **Hipótesis h1:**

  Esta hipótesis consiste en que aplicando las de los anillos, se tiene el siguiente resultado:

  $$b^4 + c^4 + a^4 = a^4 + b^4 + c^4. \tag{h1}$$

- **Hipótesis h2:**

  De manera totalmente análoga al resultado anterior, (h1), se tiene que:

$$c^4 + a^4 + b^4 = a^4 + b^4 + c^4. \tag{h2}$$

- **Hipótesis h3:**

  Esta hipótesis consiste en que para cualesquiera tres números reales a, b y c que sean positivos, se verifica la siguiente desigualdad:

$$\frac{a^4}{a^4+b^4+c^4} + \frac{b^4}{b^4+c^4+a^4} \leq \frac{a^3}{\sqrt{(a^3)^2+8b^3c^3}} + \frac{b^3}{\sqrt{(b^3)^2+c^3a^3}} \tag{h3}$$

  Para ver que se verifica esta desigualdad se hará uso del lema auxiliar 3.4.2. Si aplicamos dicho lema sobre los números a, b y c teniendo en cuenta las hipótesis (ha), (hb) y (hc), se tiene:

$$\frac{a^4}{a^4+b^4+c^4} \leq \frac{a^3}{\sqrt{(a^3)^2+8b^3c^3}}. \tag{3.91}$$

  Mientras que si se lo aplicamos a los números b, c y a (en ese orden) teniendo en cuenta las mismas hipótesis; se verifica la siguiente desigualdad:

$$\frac{b^4}{b^4+c^4+a^4} \leq \frac{b^3}{\sqrt{(b^3)^2+c^3a^3}} \tag{3.92}$$

  A partir de estas dos desigualdades, se tiene que si sumamos los dos términos que son menores de (3.91) y de (3.92) serán menor o igual que la suma de los dos términos mayores de las mismas desigualdades. Es decir, se tendría probado (h3).

- **Hipótesis h4:**

  Esta hipótesis consiste en demostrar que para cualesquiera tres números reales a, b y c que sean positivos, se verifica la siguiente desigualdad:

$$\begin{aligned}&\frac{a^4}{a^4+b^4+c^4} + \frac{b^4}{b^4+c^4+a^4} + \frac{c^4}{c^4+a^4+b^4} \\ &\leq \frac{a^3}{\sqrt{(a^3)^2+8b^3c^3}} + \frac{b^3}{\sqrt{(b^3)^2+c^3a^3}} + \frac{c^3}{\sqrt{(c^3)^2+a^3b^3}}.\end{aligned} \tag{h4}$$

  Para ello, se procederá de manera análoga a como se hizo en la hipótesis (h3). En primer lugar, aplicando el lema auxiliar 3.4.2 sobre los números

reales c, a y b (en dicho orden) y teniendo en cuenta las hipótesis (hc), (ha) y (hb), se tendría:

$$\frac{c^4}{c^4+a^4+b^4} \leq \frac{c^3}{\sqrt{(c^3)^2+8a^3b^3}}. \tag{3.93}$$

Por tanto, procediendo de manera totalmente análoga a como se hizo en la hipótesis anterior y aplicando el resultado sobre la suma de las desigualdades, pero ahora en este caso con (h3) y (3.93), se obtiene (h4).

- **Hipótesis igualdad:**

  La última hipótesis a demostrar consiste en que para cualesquiera tres números reales y positivos a, b y c, se ha de verificar la siguiente igualdad:

$$\frac{a^4}{a^4+b^4+c^4}+\frac{b^4}{b^4+c^4+a^4}+\frac{c^4}{c^4+a^4+b^4}=1. \tag{igualdad}$$

  En primer lugar, utilizamos las hipótesis (h1) y (h2) para reescribir los denominadores de la segunda y tercera fracción, respectivamente. Se obtiene que el objetivo a demostrar que antes era (igualdad), ahora es:

$$\frac{a^4}{a^4+b^4+c^4}+\frac{b^4}{a^4+b^4+c^4}+\frac{c^4}{a^4+b^4+c^4}=1. \tag{3.94}$$

  Introduzcamos el siguiente lema:

  Lema 3.4.5. Supongamos que tenemos la suma de dos fracciones divididas por el mismo denominador. Entonces se tiene que se puede sacar denominador común y que el numerador sería la suma de los dos numeradores iniciales.

  Cuya formalización en Lean es:

```
lemma add_div (a b c : K) : (a + b) / c = a / c + b / c
```

  A continuación, aplicando dicho resultado dos veces en (3.94) (pues tenemos la suma de tres fracciones), se tendría que:

$$\frac{a^4+b^4+c^4}{a^4+b^4+c^4}=1. \tag{3.95}$$

Como tenemos que el numerador y el denominador de (3.95) es el mismo, se tendría que esa fracción es uno, siempre y cuando el número que aparece sea distinto de cero. Es decir, se ha de verificar que

$$0 < a^4 + b^4 + c^4. \tag{3.96}$$

Lo cual se tiene de manera inmediate a través del primer lema que se ha demostrado en esta sección, es decir, el lema 3.4.1.

De esta forma, una vez se han introducido estas cinco hipótesis se puede concluir la demostración del teorema muy fácilmente. Bastaría con introducir la expresión (igualdad) en la hipótesis (h4) y ya se tendría el resultado; es decir, se tendría (3.90).

□

```
theorem imo2001_q2_aux
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
  : 1 ≤ a ^ 3 / sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) +
        b ^ 3 / sqrt ((b ^ 3) ^ 2 + 8 * c ^ 3 * a ^ 3) +
        c ^ 3 / sqrt ((c ^ 3) ^ 2 + 8 * a ^ 3 * b ^ 3) :=
begin
  have h1 : b ^ 4 + c ^ 4 + a ^ 4 = a ^ 4 + b ^ 4 + c ^ 4,
    by ring,
  have h2 : c ^ 4 + a ^ 4 + b ^ 4 = a ^4 + b ^ 4 + c ^ 4,
    by ring,
  have h3 : a ^ 4 / (a ^ 4 + b ^ 4 + c ^ 4) +
            b ^ 4 / (b ^ 4 + c ^ 4 + a ^ 4) ≤
            a ^ 3 / sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) +
            b ^ 3 / sqrt ((b ^ 3) ^ 2 + 8 * c ^ 3 * a ^ 3) :=
    add_le_add (cota ha hb hc) (cota hb hc ha),
  have h4 : a ^ 4 / (a ^ 4 + b ^ 4 + c ^ 4) +
            b ^ 4 / (b ^ 4 + c ^ 4 + a ^ 4) +
            c ^ 4 / (c ^ 4 + a ^ 4 + b ^ 4) ≤
            a ^ 3 / sqrt ((a ^ 3) ^ 2 + 8 * b ^ 3 * c ^ 3) +
            b ^ 3 / sqrt ((b ^ 3) ^ 2 + 8 * c ^ 3 * a ^ 3) +
            c ^ 3 / sqrt ((c ^ 3) ^ 2 + 8 * a ^ 3 * b ^ 3) :=
    add_le_add h3 (cota hc ha hb),
  have igualdad : a ^ 4 / (a ^ 4 + b ^ 4 + c ^ 4) +
                  b ^ 4 / (b ^ 4 + c ^ 4 + a ^ 4) +
                  c ^ 4 / (c ^ 4 + a ^ 4 + b ^ 4) = 1,
    { rw h1,
```

```
      rw h2,
      rw ← add_div,
      rw ← add_div,
      rw div_self,
      apply ne_of_gt,
      exact suma_pos ha hb hc,},
  rw igualdad at h4,
  exact h4,
end
```

Para llevar a cabo la formalización de este teorema en Lean han sido necesarios el uso de los siguiente lemas y teoremas:

- `add_assoc ∀ {G : Type} {a b c : G}, a + b + c = a + (b + c)`
- `add_comm ∀ {G : Type} {a b c : G}, a + b  = b + a`
- `add_le_add : a ≤ b → c ≤ d → a + c ≤ b + d`
- `add_div (a b c : K) : (a + b) / c = a / c + b / c`
- `div_self {a : G₀} (h : a ≠ 0) : a / a = 1`
- `ne_of_gt {a b : α} (h : b < a) : a ≠ b`

Además,se han usado las tácticas apply, exact, have y rw.

