Elementos de matemáticas formalizados en Isabelle/HOL

Carlos Núñez Fernández

18 de noviembre de 2019

Índice

Capítulo 1

Suma de los primeros números impares

1.1 Demostración en lenguaje natural

El primer teorema es una propiedad de los números naturales.

Teorema 1.1.1 *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, $k_j = 2j - 1$.

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

□

1.2 Especificación en Isabelle/HOL

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

1.3 Demostración aplicativa

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

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

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 :

$$Suc\ m * n = n + m * n \tag{mult-Suc}$$

$$m * Suc\ n = m + m * n \tag{mult-Suc-right}$$

Se puede eliminar los detalles de la demostración anterior.

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

1.4 Demostración automática

La correspondiente demostración automática es

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

1.5 Demostración estructurada

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 $\mathit{suma\text{-}impares}\ n = n * n$
proof $(\mathit{induct}\ n)$
 show $\mathit{suma\text{-}impares}\ 0 = 0 * 0$ **by** *simp*
next
 fix n
 assume HI: $\mathit{suma\text{-}impares}\ n = n * n$
 have $\mathit{suma\text{-}impares}\ (\mathit{Suc}\ n) = (2 * (\mathit{Suc}\ n) - 1) + \mathit{suma\text{-}impares}\ n$
  by *simp*
 also have $\ldots = (2 * (\mathit{Suc}\ n) - 1) + n * n$
  using HI **by** *simp*
 also have $\ldots = (\mathit{Suc}\ n) * (\mathit{Suc}\ n)$
  by *simp*
 finally show $\mathit{suma\text{-}impares}\ (\mathit{Suc}\ n) = (\mathit{Suc}\ n) * (\mathit{Suc}\ n)$
  by *simp*
qed

1.6 Demostración con patrones

La demostración anterior se puede simplificar usando patrones.

lemma $\mathit{suma\text{-}impares}\ n = n * n$ (**is** $?P\ n = ?Q\ n$)
proof $(\mathit{induct}\ n)$
 show $?P\ 0 = ?Q\ 0$ **by** *simp*
next
 fix n
 assume HI: $?P\ n = ?Q\ n$
 have $?P\ (\mathit{Suc}\ n) = (2 * (\mathit{Suc}\ n) - 1) + \mathit{suma\text{-}impares}\ n$
  by *simp*
 also have $\ldots = (2 * (\mathit{Suc}\ n) - 1) + n * n$ **using** HI **by** *simp*
 also have $\ldots = ?Q\ (\mathit{Suc}\ n)$ **by** *simp*
 finally show $?P\ (\mathit{Suc}\ n) = ?Q\ (\mathit{Suc}\ n)$ **by** *simp*
qed

La demostración usando otro patrón es

lemma $\mathit{suma\text{-}impares}\ n = n * n$ (**is** $?P\ n$)
proof $(\mathit{induct}\ n)$
 show $?P\ 0$ **by** *simp*
next
 fix n

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

Capítulo 2

Cancelación de funciones inyectivas

2.1 Demostración en lenguaje natural

Comentario 1: Estructurar en secciones.

Comentario 2: Hacer demostraciones detalladas.

Comentario 3: Añadir lemas usados al Soporte.

El siguiente teorema que se va a probar es una caracterización de las funciones inyectivas. Primero se definirá el significado de inyectividad de una función y la propiedad de ser cancelativa por la izquierda.
Una función $f : B \longrightarrow C$ es inyectiva si

$$\forall x, y \in \ B : f(x) = f(y) \Longrightarrow x = y.$$

Una función $f : B \longrightarrow C$ es cancelativa por la izquierda si

$$\forall A : (\forall g, h : X \longrightarrow Y) : f \circ g = f \circ h \Longrightarrow g = h.$$

Luego el teorema es el siguiente:

Luego el teorema es el siguiente:

Teorema 2.1.1 *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 se descompone la doble implicación en dos implicaciones y se va a demostrar cada una de ellas por separado.

Lema 2.1.2 (Condición necesaria) *Si f es una función inyectiva entonces para todas funciones g y h tales que $f \circ g = f \circ h$ se tiene que $g = h$.*

Demostración: Por hipótesis se tiene que $f \circ g = f \circ h$, hay que probar que $g = h$. Usando que f es inyectiva tenemos que:

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

□

Lema 2.1.3 (Condición suficiente) *Si para toda g y h tales que $f \circ g = f \circ h$ se tiene que $g = h$ entonces f es inyectiva.*

Demostración: Si el dominio de la función f fuese vacío, f es inyectiva. Supongamos que el dominio de la función f es distinto del vacío y que f verifica la propiedad de ser cancelativa por la izquierda. Hay que demostrar que $\forall a, b$ tales que $f(a) = f(b)$, esto implica que $a = b$.
Sean a, b tales que $f(a) = f(b)$.
Definiendo $g(x) = a\ \forall x$ 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, entonces $a = b$, como se quería demostrar.

