Información y Azar*

Gregory Chaitin

La primera cuestión es qué interés tiene este campo de información y azar, que yo llamo Teoría Algorítmica de la Información, *TAI.* ¿Por qué puede interesar?

La razón por la que este campo me interesó a mí es que quise comprender a fondo el Teorema de Incompletitud de Gödel. La prueba normal, la original de Gödel, no me gustaba y pensé que quizás tenía algo que ver con el azar, un dato que viene de la física.

Y esta fue la pista que me condujo a todo esto y ya hace 35 años que estoy trabajando en este campo. Pero sigo tratando de comprenderlo cada vez con otro enfoque y de una forma más profunda. ¡A esto he dedicado mi vida!

Esto desde el punto de vista personal. Desde el punto de vista intelectual el teorema de Gödel es muy sorprendente porque demuestra que la razón tiene límites, y lo hace de una forma muy paradójica creando un enunciado que dice de sí mismo que es falso; o que no es demostrable, para ser más preciso. Y esto fue muy sorprendente en el año 1931, cuando Gödel lo hizo por primera vez.

Pero con el campo de la computabilidad, con este maravilloso trabajo de Turing de 1936, que habla de la máquina de Turing y computabilidad, y números reales computables y no computables –introduciendo estos nuevos conceptos, la incompletitud se encuentra de una forma mucho más directa, no parece tan extraña, porque Turing demuestra que no se puede saber si un programa se va a detener eventualmente o nunca.

Y esto en seguida da algo que escapa al poder del método axiomático. No siempre se puede decidir si un programa se detiene, no hay algoritmos para determinar en general si un programa se va a detener y tampoco puede haber sistemas de axiomas y reglas de razonamiento que permitan deducirlo. Esto lo dice Turing, lo demuestra Turing en su trabajo de 1936.

Así que éste ya fue un paso enorme y la incompletitud ahora parece algo mucho más real que cuando Gödel la descubrió.

Mi próximo paso fue crear esta teoría que habla de la longitud de los programas, del tamaño de los programas, de complejidad de información, de cantidad

*Este artículo está basado en la primera clase de un curso que dictó el autor en la Universidad de Buenos Aires en Octubre de 1998. La transcripción de esa primera clase y posterior revisión y edición de este artículo fueron elaboradas por Verónica Becher

de bits. Es una teoría del largo del programa, del tamaño del programa, no del tiempo de su corrida; no importa si es un programa rápido o no. Se mide la complejidad H de una tarea de cómputo por la cantidad de información que es necesario tener en el programa, la mínima cantidad de información que se requiere.

Ahora bien, cuando ya se empieza a trabajar con este concepto de complejidad de información, de longitud de programas, la incompletitud se encuentra en todas partes. La incompletitud es ineludible. No hay forma de escapar porque no importa la dirección en la cual uno va, choca contra esa pared en seguida. Es decir que con casi todos los problemas que surjen de inmediato en este campo de la complejidad medida por el tamaño de los programas, resulta que no hay un algoritmo para contestar la pregunta y tampoco se puede deducir la respuesta a partir de ningún conjunto finito de axiomas. No sólo no hay un algoritmo sino que ningún sistema de razonamiento puede resolver estas preguntas. Además esto se cumple de la peor forma posible, porque muchas veces la única forma de resolver estas preguntas es agregando las respuestas como nuevos axiomas, algo que a los matemáticos no les agrada para nada.

¿Por qué no? Los físicos dicen que algo es cierto porque tienen pruebas empíricas; ellos toman hechos experimentales individuales como axiomas nuevos. Bueno, quién va a pensar que la matemática es así. ¿Por qué da más seguridad la matemática? Porque comprime muchos teoremas matemáticos en una cantidad pequeña de axiomas. Los axiomas siempre son inciertos. No se sabe si son ciertos, aunque Euclides habla de axiomas autoevidentes, pero francamente, no creo en axiomas evidentes de por sí.

