Ideas de constructivismo y computabilidad

Mario Román

21 de septiembre de 2018

1. Constructivismo

El constructivismo en matemáticas tiene una larga y compleja historia filosófica (véase [oP] o [TVD14]). Nuestra perspectiva, sin embargo, será tan puramente práctica y matemática como sea posible, sin entrar en ningún debate de este tipo. En términos generales, podríamos decir que el constructivismo exige *evidencia positiva* de una proposición, mientras que la lógica clásica simplemente pide *falta de evidencia negativa.*

Supongamos que nos preguntan si existe un programa que termine en tiempo finito devolviendo 1 si la hipótesis de Riemann es cierta y 0 en otro caso. Una respuesta posible desde el punto de vista clásico es: "Sí, si la hipótesis es cierta, el programa es print(1), si no, el programa es print(0)". Pero esto no es lo que queremos, esta solución no está *construyendo* la evidencia que queríamos. Si la hipótesis es R, esta respuesta inesperada aparece cuando permitirmos analizar por casos $(R \vee \neg R)$ sin tener evidencia de ninguna de las dos. [Shu17]

El asumir ciegamente que $(A \vee \neg A)$ en general (o condiciones equivalentes) es la principal fuente de *no-constructivismo* en nuestras matemáticas. Lo que demostramos sin asumir esa disyunción es en general constructible. En cualquier caso, si retiramos un axioma, las matemáticas que obtenemos son necesariamente menos potentes pero más generales.

1.1. La ley del tercio excluso

De forma simplificada, la matemática constructiva es la que se hace sin asumir el **tercio excluso**, es decir, sin asumir que para cualquier proposición A se tiene

que $(A \vee \neg A)$. Esto hace que no podamos escribir demostraciones por reducción al absurdo.

Definición 1.1.1 (Absurdo y negación)**.** Definimos $\bot$ como la proposición falsa. De una proposición falsa se sigue cualquier cosa. Es decir, para cualquier proposición B, se tiene $\bot \Rightarrow B$. Definimos $\neg A$, la negación de una proposición A, como $\neg A := (A \Rightarrow \bot)$.

Podríamos pensar que al no asumir el llamado *tercio excluso*, estamos abriendo la puerta a otros valores de verdad; pero eso no es cierto. Tenemos los dos valores de verdad usuales, simplemente no asumimos tener una demostración de ninguno de ellos.

Teorema 1.1.2. *No existe ninguna proposición Q que no sea verdadera y que no sea falsa a la vez.*

Demostración. Si no fuera verdadera, tendríamos $\neg Q$; pero si además no fuera falsa tendríamos $\neg\neg Q = (\neg Q \Rightarrow \bot)$, y uniendo ambas, llegaríamos a contradicción, $\bot$. Hemos demostrado que $(\neg Q \wedge \neg\neg Q) \Rightarrow \bot$. □

Aquí hay una diferenciación importante [Bau17]. En matemáticas se suele llamar *reducción al absurdo* a dos cosas distintas. Una es una verdadera demostración por contradicción y está prohibida en matemática constructiva,

- *supongamos $\neg P$, …, llegamos a contradicción $\bot$, luego P;*

pero la otra es simplemente la definición de la negación y es perfectamente válida constructivamente,

- *supongamos P, …, llegamos a contradicción $\bot$, luego $\neg P$.*

Si parecen iguales es porque tenemos la costumbre de aplicar la regla de la doble negación, $\neg\neg A = A$, casi sin pensar; ¡pero la regla de la doble negación no es cierta en matemática constructiva! porque equivale a la ley del tercio excluso.

Teorema 1.1.3. *Se tiene $\neg\neg A = A$ para cualquier proposición A si y sólo si se tiene $(A \vee \neg A)$ cualquier proposición A. De hecho, en general se tiene que $\neg\neg(A \vee \neg A)$.*

Demostración. Primero nótese que $A \Rightarrow \neg\neg A$ trivialmente. Pero además, si sabemos que $(A \vee \neg A)$, podemos demostrar $\neg\neg A \Rightarrow A$ partiendo en dos casos

- si A, entonces hemos terminado;
- si $\neg A$, entonces, como $\neg\neg A$, llegamos a $\bot$, pero de aquí se sigue cualquier cosa, en particular, se sigue que A.