□

2.2 Especificación en Isabelle/Hol

Su especificación es la siguiente, pero al igual que se ha hecho en la demostración a mano se va a demostrar a través de dos lemas:

theorem *caracterizacion-funcion-inyecctiva*:
$inj\, f \longleftrightarrow (\forall\, g\, h.\ (f \circ g = f \circ h) \longrightarrow (g = h))$
oops

Sus lemas asociados a cada implicación 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.

2.3 Demostración aplicativa lemas

Las demostraciones aplicativas de los lemas son :

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

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

En las demostraciones anteriores se han usado los siguientes lemas:

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

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

$(f = g) = (\forall x.\, f\, x = g\, x)$ (*fun-upd-comp*)

2.4 Demostración estructurada lemas

Las demostraciones declarativas son las siguientes:

lemma *condicion-necesaria-detallada*:
 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** $\langle f \circ g = f \circ h \rangle$ **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 *condicion-suficiente-detallada*:
 fixes $f :: 'b \Rightarrow 'c$
 assumes $\forall (g :: 'a \Rightarrow 'b)\ (h :: 'a \Rightarrow 'b).$
   $(f \circ g = f \circ h \longrightarrow g = h)$
shows *inj f*
proof (*rule injI*)
 fix $a\ b$
 assume *3*: $f\, a = f\, b$
 let $?g = \lambda x :: 'a.\ a$
 let $?h = \lambda x :: 'a.\ b$
 have $\forall (h :: 'a \Rightarrow 'b).\ (f \circ ?g = f \circ h \longrightarrow ?g = h)$

using *assms* **by** $(rule\ allE)$
hence *1*: $(f \circ ?g = f \circ ?h \longrightarrow ?g = ?h)$ **by** $(rule\ allE)$
have *2*: $f \circ ?g = f \circ ?h$
proof
fix x
have $(f \circ (\lambda x :: 'a.\ a))\ x = f(a)$ **by** *simp*
also have ... $= f(b)$ **using** *3* **by** *simp*
also have ... $=\ (f \circ (\lambda x :: 'a.\ b))\ x$ **by** *simp*
finally show $(f \circ (\lambda x :: 'a.\ a))\ x =\ (f \circ (\lambda x :: 'a.\ b))\ x$
by *simp*
qed
have $?g = ?h$ **using** *1 2* **by** $(rule\ mp)$
then show $a = b$ **by** $(rule\ fun\text{-}cong)$
qed

En la anterior demostración se ha introducito la regla:

$$\frac{(f :: 'a \Rightarrow 'b) = (g :: 'a \Rightarrow 'b)}{f\ (x :: 'a) = g\ x} \qquad (fun\text{-}cong)$$

Otras demostraciones declarativas usando auto y blast son:

lemma *condicion-necesaria-detallada1*:
assumes *inj f*
shows $(f \circ g = f \circ h) \longrightarrow (g = h)$
proof
assume $f \circ g = f \circ h$
then show $g = h$ **using** $\langle inj\ f \rangle$ **by** $(simp\ add: inj\text{-}on\text{-}def\ fun\text{-}eq\text{-}iff)$
qed

lemma *condicion-suficiente-detallada1*:
fixes $f :: 'b \Rightarrow 'c$
assumes $\forall (g :: 'a \Rightarrow 'b)\ (h :: 'a \Rightarrow 'b).$
$(f \circ g = f \circ h \longrightarrow g = h)$
shows *inj f*
proof $(rule\ injI)$
fix $a\ b$
assume *1*: $f\ a = f\ b$
let $?g = \lambda x :: 'a.\ a$
let $?h = \lambda x :: 'a.\ b$
have 2: $(f \circ ?g = f \circ ?h \longrightarrow ?g = ?h)$ **using** *assms* **by** *blast*
have *3*: $f \circ ?g = f \circ ?h$

proof
  fix x
  have $(f \circ (\lambda x :: 'a.\ a))\ x = f(a)$ **by** *simp*
  also have ... $= f(b)$ **using** *1* **by** *simp*
  also have ... $= (f \circ (\lambda x :: 'a.\ b))\ x$ **by** *simp*
  finally show $(f \circ (\lambda x :: 'a.\ a))\ x = (f \circ (\lambda x :: 'a.\ b))\ x$
    by *simp*
qed
show $a = b$ **using** *2 3* **by** *meson*
qed

2.5 Demostración teorema en Isabelle/Hol

En consecuencia, la demostración de nuestro teorema:

theorem *caracterizacion-inyectividad*:
  $inj\, f \longleftrightarrow (\forall g\, h.\ (f \circ g = f \circ h) \longrightarrow (g = h))$
  using *condicion-necesaria-detallada condicion-suficiente-detallada*
  by *auto*

Capítulo 3

Cancelación de las funciones sobreyectivas

Comentario 4: Estructurar en secciones.

