# Qué es una demo?

Eric Brandwein
25/08/2025
Cubaweeki 2025

**Pregunta:** *De qué carreras son? Quién es de compu? Quién es de datos? Quién es de otra carrera, y de qué carrera son?*

**Pregunta:** *Quiénes cursaron álgebra? Quiénes cursaron algo3?*

**Pregunta:** *Quiénes se sienten seguros de poder escribir una demo suficientemente formal para publicar en un paper?*

Primero tenemos que definir lo que es una demostración.

- Tírenme ideas de lo que podría ser una demo:
  1. "Una forma de convencer a alguien de una verdad matemática"
  2. "Una secuencia de aplicaciones de axiomas y teoremas"
  3. etc?

A mí me gusta más la segunda, porque puedo convencer a alguien de algo que no sea verdad. Pero pará, si una demo es una secuencia de llamados a teoremas y axiomas, cómo puede ser que las escribamos en español? Bueno, lo que escribimos es un *boceto* de la demo, no la demo en sí. La demo propiamente dicha es un algoritmo. No lo digo en el sentido figurativo; es literalmente un algoritmo. Lo podés programar en el lenguaje de unas cosas que se llaman "interactive theorem provers" y correrlos en tu compu para ver si la demo vale[1].

**Pregunta:** *Alguno usó un interactive theorem prover alguna vez?*

Por ejemplo, ponele que queremos probar lo siguiente:

**Teorema 1** $(a + 0) + c = a + (c + 0)$.

**Pregunta:** *A ver, dénme una demo bien formal de esto.*

Una demo bien formal escrita en español podría ser así:

**Demostración**

[1] Hubo una ECI hace poco que enseñaban la teoría y práctica de los interactive theorem provers, y también hay una optativa de Pablo Barenbaum este cuatri sobre estas cosas.

$$
\begin{aligned}
(a+0)+c = a+(c+0) &\Leftrightarrow (a)+c = a+(c+0) && \rightarrow \text{por axioma de la suma de 0.}\\
&\Leftrightarrow a+c = a+(c+0) && \rightarrow \text{por regla de paréntesis extra.}\\
&\Leftrightarrow a+c = a+(c) && \rightarrow \text{por axioma de la suma de 0.}\\
&\Leftrightarrow a+c = a+c && \rightarrow \text{por regla de paréntesis extra.}\\
&\Leftrightarrow \text{True} && \rightarrow \text{por identidad de términos.}
\end{aligned}
$$

□

Esto es "fácil" de pasar a un lenguaje de programación. Podríamos hacer un compilador que convierta lo de la izquierda en las conclusiones, o "goals", y lo de la derecha en las aplicaciones de resultados previos. El resultado podría ser algo así en el lenguaje Lean 4.

**Demostración**

```
import Mathlib.Data.Real.Basic

example (a c : Nat) : (a + 0) + c = a + (c + 0) := by
    rewrite [add_zero] -- a + c = a + (c + 0)
    rewrite [add_zero] -- a + c = a + c
```

□

Los "--" son comentarios para ayudar a la lectura, no hace falta ponerlos. Resulta que las líneas que dicen "por regla de ..." en la anterior demo son parte del intérprete de Lean, así que tampoco hace falta ponerlos. También el rewrite se fija automáticamente si lo que te queda es igual, y aplica la reflexividad de la igualdad.

Esta última demo la pueden analizar en el intérprete online de Lean 4 e ir línea por línea viendo cómo se va tranformando el "goal".[2]

Volviendo, uno podría escribir lo que dije en la demo 1 más en palabras, y sería lo mismo.

**Pregunta:** *Cómo escribirían esa demo en palabras?*

**Demostración** Nótese que $a$ sumado a $c$ es equivalente a sí mismo. Luego, por axioma de suma de cero, podemos sumar un cero a la $a$ de la izquierda, y sumar un cero a la $c$ de la derecha, y obtener el enunciado del teorema. □

Esta es la idea de las demos que ustedes deberían escribir en una materia como Algo 3. Aunque el texto esté escrito en español, debería poder pasarse fácilmente a un lenguaje formal. Imagínense a un estudiante como ustedes que además se sepa la sintaxis de Lean y que se sepa los nombres que tienen los teoremas y los axiomas en Lean, de la misma manera que ustedes saben cómo escribir un `for` en Python; esa persona debería poder traducir su demostración a Lean de la misma manera que ustedes leerían un código en C++ y lo traducirían a Python, haciendo de "compiladores" de español a Lean.

[2] Para aprender a programar en Lean pueden empezar con el natural numbers game.

**Pregunta:** *Cuáles de estas demos podrían encontrar en un paper?*

Como esta demo es demasiado básica, quizá en realidad aparecería algo así en un paper, o ni siquiera aparecería:

**Demostración** Vale por propiedades básicas de la aritmética. □

Nótese que esto es más difícil de pasar a Lean. Se te tiene que ocurrir que hay que usar el axioma de que sumar cero es lo mismo que no hacer nada. Esta demo sólo te da una ayudita de dónde tenés que buscar. Yo digo que esta demo es menos "formal" que las otras; como que cubre todos los casos pero no te tira todos los pasos para resolverlos. Otros quizá dicen que "rigurosidad" es otra cosa (y no digo que no tengan razón) pero vamos a usar el término así en este tallercito.