3.4.2. Conclusión del problema

Una vez ya se han introducido y formalizado los tres resultados axiliares, se procederá a la formalización en sí de la resolución como tal del problema. Para llevar a cabo tal resolución se hará uso de un teorema que demuestra el resultado que nos interesa.

Teorema 3.4.6. Sean a, b y c tres números reales y positivos, esto es:

$$0 < a, \tag{ha}$$
$$0 < b, \tag{hb}$$
$$0 < c. \tag{hc}$$

Entonces, se tiene que para estos números a, b y c se verifica la siguiente desigualdad:

$$1 \leq \frac{a}{\sqrt{a^2+8bc}} + \frac{b}{\sqrt{b^2+8ca}} + \frac{c}{\sqrt{c^2+8ab}}. \tag{3.97}$$

Demostración: Al igual que en los resultados anteriores, las hipótesis a considerar son (ha), (hb) y (hc). En este caso, el objetivo a demostrar es la expresión (3.97).

Para llevar a cabo la prueba, se comenzarán demostrando dos hipótesis a partir de las cuales se tendrá de manera inmediata el resultado deseado.

- **Hipótesis h:**

$$\forall x \in \mathbb{R}, x > 0 \longrightarrow ((x^3)^{-1})^3 = x. \tag{h}$$

Para demostrar esta hipótesis, comencemos fijando un número real x, a continuación, a la hipótesis de que este número es positivo la denotaremos como sigue:

$$0 < x. \tag{hx}$$

Además, se tiene de manera trivial las dos siguientes afirmaciones:

$$0 < 3, \tag{h1}$$
$$3 \in \mathbb{N}. \tag{h2}$$

Introduzcamos ahora el siguiente lema:

Lema 3.4.7. Sean x un número real no negativo y n un número natural positivo. Entonces, se verifica la siguiente igualdad:

$$(x^{n^{-1}})^n = x, \tag{3.98}$$

donde en esa expresión n^{-1} es considerado como un número real.

La formalización en Lean de este resultado es la siguiente:

```
lemma rpow_nat_inv_pow_nat {x : ℝ} (hx : 0 ≤ x) {n : ℕ} (hn : 0 < n) :
(x ^ (n⁻¹ : ℝ)) ^ n = x
```

Entonces, se tiene que haciendo uso del lema 3.4.7, teniendo en cuenta las hipótesis (h1) y (h2) y sabiendo que como el x considerado es mayor estrictamente que cero, (hx), entonces éste es mayor o igual que cero; se demuestra (h).

- **Hipótesis aux:**

  En esta nueva hipótesis se hará uso del teorema auxiliar 3.4.4. No obstante, previo a ello usaremos un resultado y es el siguiente:

  Lema 3.4.8. Sean x e y dos números reales tal que x es positivo. Entonces, x^y es positivo. Simbólicamente:

  $$\forall x, y \in \mathbb{R}, x > 0 \longrightarrow x^y > 0. \tag{3.99}$$

  Cuya formalización en Lean es:

```
lemma rpow_pos_of_pos {x : ℝ} (hx : 0 < x) (y : ℝ) : 0 < x ^ y
```

  De esta manera, se tiene que si aplicamos el lema 3.4.8 considerando como x al número real a que por hipótesis es positivo, (ha), y como número y al número real 3^{-1}. De esta forma se tendría que:

  $$a^{3^{-1}} > 0. \tag{3.100}$$

  Si en lugar de considerar como x al número a, consideramos los números b y c, respectivamente se obtiene lo siguiente:

  $$b^{3^{-1}} > 0, \tag{3.101}$$
  $$c^{3^{-1}} > 0. \tag{3.102}$$

  Entonces, teniendo estas tres hipótesis, (3.100) (3.101) y (3.102), se puede aplicar el teorema auxiliar 3.4.4 sobre los números reales $a^{3^{-1}}$, $b^{3^{-1}}$ y $c^{3^{-1}}$. Se obtendría entonces la hipótesis aux:

  $$\begin{aligned} 1 \leq & \frac{(a^{3^{-1}})^3}{\sqrt{((a^{3^{-1}})^3)^2 + 8(b^{3^{-1}})^3(c^{3^{-1}})^3}} \\ & + \frac{(b^{3^{-1}})^3}{\sqrt{((b^{3^{-1}})^3)^2 + 8(c^{3^{-1}})^3(a^{3^{-1}})^3}} \\ & + \frac{c^3}{\sqrt{((c^{3^{-1}})^3)^2 + 8(a^{3^{-1}})^3(b^{3^{-1}})^3}}. \end{aligned} \tag{aux}$$

De forma que, una vez tenemos ya probadas estas dos hipótesis, es bastante sencilla la resolución del problema original.

Como consecuencia de que los números reales en consideración son mayor que cero por hipótesis, (ha), (hb) y (hc), se tiene que podemos aplicar la hipótesis (h) en los diferentes términos que aparecen en la expresión de la hipótesis (aux). De esta forma se obtendría que:

$$1 \leq \frac{a}{\sqrt{a^2+8bc}} + \frac{b}{\sqrt{b^2+8ca}} + \frac{c}{\sqrt{c^2+8ab}}, \tag{3.103}$$

que era el resultado deseado.

□

A continuación, presentemos la formalización en Lean de este resultado:

```
theorem imo2001_q2
  (ha : 0 < a)
  (hb : 0 < b)
  (hc : 0 < c)
  : 1 ≤ a / sqrt (a ^ 2 + 8 * b * c) +
        b / sqrt (b ^ 2 + 8 * c * a) +
        c / sqrt (c ^ 2 + 8 * a * b) :=
begin
  have h : ∀ {x : ℝ}, 0 < x → (x ^ (3 : ℝ)⁻¹) ^ 3 = x,
    { intros x hx,
      have h1 : 0 < 3 :=
        zero_lt_three,
      have h2 : ↑3 = (3 : ℝ) :=
        by norm_num,
      have htesis : (x ^ (↑3)⁻¹) ^ 3 = x :=
        rpow_nat_inv_pow_nat (le_of_lt hx) h1,
      rw h2 at htesis,
      exact htesis,},
  have aux :=
    imo2001_q2_aux (rpow_pos_of_pos ha 3⁻¹)
                   (rpow_pos_of_pos hb 3⁻¹)
                   (rpow_pos_of_pos hc 3⁻¹),
  rw h ha at aux,
  rw h hb at aux,
  rw h hc at aux,
  exact aux,
end
```

Para la formalización en Lean de este teorema han sido necesarios los siguientes lemas y teoremas auxiliares:

- `zero_lt_three : 0 < (3:α)`
- `rpow_nat_inv_pow_nat : ∀ (n : ℕ), 0 ≤ x → 0 < n → (x ^ (n⁻¹: ℝ)) ^ n = x`
- `rpow_pos_of_pos {x : ℝ} (hx : 0 < x) (y : ℝ) : 0 < x ^ y`

Además, se han usado las tácticas exact, have, intros, norm_num y rw.

