Elementos de matemáticas formalizados en Isabelle/HOL

Carlos Núñez Fernández

18 de noviembre de 2019

Resumen

En este trabajo vamos a presentar la formalización en Isabelle/HOL de una selección de teoremas de distintos campos de las matemáticas.

Índice

1 Suma de los primeros números impares

El primer teorema es una propiedad de los números naturales.

Teorema 1.1 *La suma de los n primeros números impares es n^2.*

Demostración: La demostración la haremos en inducción sobre n.

(Base de la inducción) El caso $n = 0$ es trivial.

(Paso de la inducción) Supongamos que la propiedad se verifica para n y veamos que también se verifica para $n+1$.

Tenemos que demostrar que $\sum_{j=1}^{n+1} k_j = (n+1)^2$ donde k_j el j–ésimo impar; es decir, $k_j = 2j - 1$.

$$\begin{aligned}
&\textstyle\sum_{j=1}^{n+1} k_j \\
&= k_{n+1} + \textstyle\sum_{j=1}^{n} k_j \\
&= k_{n+1} + n^2 \\
&= 2(n+1) - 1 + n^2 \\
&= n^2 + 2n + 1 \\
&= (n+1)^2
\end{aligned}$$

□

Para especificar el teorema en Isabelle, se comienza definiendo la función *suma-impares* tal que *suma-impares n* es la suma de los n primeros números impares

```
fun suma-impares :: nat ⇒ nat where
  suma-impares 0 = 0
| suma-impares (Suc n) = (2∗(Suc n) − 1) + suma-impares n
```

El enunciado del teorema es el siguiente:

```
lemma suma-impares n = n ∗ n
oops
```

En la demostración se usará la táctica *induct* que hace uso del esquema de inducción sobre los naturales:

$$\frac{P\ 0 \qquad \bigwedge nat.\ \dfrac{P\ nat}{P\ (Suc\ nat)}}{P\ nat} \qquad (nat.induct)$$

Vamos a presentar distintas demostraciones del teorema. La primera es la demostración aplicativa detallada.

```
lemma suma-impares n = n ∗ n
  apply (induct n)
   apply (simp only: suma-impares.simps(1))
  apply (simp only: suma-impares.simps(2))
  apply (simp only: mult-Suc mult-Suc-right)
  done
```

En la demostración anterior hemos usado dentro del método *simp* únicamente la definición de *suma-impares*, para ello lo hemos indicado con *simp only*. A parte hemos usado lo siguiente :

$$Suc\ m * n = n + m * n \qquad (mult\text{-}Suc)$$

$$m * Suc\ n = m + m * n \qquad (mult\text{-}Suc\text{-}right)$$

Se puede eliminar los detalles de la demostración anterior.

lemma *suma-impares* $n = n * n$
  apply (*induct* n)
   apply *simp-all*
  done

La correspondiente demostración automática es

lemma *suma-impares* $n = n * n$
  by (*induct* n) *simp-all*

La demostración estructurada y detallada del lema anterior es

lemma *suma-impares* $n = n * n$
proof (*induct* n)
  have *suma-impares* $0 = 0$
    by (*simp only*: *suma-impares.simps*(*1*))
  also have $\ldots = 0 * 0$
    by (*simp only*: *mult-0*)
  finally show *suma-impares* $0 = 0 * 0$
    by *simp*
next
  fix n
  assume *HI*: *suma-impares* $n = n * n$
  have *suma-impares* $(Suc\ n) = (2 * (Suc\ n) - 1) +$ *suma-impares* n
    by (*simp only*: *suma-impares.simps*(*2*))
  also have $\ldots = (2 * (Suc\ n) - 1) + n * n$
    by (*simp only*: *HI*)
  also have $\ldots = n * n + 2 * n + 1$
    by (*simp only*: *mult-Suc-right*)
  also have $\ldots = (Suc\ n) * (Suc\ n)$
    by (*simp only*: *mult-Suc mult-Suc-right*)
  finally show *suma-impares* $(Suc\ n) = (Suc\ n) * (Suc\ n)$

```
  by simp
qed
```

En la demostración anterior se pueden ocultar detalles.

```
lemma suma-impares n = n * n
proof (induct n)
  show suma-impares 0 = 0 * 0 by simp
next
  fix n
  assume HI: suma-impares n = n * n
  have suma-impares (Suc n) = (2 * (Suc n) - 1) + suma-impares n
    by simp
  also have ... = (2 * (Suc n) - 1) + n * n
    using HI by simp
  also have ... = (Suc n) * (Suc n)
    by simp
  finally show suma-impares (Suc n) = (Suc n) * (Suc n)
    by simp
qed
```