Siempre hay un riesgo, pero cuando hay menos axiomas hay menos riesgo, porque si estoy partiendo de muchos axiomas hay muchas más posibilidades de que me equivoque. De modo que es porque se comprimen muchos resultados en pocas hipótesis, que son los axiomas, que una teoría es buena. El grado de compresión de muchos casos partiendo de pocos axiomas, de pocos principios, es la razón de ser de la matemática y la razón por la cual da más confianza. Pero en mi teoría se encuentran hechos matemáticos que no se pueden comprimir para nada, que no se pueden reducir a hipótesis más simples que ellos mismos. En otras palabras, aquí el razonamiento no sirve para nada porque no comprime nada, no nos da más confianza, no puede darnos más confianza. Estamos en un caso extremo.

A continuación veamos un poco de la historia de este campo, desde mi punto de vista, es decir, visto por uno de los participantes, desde las trincheras de este campo.

Primero, esto comienza en la década del sesenta, y los iniciadores fuimos tres y lo hicimos en forma independiente:

$$\underline{TAI_{60}}$$
Ray Solomonoff

A. N. Kolmogorov
Greg Chaitin

Algunos de estos nombres parecen rusos pero a decir verdad el único que estaba en Rusia es Kolmogorov.

$\underline{TAI_{60}}$
Ray Solomonoff
A. N. Kolmogorov
Greg Chaitin

Kolmogorov era un matemático al final de una carrera de gran éxito. Ya en esa época era bastante mayor, mayor que yo ahora.

$\underline{TAI_{60}}$
Ray Solomonoff
A. N. Kolmogorov
Greg Chaitin

Solomonoff era un muchacho en esa época y no es un matemático, su especialidad es la Inteligencia Artificial. Es amigo de Marvin Minsky, quien es un gran nombre en Inteligencia Artificial.

$\underline{TAI_{60}}$
Ray Solomonoff
A. N. Kolmogorov
Greg Chaitin

Y yo era un estudiante. Había tenido la idea a los quince años de edad y estaba escribiendo mi primer trabajo sobre este tema cuando tenía dieciocho; se publicó cuando tenía diecinueve.

Todos nosotros propusimos independientemente la idea de medir el tamaño del programa como concepto importante.

Solomonoff lo propuso para Inteligencia Artificial, para predecir el futuro, para dar una forma precisa a la navaja de Occam, que dice que la teoría más simple es la mejor. Pero no lo logró del todo. Este trabajo de Solomonoff es hermoso, es muy estimulante, pero él no pudo encontrar, a mi juicio, la forma de llevarlo adelante. Además como él no era matemático, algunos detalles matemáticos están mal, pero a pesar de esto el trabajo es muy bueno.

Ahora bien, Kolmogorov y yo somos matemáticos, e hicimos un poco más que Solomonoff. Además de proponer que se mida el tamaño del programa en bits, Kolmogorov y yo propusimos una definición del azar basada en esto: Propusimos definir una cadena de bits, una cadena finita de unos y ceros, como aleatoria si no se la puede comprimir en un programa mucho más conciso que la cadena misma.

El azar es un concepto que no existe dentro del formalismo que se usa para la teoría de probabilidades ordinaria, en la cual trabajó mucho Kolmogorov. El concepto de una sucesión sin estructura realmente no existe dentro de esa teoría. Porque si se echa una moneda al aire, una moneda equilibrada y con echadas independientes, hay 2^n posibilidades y todas tienen igual probabilidad. Por lo tanto todas son igualmente "aleatorias":

0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
1 1 1 1 1 1 1 1 1 1
1 1 0 1 0 0 1 1 1 0

Pero intuitivamente la sucesión cara, cara, cara, cara, ... tiene estructura, y una sucesión en la cual todo está mezclado no tiene estructura. Pero este concepto no tiene cabida dentro de la teoría de probabilidades clásica.