Capítulo 4

Nuevos problemas de las IMO en Lean

En este capítulo se presentará las soluciones de problemas de las Olimpiadas Internacionales de Mátematicas (IMO) cuyas formalizaciones en Lean no se ha llevado a cabo con anterioridad.

4.1. IMO 2001 Q6

En esta nueva sección, se va a detallar la resolución del problema Q6 propuesto en el año 2001. Como en el resto de planteamientos anteriores que se han detallado en el capítulo previo, se comenzará presentado la demostración en lenguaje natural del problema y luego su correspondiente formalización en Lean.

Problema 1 (2001–Q6). Sean a, b, c y d cuatro números enteros tales que $a > b > c > d > 0$. Supongamos que

$$ac + bd = (a + b - c + d)(-a + b + c + d).$$

Demostrar que $ab + cd$ no es primo.

Para llevar a cabo la formalización en Lean de la resolución de este problema, ha sido necesaria la importación de las siguientes teorías:

- data.int.basic, para poder trabajar con el conjunto de los números enteros.

- algebra.associated, para trabajar con las definición de número primo.
- tactic, para poder acceder a las diferentes tácticas.

La importación en Lean de todas estas teorías se formaliza como sigue:

```
import data.int.basic
import algebra.associated
import tactic
```

Además, en este problema hay que definir las variables a, b, c y d que pertenecen al conjunto de los números enteros. Las diferentes desigualdades que verifican se enunciarán en los lemas en los que sean necesarias dichas condiciones. En Lean esto se formaliza de la siguiente manera:

```
variables {a b c d : ℤ}
```

A continuación, se va a dividir en dos partes esta sección: en la primera parte probaremos cinco resultados auxiliares que han sido necesarios para la formalización del problema y en la segunda parte se formalizará el teorema final que nos da el resultado deseado.

4.1.1. Resultados auxiliares

Como ya se ha dicho anteriormente, en esta sección del problema se van a formalizar cinco resultados auxiliares. Veamos estos lemas con sus correspondientes formalizaciones en Lean.

Destacar que en los lemas auxiliares que se van a presentar a continuación no siempre es necesario usar todas las hipótesis del teorema en sí. Es por eso, que los resultados se han probado usando el menor número de hipótesis posible, haciendo el resultado lo más general posible.

Lema 4.1.1 (sumas_equivalentes)**.** Sean a, b, c y d cuatro números enteros tales que

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Entonces se verifica que

$$b^2 + bd + d^2 = a^2 - ac + c^2. \tag{4.1}$$

Demostración: Para demostrar (4.1) se comenzará desarrollando la segunda parte de la igualdad de la hipótesis (h), obteniendo el siguiente resultado:

$$(a+b-c+d)(-a+b+c+d) = -a^2+b^2+ac-c^2+bd+d^2+ac+bd. \qquad \text{(h1)}$$

A continuación, reescribimos el segundo término de la hipótesis (h) como se ha descrito en (h1). De esta forma se llega a la siguiente igualdad:

$$ac+bd = -a^2+b^2+ac-c^2+bd+d^2+ac+bd. \qquad (4.2)$$

La conclusión de la prueba ya se tiene, de manera trivial, sin más que simplificar la ecuación (4.2) y redistribuir los términos se obtiene (4.1).

□

Este resultado se formaliza en Lean como a continuación se plantea:

```
lemma sumas_equivalentes
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : b^2 + b*d + d^2 = a^2 - a*c + c^2 :=
begin
  have h1: (a + b - c + d) * (-a + b + c + d) =
           -a^2  + b^2  + a*c - c^2 + b*d + d^2 + a*c + b*d,
    by ring,
  rw h1 at h,
  by nlinarith,
end
```

Para levar a cabo la formalización de este lema no hemos hecho uso de ningún lema auxiliar. No obstante, sí se han utilizado las tácticas de have, rw, ring y nlinarith.

Prosigamos con la demostración del problema, para ello el lema que se presenta a continuación nos dará la igualdad de dos productos que será de gran utilidad en la conclusión del problema.

Lema 4.1.2 (productos_equivalentes)**.** Sean a, b, c y d cuatro números enteros verificando que

$$ac+bd = (a+b-c+d)(-a+b+c+d). \qquad \text{(h)}$$

Entonces se verifica que

$$(ac+bd)(b^2+bd+d^2) = (ab+cd)(ad+bc). \qquad (4.3)$$

Demostración: Para llevar a cabo la prueba de este resultado, se van a desarrollar los dos lados de la igualdad que se quiere demostrar, (4.3), y se va a comprobar que son iguales.

Comencemos desarrollando la parte izquierda de (4.3):

$$(ac + bd)(b^2 + bd + d^2) = ac(b^2 + bd + b^2) + bd(b^2 + bd + b^2). \tag{h1}$$

Ahora bien, haciendo uso del lema que hemos demostrado antes, es decir, el lema 4.1.1 sobre el segundo término que aparece sobre el que se puede aplicar, se tiene que

$$ac(b^2 + bd + b^2) + bd(b^2 + bd + b^2) = ac(b^2 + bd + b^2) + bd(a^2 - ac + c^2). \tag{h2}$$

De esta forma, introduciendo el resultado obtenido en (h2) en (h1), se llega a que:

$$(ac + bd)(b^2 + bd + d^2) = ac(b^2 + bd + b^2) + bd(a^2 - ac + c^2). \tag{4.4}$$

Ahora bien, si seguimos desarrollando la segunda parte de la igualdad (4.4), se llega a que:

$$(ac + bd)(b^2 + bd + d^2) = acb^2 + acd^2 + a^2bd + bcd^2. \tag{4.5}$$

Por otro lado, se tiene que reescribiendo la parte derecha de la ecuación que queremos demostrar,es decir, de (4.3), se llega a:

$$(ab + cd)(ad + bc) = acb^2 + acd^2 + a^2bd + bcd^2. \tag{4.6}$$

Se puede comprobar a través de las expresiones (4.5) y (4.6) que llegamos a lo mismo al desarrollar los dos términos de la expresión (4.3). Por tanto, ya se tendría la prueba del lema.

□

En este caso, la formalización en Lean es:

```
lemma productos_equivalentes
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : (a*c + b*d) * (b^2 + b*d + d^2) = (a*b + c*d) * (a*d + b*c) :=
begin
  have h1: (a*c + b*d) * (b^2 + b*d + d^2) =
           a*c * (b^2 + b*d + d^2) + b*d * (b^2 + b*d + d^2),
    by ring,
```

```
  have h2: a*c * (b^2 + b*d + d^2) + b*d * (b^2 + b*d + d^2) =
           a*c * (b^2 + b*d + d^2) + b*d * (a^2 - a*c + c^2),
    by rw sumas_equivalentes h,
  rw h2 at h1,
  by nlinarith,
end
```

A continuación, se van a probar dos lemas muy parecidos (puesto que ambos nos afirman la veracidad de una desigualdad), que a su vez son muy sencillos de formalizar en Lean bajo las hipótesis del problema.

Lema 4.1.3 (desigualdad_auxiliar1)**.** Sean a, b, c y d cuatro números enteros tales que $a > b > c > d$. Además, supongamos que estos números enteros verifican que

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Entonces se tiene que

$$ac + bd < ab + cd \tag{4.7}$$

Demostración: En primer lugar, denotemos las hipótesis del enunciado como sigue:

$$b < a, \tag{hba}$$
$$c < b, \tag{hcb}$$
$$d < c. \tag{hdc}$$

Por otro lado, se tiene que demostrar (4.7) es equivalente a demostrar la siguiente desigualdad:

$$0 < ab + cd - (ac + bd). \tag{4.8}$$

Sin embargo, desarrollando la resta de (4.8), se obtiene que el objetivo a probar se convierte en:

$$0 < (a - d)(b - c). \tag{4.9}$$

No obstante, este resultado es inmediato a partir de las desigualdades que tenemos de hipótesis, es decir, (hba), (hcb) y (hdc).