Para la conversa, simplemente probaremos $\neg\neg(A \vee \neg A)$. Para ello asumimos $\neg(A \vee \neg A)$ y llegamos a contradicción. Esto lo haremos probando $\neg A$ bajo la asunción $\neg(A \vee \neg A)$, entonces tendremos que $A \vee \neg A$, simplemente porque ya tenemos $\neg A$, y habremos llegado a contradicción.

Para probar $\neg A$ bajo la asunción $\neg(A \vee \neg A)$, simplemente vemos que si se tuviera A, se tendría $A \vee \neg A$ y se llegaría a contradicción. □

Muchos de los ejemplos comunes de reducción al absurdo son realmente demostraciones de una negación. Y no los perdemos cuando trabajamos sin tercio excluso.

Proposición 1.1.4. *La raíz de* 2 *es irracional.*

Demostración. Irracional significa *no racional*. Supongamos que fuera racional, entonces sería de la forma $\sqrt{2} = a/b$, para algunos a y b coprimos. Pero entonces $a^2 = 2b^2$, con lo que $4 \mid a$ y $2 \mid b$, llegando a contradicción. □

1.2. Realizabilidad y la interpretación BHK

Para ganar intuición sobre cómo funcionan las matemáticas una vez que excluímos algunos principios clásicos, recurriremos a la interpretación informal de Brower-Heyting-Kolmogorov, que tiene su formalización en la teoría de realizabilidad.

Establecemos una relación de realizabilidad $p \Vdash \varphi$. Y la leemos diciendo que p, que podría ser un programa o una secuencia de bits, **realiza** o *representa* a φ (o *presenta evidencia de*), que podría ser una fórmula o un objeto matemático. Establecemos ciertas reglas y operaciones entre realizadores que gobiernan cómo interpretan la lógica. [Bau13]

- $\langle p, q\rangle \Vdash \phi \wedge \psi$, cuando $p \Vdash \phi$ y $q \Vdash \psi$;
- $\langle p, q\rangle \Vdash \phi \vee \psi$, cuando p representa un bit que nos dice si vamos a demostrar ϕ o si vamos a demostrar ψ y q realiza al elegido;
- $p \Vdash \phi \Rightarrow \psi$, cuando p es un programa que toma como entrada un $s \Vdash \phi$ y devuelve un $p(s) \Vdash \psi$;
- $p \Vdash \forall x \in A, \phi(x)$, cuando p es un programa que toma como entrada $s \Vdash a \in A$ y devuelve $p(s) \Vdash \phi(a)$;
- $\langle p, q\rangle \Vdash \exists x \in A, \phi(x)$, cuando $p \Vdash a \in A$ y $q \Vdash \phi(a)$;
- $p \Vdash \bot$ en ningún caso;
- $p \Vdash \top$ en todos los casos.

Estos conceptos son claros en los asistentes de demostraciones y en las teorías de tipos. Podemos sin embargo imaginarnos cómo serían los realizadores de ciertos principios. Como primer ejemplo, sea el principio de conmutatividad de la conjunción $\phi \wedge \psi \Rightarrow \psi \wedge \phi$; su realización podría ser un programa que toma una tupla (a, b) como entrada y devuelve la tupla (b, a), es decir, que *intercambia*.

O, por ejemplo, el principio de inducción de los números naturales

$$\Big(\phi(0) \wedge (\forall k \in \mathbb{N}\colon \phi(k) \Rightarrow \phi(k+1))\Big) \Rightarrow \forall n \in \mathbb{N}\colon \phi(n)$$

es realizado por un programa que implementa recusión primitiva. En Python, por ejemplo, si escogemos los naturales de Python como realizadores de los naturales, sería el siguiente.

```python
def recursion (zero, succ, n):
  if n == 0: return zero
  return succ(recursion(zero,succ,n-1))
```

En un lenguaje fuertemente tipado, sin embargo, es más fácil llevar la cuenta de qué estamos representando en cada función.

```haskell
recursion :: a -> (a -> a) -> Integer -> a
recursion z s 0 = z
recursion z s n = s (recursion z s (n-1))
```

Todavía mejor sería si los tipos fueran suficientemente fuertes como para asegurarnos que los realizadores que construimos son válidos. Agda es un asistente de demostración con tipos dependientes.

```
recursion : {ϕ : Nat → Set}
  → ϕ 0
  → ({k : Nat} → ϕ k → ϕ (succ k))
  → (n : Nat) → ϕ n
recursion {ϕ} z s zero = z
recursion {ϕ} z s (succ n) = s (recursion {ϕ} z s n)
```