La demostración anterior se puede simplificar usando patrones.

```
lemma suma-impares n = n * n (is ?P n = ?Q n)
proof (induct n)
  show ?P 0 = ?Q 0 by simp
next
  fix n
  assume HI: ?P n = ?Q n
  have ?P (Suc n) = (2 * (Suc n) - 1) + suma-impares n
    by simp
  also have ... = (2 * (Suc n) - 1) + n * n using HI by simp
  also have ... = ?Q (Suc n) by simp
  finally show ?P (Suc n) = ?Q (Suc n) by simp
qed
```

La demostración usando otro patrón es

```
lemma suma-impares n = n * n (is ?P n)
proof (induct n)
  show ?P 0 by simp
next
  fix n
```

assume *?P n*
then show *?P* (*Suc n*) **by** *simp*
qed

2 Cancelación de funciones inyectivas

El siguiente teorema prueba una caracterización de las funciones inyectivas, en otras palabras, las funciones inyectivas son monomorfismos en la categoría de conjuntos. Un monomorfismo es un homomorfismo inyectivo y la categoría de conjuntos es la categoría cuyos objetos son los conjuntos.

Teorema 2.1 *f es una función inyectiva, si y solo si, para todas funciones g y h tales que* $f \circ g = f \circ h$ *se tiene que* $g = h$.

Vamos a hacer dos lemas de nuestro teorema, ya que podemos la doble implicación en dos implicaciones y demostrar cada una de ellas por separado.

Lema 2.2 *f es una función inyectiva si para todas funciones g y h tales que* $f \circ g = f \circ h$ *se tiene que* $g = h$.

Demostración: La demostración la haremos por doble implicación:

1. Supongamos que tenemos que $f \circ g = f \circ h$, queremos demostrar que $g = h$, usando que f es inyectiva tenemos que:

$$(f \circ g)(x) = (f \circ h)(x) \Longrightarrow f(g(x)) = f(h(x)) = g(x) = h(x)$$

2. Supongamos ahora que $g = h$, queremos demostrar que $f \circ g = f \circ h$.

$$(f \circ g)(x) = f(g(x)) = f(h(x)) = (f \circ h)(x)$$

.

□

Lema 2.3 *Si para toda g y h tales que* $f \circ g = f \circ h$ *se tiene que* $g = h$ *entonces f es inyectiva.*

Demostración:

Supongamos que el dominio de nuestra función f es distinto del vacío. Tenemos que demostrar que $\forall a, b$ tales que $f(a) = f(b)$, esto implica que $a = b$.
Sean a, b tales que $f(a) = f(b)$, y definamos $g(x) = a\ \forall x$ y $h(x) = b\ \forall x$ entonces

$$(f \circ g) = (f \circ h) \Longrightarrow f(g(x)) = f(h(x)) \Longrightarrow f(a) = f(b)$$

Por hipótesis tenemos entonces que $a = b$, como queríamos demostrar.

□

Su especificación es la siguiente, pero al igual que hemos hecho en la demostración a mano vamos a demostrarlo a través de dos lemas:

theorem *caracterizacionineyctiva*:
$inj\ f \longleftrightarrow (\forall\, g\ h.\ (f \circ g = f \circ h) \longrightarrow (g = h))$
oops

Sus lemas son los siguientes:

lemma
$\forall\, g\ h.\ (f \circ g = f \circ h \longrightarrow g = h) \Longrightarrow inj\ f$
oops

lemma
$inj\ f \Longrightarrow (\forall\, g\ h.(f \circ g = f \circ h) \longrightarrow (g = h))$
oops

En la especificación anterior, *inj f* es una abreviatura de *inj f* definida en la teoría Fun.thy. Además, contiene la definición de *inj-on*

$$inj\text{-}on\ f\ A = (\forall\, x{\in}A.\ \forall\, y{\in}A.\ f\ x = f\ y \longrightarrow x = y) \qquad (inj\text{-}on\text{-}def)$$

Por su parte, *UNIV* es el conjunto universal definido en la teoría Set.thy como una abreviatura de *top* que, a su vez está definido en la teoría Orderings.thy mediante la siguiente propiedad

$$\frac{ordering\text{-}top\ less\text{-}eq\ less\ top}{less\text{-}eq\ a\ top} \qquad (ordering\text{-}top.extremum)$$

En el caso de la teoría de conjuntos, la relación de orden es la inclusión de conjuntos.