□

Formalizar este lema en Lean es bastante sencillo y directo ya que existe una táctica que a través de las desigualdades que se tienen por hipótesis, deduce el resultado. La formalización sería:

```
lemma desigualdad_auxiliar1
  (hba : b < a)
  (hcb : c < b)
  (hdc : d < c)
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : a*c + b*d < a*b + c*d:=
by nlinarith
```

En la demostración de este lema sólo ha sido necesario el uso de la táctica nlinarith.

Destacar que la la táctica nlinarith automatiza todos los pasos que se han realizado en la demostración en lenguaje natural.

Lema 4.1.4 (desigualdad_auxiliar2)**.** Sean a, b, c y d cuatro números enteros tales que a es mayor que b y c mayor que d. Es decir,

$$b < a, \tag{hba}$$
$$d < c. \tag{hdc}$$

Además, los números enteros en consideración verifican la siguiente relación:

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Entonces se tiene que

$$ad + bc < ac + bd \tag{4.10}$$

Demostración: En primer lugar, se tiene que demostrar (4.10) es equivalente a probar la siguiente expresión:

$$0 < ac + bd - (ad + bc), \tag{4.11}$$

donde simplemente se ha pasado el término de la izquierda hacia la parte derecha de la desigualdad.

Desarrollando la resta que se puede observar en (4.11), se obtiene lo siguiente:

$$0 < (a - b)(c - d). \tag{4.12}$$

No obstante, este resultado se tiene de manera inmediate a partir de las hipótesis (hba) y (hdc).

□

Análogamente al lema anterior, 4.1.3, la formalización en Lean de este lema sería inmediata:

```
lemma desigualdad_auxiliar2
  (hba : b < a)
  (hdc : d < c)
  (h : a*c + b*d = (a + b - c + d)*(-a + b + c + d))
  : a*d + b*c < a*c + b*d:=
by nlinarith
```

Al igual que en el lema 4.1.3, sólo ha sido necesario el uso de la táctica nlinarith para su formalización. Al igual que antes, esta táctica prueba de manera automática el resultado.

Finalmente, introduzcamos el último lema auxiliar antes de llevar a cabo la formalización del problema como tal.

Lema 4.1.5 (division)**.** Sean a, b, c y d cuatro números enteros tales que verifican la siguiente relación:

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Entonces se tiene que el número entero $ac + bd$ divide al producto de los números enteros $ab + cd$ y $ad + bd$. Simbólicamente:

$$ac + bd \mid (ab + cd)(ad + bc). \tag{4.13}$$

Demostración: Previo a la demostración como tal del resultado, recordemos qué significa que un número entero divida a otro.

Definición 4.1.6. Sean a y b dos números enteros. Se dirá que a **divide a** b si y solamente si existe otro número entero k tal que $b = a \cdot k$. Simbólicamente,

$$a \mid b \iff \exists k \in \mathbb{Z}, b = a \cdot k. \tag{4.14}$$

Por tanto, se tiene que haciendo uso de la definición 4.1.6, la prueba del lema consiste en probar que existe un número entero k verificando que

$$(ab + cd)(ad + bc) = (ac + bd)k. \tag{4.15}$$

Usando como número entero k el que viene dado por la suma $b^2 + bd + d^2$, se tendría que para obtener el resultado que se quería demostrar, bastaría

con hacer uso del Lema 4.1.2.

□

La formalización en Lean de este resultado sería:

```
lemma division
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : a*c + b*d | (a*b+c*d) * (a*d+b*c):=
begin
  use (b^2 + b*d +d^2),
  by nlinarith [productos_equivalentes h],
end
```

Para la formalización de este último lema auxiliar sólo han sido necesario el uso de dos tácticas: nlinarith y use.

4.1.2. Conclusión del problema

Concluiremos el problema en esta sección, para ello se hará uso de los lemas auxiliares probados anteriormente. Para resolver el problema que nos interesa, se enunciará el siguiente teorema:

Teorema 4.1.7 (imo2001q6)**.** Sean a, b, c y d cuatro números enteros tales que $a > b > c > d > 0$. Además, supongamos que estos números enteros verifican que

$$ac + bd = (a + b - c + d)(-a + b + c + d). \tag{h}$$

Entonces se tiene que $ab + cd$ no es un número primo.

Demostración: En primer lugar, a parte de la hipótesis (h) enunciada, denotemos el resto de hipótesis del teorema como sigue:

$$0 < d, \tag{hd}$$
$$d < c, \tag{hdc}$$
$$c < b, \tag{hcb}$$
$$b < a. \tag{hba}$$

A continuación, la prueba del resultado se va a hacer mediante la **introducción de la negación**. Esto es, como el objetivo es demostrar que el número $ab+cd$ no es un número primo, supondremos que sí lo es y buscaremos llegar a una contradicción.

Comencemos introduciendo la hipótesis de que $ab + cd$ es un número primo:

$$\text{primo}(ab + dc) \tag{h1}$$

Encadenando las desigualdades que tenemos como hipótesis, es decir, (hd), (hdc), (hcb) y (hba), se puede llegar fácilmente a los siguientes tres resultados:

$$0 < a, \tag{ha}$$
$$0 < b, \tag{hb}$$
$$0 < c. \tag{hc}$$

Ahora bien, haciendo uso del lema 4.1.2 (lo cual es posible pues tenemos como hipótesis (h)), se obtiene la siguiente expresión:

$$(ac + bd)(b^2 + bd + d^2) = (ab + cd)(ad + bc). \tag{h2}$$

Asimismo, haciendo uso del lema 4.1.5, se llega a que

$$(ac + bd) | (ab + cd)(ad + bc). \tag{h3}$$

Para proseguir con la prueba del teorema es necesario introducir un resultado auxiliar y que es el siguiente:

Lema 4.1.8. Sean a, b y p tres números cualquiera, verificando que p es un número primo y que a divide al producto de p y b. Entonces se tiene que o bien p divide a a ó a divide a b. Simbólicamente:

$$\forall a, b, p,\ \text{primo}(p), a \mid pb \longrightarrow p \mid a \vee a \mid b \tag{4.16}$$

Cuya formalización en Lean es la siguiente:

```
lemma left_dvd_or_dvd_right_of_dvd_prime_mul
[comm_cancel_monoid_with_zero α] {a : α} :
∀ {b p : α}, prime p → a ∣ p * b → p ∣ a ∨ a ∣ b
```

Entonces, como nosotros tenemos las hipótesis (h1) y (h3), podemos hacer uso del lema 4.1.8 y se tendría que ha de verificarse la siguiente disyunción:

$$(ab + cd \mid ac + db) \vee (ac + bd \mid ad + bc). \tag{h4}$$

Con el objetivo de llegar a la contradicción que buscamos, vamos a separar la disyunción (h4) y consideraremos los dos casos posibles. Se va a comprobar que en ambos casos llegamos a una contradicción.

- **Primer caso**

  Supongamos que se verifica la primera parte de la disyunción, es decir, que tenemos que

  $$ab + cd \mid ac + db. \tag{hj}$$

  En primer lugar, haciendo uso del lema 4.1.3, probado en la sección anterior, se tiene que

  $$ab + cd > ac + bd. \tag{hj1}$$

  Además, haciendo uso de que la multiplicación de numeros positivos es positiva y la suma también, se tiene el siguiente resultado:

  $$0 < ac + bd. \tag{hpj}$$

  A continuación, introduzcamos el siguiente teorema:

  Teorema 4.1.9. Sean a y b dos números enteros tales que b es estrictamente mayor que cero y a divide a b. Entonces, se tiene que a es un número menor o igual que b.