Nótese también que hace falta saberse (o saber buscar) los axiomas de la suma para poder pasar una demo no formal a otra formal. De alguna manera, la no formal depende más de la intuición que tiene uno de las cosas que son verdaderas. Esto es más obvio cuando se demuestran cosas sobre objetos más "dibujables", como por ejemplo si quisiéramos probar que un círculo no tiene ningún vértice. Todos lo podemos ver, pero qué se yo cómo se demuestra formalmente. Tendría que tener en mente la definición de círculo y de vértice, que como no hice matemática no sé cuáles son. Personalmente no me creo capaz de demostrar esto sin hacer una materia o leer un libro de, qué sería, topología? Ni idea, los matemáticos me podrán decir.

## Cosas malas en demos

Nótese que todas estas demos que vimos son correctas en el sentido de que todas se pueden transformar en una demostración completamente formal. Podríamos tener una que no fuese correcta, por ejemplo:

**Demostración** Vale porque todo natural es positivo. □

Es mentira que el teorema vale por esto; son las reglas de la suma de naturales las que nos dan el resultado. Podría ser más sutil, por ejemplo:

**Demostración** Como todo natural es positivo, la suma de $a$ con 0 es igual a $a$, y lo mismo con $c$. □

De nuevo, no tiene nada que ver que $a$ sea positivo con que $a + 0 = a$. Podría sin embargo uno decir que esta demo es más *formal* que la anterior, porque te dice más el paso a paso, aunque los pasos estén mal.

**Pregunta:** *Díganme errores comunes que se les ocurran en demos.*

Por suerte no hay manera que sepamos de llegar a un resultado falso haciendo todos pasos correctos, así que el único verdadero error que pueden tener es aplicar mal los axiomas y teoremas. Pero igual hay cosas que tenemos que fijarnos que no pasen.

**Errores comunes**

1. Deducir algo falso usando mal las conclusiones de un teorema/axioma/definición.

   **Pregunta:** *Dar ejemplo.*

2. Deducir algo verdadero usando las hipótesis incorrectas.

   **Pregunta:** *Dar ejemplo.*

3. No contemplar todos los casos.

   **Pregunta:** *Dar ejemplo.*

4. Falta de formalismo, o sea, decir los pasos muy por arriba.

   **Pregunta:** *Dar ejemplo.*

5. Terminar demostrando otra cosa.

   **Pregunta:** *Dar ejemplo.*

6. Que no se entienda nada, i.e. errores de escritura (ambigüedad, gramática, caligrafía, palabras raras o frases largas).

   **Pregunta:** *Dar ejemplo.*

**Cosas al borde**

**Repetir lo mismo que dijiste antes.** Algunas veces esto es porque no están 100% convencidos de algo. Por ejemplo:

**Demostración** No podría pasar que $(a + 0) + c \neq a + (c + 0)$, porque la suma no funciona así. Como los números son naturales, $(a + 0) + c \neq a + (c + 0)$ es falso. □

Hay seguro mejores ejemplos en resueltos de Algo 3 de CubaWiki.

**Cómo encontrar estos errores**

Encontrar estos errores es muy parecido a debuggear un programa, porque literalmente están buscando un bug en un algoritmo: su demostración.

1. Si tienen un contraejemplo, corran la demo en su contraejemplo para ver dónde se equivocaron. Alguna de las afirmaciones va a ser la primera en la que se equivocaron; identifiquen cuál de los errores de arriba es el causante y arréglenlo. Obviamente, sigan corriendo la demo con este ejemplo para ver si no hay más errores.
2. Como en programación, tengan una "batería de tests". Busquen casos borde que puedan romper todo. Qué pasa si el conjunto tiene 0, 1 o 2 elementos? Qué pasa si el número es negativo? etc.
3. Si no es suficiente, hay que volver al debuggeo manual: ir línea por línea y ver si es correcta. Cada afirmación tiene que deducirse sólamente de las cosas que dijeron y nada más.

Igual a debuggear un código, uno necesita práctica para detectar los errores más comunes, tipo dividir por cero, acceder a una posición de memoria inválida, etc. Lo malo es que a diferencia del código en Python, para correr este algoritmo en una computadora hay que pasarlo a un lenguaje formal, lo cual tarda el cuádruple[3], así que estamos cuasi obligados a correrlos en la cabeza.

**Ejercicio:** *Entren a la materia que más les guste de CubaWiki (álgebra, análisis, Algo 2, Algo 3) y busquen cualquier demo de un alumno. Traten de ver qué errores de estos tienen.*

## Recomendaciones

### Practiquen, practiquen, practiquen.

No importa cuántas demos lean, sin práctica no van a darse cuenta qué cosas hacen mal ustedes en demos, y no las van a poder mejorar.