Presentaremos distintas demostraciones de los lemas. La primera demostración es applicativa:

lemma *inyectivapli*:
inj f $\Longrightarrow$ $(\forall\, g\ h.(f \circ g = f \circ h) \longrightarrow\ (g = h))$
apply (*simp add*: *inj-on-def fun-eq-iff*)
done

lemma *inyectivapli2*:
$\forall\, g\ h.\ (f \circ g = f \circ h \longrightarrow g = h) \Longrightarrow$ *inj f*
apply (*rule injI*)
by (*metis fun-upd-apply fun-upd-comp*)

En las demostraciones anteriores se han usado los siguientes lemas:

$$(f = g) = (\forall\, x.\ f\, x = g\, x) \qquad (\textit{fun-eq-iff})$$

$$(f(x := y))\ z = (\textit{if}\ z = x\ \textit{then}\ y\ \textit{else}\ f\, z) \qquad (\textit{fun-upd-apply})$$

$$(f = g) = (\forall\, x.\ f\, x = g\, x) \qquad (\textit{fun-upd-comp})$$

La demostración applicativa1 sin auto es

lemma
inj f $\Longrightarrow \forall\, g\ h.\ (f \circ g = f \circ h) \longrightarrow\ (g = h)$
apply (*unfold inj-on-def*)
apply (*unfold fun-eq-iff*)
apply (*unfold o-apply*)
apply *simp+*
done

lemma
$\forall\, g\ h.\ (f \circ g = f \circ h \longrightarrow g = h) \Longrightarrow$ *inj f*
oops

En la demostración anterior se ha introducido los siguientes hechos

- $(f \circ g)\ x = f\ (g\ x)$ (*o-apply*)
- $[\![P \Longrightarrow Q;\ Q \Longrightarrow P]\!] \Longrightarrow P = Q$ (*iffI*)

La demostración automática es

lemma *inyectivaut*:
assumes *inj f*
shows $\forall\, g\ h.\ (f \circ g = f \circ h) \longrightarrow (g = h)$
using *assms*
by (*auto simp add*: *inj-on-def fun-eq-iff*)

lemma *inyectivaut2*:
assumes $\forall\, g\ h.\ ((f \circ g = f \circ h) \longrightarrow (g = h))$
shows *inj f*
using *assms*
oops

La demostración declarativa

lemma *inyectdeclarada*:
assumes *inj f*
shows $\forall\, g\ h.(f \circ g = f \circ h) \longrightarrow (g = h)$
proof
fix g:: $'c \Rightarrow 'a$
show $\forall\, h.(f \circ g = f \circ h) \longrightarrow (g = h)$
proof (*rule allI*)
fix h
show $f \circ g = f \circ h \longrightarrow (g = h)$
proof (*rule impI*)
assume $f \circ g = f \circ h$
show $g = h$
proof
fix x
have $(f \circ g)(x) = (f \circ h)(x)$ **using** ‹$f \circ g = f \circ h$› **by** *simp*
then have $f(g(x)) = f(h(x))$ **by** *simp*
thus $g(x) = h(x)$ **using** ‹*inj f*› **by** (*simp add*:*inj-on-def*)
qed
qed
qed
qed

lemma *inyectdeclarada2*:
fixes f :: $'b \Rightarrow 'c$

```
  assumes ∀(g :: 'a ⇒ 'b) (h :: 'a ⇒ 'b).
        (f ∘ g = f ∘ h ⟶ g = h)
shows  inj f
proof (rule injI)
  fix a b
  assume 3: f a = f b
  let ?g = λx :: 'a. a
  let ?h = λx :: 'a. b
  have ∀(h :: 'a ⇒ 'b). (f ∘ ?g = f ∘ h ⟶ ?g = h)
    using assms by (rule allE)
  hence 1:  (f ∘ ?g = f ∘ ?h ⟶ ?g = ?h)  by (rule allE)
  have 2: f ∘ ?g = f ∘ ?h
  proof
    fix x
    have  (f ∘ (λx :: 'a. a)) x = f(a)  by simp
    also have ... = f(b) using 3 by simp
    also have ... =  (f ∘ (λx :: 'a. b)) x by simp
    finally show  (f ∘ (λx :: 'a. a)) x =  (f ∘ (λx :: 'a. b)) x
      by simp
  qed
  have ?g = ?h using 1 2 by (rule mp)
  then show  a = b by meson
qed
```