La interpretación nos sirve para justificar informalmente por qué no es posible asumir tercio excluso en general. Supongamos que queremos demostrar el siguiente enunciado buscando un programa que lo realice.

$$\forall x \in \mathbb{R}\colon (x = 0) \vee (x \neq 0)$$

Aquí asumiríamos que los reales son la compleción de Cauchy de los números racionales. Esto quiere decir que un programa realiza el real $r \in \mathbb{R}$ si toma como entrada un natural $k \in \mathbb{N}$ y devuelve en su salida un racional $q \in \mathbb{Q}$ tal que $|r - q| < 2^{-k}$. Un programa que realizara el enunciado anterior debería ser un programa que tomara como entrada el programa que representa a un real y que devolviera en su salida un programa que realizara $x = 0$ o un programa que realizara $x \neq 0$. ¡Pero esto no parece posible! Necesitaríamos que nuestro programa supiera si x es efectivamente 0 pero la única información que tenemos de él, aunque nos permite aproximarlo tanto como queramos, no nos permite comprobar que sea 0.

Esto tampoco quiere decir que todas las instancias del tercio excluso sean falsas, por ejemplo, realizar $\forall n \in \mathbb{N}\colon (n = 0) \vee (n \neq 0)$ es sencillo.

1.3. Axioma de elección. Teorema de Diaconescu

Teorema 1.3.1 (Diaconescu)**.** *El axioma de elección implica el tercio excluso.*

Demostración. Sea P una proposición arbitraria. Consideramos los conjuntos $A = \{x \in \{0,1\} \mid P \vee (x = 0)\}$ y $B = \{x \in \{0,1\} \mid P \vee (x = 1)\}$, que son claramente no vacíos.

Por axioma de elección, existe una función $f\colon \{A, B\} \to \{0,1\}$ tal que $f(A) \in A$ y $f(B) \in B$. Como la igualdad entre números naturales sí es decidible, podemos dividir en casos.

- Si $f(A) = 1$, debemos tener P.
- Si $f(B) = 0$, debemos tener P.
- Si $f(A) = 0$ y $f(B) = 1$; suponemos que P fuera cierto y entonces tenemos $A = \{0,1\} = B$, luego $0 = f(A) = f(B) = 1$, llegando a contradicción. Habríamos probado $\neg P$. □

1.4. Las matemáticas constructivas

El no asumir el tercio excluso nos da libertad axiomática. Podemos tomar como axiomas proposiciones que contravendrían el tercio excluso en condiciones normales pero que aquí nos permiten desarrollar axiomatizaciones de la matemática perfectamente válidas. Hay modelos de la matemática constructiva donde todas las funciones son continuas, y modelos en los que todas las funciones son computables. Todas las categorías con cierta estructura (de topos elemental, y podemos

asumir existencianúmeros naturales o extensionalidad) sirven como modelos para este tipo de matemáticas.

Por otra parte, muchas de las intuiciones que teníamos como obvias en la matemática clásica empiezan a fallar. El hecho de que el subconjunto de un conjunto finito sea finito equivale al tercio excluso.

Proposición 1.4.1. *Dada una proposición no decidible P, el conjunto $A = \{0 \mid P \text{ es cierto}\}$ no puede probarse finito, pero puede probarse que no es infinito. El conjunto A es $\neg\neg$-finito.*

Demostración. Para demostrar que es finito necesitaríamos una biyección con un conjunto de la forma $\{0, ..., n-1\}$, pero esa biyección nos serviría para decidir P.

Por otro lado, supongamos que fuera infinito. Entonces no podría ser vacío, luego no podríamos tener $\neg P$; pero tampoco podría ser de cardinalidad 1, luego no podríamos tener P. Hemos obtenido que $\neg\neg P \wedge \neg P$, contradicción. □

Y no es lo único que no se comporta como esperamos.

- No podemos probar que **cada espacio vectorial tiene una base**.
- El **teorema de Bolzano** tampoco puede probarse. Podremos de todas formas probar versiones que intentan construir el cero aproximándolo, pero necesitamos hipótesis adicionales.
- El **teorema de Tychonoff** o el **lema de Zorn** son equivalentes al axioma de elección, no podemos esperar probarlos en este contexto.
- Existe una función $f\colon [0,1] \to \mathbb{R}$ **no acotada**.
- Es un problema abierto determinar si existe una inyección $\mathbb{R} \to \mathbb{N}$.