  Cuya formalización en Lean es la siguiente

```
theorem le_of_dvd {a b : ℤ}
(bpos : 0 < b)
(H : a ∣ b) :
a ≤ b
```

  Entonces, se tiene que haciendo uso del teorema 4.1.9 sobre los números enteros $ab + cd$, $ac + bd$ y las hipótesis (hpj) y (hj1), se llega a que

  $$ab + cd \leq ac + bd. \tag{hj2}$$

  Ahora bien, se sabe que la hipótesis (hj1) es equivalente a esta otra:

  $$\neg(ab + cd \leq ac + bd). \tag{hj3}$$

  De esta forma, si vemos las hipótesis (hj2) y (hj3), se puede observar que hemos llegado a una contradicción, que era lo deseado.

- **Segundo caso**

  Supongamos ahora que se verifica la segunda parte de la disyunción, es decir, que tenemos que

  $$ac + bd \mid ad + bc \tag{hk}$$

Se va a proceder de manera totalmente análoga a como se ha hecho en el caso anterior.

Comenzamos haciendo uso del lema auxliar 4.1.4, a partir del cual se obtiene que

$$ac + bd > ad + bc. \tag{hk1}$$

Asimismo, como consecuencia de que el producto de números positivos es positivo y la suma de los mismos también es positiva, se puede afirmar que se verifica el siguiente resultado:

$$0 < ad + bc. \tag{4.17}$$

Por otro lado, se tiene que haciendo uso del teorema 4.1.9, visto anteriormente, se llega a que

$$ac + bd \leq ad + bc. \tag{hk2}$$

Ahora bien, se tiene que la hipótesis (hk1) puede ser reescrita de la siguiente manera:

$$\neg(ac + bd \leq ad + bc). \tag{hk3}$$

Entonces, observando las hipótesis (hk2) y (hk3) se llega de nuevo a contradicción.

De esta forma, al haber llegado a contradicción en los dos casos que se diferenciaban en la disyunción (h4), se tendría que la hipótesis que se llevó a cabo al principio de que el número $ab + cd$ era primo es falsa, concluyendo así la prueba.

□

La formalización en Lean de este teorema sería:

```
theorem imo2001q6
  (hd  : 0 < d)
  (hdc : d < c)
  (hcb : c < b)
  (hba : b < a)
  (h : a*c + b*d = (a + b - c + d)*(-a + b + c + d))
  : ¬ prime (a*b + c*d) :=
begin
  intro h1,
  have ha : 0 < a,
    by linarith,
```

```
  have hb : 0 < b,
    by linarith,
  have hc : 0 < c,
    by linarith,
  have h2 : (a*c + b*d) * (b^2 + b*d + d^2) =
            (a*b + c*d) * (a*d + b*c),
    from productos_equivalentes h,
  have h3 : a*c + b*d ∣ (a*b + c*d) * (a*d + b*c),
    from division h,
  have h4: (a*b + c*d ∣ a*c + b*d) ∨ (a*c + b*d  ∣ a*d + b*c),
    from left_dvd_or_dvd_right_of_dvd_prime_mul h1 h3,
  cases h4 with hj hk,
  { have hj1: a*b + c*d > a*c + b*d,
      from desigualdad_auxiliar1 hba hcb hdc h,
    have hpj: 0 < a*c + b*d,
      from add_pos (mul_pos ha hc) (mul_pos hb hd),
    have hj2: a*b + c*d ≤ a*c + b*d,
      from int.le_of_dvd hpj hj,
    have hj3: ¬ (a*b + c*d ≤ a*c + b*d),
      from not_le.mpr hj1,
    exact hj3 hj2, },
  { have hk1: a*c + b*d > a*d + b*c,
      from desigualdad_auxiliar2 hba hdc h,
    have hpk: 0 < a*d + b*c,
      from add_pos (mul_pos ha hd) (mul_pos hb hc),
    have hk2: a*c + b*d ≤ a*d + b*c,
      from int.le_of_dvd hpk hk,
    have hk3: ¬ (a*c+b*d ≤  a*d+b*c),
      from not_le.mpr hk1,
    exact hk3 hk2, },
end
```

Para la formalización de este teorema en Lean han sido necesarios los siguiente lemas y teoremas auxiliares:

- `left_dvd_or_dvd_right_of_dvd_prime_mul: prime p → a ∣ p*b → p∣a ∨ a∣b`
- `add_pos : 0 < a → 0 < b → 0 < a + b`
- `mul_pos : 0 < a → 0 < b → 0 < a * b`
- `le_of_dvd : 0 < b → a ∣ b → a ≤ b`

- `not_le : ¬ a ≤ b ↔ b < a`

Además, se han usado las tácticas exact, have, intro, cases y linarith.

Apéndice A

Tácticas en Lean

En este apéndice se van a explicar de manera detallada las diferentes tácticas usadas durante el desarrollo del trabajo. Con el objetivo de clarificar las ideas, se plantearán diferentes ejemplos para ver cómo actuan las diversas tácticas.

Muchos de los ejemplos que he usado para la clarificación de las diferentes tácticas han sido obtenidos del tutorial que estudié para comenzar a trabajar con Lean, en concreto [6].

A.1. Táctica sorry

La primera táctica que vamos a estudiar es la táctica sorry. En particular, esta táctica es muy útil y tiene una función muy sencilla: es capaz de aceptar cualquier resultado. Ahora bien, a nosotros no nos interesa usar esta táctica para la formalización como tal de las pruebas pero sí como herramienta de ayuda que nos permita formalizar resultados auxiliares o hechos dentro de una prueba.

Realmente cualquier ejemplo, lema o teorema sería válido para ver cómo funciona esta táctica. Veamos el siguiente:

```
example (h₁ : a ∣ b) (h₂ : b ∣ c) : a ∣ c :=
begin
  sorry,
end
```

Asimismo, veamos un ejemplo en el que la táctica sorry sea utilizada con la función de probar un hecho dentro de la prueba. Es más, en este ejemplo aparecen las dos funciones mencionadas de la táctica:

```
example (a b : ℝ) : a = a*b → a = 0 ∨ b = 1 :=
begin
  have H : a*(1 - b) = 0, by sorry,
  sorry,
end
```

A.2. Táctica rewrite

A continuación, se presenta la táctica rw. Esta táctica es muy usada y su forma de funcionar es muy intuitiva: consiste en reemplazar la ecuación o el si y solamente si que se encuentre después de la táctica rw sobre el objetivo a demotrar.

Si a continuación de la táctica rw se encuentra una flecha hacia la izquierda (←) se tiene que la sustitución es aplicada al revés.

En el ejemplo que se presenta a continuación se pueden observar los dos usos de la táctica rw que hemos mencionado:

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

Asimismo, otra función de la táctica rw de la cual se ha hecho uso durante el desarrollo del trabajo es aplicar la sustitución en alguna hipótesis y no en el objetivo a demostrar. Esto se hace haciendo uso del predicado `at`; veámoslo en Lean:

```
example (a b c d : ℝ) (hyp : c = d*a + b) (hyp' : b = a*d) : c = 2*a*d :=
begin
  rw hyp' at hyp,
  rw mul_comm d a at hyp,
  rw ← two_mul (a*d) at hyp,
  rw ← mul_assoc 2 a d at hyp,
  exact hyp,
end
```

En este ejemplo se pueden observar las tres funciones descritas de la táctica rw.