Otra demostración declarativa es

```
lemma inyectdetalladacorta1:
  assumes inj f
  shows (f ∘ g = f ∘ h) ⟶(g = h)
proof
  assume f ∘ g = f ∘ h
  then show g = h using ‹inj f› by (simp add: inj-on-def fun-eq-iff)
qed

lemma inyectdetalladacorta2:
  fixes f :: 'b ⇒ 'c
  assumes ∀(g :: 'a ⇒ 'b) (h :: 'a ⇒ 'b).
        (f ∘ g = f ∘ h ⟶ g = h)
  shows  inj f
proof (rule injI)
  fix a b
  assume 1: f a = f b
```

```
  let ?g = λx :: 'a. a
  let ?h = λx :: 'a. b
  have 2: (f ∘ ?g = f ∘ ?h ⟶ ?g = ?h) using assms by blast
  have 3: f ∘ ?g = f ∘ ?h
  proof
    fix x
    have (f ∘ (λx :: 'a. a)) x = f(a) by simp
    also have ... = f(b) using 1 by simp
    also have ... = (f ∘ (λx :: 'a. b)) x by simp
    finally show (f ∘ (λx :: 'a. a)) x = (f ∘ (λx :: 'a. b)) x
      by simp
  qed
  show a = b using 2 3 by meson
qed
```

En consecuencia, la demostración de nuestro teorema:

```
theorem caracterizacioninyectiva:
  inj f ⟷ (∀ g h. (f ∘ g = f ∘ h) ⟶ (g = h))
  using inyectdetalladacorta1 inyectdetalladacorta2 by auto
```

3 Cancelación de las funciones sobreyectivas

El siguiente teorema prueba una caracterización de las funciones sobreyectivas, en otras palabras, las funciones sobreyectivas son epimorfismos en la categoría de conjuntos. Donde un epimorfismo es un homomorfismo sobreyectivo y la categoría de conjuntos es la categoría donde los objetos son conjuntos.

Teorema 3.1 *f es sobreyectiva si y solo si para todas funciones g y h tal que g o f = h o f se tiene que g = h.*

El teorema lo podemos dividir en dos lemas, ya que el teorema se demuestra por una doble implicación, luego vamos a dividir el teorema en las dos implicaciones.

Lema 3.2 *f es sobreyectiva entonces para todas funciones g y h tal que g o f = h o f se tiene que g = h.*

Demostración:

- Supongamos que tenemos que $g \circ f = h \circ f$, queremos probar que $g = h$. Usando la definición de sobreyectividad ($\forall y \in Y, \exists x | y = f(x)$) y nuestra hipótesis, tenemos que:

$$g(y) = g(f(x)) = (gof)(x) = (hof)(x) = h(f(x)) = h(y)$$

- Supongamos que $g = h$, hay que probar que $gof = hof$. Usando nuestra hipótesis, tenemos que:

$$(gof)(x) = g(f(x)) = h(f(x)) = (hof)(x).$$

(*¡*).(*¿*)

□

Lema 3.3 *Si para todas funciones g y h tal que g o f = h o f se tiene que g = h entonces f es sobreyectiva.*

Demostración: Para la demostración del ejercicios, primero debemos señalar los dominios y codominios de las funciones que vamos a usar. $f : C \longrightarrow A$, $g, h : A \longrightarrow B$. También debemos notar que nuestro conjunto B tiene que tener almenos dos elementos diferentes, supongamos que $B = \{a, b\}$.
La prueba la vamos a realizar por reducción al absurdo. Luego supongamos que nuestra función f no es sobreyectiva, es decir, $\exists y_1 \in A$ *tal que* $\nexists x \in C$: $f(x) = y$.
Definamos ahora las funciones g, h :

$$g(y) = a \ \forall y \in A$$

$$h(y) = a \ si \ y \neq y_1 \ h(y) = b \ si \ y = y_1$$

Entonces sabemos que $g(y) \neq h(y) \forall y \in A$. Sin embargo, por hipótesis tenemos que si $g \circ f = h \circ f$, lo cual es cierto, se tiene que $h = g$. Por lo que hemos llegado a una contradicción, entonces f es sobreyectiva.