2. Geometría diferencial sintética

Cuando hacemos matemática aplicada, toda función tiene derivada (suele decirse "asumo lo que haga falta para derivar"); y las derivadas suelen calcularse usando infinitesimales.

$$(x^2)' = \frac{(x+dx)^2 - x^2}{dx} = 2x + dx \overset{?!}{=} 2x.$$

Pero si en el último paso decimos simplemente que "podemos eliminar dx porque es infinitesimalmente pequeño", ¿por qué no eliminarlo directamente al principio?

$$(x^2)' = \frac{(x+dx)^2 - x^2}{dx} \stackrel{?!}{=} \frac{x^2 - x^2}{dx} = 0.$$

Si queremos usar infinitesimales de una manera formal, tendremos que dar un planteamiento que evite todos estos problemas. Existen en matemáticas varias formas de trabajar con los infinitesimales evitando paradojas; en nuestro caso, usaremos la geometría diferencial sintética de Kock-Lawvere [Koc06].

2.1. Microafinidad

Empezamos definiendo los **infinitesimales** (de grado 2) como los números cuyo cuadrado es 0, es decir, los elementos del conjunto $D = \{d \in R \mid d^2 = 0\}$.

Axioma 2.1.1 (Axioma de microafinidad de Kock-Lawvere)**.** Toda función $g\colon D \to R$ es lineal de forma única.

Esto es lo que nos proporciona derivadas para cada $f\colon R \to R$. Dado un punto $x \in R$, la función $g(d) = f(x+d)$ debe ser lineal, y además sabemos que $g(0) = f(x)$, por lo que existe un único número $f'(x)$ al que llamamos *derivada de f en el punto x* cumpliendo que

$$f(x+d) = f(x) + f'(x)d.$$

Así, **toda función tiene derivada**, y tomando δ infinitesimal demostramos que también es derivada en el sentido $\varepsilon - \delta$.

Pero claro, estamos obviando un problema importante; dado $d \in D$, no es muy difícil deducir que $d^2 = 0$ implica $d = 0$ y que por tanto, $D = \{0\}$. Todavía peor, la función $f(x) = x$ tendría como derivadas a 0 y a 1 a la vez, $f(x+0) = f(x) + 1 \cdot 0 = f(x) + 0 \cdot 0$, obteniendo $0 = 1$. La solución es debilitar la lógica; esta contradicción no se puede alcanzar si no asumimos el tercio excluso.

Para todo lo demás, podemos asumir que R sigue siendo un cuerpo, en el sentido de que

$$(x \neq 0) \implies x \text{ tiene inversa.}$$

Esto nos prohíbe dividir por infinitesimales particulares, ya que no podemos demostrar que sean distintos de 0. Sin embargo, podremos dividir por infinitesimales cuando estén cuantificados universalmente.

Teorema 2.1.2 (Ley de cancelación)**.** *Si tenemos $ad = bd$ para todo $d \in D$, entonces $a = b$.*

Demostración. Tomamos $f(x) = ax - bx$, y tenemos que $f(d) = (a - b)d = 0d$. Como la derivada es única, $a - b = 0$. □

2.2. Ejemplo: derivación con infinitesimales

Ahora somos capaces de formalizar el ejemplo inicial. Si llamamos $f(x) = x^2$ tendremos que para cualquier infinitesimal d se tiene la siguiente igualdad.

$$f'(x)d = f(x + d) - f(x) = x^2 + 2xd - x^2 = 2xd.$$

Por lo que, por ley de cancelación, $f'(x) = 2x$.

3. Omnisciencia en espacios infinitos

En esta sección construiremos un espacio que permite una inyección desde los números naturales pero con la propiedad de que podemos encontrar en tiempo finito ejemplos de cada propiedad o demostraciones de que no existe ningún ejemplo. Las propiedades estarán limitadas por la estructura de este espacio: nótese que sería imposible tener esta misma propiedad para los números naturales. Escribiremos una implementación en Haskell.

```
forsome (\n -> 2 * n ^ 3 == 245 + n)      -- true
forsome (\n -> n * n == 28)               -- false
epsilon (\n -> n * n + 4 * n == 32)       -- 4
```

3.1. Omnisciencia

Definición 3.1.1 (Omnisciencia)**.** Un conjunto X es **omnisciente** si para cualquier proposición booleana $p\colon X \to 2$, podemos o encontrar un $x \in X$ tal que $p(x) = \mathsf{true}$ o podemos encontrar una demostración de para cualquier $x \in X$ se tiene que $p(x) = \mathsf{false}$.