Comentario 5: Hacer demostraciones detalladas.

Comentario 6: Añadir lemas usados al Soporte.

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 3.0.1 *f es sobreyectiva si y solo si para todas funciones g y h tal que $g \circ f = h \circ f$ 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 3.0.2 *f es sobreyectiva entonces para todas funciones g y h tal que $g \circ f = h \circ f$ 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:

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

- Supongamos que $g = h$, hay que probar que $g \circ f = h \circ f$. Usando nuestra hipótesis, tenemos que:

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

(*¡*).(*¿*)

□

Lema 3.0.3 *Si para todas funciones g y h tal que $g \circ f = h \circ f$ 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 $\exists y\ .\ x = f(y)$ **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 $... = (g \circ f)\ (y)$ **by** *simp*
        also have $... = (h \circ f)\ (y)$ **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 $f :: {}'c \Rightarrow {}'a$
assumes $\forall\, (g :: {}'a \Rightarrow {}'b)\ (h :: {}'a \Rightarrow {}'b).\ (\ g \circ f = h \circ f\ ) \longrightarrow (g = h)$
shows *surj f*
proof (*rule surjI*)
assume *1*: $\neg$ *surj f*
have $\neg(\forall\, y.\ \exists\, x.\ y = f\ x)$ **using** *1* **by** (*simp add*: *surj-def*)
then have $\exists\, y.\ \nexists\, x.\ y = f\ x$ **by** *simp*
then obtain *y1* **where** $\nexists\, x.\ y1 = f\ x$ **by** (*rule exE*)
then have $\forall\, x.\ y1 \neq f\ x$ **by** *simp*
let $?g = \lambda x :: {}'a.\ a :: {}'b$
let $?h =$ *fun-upd* $?g\ y1\ (b :: {}'b)$
have $2{:}?g \circ f = ?h \circ f \longrightarrow ?g = ?h$ **using** *assms* **by** *blast*
have $3{:}?g \circ f = ?h \circ f$
by (*metis* (*mono-tags*, *lifting*) *fun-upd-def* ‹$\nexists\, x :: {}'c.\ (y1 :: {}'a) =$
$(f :: {}'c \Rightarrow {}'a)\ x$› *f-inv-into-f fun.map-cong0*)
have $?g = ?h$ **using** *2 3* **by** (*rule mp*)
have $?g \neq ?h$
proof
assume *4*: $?g = ?h$
show *False*
proof $-$
have $?g =$ *fun-upd* $?g\ y1\ (b :: {}'b)$ **using** *4* **by** *simp*
also have ... $=\ (\lambda x.\ \mathit{if}\ x = y1\ \mathit{then}\ b\ \mathit{else}\ ?g\ x)$ **by** (*simp add*:
fun-upd-def)
finally have *5*: $?g = (\lambda x.\ \mathit{if}\ x = y1\ \mathit{then}\ b\ \mathit{else}\ ?g\ x)$ **by** *simp*
show *False*
proof (*cases*)
oops

En la demostración hemos introducido:

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

$$[\![P :: bool \Longrightarrow Q :: bool;\, Q \Longrightarrow P]\!] \Longrightarrow P = Q \tag{iffI}$$

La demostración aplicativa es:

lemma *surj f* $\Longrightarrow ((g \circ f) = (h \circ f)\ ) \longrightarrow (g = h)$
apply (*simp add*: *surj-def fun-eq-iff*)
apply *metis*
done

lemma *surj f* $\Longrightarrow ((g \circ f) = (h \circ f)\ ) \longrightarrow (g = h)$
apply (*simp add*: *surj-def fun-eq-iff*)
by *metis*

En esta demostración hemos introducido:

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

Capítulo 4

Propiedad de los conjuntos finitos de números naturales

Comentario 7: Estructurar en secciones.

Comentario 8: Hacer demostraciones detalladas.

Comentario 9: Añadir lemas usados al Soporte.