Entonces Kolmogorov y yo propusimos esto como un concepto nuevo. Y también hay un muchacho sueco que estaba visitando a Kolmogorov en Moscú, que se llama

Per Martin-Löf

Yo tuve el gusto de encontrarme con Solomonoff y con Martin-Löf pero nunca con Kolmogorov. Por otra parte, ya falleció.

Per Martin-Löf era un joven estudiante de Kolmogorov. Venía de Estocolmo, estaba en Moscú, y yo me pregunto a veces si fue el joven, Martin-Löf, o el viejo, Kolmogorov, quien tuvo la idea original en Moscú.

Pero hay algunos problemas. La definición original de nosotros tres (Solomonoff, Kolmogorov y yo) de complejidad asigna a la mayoría de las cadenas de n bits la complejidad n. La mayoría son aleatorias, en el sentido que Kolmogorov y yo propusimos. Es decir que para la mayoría de las cadenas de n bits de largo, el programa más corto es casi del mismo tamaño que la cadena misma. Pero resulta que esa definición tiene algunos problemas muy serios cuando se trata de definir una cadena aleatoria infinita.

Una cadena que es infinita en una dirección, es como el desarrollo en binario de un número real

,01 ...

Y se puede considerar que algo así es un número real, si uno pone una coma adelante.

Entonces, definir aleatoriedad para una cadena binaria infinita (infinita en una dirección) es como definir un número real aleatorio, y resulta que hay algunos inconvenientes técnicos con la definición de complejidad original. Anda

muy bien para cadenas finitas pero para cadenas infinitas tiene algunos problemas.

Entonces Martin-Löf resolvió el problema diciendo "olvidemos la complejidad del largo del programa", y formuló una definición muy razonable de aleatoriedad de cadenas binarias infinitas que no tiene nada que ver con la complejidad sino que tiene que ver con la teoría de la medida hecha en forma constructiva.

Su definición es más o menos así: un número real es aleatorio si cumple con todas las propiedades constructivas de aleatoriedad. Y una propiedad constructiva de aleatoriedad es una prueba de aleatoriedad, y una prueba de aleatoriedad es un algoritmo que ... Bueno, tendría que darles más detalles acerca del concepto de aleatoriedad para números reales, lo cual no puedo hacer por razones de espacio. Pero ustedes se dan cuenta de que la definición de aleatoriedad de Martin-Löf no toma en cuenta la complejidad. La razón por la cual Martin-Löf la propuso es que Kolmogorov había propuesto, con optimismo, que en el caso de una cadena aleatoria infinita todas sus partes iniciales también tienen que ser aleatorias.

¡Pero esto no camina! Yo me di cuenta de que esto no camina porque resulta que muchas veces ocurre, cuando se trata de una cadena infinita, que hay una corrida larga de ceros sucesivos y la complejidad baja. Por lo tanto hay oscilaciones en la complejidad y a la fuerza tiene que haber bajadas bastante fuertes. Más precisamente, en el caso de una cadena infinita, con probabilidad uno infinitas veces tiene que haber una bajada de más o menos $\log_2 n$ bits en la complejidad de los primeros n bits. Porque a la fuerza, con probabilidad uno, tiene que haber corridas de este largo de "caras" si se juega a "caras" y "secas", que es lo mismo que ceros y unos.

Es decir, se demuestra en la teoría de probabilidades que en una sucesión infinita de echadas al aire independientes de una moneda equilibrada, infinitas veces los primeros n resultados tienen que terminar con más o menos logaritmo de n sucesivos resultados que son idénticos. Pero esto permite comprimir la sucesión, y la complejidad no es la mayor posible, porque no hace falta dar toda la parte repetida. Primero se indica la parte inicial, después se dice "y después sigue una cantidad determinada que es siempre igual" y esto permite una compresión.