En matemática clásica todo conjunto es omnisciente en virtud del tercio excluso, así que lo que nos interesa es estudiarlo bajo una interpretación constructivista. El propósito de esta sección es entonces el de buscar espacios cumpliendo la

siguiente sentencia.

$$\forall p \in 2^X : \Big(\exists x \in X : p(x) = \mathsf{true}\Big) \vee \Big(\forall x \in X : p(x) = \mathsf{false}\Big)$$

Esto lo haremos construyendo una **función de búsqueda** $\varepsilon : (X \to 2) \to X$ tal que $p(\varepsilon(p)) = 1$ determinará que hemos encontrado un ejemplo y tal que $p(\varepsilon(p)) = 0$ implica que no hay ningún ejemplo.

Proposición 3.1.2. *El espacio de los números naturales $\mathbb{N}$ no es omnisciente.*

Demostración. Si fuera omnisciente podríamos resolver el problema de la parada. Dada una máquina de Turing M construiríamos $p(n)$ como el programa que simula n pasos de computación de M y devuelve un booleano que indica si *no* ha terminado. □

Podría parecer por esta proposición que ningún espacio infinito va a ser omnisciente: al fin y al cabo, esperamos que cualquier espacio infinito 'como conjunto' sea más grande que los números naturales. Lo interesante es notar que, en matemática constructiva, los conjuntos tienen una estructura que no queda capturada por su cardinalidad. Encontraremos incluso conjuntos omniscientes no numerables.

3.2. El espacio de Cantor

Definición 3.2.1. El **espacio de Cantor** es $2^{\mathbb{N}}$, estando formado por las secuencias binarias.

Nótese que el espacio de Cantor es trivialmente un conjunto no numerable. Esto puede causar confusión cuando lo vemos desde fuera del sistema: si el espacio de Cantor está dado sólo por programas que toman un natural y devuelven un booleano ¿cómo puede no ser numerable si estos programas lo son? Pero esto es simplemente una observación que hacemos desde fuera del sistema formal en el que estamos trabajando. El hecho es que no podemos encontrar funciones del sistema formal $\mathbb{N} \to 2^{\mathbb{N}}$ que sean sobreyectivas, el argumento clásico de diagonalización de Cantor sirve en este caso.

Si realizamos el hecho de que es omnisciente, además de poder encontrar ejemplos de propiedades en general, ganaríamos la posibilidad de decidir igualdades de funciones en un conjunto infinito. Supongamos que tenemos un conjunto B con **igualdad decidible**, es decir,

$$\forall x, y \in B : (x = y) \vee \neg(x = y),$$

entonces tendremos igualdad decidible también en $2^{\mathbb{N}} \to B$, y un programa será capaz de determinar si para cualesquiera dos funciones $f, g \colon 2^{\mathbb{N}} \to B$ se tiene $(f = g) \vee \neg(f = g)$. Planteamos ejemplos de los dos fenómenos en el siguiente código.

```
n b = if b then 1 else 0 -- Auxiliar Bool -> Integer
-- Ejemplos:
forsome (\f -> n(f 1) + n(f 2) + n(f 3) == 4)
-- False
w1 = (\f -> f ( n(f 2) * n(f 4) + n(f 3) * n(f 4)))
w2 = (\g -> g ((n(g 3) + n(g 2)) * n(g 4)))
w1 == w2
-- True
v1 = ( \g -> let ng = n . g in ng(2*ng 0 + 3*ng 2 + 2*ng 1) )
v2 = ( \g -> let ng = n . g in ng(2*ng 0 + 3*ng 2 + 2*ng 2) )
v1 == v2
-- False
```