□

Su especificación es la siguiente, que la dividiremos en dos al igual que en la demostración a mano:

theorem
$surj\ f \longleftrightarrow (\forall\, g\ h.(g \circ f = h \circ f) \longrightarrow (g = h))$
oops

lemma
$surj\ f \Longrightarrow\ (\forall\, g\ h.\ (g \circ f = h \circ f) \longrightarrow (g = h))$
oops

lemma
$\forall\, g\ h.\ (g \circ f = h \circ f \longrightarrow g = h) \longrightarrow surj\ f$
oops

En la especificación anterior, *surj f* es una abreviatura de *range f* = *UNIV*, donde *range f* es el rango o imagen de la función f. Por otra parte, *UNIV* es el conjunto universal definido en la teoría Set.thy como una abreviatura de *top* que, a su vez está definido en la teoría Orderings.thy mediante la siguiente propiedad

$$\frac{ordering\text{-}top\ less\text{-}eq\ less\ top}{less\text{-}eq\ a\ top} \qquad (ordering\text{-}top.extremum)$$

Además queda añadir que la teoría donde se encuentra definido *surj f* es en Fun.thy. Esta teoría contiene la definicion *surj-def*.

$$surj\ f = (\forall\, y.\ \exists\, x.\ y = f\ x) \qquad (inj\text{-}on\text{-}def)$$

Presentaremos distintas demostraciones de los lemas. Las primeras son las detalladas:

lemma *sobreyectivadetallada*:
assumes *surj f*
shows $\forall\, g\ h.\ (\ g \circ f = h \circ f\ ) \longrightarrow (g = h)$
proof (*rule allI*)
fix $g :: 'a \Rightarrow 'c$
show $\forall\, h.\ (g \circ f = h \circ f) \longrightarrow (g = h)$
proof (*rule allI*)
fix h
show $(g \circ f = h \circ f) \longrightarrow (g = h)$
proof (*rule impI*)
assume *1*: $g \circ f = h \circ f$
show $g = h$

```
      proof
        fix x
        have  ∃ y . x = f(y) using assms by (simp add:surj-def)
        then obtain  y where 2:x = f(y) by (rule exE)
        then have  g(x) = g(f(y)) by simp
        also have  ... = (g ∘ f) (y)   by simp
        also have  ... = (h ∘ f) (y) using 1 by simp
        also have  ... = h(f(y)) by simp
        also have  ... = h(x) using 2   by (simp add: ‹x = f y›)
        finally show   g(x) = h(x)  by simp
      qed
    qed
  qed
qed


lemma sobreyectivadetallada2:
  fixes f :: 'c ⇒ 'a
  assumes ∀ (g :: 'a ⇒ 'b) (h :: 'a ⇒ 'b). ( g ∘ f = h ∘ f ) ⟶ (g = h)
  shows surj f
proof (rule surjI)
  assume 1: ¬ surj f
  have  ¬(∀ y. ∃ x. y = f x) using 1 by (simp add: surj-def)
  then have ∃ y. ∄ x. y = f x by simp
  then obtain y1 where ∄ x. y1 = f x by (rule exE)
  then have ∀ x. y1 ≠ f x  by simp
  let ?g = λx :: 'a. a :: 'b
  let ?h = fun-upd ?g y1 (b :: 'b)
  have 2:?g ∘ f = ?h ∘ f ⟶ ?g = ?h using assms by blast
  have 3:?g ∘ f = ?h ∘ f
    by (metis (mono-tags, lifting) fun-upd-def ‹∄ x :: 'c. (y1 :: 'a) = (f :: 'c
⇒ 'a) x› f-inv-into-f fun.map-cong0)
  have ?g = ?h using 2 3 by (rule mp)
  have ?g ≠ ?h
  proof
    oops
```

En la demostración hemos introducido:

$$\frac{\exists\, x :: 'a.\ (P :: 'a \Rightarrow bool)\ x \qquad \bigwedge x :: 'a.\ \dfrac{P\ x}{Q :: bool}}{Q} \qquad (rule\ exE)$$

$$[\![P :: bool \Longrightarrow Q :: bool;\ Q \Longrightarrow P]\!] \Longrightarrow P = Q \qquad (iffI)$$

La demostración aplicativa es:

```
lemma surj f ⟹ ((g ∘ f) = (h ∘ f) ) = (g = h)
  apply (simp add: surj-def fun-eq-iff)
  apply (rule iffI)
   prefer 2
  apply auto

  apply  metis

  done

lemma surj f ⟹ ((g ∘ f) = (h ∘ f) ) = (g = h)
  apply (simp add: surj-def fun-eq-iff  )
  by metis
```

En esta demostración hemos introducido:

$$((f :: 'a \Rightarrow 'b) = (g :: 'a \Rightarrow 'b)) = (\forall\, x :: 'a.\ f\ x = g\ x) \qquad (fun\text{-}eq\text{-}iff)$$

4 Propiedad de los conjuntos finitos de números naturales

El siguiente teorema es una propiedad que verifican todos los conjuntos finitos de números naturales estudiado en el tema 10 de la asignatura de LMF. Su enunciado es el siguiente

Teorema 4.1 *Sea S un conjunto finito de números naturales. Entonces todos los elementos de S son menores o iguales que la suma de los elementos de S, es decir,*

$$\forall m, m \in S \Longrightarrow m \leq \sum S$$

donde $\sum S$ denota la suma de todos los elementos de S.