No sé por qué Kolmogorov no se dió cuenta de esto; entonces Martin-Löf parece que se lo indicó. Yo sí me di cuenta y en lugar de exigir que sea aleatoria al máximo e incompresible la complejidad de cada parte inicial, le puse una cota más generosa, permitiendo que la complejidad baje un poco. Pero el problema con esta definición de aleatoriedad es que permite que cadenas que no son aleatorias cumplan con la definición de aleatoriedad. Martin-Löf resolvió este problema con su definición estadística de aleatoriedad pero abandonó el concepto de complejidad. Esa fue su contribución. Afortunadamente, resulta que cuando se corrige la definición de complejidad, todo anda bien con la definición

de aleatoriedad aún en el caso de cadenas infinitas. Este fue el próximo paso importante en el desarrollo de la teoría, el próximo paso histórico importante, diez años después, en la década del setenta, y lo di yo.

$$\underline{TAI_{70}}$$
Greg Chaitin

Solomonoff desaparece de la escena, a mi juicio, en este momento; Kolmogorov solamente tiene dos trabajos publicados en la década del sesenta, de cuatro páginas más o menos cada uno, pero tuvo discípulos que trabajaron en esto. Uno de estos discípulos también se dio cuenta de que había que cambiar la definición de complejidad. Pero yo creo que no lo hizo bien, que hizo solamente la mitad. Este discípulo, que después se fue de Rusia a los Estados Unidos, se llama Leonid Levin.

$$\underline{TAI_{70}}$$
Greg Chaitin
Leonid Levin

Nosotros nos dimos cuenta que había que cambiar la definición de complejidad, y el resultado es la definición con programas autodelimitantes. Es muy importante entender por qué.

Mi idea, para empezar, es que una definición de información o de complejidad tiene que ser aditiva, es decir que si la complejidad de algo es tantos bits, y la de otra cosa tantos bits, al tener las dos cosas, la suma de estos mismos bits tiene que alcanzar. Pero yo me di cuenta de que esto no es cierto con la definición original porque si tomo dos programas y los concateno, no sé dónde termina el primero y empieza el segundo. Tengo que agregar algo para indicar dónde hay que cortar. Con lo cual en la teoría TAI_{60} la complejidad tiene esta propiedad:

$$H(A,B) \leq H(A) + H(B) + \Delta$$

donde Δ es algo del orden del logaritmo de ambos lados. Pero en la nueva teoría, TAI_{70}, Δ queda como una constante. Además hay otra cosa importante que Leonid Levin nunca tomó en cuenta. Y lamentablemente en mi libro sobre los límites de la matemática[1] el capítulo que explica cómo programar mi teoría en LISP no lo menciona.

La definición de complejidad condicional o relativa también debe cambiarse, por lo siguiente: ¿Cuántos bits se requieren para computar algo si a uno le han dado gratis otras cosas? Esa definición, la evidente que se usó en TAI_{60}, está mal. Levin no se dio cuenta, pero yo sí. En TAI_{70} a uno no le dan algo gratis directamente, sino que le dan un programa de tamaño mínimo para calcularlo.

[1] G. Chaitin, The Limits of Mathematics, Springer-Verlag, 1998.

Esto es lo que yo propuse en la década del setenta. Desde este momento se la puede llamar realmente una teoría algorítmica de la información.

Después hubo más. Hace aproximadamente cinco años, cambié todo de nuevo. Rehice la teoría una vez más y esto lo explico en mi libro sobre los límites de la matemática. Porque hasta ahora esta teoría era bastante teórica, era una teoría del largo de programas pero no eran programas que uno puede correr en la computadora. Eran computadoras abstractas, eran programas teóricos para máquinas de Turing universales. Esto era muy teórico.

Yo me di cuenta de que se podía realmente programar todo, transformando la teoría en una sobre el largo de programas reales, que se pueden correr, y lo hice mediante una versión de LISP que inventé para tomar la teoría algorítmica de la información y ponerla a andar en la computadora. Desde el punto de vista matemático es la misma teoría de la complejidad, o sea $TAI_{70} = TAI_{90}$. La única diferencia es que ahora realmente uno puede tener los programas en la mano, uno los puede elaborar y después correrlos en una computadora, y entonces, si uno tiene el programa, puede medir su largo en bits y esto da una cota a la complejidad del resultado del programa.