Demostraremos que es omnisciente construyendo un programa que realice la función de búsqueda $\varepsilon \colon (2^{\mathbb{N}} \to 2) \to 2^{\mathbb{N}}$. Usaremos en este caso Haskell. Lo interesante en este código será demostrar que siempre debe terminar; entender el papel que juega la evaluación perezosa del lenguaje es crucial, así como el hecho de que cada proposición sobre el espacio debe poder calcularse en tiempo finito y por tanto sólo puede consumir una cantidad finita de información de cada secuencia.

```
{-# LANGUAGE FlexibleInstances #-}

-- Empezamos definiendo el espacio de Cantor. Incluimos una función
-- auxiliar que añade un elemento al inicio de la secuencia.
type Cantor = Integer -> Bool

(#) :: Bool -> Cantor -> Cantor
(b # f) 0 = b
(b # f) n = f (n-1)

-- Usaremos una definición de epsilon que compondrá una inducción
-- mutua con otra función que comprueban si existen ejemplos
-- en cada rama del árbol binario.

-- Esta definición de epsilon es debida a Ulrich Berger.
epsilon :: (Cantor -> Bool) -> Cantor
epsilon p =
  if forsome (\a -> p (False # a))
```

```
    then False # epsilon (\a -> p (False # a))
    else True  # epsilon (\a -> p (True  # a))

forsome :: (Cantor -> Bool) -> Bool
forsome p = p (epsilon p)

forevery :: (Cantor -> Bool) -> Bool
forevery p = not (forsome (not . p))

-- Igualdad para funciones.
instance (Eq b) => Eq (Cantor -> b) where
  f == g = forevery (\u -> f u == g u)
```

Esta implementación presenta un problema: es extremadamente lenta. En el fichero `Omnsiciente.hs` se presenta otra versión de la función epsilon debida a Martín Escardó que es más rápida por un factor exponencial.

3.3. Los números conaturales

Los números naturales pueden caracterizarse en la categoría Set por su principio de inducción. La inducción nos permite, dado un conjunto A con un elemento y una función $f\colon A \to A$, definir una función de los naturales al conjunto, $\mathsf{rec}\colon \mathbb{N} \to A$, que cumpla $\mathsf{rec}(0) = a$ y $\mathsf{rec}(n+1) = f(\mathsf{rec}(n))$.

Esta misma condición se expresa categóricamente diciendo que los naturales son un álgebra inicial del funtor $(+1)$. Es decir, existe una única r haciendo conmutar al siguiente diagrama.

$$\begin{array}{ccc} 1+\mathbb{N} & \xrightarrow{1+\mathsf{rec}} & 1+A \\ {\scriptstyle \text{zero+succ}}\downarrow & & \downarrow{\scriptstyle a+f} \\ \mathbb{N} & \xrightarrow{\mathsf{rec}} & A \end{array}$$

Esta caracterización puede dualizarse para obtener los **números conaturales** $\mathbb{N}_\infty$ como la coálgebra final del funtor $(+1)$.

$$\begin{array}{ccc} 1+A & \xrightarrow{1+\mathsf{corec}} & 1+\mathbb{N}_\infty \\ {\scriptstyle f}\uparrow & & \uparrow{\scriptstyle \text{pred}} \\ A & \xrightarrow{\mathsf{corec}} & \mathbb{N}_\infty \end{array}$$

Esta construcción tiene la ventaja de que la hemos hecho en categorías y puede ser replicada en cualquier lenguaje que soporte estructuras coinductivas. Una cons-

trucción más intuitiva es tomar $\mathbb{N}_\infty$ como la compatificación de Alexandrov de $\mathbb{N}$: consideramos todos los naturales pero además un punto $\infty \in \mathbb{N}_\infty$, que puede verse como un número natural cuyo predecesor es él mismo. Esto es distinto de $\mathbb{N} \cup \{\infty\}$, que tendría en principio topología discreta.

Todavía podemos tomar otra construcción más fácil de implementar en la mayoría de lenguajes de programación. Podemos tomar $\mathbb{N}_\infty$ como el espacio de sucesiones binarias decrecientes.

$$\mathbb{N}_\infty = \left\{ x \in 2^{\mathbb{N}} \mid \forall i \in \mathbb{N}.\, x_i \geq x_{i+1} \right\}$$

Aquí representamos los naturales como $i(n) = 1^n 0^\omega$ y el punto de compactificación como $\infty = 1^\omega$. Estas son además las únicas sucesiones posibles.

Proposición 3.3.1. *El espacio de los números conaturales es omnisciente.*

Demostración. Usando la última representación que hemos descrito, podemos definir una función que encontrará un ejemplo si existe como sigue.

$$\varepsilon(p)(n) = \begin{cases} 0 & \text{si } \exists k \leq n \in \mathbb{N} \colon p(i(k)) = \mathsf{false} \\ 1 & \text{si } \forall k \leq n \in \mathbb{N} \colon p(i(k)) = \mathsf{true} \end{cases}$$

Nótese que está bien definida y termina para cualquier entrada.