### Pasen las ideas a definiciones formales.

Por ejemplo, qué quiere decir "círculo"? En grafos, qué quiere decir "camino"? Qué quiere decir "el máximo de un conjunto"? Sin pasar estas ideas a definiciones formales no podemos aplicarles los teoremas y axiomas que son súper útiles, y nos quedamos siempre en el mundo de la intuición.

### Sépanse los axiomas y teoremas comunes del área.

De la misma manera que no podíamos probar formalmente el teoremita del principio sin saber los axiomas de la suma, no vamos a poder probar nada formalmente sin saber los axiomas y teoremas de la cosa que estamos tratando de probar. Por ejemplo, en Algo 3, si quieren demostrar cosas sobre grafos, como los grafos son solo un par de conjuntos, tienen que saber los axiomas y teoremas de conjuntos, secuencias, y funciones.

### Ante la duda, háganlo más riguroso.

Hay algunas cosas que terminan siendo obvias y no queremos tener que demostrar cada vez, como el primer teorema que vimos hoy. Pero hay otras cosas que no estamos seguros si son obvias o no para el lector. Ahí es cuando tenemos que ser más rigurosos para convencerlo.

Por otra parte, también puede pasar que nos encontremos con que escribimos una cosa, y no sabemos bien por qué pasa. Por cada afirmación que hagan, pregúntense cuán seguros están de que nadie podría encontrar un contraejemplo. Si la respuesta de por qué nadie podría encontrar un contrajemplo no está escrita en el texto, escríbanla.

### Usen expresiones estándar.

Es más fácil transformar la demo en un lenguaje tipo Lean si lo que escribieron no tiene ambigüedades, y para eso está bueno usar expresiones que ya se sabe qué significan.

[3]Resulta que a de Bruijn se le ocurrió primero pensar en la comparación entre escribir una demo en lenguaje natural y en un lenguaje formal, y parece que más o menos para cualquier demostración se mantiene constante la relación; es decir, no es que demostraciones informales más largas ahorran cada vez más cosas, sino que la cantidad de líneas que se ahorran por cada línea informal escrita es constante y más o menos 4, dependiendo del área. A esto se lo llama el *factor de de Bruijn* y hay gente que lo analizó.

**Pregunta:** *Qué expresiones comunes conocen?*

- "Sea $x$ un natural..."
- "Existe un conjunto $S$ tal que..."
- "Todo $y$ en $S$ cumple que..."
- "Si ..., entonces..."
- "Tal cosa pasa si y solo si pasa tal otra."
- etc.

**Separen sus demos en lemitas.**

Agarren una afirmación que no estén seguros si está suficientemente justificada, y escriban un Lema que sea sólo esa afirmación. Pónganle la mínima cantidad de hipótesis que necesite la afirmación para ser cierta. Demuestren ese lemita por separado. Es más fácil entender una demo si está dividida en afirmaciones. Además, ayuda a identificar si las hipótesis que tenés son suficientes para demostrar la afirmación, o si necesitás algo más.

**Lean muchas demos bien escritas del área.**

Por ej, el Cormen en el caso de Algo 3.

**Muestren sus demos a sus compañeros y a los profes.**

Al ser medio subjetivo cuándo una demo es suficientemente formal, cuantas más opiniones tengan más seguros van a estar de que está bien lo que escribieron.

**Lean sus demos de nuevo, y corrijan los errores que tuvieron.**

Siempre que escribo una demo para un artículo, la versión que envío al journal es como la 5ta versión a veces, nunca la primera. No tengan miedo de reescribir todo de nuevo, total las ideas no se pierden, ya las tienen en la cabeza.

**Escriban bien, loco.**

Es muy difícil leer una demo con frases gigantes con comas en cualquier lado, o con caligrafía que parece hecha con el pie. El que los corrige va a tener super mal humor si tiene que leer una cosa así, y va a tener cero ganas de corregirla bien.

**Ejercicio:** *Agarren la demo que encontraron antes que tenía el error y escríbanla de cero. Hagan muchas versiones, y hagan que cada versión sea más formal que la anterior, hasta llegar a usar sólo axiomas, teoremas y definiciones que sepan que son verdaderos. Si en un momento llegan a que no pueden formalizarla más, consulten con un compañero y/o conmigo.*

## Biblografía útil

Hay un libro de demostraciones que escribió Fede Lebrón (para los que no lo conocen, ayuda mucho en el canal de Telegram de Algo3) que está lleno de demos y tiene muchísimos tips de qué hacer para demostrar en lenguaje natural. Otros libros en inglés que encontré que ayudan con demostraciones son https://users.metu.edu.tr/serge/courses/111-2011/textbook-math111.pdf y https://longformmath.

com/proofs-book/. Y siempre está bueno leer la bibliografía recomendada del curso que estén haciendo.

Para otra forma de escribir demostraciones formales más cercana a lenguaje natural, pueden ver si les gusta https://github.com/PatrickMassot/verbose-lean4 o https://impermeable.github.io/. Mención especial a ProofGrader que es lo que más se parece a estar escribiendo una demo en lenguaje natural.