A.3. Táctica have

La siguiente táctica a estudiar será la táctica have. Esta táctica es usada cuando se quieren introducir nuevos lemas al problema que tendrán que ser demostrados luego.

En este caso el mismo ejemplo que uno de los que usamos para estudiar la táctica sorry. Este ejemplo era:

```
example (a b : ℝ) : a = a*b → a = 0 ∨ b = 1 :=
begin
  intro hyp,
  have H : a*(1 - b) = 0, by sorry,
  sorry,
end
```

A.4. Táctica exact

La táctica exact ha sido una de las más utilizadas en el trabajo y cuya función es muy simple: nos introduce una prueba directa del objetivo a demostrar.

Para ver un ejemplo del uso de esta táctica, también se va a hacer uso de uno ya propuesto anteriormente y es uno de los que se estudió al ver la táctica rw. El ejemplo era el siguiente:

```
example (a b c d : ℝ) (hyp : c = d*a + b) (hyp' : b = a*d) : c = 2*a*d :=
begin
  rw hyp' at hyp,
  rw mul_comm d a at hyp,
  rw ← two_mul (a*d) at hyp,
  rw ← mul_assoc 2 a d at hyp,
  exact hyp,
end
```

A.5. Táctica intro

La siguiente táctica que se va a introducir es intro, esta táctica puede ser utilizada de diversas maneras. Una manera muy común de usarla es cuando tenemos que demostrar una implicación, entonces se supone como cierta la primera parte de la implicación y esto se hace a través de la táctica intro.

En el siguiente ejemplo se ve muy claro:

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

En el siguiente ejemplo que se plantea, se hace uso intros, en este caso como se quiere demostrar que Q implica no P, al decirle intros q p, le estamos diciendo que introduzca las hipótesis de que se verifica Q y también P. De esta manera, el objetivo a demostrar pasa a ser false.

```
example (P Q : Prop) (h₁ : P ∨ Q) (h₂ : ¬ (P ∧ Q)) : Q → ¬P :=
begin
  intros q p,
  exact h₂ ⟨p,q⟩ ,
end
```

A.6. Táctica apply

La táctica apply es la siguiente que vamos a introducir. La función de esta táctica trata de unificar el objetivo a demostrar con la conclusión de un resultado auxiliar (el que se especifique justo después de aplicar la táctica). De manera que si unifica, los objetivos a demostrar pasan a ser las diversas premisas que tuviese el resultado usado.

Veamos un ejemplo en el que se utiliza esta táctica:

```
def non_decreasing (f : ℝ → ℝ) := ∀ x₁ x₂, x₁ ≤ x₂ → f x₁ ≤ f x₂
def non_increasing (f : ℝ → ℝ) := ∀ x₁ x₂, x₁ ≤ x₂ → f x₁ ≥ f x₂

example (f g : ℝ → ℝ) (hf : non_decreasing f) (hg : non_increasing g) :
non_increasing (g ∘ f) :=
begin
  intros x1 x2 h,
  apply hg,
  apply hf,
  exact h,
end
```

A.7. Táctica linarith

La siguiente táctica ha sido otra muy utilizada en el trabajo y es linarith. Esta táctica es capaz de probar multitud de igualdades y desigualdades de manera directa; realmente, es capaz de probar casi cualquier problema lineal. En el ejemplo que se plantea a continuación se puede ver de manera muy clara:

```
example (a b : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) : 0 ≤ a + b :=
begin
  linarith,
end
```

Se puede observar que a través de las dos hipótesis que se plantean, ha y hb, la táctica linarith es capaz de probar directamente el resultado.

A.8. Táctica nlinarith

La táctica nlinarith, que es la que vamos a estudiar a continuaón, es muy parecida a la táctica linarith que acabamos de describir. Realmente la táctica nlinarith no es más que una extensión de la táctica linarith que puede resolver problemas no lineales.

Para ver un ejemplo sobre esta táctica, se ha usado uno que se corresponde con un lema auxiliar del problema Q6 de 2001 de las Olimpiadas Internacionales de Mátemáticas. El ejemplo es el que se plantea a continuación:

```
example (a b c d : ℤ)
  (hba : b < a)
  (hcb : c < b)
  (hdc : d < c)
  (h : a*c + b*d = (a + b - c + d) * (-a + b + c + d))
  : a*c + b*d < a*b + c*d:=
begin
  nlinarith,
end
```

A.9. Táctica assume

La siguiente táctica que se estudia es la táctica assume. Como su propio nombre indica nos sirve para asumir o fijar una variable o incluso una hipótesis.

Cuando nos encontramos ante un resultado en el que hay que demostrar un para todo, esta táctica es muy útil. Veamos el siguiente ejemplo:

```
variables{n : ℕ}
example : ∀ n, 3*n=n*3 :=
begin
  assume n,
  sorry,
end
```

Se puede observar que el objetivo a demostrar pasa a ser sólo y excluivamente probar la igualdad puesto que hemos fijado el número n perteneciente al conjunto de los números naturales.

A.10. Táctica by contradiction

La táctica by_contradiction es muy usada en las pruebas matemáticas. Esta táctica consiste en suponer que el objetivo a demostrar no se verifica y que se acabe llegando a una contradicción.

Un ejemplo en el que el uso de esta táctica se ver muy claro es el siguiente:

```
example (P Q : Prop) (h : ¬ Q → ¬ P) : P → Q :=
begin
  intro hP,
  by_contradiction hnQ,
  exact h hnQ hP,
end
```

En la primera línea, se introduce como hipótesis hP que se tiene la proposición P y luego se denota como hipótesis hnQ que no se tiene Q. En la última línea es cuando ya se llega a la contradicción.

A.11. Táctica let

La táctica let que ha sido usada en alguna formalización que se ha realizado en Lean consiste en introducir una nueva hipótesis (a la que se puede nombrar si así se desea).

En el ejemplo que se tiene a continuación, se puede ver que se ha introducido una nueva hipótesis a través de la táctica let:

```
variables (u v w : ℕ → ℝ) (l l' : ℝ)
notation `|`x`|` := abs x
def seq_limit (u : ℕ → ℝ) (l : ℝ) : Prop :=
∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - l| ≤ ε

def tendsto_infinity (u : ℕ → ℝ) := ∀ A, ∃ N, ∀ n ≥ N, u n ≥ A

example {u : ℕ → ℝ} : tendsto_infinity u → ∀ l, ¬ seq_limit u l :=
begin
  intro h,
  intro l,
  intro lim,
```

```
  cases lim 1 (by linarith) with N1 hN1,
  cases h (l+2) with N2 hN2,
  let N3 := max N1 N2,
  sorry,
end
```

A.12. Táctica use

La táctica use irá acompañado de una expresión. Esta táctica será utilizada cuando nos encontremos que hay que probar que existe un elemento tal que verifica una condición; entonces, cuando le queremos decir que considere uno en concreto se lo decimos a Lean a través de la táctica use.

En el siguiente ejemplo se puede ver de manera inmediata:

```
example : ∃ n : ℕ, 8 = 2*n :=
begin
  use 4,
  refl,
end
```

A.13. Táctica induction

La táctica induction es uno de los mecanismos más famosos para demostrar resultados en Matemáticas. Simplemente consiste en demostrar un resultado por inducción, es decir, si hacemos inducción el número b, se tiene que demostrar el resultado para b igual a 0 y luego, suponiendo que se verifica para $k = b$ hay que demostrar el mismo resultado para $k + 1$.