Entonces la teoría se hace bastante más concreta. En otras palabras, además de ser algo teórico, se convierte en algo donde realmente uno puede correr los programas, o mejor dicho, lo hace la computadora de cada uno. Y esto lo explico en mi libro sobre los límites de la matemática.

Entonces hay estas tres etapas: TAI_{60}, TAI_{70}, TAI_{90}. Y en la teoría TAI_{70} ¿qué pasa con la complejidad de las cadenas de n bits de largo? Bueno, ahora su complejidad no va a ser de n bits, va a ser mayor. Más o menos va a ser normalmente este número de bits:

$$n + \log_2 n.$$

Es decir que en mi teoría un programa debe indicar dentro del programa mismo dónde termina, cuan largo es. Son programas autodelimitantes, es un poco como poner una cabecera a algo de largo variable indicando su largo. Pero no hace falta indicar directamente cuantos bits vienen a continuación. A veces uno puede ser más ingenioso y dar un programa para calcular el número de bits. Porque puede ser que el número de bits que vienen a continuación es un número compresible, no de máxima complejidad. En otras palabras, el resultado presentado en forma correcta es que las cadenas de n bits más complejas tienen una complejidad de n bits más la complejidad de n, o sea

$$n + H(n),$$

que normalmente es más o menos $n + \log_2 n$. Pero a veces, por ejemplo, si n es una potencia de 2, $H(n)$ va a ser mucho menor, va a ser del orden del logaritmo del logaritmo de n, en lugar de $\log_2 n$. Para números n de forma especial $H(n)$ podría ser mucho más chico que $\log_2 n$. Además en esta teoría se puede definir

un número que me encanta, que llamé Ω, que es la probabilidad de detención de una computadora universal, que realmente da una versión muy diferente y muy fuerte del teorema de Turing de que no se puede decidir si un programa se detiene. Porque cuando uno pregunta cuál es la probabilidad de detención de un programa escogido al azar, la respuesta es el número real Ω cuyos bits son realmente imposibles de conocer, son aleatorios, en el sentido de mi teoría, la *TAI*.

Otra cosa buena es que con la definición de complejidad en TAI_{70} la definición natural de una cadena aleatoria infinita, es decir, incompresible, resulta ser equivalente a la definición por pruebas estadísticas de Martin-Löf de un número real aleatorio. Es decir que con el enfoque de TAI_{70} se unifica la definición de aleatoriedad para cadenas finitas e infinitas, mientras que en TAI_{60} había que usar diferentes métodos en cada uno de estos casos.

Es por esto que creo que aunque la definición de Martin-Löf es hermosa, no va al fondo del caso, porque su definición usando pruebas estadísticas realmente se aplica solamente a cadenas infinitas. Pero usando la complejidad, se define algo incompresible o aleatorio en el caso finito e infinito a la vez. Creo que esto es importante subrayarlo.

Yo estoy convencido de que TAI_{70} es la formulación definitiva de la teoría, y TAI_{60} no lo es; fue un esfuerzo preliminar, y ahora creo que solamente tiene interés histórico. Hay algunas preguntitas que todavía quedan, pero realmente creo que habría que olvidar esta versión de la teoría. No sé si todo el mundo lo hace, pero yo lo hago.

Ahora bien, hay algunos temas adicionales que fueron estudiados y yo elaboré muchas teorías de complejidad del largo de programas además de estas que acabo de presentar. Empecé realmente en la década del sesenta con dos teorías del largo de programas de máquinas de Turing, donde mido el número de estados, tomo el número de estados como la medida de complejidad. Empecé con esto realmente, y además, para dos tipos diferentes de máquinas de Turing, uno con una definición un poco rebuscada que inventé especialmente, pero lo hice por una razón.