Demostración: La demostración del teorema la haremos por inducción sobre conjuntos finitos naturales.
Primero veamos el caso base, es decir, supongamos que $S = \emptyset$:

Tenemos que:

$$\forall n, n \in \emptyset \Longrightarrow n \leq \sum \emptyset.$$

Ya hemos probado el caso base, veamos ahora el paso inductivo:
Sea S un conjunto finito para el que se cumple la hipótesis, es decir, todos los elementos de S son menores o iguales que la suma de todos sus elementos, sea a un elemento tal que $a \notin S$, ya que si $a \in S$ entonces la demostración es trivial.
Hay que probar:

$$\forall n, n \in S \cup \{a\} \Longrightarrow n \leq \sum (S \cup \{a\})$$

Vamos a distinguir dos casos:
Caso 1: $n = a$
Si $n = a$ tenemos que $n = a \leq a + \sum S = \sum (S \cup \{a\})$
Caso 2: $n \neq a$
Si $n \neq a$ tenemos que $a \notin S$ y que $n \in S \cup \{a\}$, luego esto implica que $n \in S$ y usando la hipótesis de inducción

$$n \in S \Longrightarrow n \leq \sum S \leq \sum S + a = \sum (S \cup \{a\})$$

□

En la demostración del teorema hemos usado un resultado, que vamos a probar en Isabelle después de la especificación del teorema, el resultado es $\sum S + a = \sum (S \cup \{a\})$.

Para la especificación del teorema en isabelle, primero debemos notar que *finite S* indica que nuestro conjunto S es finito y definir la función *sumaConj* tal que *sumaConj n* esla suma de todos los elementos de S.

definition *sumaConj* :: *nat set* ⇒ *nat* **where**
sumaConj S ≡ ∑ *S*

El enunciado del teorema es el siguiente :

lemma *finite S* ⟹ ∀ *x*∈*S*. *x* ≤ *sumaConj S*

oops

Vamos a demostrar primero el lema enunciado anteriormente

lemma $x \notin S \wedge \mathit{finite}\ S \longrightarrow \mathit{sumaConj}\ S + x = \mathit{sumaConj}(\mathit{insert}\ x\ S)$
by (*simp add*: *sumaConj-def*)

La demostración del lema anterior se ha incluido *sumConj-def*, que hace referencia a la definición sumaConj que hemos hecho anteriormente.
En la demostración se usará la táctica *induct* que hace uso del esquema de inducción sobre los conjuntos finitos naturales:

$[\![\mathit{finite}\ x;\ P\ \emptyset;\ \bigwedge A\ a.\ \mathit{finite}\ A \wedge P\ A \Longrightarrow P\ (\{a\} \cup A)]\!] \Longrightarrow P\ x$
(*finite.induct*)

Vamos a ver presentar las diferentes formas de demostración.

La demostración aplicativa es:

lemma $\mathit{finite}\ S \Longrightarrow \forall\, x{\in}S.\ x \leq \mathit{sumaConj}\ S$
apply (*induct rule*: *finite-induct*)
apply *simp*
apply (*simp add*: *add-increasing sumaConj-def*)
done

En la demostración anterior se ha introducido:

$$\frac{(0 :: 'a) \leq a \wedge b \leq c}{b \leq a + c} \qquad (\mathit{add\text{-}increasing})$$

La demostración automática es:

lemma $\mathit{finite}\ S \Longrightarrow \forall\, x{\in}S.\ x \leq \mathit{sumaConj}\ S$
by (*induct rule*: *finite-induct*)
(*auto simp add*: *sumaConj-def*)