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 4.0.1 *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* $\Rightarrow$ *nat* **where**
sumaConj S $\equiv \sum S$

El enunciado del teorema es el siguiente :

lemma *finite S* $\Longrightarrow \forall\, x{\in}S.\ x \leq$ *sumaConj S*

oops

Vamos a demostrar primero el lema enunciado anteriormente

lemma $x \notin S \wedge$ *finite S* $\longrightarrow$ *sumaConj S* $+\ x =$ *sumaConj*(*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:

$$\llbracket \textit{finite } x;\ P\ \emptyset;\ \bigwedge A\ a.\ \textit{finite } A \wedge P\ A \Longrightarrow P\ (\{a\} \cup A) \rrbracket \Longrightarrow P\ x \qquad (\textit{finite.induct})$$

Vamos a ver presentar las diferentes formas de demostración.

La demostración aplicativa es:

```
lemma finite S ⟹ ∀ x∈S. x ≤ sumaConj S
 apply (induct rule: finite-induct)
  apply simp
 apply (simp add: add-increasing sumaConj-def)
 done
```

En la demostración anterior se ha introducido:

$$\frac{(0 :: 'a) \le a \wedge b \le c}{b \le a + c} \qquad (\textit{add-increasing})$$

La demostración automática es:

```
lemma finite S ⟹ ∀ x∈S. x ≤ sumaConj S
 by (induct rule: finite-induct)
   (auto simp add:  sumaConj-def)
```

La demostración declarativa es:

```
lemma sumaConj-acota:
 finite S ⟹ ∀ x∈S. x ≤ sumaConj S
proof (induct rule: finite-induct)
 show ∀ x ∈ {}. x ≤ sumaConj {} by simp
next
 fix x and F
 assume fF: finite F
   and xF: x ∉ F
   and HI: ∀  x∈F. x ≤ 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 $y \neq x$
then have $y \in F$ **using** ⟨$y \in$ *insert x F*⟩ **by** *simp*
then have $y \leq$ *sumaConj F* **using** *HI* **by** *simp*
also have $\ldots \leq x +$ (*sumaConj F*) **by** *simp*
also have $\ldots =$ *sumaConj* (*insert x F*) **using** *fF xF*
by (*simp add*: *sumaConj-def*)
finally show *?thesis* **.**
qed
qed
qed

Capítulo 5

Teorema de Cantor

Comentario 10: Estructurar en secciones.

Comentario 11: Hacer demostraciones detalladas.

Comentario 12: Añadir lemas usados al Soporte.

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 5.0.1 *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 5.0.2 *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.

$\square$

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:

$$\cfrac{\cfrac{P}{False}}{\neg P} \qquad (notI)$$

$$\frac{\exists x.\, P\, x \qquad \bigwedge x.\, \dfrac{P\, x}{Q}}{Q} \tag{exE}$$

$$\frac{\forall x.\, P\, x \qquad \dfrac{P\, x}{R}}{R} \tag{allE}$$

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

Apéndice A

Métodos de pruebas y reglas

Métodos de pruebas de demostraciones:

$$\llbracket P\ 0;\ \bigwedge nat.\ P\ nat \Longrightarrow P\ (Suc\ nat)\rrbracket \Longrightarrow P\ nat \qquad (nat.induct)$$

$$\llbracket P \Longrightarrow Q;\ Q \Longrightarrow P\rrbracket \Longrightarrow P = Q \qquad (iffI)$$

$$\llbracket finite\ x;\ P\ \emptyset;\ \bigwedge A\ a.\ finite\ A \wedge P\ A \Longrightarrow P\ (\{a\} \cup A)\rrbracket \Longrightarrow P\ x \qquad (finite.induct)$$

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

Reglas usadas:

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

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

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

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

$$\frac{\dfrac{P}{Q} \qquad \dfrac{Q}{P}}{P = Q} \qquad (iffI)$$

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

$$\frac{\exists\, x.\ P\ x \qquad \bigwedge x.\ \dfrac{P\ x}{Q}}{Q} \tag{exE}$$

$$\frac{\forall\, x.\ P\ x \qquad \dfrac{P\ x}{R}}{R} \tag{allE}$$

$$\frac{\dfrac{P}{\mathit{False}}}{\neg\, P} \tag{notI}$$

$$((P \longrightarrow Q) \wedge (\neg\, P \longrightarrow Q)) = Q \tag{cases}$$

Apéndice B

Lemas de HOL usados

En este apéndice se recogen la lista de los lemas usados en el trabajo indicando la página del libro de HOL donde se encuentra.

Comentario 13: Añadir el libro de HOL a la bibliografía.

Comentario 14: Completar la lista de lemas usados.

B.1 Números naturales (16)

B.1.1 Operaciones aritméticas (16.3)

- (p. 348) $0 * n = 0$ (*mult-0*)
- (p. 348) $Suc\ m * n = n + m * n$ (*mult-Suc*)
- (p. 348) $m * Suc\ n = m + m * n$ (*mult-Suc-right*)

Bibliografía

[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.

Lista de tareas pendientes