Ahora podemos comprobar por casos. Si $\varepsilon(p) = i(n)$ para algún $n \in \mathbb{N}$, será porque $p(i(n)) = 0$ por definición y habremos terminado. Si $\varepsilon(p) = \infty$, tendremos forzosamente que $p(i(n)) = 1$ para todo $n \in \mathbb{N}$; así si existe algún ejemplo será precisamente ∞. □

Para la implementación en Haskell de este espacio vamos a usar una técnica distinta. Si intentamos definir los números naturales, la evaluación perezosa nos dará directamente los números conaturales: existirá el punto fijo de la función sucesor.

```
{-# LANGUAGE FlexibleInstances #-}

-- CONATURAL.
-- Una representación de los números conaturales.
data Conat = Zero | Succ Conat deriving (Eq, Show)

infinity :: Conat
infinity = Succ infinity

-- Hacerlos instancia de Num nos permitirá usar los enteros con
```

```haskell
-- notación usual.
instance Num Conat where
  Zero + y = y
  Succ x + y = Succ (x + y)
  Zero * y = Zero
  Succ x * y = y + (x * y)
  fromInteger 0 = Zero
  fromInteger n = Succ (fromInteger (n-1))


-- Búsqueda usando las mismas funciones auxiliares que en el
-- caso de los números conaturales.
epsilon :: (Conat -> Bool) -> Conat
epsilon p = if p Zero
  then Zero
  else Succ $ epsilon (p . Succ)

forsome :: (Conat -> Bool) -> Bool
forsome p = p (epsilon p)

forevery :: (Conat -> Bool) -> Bool
forevery p = not (forsome (not . p))

-- Igualdad para funciones.
instance (Eq b) => Eq (Conat -> b) where
  f == g = forevery (\u -> f u == g u)
```

3.4. Topología y computación

La diferencia entre $\mathbb{N}$, que no es omnisciente, y los espacios $\mathbb{N}_\infty$ y $2^{\mathbb{N}}$ reside en que estos últimos son compactos. ¿Cómo hemos acabado trabajando con topología? Podemos trazar una identificación entre computabilidad y continuidad, y cada conjunto en matemática constructiva tiene una estructura más allá de su cardinalidad que puede ser entendida como una estructura de espacio.

Informalmente, se pueden hacer las siguientes identificaciones.

Computación	Topología
Tipo de datos	Espacio
Elemento del tipo	Punto del espacio
Propiedad semidecidible	Conjunto abierto
Función computable	Función continua
forevery es semidecidible	Compacto
forsome es semidecidible	Disperso

Un tratamiento formal de este tema puede encontrarse en [Esc04].

Por ejemplo, $\mathbb{N}_\infty$ tiene la topología de la secuencia convergente genérica. El conjunto $\{\infty\}$ sería cerrado, pero no abierto; si traducimos esto, tenemos que es semidecidible ver si un conatural es finito (simplemente ir tomando su predecesor hasta llegar a 0) pero no decidible (el infinito no puede distinguirse de un número suficientemente grande).

3.5. Lema de Konig

El espacio de Cantor es compacto; esto es lo que le hace ser omnisciente. Pero no es computablemente compacto, en el sentido de que existe una secuencia computable de abiertos que lo cubren pero que no tiene ningún subrecubrimiento finito.

4. Teoría de tipos

Usaremos el lenguaje de programación Agda para probar teoremas en matemática constructiva y extraer realizadores de ellos.

4.1. Tipos básicos para la teoría de Martin-Löf

La proposición verdadera $\top$ la realiza el único elemento de su tipo. La proposición falsa no la realiza ningún elemento. Para realizar una conjunción necesitamos realizar sus dos factores, mientras que para realizar una disyunción necesitamos elegir uno de los dos y realizarlo.

```
-- Proposición verdadera.
record ⊤ : Set where
  constructor *
```

```
-- Proposición falsa.
data ⊥ : Set where

-- Conjunción.
record ∧ (A B : Set) : Set where
  constructor ,
  field
    fst : A
    snd : B

-- Disyunción.
data ∨ (A B : Set) : Set where
  inl : A → A ∨ B
  inr : B → A ∨ B
```