La demostración declarativa es:

lemma *sumaConj-acota*:
$\mathit{finite}\ S \Longrightarrow \forall\, x{\in}S.\ x \leq \mathit{sumaConj}\ S$
proof (*induct rule*: *finite-induct*)
show $\forall\, x \in \{\}.\ x \leq \mathit{sumaConj}\ \{\}$ **by** *simp*
next
fix x **and** F
assume *fF*: *finite F*
and *xF*: $x \notin F$
and *HI*: $\forall\ x{\in}F.\ x \leq \mathit{sumaConj}\ F$

```
  show ∀ y ∈ insert x F. y ≤ sumaConj (insert x F)
  proof
    fix y
    assume y ∈ insert x F
    show y ≤ sumaConj (insert x F)
    proof (cases y = x)
      assume y = x
      then have y ≤ x + (sumaConj F) by simp
     also have ... = sumaConj (insert x F)   by (simp add: fF sumaConj-def
xF)
      finally show ?thesis .
    next
      assume y ≠ x
      then have y ∈ F using ‹y ∈ insert x F› by simp
      then have y ≤ sumaConj F using HI by simp
      also have ... ≤ x + (sumaConj F) by simp
      also have ... = sumaConj (insert x F) using fF xF
        by (simp add: sumaConj-def)
      finally show ?thesis .
    qed
  qed
qed
```

5 Teorema de Cantor

El siguiente, denominado teorema de Cantor por el matemático Georg Cantor, es un resultado importante de la teoría de conjuntos.

El matemático Georg Ferdinand Ludwig Philipp Cantor fue un matemático y lógico nacido en Rusia en el siglo XIX. Fue inventor junto con Dedekind y Frege de la teoría de conjuntos, que es la base de las matemáticas modernas.

Para la comprensión del teorema vamos a definir una serie de conceptos:

- Conjunto de potencia A ($\mathcal{P}(A)$): conjunto formado por todos los subconjuntos de A.
- Cardinal del conjunto A (Denotado $\#A$): número de elementos del propio conjunto.

El enunciado original del teorema es el siguiente :

Teorema 5.1 *El cardinal del conjunto potencia de cualquier conjunto A es estrictamente mayor que el cardinal de A, o lo que es lo mismo, $\#\mathcal{P}(A) > \#A$.*

Pero el enunciado del teorema lo podemos reformular como:

Teorema 5.2 *Dado un conjunto A, $\nexists f : A \longrightarrow \mathcal{P}(A)$ que sea sobreyectiva.*

El teorema lo hemos podido reescribir de la anterior forma, ya que si suponemos que $\exists f$ tal que $f : A \longrightarrow \mathcal{P}(A)$ es sobreyectiva, entonces tenemos que $f(A) = \mathcal{P}(A)$ y por lo tanto, $\#f(A) \geq \#\mathcal{P}(A)$, de lo que se deduce esta reformulación. Reciprocamente, es trivial ver que esta reformulación implica la primera. con el teorema.
El teorema de Cantor es trivial para conjuntos finitos, ya que el conjunto potencia, de conjuntos finitos de n elementos tiene 2^n elementos.

Por ello, vamos a realizar la prueba para conjuntos infinitos.

Demostración:

Vamos a realizar la prueba por reducción al absurdo.
Supongamos que $\exists f : A \longrightarrow \mathcal{P}(A)$ sobreyectiva, es decir, $\forall C \in \rho(A), \exists x \in A$ tal que $C = f(x)$. En particular, tomemos el conjunto

$$B = \{x \in A : x \notin f(x)\}$$

y supongamos que $\exists a \in A : B = f(a)$, ya que B es un subconjunto de A, luego podemos distinguir dos casos :
1. Si $a \in B$, entonces por definición del conjunto B tenemos que $a \notin B$, luego llegamos a una contradicción.
2. Si $a \notin B$, entonces por definición de B tenemos que $a \in B$, luego hemos llegado a otra contradicción.

En las dos hipótesis hemos llegado a una contradicción, por lo que no existe a y f no es sobreyectiva.

□

Para la especificación del teorema en Isabelle, primero debemos notar que

$$f :: 'a \Rightarrow 'a\ set$$

significa que es una función de tipos, donde $'a$ significa un tipo y para poder denotar el conjunto potencia tenemos que poner $'a\ set$ que significa que es de un tipo formado por conjuntos del tipo $'a$.

El enunciado del teorema es el siguiente :

theorem *Cantor*: $\nexists f :: 'a \Rightarrow 'a\ set.\ \forall A.\ \exists x.\ A = f\ x$

oops

La demostración la haremos por la regla la introducción a la negación, la cual es una simplificación de la regla de reducción al absurdo, cuyo esquema mostramos a continuación:

$$(P \Longrightarrow False) \Longrightarrow \neg\ P \qquad (notI)$$

Esta es la demostración detallada del teorema:

theorem *CantorDetallada*: $\nexists f :: 'a \Rightarrow 'a\ set.\ \forall B.\ \exists x.\ B = f\ x$
proof (*rule notI*)
assume $\exists f :: 'a \Rightarrow 'a\ set.\ \forall A.\ \exists x.\ A = f\ x$
then obtain $f :: 'a \Rightarrow 'a\ set$ **where** $*$: $\forall A.\ \exists x.\ A = f\ x$ **by** (*rule exE*)
let *?B* $= \{x.\ x \notin f\ x\}$
from $*$ **obtain** $\exists x.\ ?B = f\ x$ **by** (*rule allE*)
then obtain *a* **where** *1*:*?B* $= f\ a$ **by** (*rule exE*)
show *False*
proof (*cases*)
assume $a \in ?B$
then show *False* **using** *1* **by** *blast*
next
assume $a \notin ?B$
thus *False* **using** *1* **by** *blast*
qed
qed