En el ejemplo que se plantea a continuación, se ve cómo se usa esta táctica: al decir induction b with k h, se está diciendo que hacemos la inducción sobre el número b denotando los casos por $k = 0$ y $k + 1$:

```
example (t a b : ℕ) : t * (a + b) = t * a + t * b :=
begin
  induction b with k h,
  {sorry,},
  {sorry,},
end
```

A.14. Táctica cases

La táctica cases es también una táctica muy usada en las formalizaciones de Lean. Existen diversas formas de usar esta táctica, no obstante, nos centraremos en las que se han usado en el trabajo y que consisten en:

1. Cuando se tiene una hipótesis que es una conjunción, se usa la táctica cases para separar sus componentes. Es decir, para la regla de eliminación de la conjunción.
2. Cuando se tiene una hipótesis que afirma que existe un número tal que verifica una determinada condición, se usa la táctica cases para por un lado tener el número en cuestión y por otro la condición que él verifica. Es decir, para la regla de eliminación del existencial.
3. Cuando se tiene una hipótesis que es una disyunción, se usa la táctica cases para dividir el problema en dos: probar el objetivo suponiendo la primera parte de la disyunción y probarlo suponiendo que se verifica la segunda parte. Es decir, para la regla de eliminación de la disyunción.

En el siguiente ejemplo que se presenta, se puede ver cómo funciona la táctica cases cuando se usa para la regla de eliminación de la conjunción:

```
example {a b : ℝ} : (0 ≤ a ∧ 0 ≤ b) → 0 ≤ a + b :=
begin
  intros hyp,
  cases hyp with ha hb,
  exact add_nonneg ha hb,
end
```

En el segundo ejemplo, se puede observar la táctica cases usada para la regla de eliminación del existencial:

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

En este último ejemplo, se presenta la táctica cases para la regla de eliminación de la disyunción:

```
example (P Q : Prop): P ∨ Q → Q ∨ P :=
begin
  intro hpq,
  cases hpq with hp hq,
  {sorry,},
  {sorry,},
end
```

A.15. Táctica split

El caso de la táctica split es bastante intuitivo: esta táctica divide cuando hay que demostrar un si y solamente si en las dos implicaciones como objetivos separados o también separa en dos objetivos cuando hay que demostrar una conjunción. Es decir, incluye las reglas de introducción del bicondicional y de la conjunción.

En este primer ejemplo, se puede observar como separa el si y solamente si en dos objetivos separados:

```
example (P Q R : Prop) : (P ∧ Q → R) ↔ (P → (Q → R)) :=
begin
  split,
  { intros h1 h2 h3,
    exact h1 ⟨ h2,h3⟩ },
  { intros h1 h2,
    cases h2 with p q,
    exact h1 p q}
end
```

Mientras que un ejemplo muy simple en el que la táctica split es usada para dividir una conjunción es el siguiente:

```
example (P Q : Prop) (hp: P) (hq : Q) : P ∧ Q :=
begin
  split,
  { exact hp,},
  { exact hq,},
end
```

A.16. Táctica from

La táctica que se va a presentar es la conocida como from, esta táctica es muy similar a la táctica exact; pero a diferencia de esta última, la táctica from se puede usar con otras tácticas como have.

Veamos un ejemplo del uso de la táctica from para probar una hipótesis que ha sido introducida con la táctica have:

```
def up_bounds (A : set ℝ) := { x : ℝ | ∀ a ∈ A, a ≤ x}
def is_max (a : ℝ) (A : set ℝ) := a ∈ A ∧ a ∈ up_bounds A
infix ` is_a_max_of `:55 := is_max

example
  (A : set ℝ)
  (x y : ℝ)
  (hx : x is_a_max_of A)
  (hy : y is_a_max_of A)
  : x = y :=
begin
  have : x ≤ y, from hy.2 x hx.1,
  have : y ≤ x, from hx.2 y hy.1,
  linarith,
end
```

A.17. Tácticas left y right

La táctica left se usa cuando se quiere quedar con la primera parte del objetivo a demostrar que está formada por dos constructores. De manera totalmente análoga, la táctica right se queda con la segunda parte del objetivo cuando está formada por dos constructores. Esta táctica incluye las reglas de introducción de la disyunción.

Continuando la formalización de uno de los ejemplos que se ha presentado en la explicación de la táctica cases (en concreto el usado para ilustrar la regla de eliminación de la disyunción), se puede ver muy bien el uso de las tácticas left y right:

```
example (P Q : Prop): P ∨ Q → Q ∨ P :=
begin
  intro hpq,
```

```
  cases hpq with hp hq,
  { right,
    exact hp,},
  { left,
    exact hq,},
end
```

A.18. Táctica library search

La táctica library_search es una táctica que para llevar a cabo formalizaciones de pruebas en Lean es muy útil. Esta táctica se suele utilizar cuando estamos cerca de concluir la demostración y no sabemos qué lema auxiliar utilizar o si existe alguno o no, library_search nos da esta información. En el caso de que haya un lema mediante el cual se pueda probar el resultado o una prueba medio directa, esta táctica nos la proporciona; mientras que en el caso de que no sea capaz de encontrarlo pues no nos proporciona ninguna respuesta.

Por ejemplo, en uno de los ejemplos propuestos analizar la táctica intro, se puede ver muy bien el uso de esta táctica:

```
example (a b : ℝ): 0 ≤ a → b ≤ a + b :=
begin
  library_search,
end
```

Si hacemos caso de la propuesta de formalización que nos propone nos quedaría:

```
example (a b : ℝ): 0 ≤ a → b ≤ a + b :=
begin
  exact le_add_of_nonneg_left,
end
```

Se puede observar que es una formalización diferente a la propuesta en el ejemplo de la táctica intro e incluso me atrevería a decir que más elegante.

A.19. Táctica norm num

La táctica norm_num es capaz de probar de manera directa multitud de igualdades y desigualdades.

Por ejemplo, la siguiente igualdad la prueba en un solo paso:

```
example : (2 : ℝ) + 2 = 4 :=
begin
  norm_num,
end
```

En el ejemplo que se presena ahora, la táctica norm_num es capaz de probar la siguiente desigualdad:

```
example : (73 : ℝ) < 789/2 :=
begin
  norm_num,
end
```

A.20. Táctica refine

La táctica refine funciona de manera totalmente análoga a como lo hace la táctica exact; no obstante, en esta nueva táctica se puede escribir _ como huecos que pueden ser rellenados. Tras el uso de esta táctica, se pasará a tener tantos objetivos como huecos se hayan puesto.

Veamos un ejemplo de esto:

```
example (p : ℕ) (h: p.prime) : 1 | p :=
begin
  refine is_unit.dvd _,
  exact is_unit_one,
end
```

A.21. Táctica ring

La táctica ring en Lean es una táctica muy útil puesto que es capaz de resolver las ecuaciones cuando se está trabajando con anillos (semi) conmutativos.

Por ejemplo, cuando nos encontramos ante situaciones triviales, se pueden demostrar directamente mediante el uso de esta táctica. El siguiente ejemplo es muy representativo:

```
example (a b : ℝ) : (a + b) + a = 2*a + b :=
begin
  by ring,
end
```

A.22. Táctica simp

La táctica simp lo que hace es simplificar el objetivo a probar haciendo uso de lemas ya definidos en Lean. En el ejemplo planteado se ve muy claro:

```
example (a b : ℝ) (h: b=0): a + b = a :=
begin
  simp,
  exact h,
end
```

Mediante el uso de la táctica simp se ha simplificado el objetivo a demostrar mediante el uso del lema auxiliar de cancelación. De esta manera, el objetivo a demostrar pasa a ser que el número b es cero.