Con estos primeros tipos de datos ya somos capaces de probar la conmutatividad de la conjunción. Agda es capaz de extraer un programa de la demostración.

```
swap : {A B : Set} → A ∧ B → B ∧ A
swap (a , b) = b , a
```

Para completar estos tipos, notamos que los cuantificadores universales están directamente implementados en Agda, así como un tipo `Set` que sirve como universo de tipos. La función `swap` es un ejemplo de ambos fenómenos usamos `(A B : Set)` para cuantificar universalmente sobre cualesquiera dos tipos del lenguaje. Construimos ahora el cuantificador existencial $\mathbf{\Sigma}$.

```
record Σ (A : Set) (B : A → Set) : Set where
  constructor **
  field
    fst : A
    snd : B fst
```

4.2. La igualdad

Lo único que nos falta para poder hacer matemáticas es el concepto de igualdad, y poder construir programas que realicen una igualdad. Nuestra igualdad vendrá definida por el **eliminador J**, que de forma simplificada es la regla que dice que: dada una proposición dependiendo en dos argumentos del mismo tipo $C(-,-)$, si podemos probar $C(x,x)$ para cada x, entonces tenemos $C(a,b)$ para cada dos elementos iguales $a = b$.

Lo interesante de esta regla, junto a la reflexividad, es que captura la mayoría de propiedades de la igualdad. Implica la transitividad, la simetría y el hecho de que toda función respeta la igualdad.

```
data ≡ {A : Set} (a : A) : A → Set where
  refl : a ≡ a

symm : {A : Set}{a b : A} → a ≡ b → b ≡ a
symm refl = refl

trans : {A : Set}{a b c : A} → a ≡ b → b ≡ c → a ≡ c
trans refl q = q

ap : {A B : Set}{a b : A} → (f : A → B) → a ≡ b → f a ≡ f b
ap f refl = refl
```

Podríamos pensar que la única instancia de la igualdad es realmente la reflexividad. Es decir, que $\forall p \in (a = a) \colon p = \mathsf{refl}$; y podríamos pensar que es otra consecuencia del eliminador J, pero no es así. De hecho existe un modelo de la teoría de Martin-Löf en grupoides que demuestra que es independiente de la teoría. Si interpretamos los grupoides como espacios, lo que estaríamos intentando demostrar es que todo espacio (dado por un tipo) es simplemente conexo; y el modelo demuestra que no tiene por qué ser el caso. Si asuimos que la única instancia de la igualdad es la reflexividad trabajaremos sólo con tipos que representen espacios conexos.

4.3. Teoremas en Agda

Como ejemplo, probamos la conmutatividad de la suma de números naturales.

```
data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ

infixl 30 +
+ : ℕ → ℕ → ℕ
zero + b = b
succ a + b = succ (a + b)

+rzero : ∀ n → n + zero ≡ n
+rzero zero = refl
+rzero (succ n) = ap succ (+rzero n)

+rsucc : ∀ n m → n + succ m ≡ succ (n + m)
+rsucc zero m = refl
+rsucc (succ n) m = ap succ (+rsucc n m)

+comm : ∀ n m → n + m ≡ m + n
+comm zero m rewrite +rzero m = refl
+comm (succ n) m rewrite +rsucc m n = ap succ (+comm n m)
```

Referencias

[Bau13] Andrej Bauer. Intuitionistic mathematics and realizability in the physical world. In *A Computable Universe: Understanding and Exploring Nature as Computation*, pages 143–157. World Scientific, 2013.

[Bau17] Andrej Bauer. Five stages of accepting constructive mathematics. *Bulletin of the American Mathematical Society*, 54(3):481–498, 2017.

[Esc04] Martín Escardó. Synthetic topology: of data types and classical spaces. *Electronic Notes in Theoretical Computer Science*, 87:21–156, 2004.

[Koc06] Anders Kock. *Synthetic differential geometry*, volume 333. Cambridge University Press, 2006.

[oP] Stanford Encyclopedia of Philosophy. Intuitionism in the Philosophy of Mathematics.

[Shu17] Michael Shulman. The weird and wonderful world of constructive mathematics. *Mathcamp*, 2017.

[TVD14] Anne Sjerp Troelstra and Dirk Van Dalen. *Constructivism in mathematics*, volume 2. Elsevier, 2014.