Esta es la demostración aplicativa del teorema:

theorem *CantorAplicativa* :
$\nexists f :: 'a \Rightarrow 'a\ set.\ \forall A.\ \exists x.\ A = f\ x$
apply (*rule notI*)
apply (*erule exE*)
apply (*erule-tac* $x = \{x.\ x \notin f\ x\}$ **in** *allE*)
apply (*erule exE*)
apply *blast*
done

Esta es la demostración automática del teorema:

theorem *CantorAutomatic*: $\nexists f :: 'a \Rightarrow 'a\ set.\ \forall B.\ \exists x.\ B = f\ x$
by *best*

En la demostración de isabelle hemos utilizado el método de prueba rule con las siguientes reglas, tanto en la aplicativa como en la detallada:

$$\frac{\dfrac{P}{False}}{\neg\ P} \qquad (notI)$$

$$\frac{\exists x.\ P\ x \qquad \bigwedge x.\ \dfrac{P\ x}{Q}}{Q} \qquad (exE)$$

$$\frac{\forall x.\ P\ x \qquad \dfrac{P\ x}{R}}{R} \qquad (allE)$$

También hacemos uso de blast, que es un conjunto de reglas lógicas y la demostración automática la hacemos por medio de "best".

6 Métodos de pruebas y reglas

Métodos de pruebas de demostraciones:

$$[\![P\ 0;\ \bigwedge nat.\ P\ nat \Longrightarrow P\ (Suc\ nat)]\!] \Longrightarrow P\ nat \qquad (nat.induct)$$

$$[\![P \Longrightarrow Q;\ Q \Longrightarrow P]\!] \Longrightarrow P = Q \qquad (iffI)$$

$[\![finite\ x;\ P\ \emptyset;\ \bigwedge A\ a.\ finite\ A \wedge P\ A \Longrightarrow P\ (\{a\} \cup A)]\!] \Longrightarrow P\ x$
$(finite.induct)$

$$(P \Longrightarrow False) \Longrightarrow \neg\ P \qquad (notI)$$

Reglas usadas:

$$inj\text{-}on\ f\ A = (\forall x{\in}A.\ \forall y{\in}A.\ f\ x = f\ y \longrightarrow x = y) \qquad (inj\text{-}on\text{-}def)$$

$$\frac{ordering\text{-}top\ less\text{-}eq\ less\ top}{less\text{-}eq\ a\ top} \qquad (ordering\text{-}top.extremum)$$

$$(f = g) = (\forall x.\ f\ x = g\ x) \qquad (fun\text{-}eq\text{-}iff)$$

$$(f \circ g)\ x = f\ (g\ x) \qquad (o\text{-}apply)$$

$$\frac{\begin{array}{cc} P & Q \\ \overline{Q} & \overline{P} \end{array}}{P = Q} \qquad (iffI)$$

$$\frac{ListMem\ x\ xs}{ListMem\ x\ (y \cdot xs)} \qquad (insert)$$

$$\frac{\exists x.\ P\ x \qquad \bigwedge x.\ \dfrac{P\ x}{Q}}{Q} \qquad (exE)$$

$$\frac{\forall x.\ P\ x \qquad \dfrac{P\ x}{R}}{R} \qquad (allE)$$

$$\frac{\dfrac{P}{False}}{\neg\ P} \qquad (notI)$$

$$((P \longrightarrow Q) \wedge (\neg\ P \longrightarrow Q)) = Q \qquad (cases)$$

Referencias

[1] José A. Alonso. Temas de "Lógica matemática y fundamentos (2018–19)". Technical report, Univ. de Sevilla, 2019. En https://www.cs.us.es/~jalonso/cursos/lmf-18/temas.php.

[2] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A proof assistant for Higher–Order Logic*. Lecture Notes in Computer Science, Vol. 2283, Springer–Verlag, 2019. En https://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2019/doc/tutorial.pdf.