A.23. Táctica simpa

La táctica simpa es muy parecida a la táctica simp; no obstante, la que se introduce ahora se trata de una táctica de conclusión de los resultados.

El mismo ejemplo que se ha propuesto para la táctica simp es válido para la táctica simpa y es más, con esta última se concluye el resultado de manera

directa. Veámoslo:

```
example (a b : ℝ) (h : b=0): a + b = a :=
begin
  simpa,
end
```

A.24. Táctica suggest

En general, cuando la táctica library_search no es capaz de solucionarnos el problema y no sabemos cómo continuar con la formalización en Lean del resultado, se suele usar la táctica suggest. Cuando se usa esta táctica en la formalización de un resultado, se nos proporciona una lista de posibles vías de continuar la formalización. En general, la mayoría de las posibilidades propuestas hacen uso de la táctica refine ya descrita.

Veamos el uso de esta táctica con un ejemplo:

```
example (a b : ℝ) : a + b ≤ a + b +1 :=
begin
  suggest,
  sorry,
end
```

Destacar que la táctica suggest no es capaz de probar el resultado, de ahí que se añada la táctica sorry. Algunas de las propuestas que se nos hace en el ejemplo propuesto son las siguientes:

```
Try this: exact (a + b).le_succ
Try this: exact nat.le.intro rfl
Try this: refine eq.ge _
Try this: refine eq.le _
Try this: refine ge.le _
Try this: refine not_lt.mp _
Try this: refine le_of_eq _
Try this: refine ge_of_eq _
Try this: refine le_of_lt _
Try this: refine ge_iff_le.mp _
Try this: refine ge_iff_le.mpr _
Try this: refine le_add_left _
Try this: refine sup_eq_left.mp _
...
```

Es más, usando cualquiera de las dos primeras propuestas se tendría demostrado el problema. Aquí planteamos las dos:

```
example (a b : ℕ) : a + b ≤ a + b +1 :=
begin
  exact (a + b).le_succ,
end

example (a b : ℕ) : a + b ≤ a + b +1 :=
begin
  exact nat.le.intro rfl,
end
```

A.25. Táctica specialize

La táctica specialize actúa sobre una hipótesis determinada y su función consiste en concretar los términos que en la hipótesis son generales o universales por los que el usuario imponga. Esta táctica se corresponde con la regla de eliminación del cuantificador universal.

Para ver mejor cómo funciona, planteemos un ejemplo:

```
lemma unique_max
  (A : set ℝ)
  (x y : ℝ)
  (hx : x is_a_max_of A)
  (hy : y is_a_max_of A) :
  x = y :=
begin
  cases hx with x_in x_up,
  cases hy with y_in y_up,
  specialize x_up y,
  specialize x_up y_in,
  specialize y_up x x_in,
  linarith,
end
```

Se puede observar que en el primer uso de la táctica specialize, se ha particularizado la hipótesis x_up para el número real y considerado.

De manera totalmente análoga se tiene en los dos siguientes usos de la táctica specialize.

A.26. Táctica push neg

La táctica push_neg puede ser aplicada sobre una hipótesis o sobre el propio objetivo a demostrar. Básicamente, lo que hace esta táctica es que cuando se tiene la negación de un cierto hecho, lo convierte en un hecho interiorizando la negación pero con el mismo significado y manteniendo el nombre de las variables.

```
example {x y : ℝ} : (∀ ε > 0, y ≤ x + ε) →  y ≤ x :=
begin
  intro h,
  by_contradiction H,
  push_neg at H,
  sorry,
end
```

Se puede observar que la hipótesis H es la negación de que y sea menor o igual que x. Tras la aplicación de la táctica push_neg, esta hipótesis se convierte en considerar que x es menor estrictamente que y.

A.27. Táctica congr'

La última táctica que se va a detallar en este apéndice es la táctica congr', la cual sólo ha sido utilizada una vez en el desarrollo del trabajo. La táctica congr' es una táctica muy parecida a congr pero menos agresiva que esta última.

Estas tácticas intentan demostrar el objetivo cuando se trata de una igualdad devolviendo unos determinados subobjetivos a probar. La táctica congr' hace uso de un argumento opcional que es el de la recursividad de las funciones.

Veamos un ejemplo de la táctica congr':

```
variables (f g: ℝ → ℝ)
example {x y : ℝ} : f (g (x + y)) = f (g (y + x)):=
begin
  congr',
  {sorry,},
  {sorry,},
end
```

Se puede observar que tras aplicar la táctica congr' el objetivo a probar pasa a dividirse en dos: por un lado, probar que el número x es igual a y y por otro que y es igual a x.

A.28. Otras tácticas

En este apéndice se han detallado todas las tácticas que se han utilizado en el desarrollo del trabajo; no obstante, existen muchas más tácticas que pueden ser utilizadas en las formalizaciones en Lean. En [7] es un buen lugar para ver y estudiar las tácticas que se pueden usar en Lean.

Bibliografía

[1] Jeremy Avigad, Kevin Buzzard, Robert Y. Lewis, and Patrick Massot. Mathematics in Lean. Technical report, Lean community, 2020. En https://leanprover-community.github.io/mathematics_in_lean/.

[2] Jeremy Avigad, Robert Y. Lewis, and Floris van Doorn. Logic and proof. Technical report, Lean community, 2020. En https://leanprover.github.io/logic_and_proof.

[3] Davide Castelvecchi. Mathematicians welcome computer-assisted proof in grand unification theory. Technical report, Nature, 2021. En https://www.nature.com/articles/d41586-021-01627-2.

[4] Tian Chen. IMO 1977 Q6. Technical report, Lean community, 2021. En https://github.com/leanprover-community/mathlib/blob/master/archive/imo/imo1977_q6.lean.

[5] Tian Chen. IMO 2001 Q2. Technical report, Lean community, 2021. En https://github.com/leanprover-community/mathlib/blob/master/archive/imo/imo2001_q2.lean.

[6] Lean community. Lean tutorials. Technical report, Lean community, 2019. En https://github.com/leanprover-community/tutorials.

[7] Lean community. Mathlib tactics. Technical report, Lean community, 2019. En https://leanprover-community.github.io/mathlib_docs/tactics.html.

[8] Kevin Lacker. IMO 1959 Q1. Technical report, Lean community, 2020. En https://github.com/leanprover-community/mathlib/blob/master/archive/imo/imo1959_q1.lean.

[9] Kevin Lacker and Healther Macbeth. IMO 1962 Q4. Technical report, Lean community, 2020. En https://github.com/leanprover-community/mathlib/blob/master/archive/imo/imo1962_q4.lean.

[10] Filip Marié and Sana Stojanović-Durdević. Formalizing IMO problems and solutions in Isabelle/HOL. Technical report, University of Belgrade, 2020. En https://eptcs.web.cse.unsw.edu.au/paper.cgi?thedu2020.3.pdf.

[11] S. Morrison. Infinitude of primes — a lean theorem prover demo. Technical report, Youtube, 2020. Disponible en https://github.com/leanprover-community/tutorials (Accedido el 16 de Junio de 2021).

[12] Scott Morrison. IMO. Technical report, Lean community, 2020. En https://github.com/leanprover-community/mathlib/tree/master/archive/imo.

[13] Stanislas Polu, Kunhao Zheng, and David Renshaw. Minif2f. Technical report, OpenAI, 2021. En https://github.com/openai/miniF2F.

[14] Daniel Selman, Leonardo de Moura, Kevin Buzzard, Reid Barton, Percy Liang, Sarah Loos, and Freek Wiedijk. IMO Grand Challenge. Technical report, IMO, 2019. En https://imo-grand-challenge.github.io/.

Índice de lemas de Lean

Índice de tácticas