Así que tengo dos teorías del largo de programas para máquinas de Turing en las cuales no uso bits sino número de estados como la cosa medida.

Después hice algo así como tres teorías de complejidad del tamaño de programas para LISP (realmente hice cuatro), para distintas versiones de LISP. Aquí la idea es medir el largo de programas y tomar un idioma de programación que la gente realmente use. Yo tomé LISP porque LISP tiene la siguiente particularidad: LISP es a la vez algo elegante y teóricamente hermoso, pero además es un idioma de programación que realmente se puede usar.

Además de ser teórico, he trabajado para IBM como programador. Me encanta programar, y jugaba con LISP por muchos años sin darme cuenta de que se podía aplicar al desarrollo teórico de la *TAI*. LISP realmente está en estos

dos mundos: el mundo de la teoría y el mundo de la computación práctica. Y es el único idioma que conozco que está en los dos mundos.

Si uno tiene un idioma de programación real, la forma natural de medir el largo del programa es contar el número de caracteres que tiene, dar su largo en caracteres. Tengo algunos artículos sobre este tema, incluso hablo de esto en un capítulo de mi libro sobre los límites de la matemática, en el segundo capítulo, que se llama "Programas elegantes en LISP". Comienza en la primera mitad de este capítulo, donde digo que un programa elegante en LISP es el que tiene la propiedad de que ningún programa más pequeño da el mismo resultado. Y allí se mide el largo de un programa en caracteres de programación LISP.

Pero éstas no son las teorías correctas. Son una aproximación a la *TAI,* que es mucho más abstracta, y que mide la información en bits en lugar de caracteres. Estas teorías son más concretas porque se trata de modelos de computación que uno puede tener en la mano. A mí me interesaba ver qué pasa si uno trabaja con un modelo computacional más real. Y finalmente los combiné, porque mi version de LISP es una forma de combinar los idiomas de programación reales con toda la parte teórica.

Otra cosa que hice, que creo que no salió, tiene que ver con la teoría de la evolución y el origen de la vida. Tengo tres trabajos sobre este tema, tratando de usar el concepto de complejidad de la *TAI* en la biología. Creo que no lograron su objetivo, pero igual les puede interesar.

Existe ahora un campo que se llama vida artificial, *Artificial Life;* empiezan a verse algunos libros sobre este tema. En este campo hacen experimentos, trabajan con modelos que andan en la computadora, y a veces encuentran la forma de crear por evolución cosas interesantes en un tiempo razonable. Lo que yo traté de hacer es diferente, una teoría que habla de la complejidad de un organismo y que muestra que con altas probabilidades en ciertas circunstancias tienen que aparecer organismos e incrementarse su complejidad, donde la complejidad 0 sería la no existencia de organismos vivos.

Esa fue la idea. Yo quería una teoría matemática general de la evolución darwiniana, que no la logré y, que yo sepa, nadie la ha hecho, y lo ofrezco como un proyecto interesante. Probablemente la definición de complejidad de la *TAI* no es adecuada para esto, pero hay quienes todavía piensan que sí. Yo lo pensaba originalmente, pero no lo pienso ahora.

Otra cosa que habría que hacer, y yo empecé a hacerlo, es que esta teoría del largo de programas principalmente habla del largo de un programa que eventualmente se acaba, es decir, que realiza un cómputo finito, que calcula algo finito. Pero también es interesante el caso infinito, es decir programas que generan una salida infinita. Uso esto un poco en mi teoría, para su aplicación a los límites de la lógica. Porque un sistema lógico es en cierto sentido un cómputo que nunca termina, es un cómputo que va haciendo todas las deducciones posibles a partir de los axiomas. Por lo tanto, si uno habla de la complejidad de un sistema

de axiomas, uno está hablando de la complejidad de un cómputo que nunca se acaba, que va a continuar deduciendo teoremas. Sería interesante –y tengo un solo trabajo sobre este tema, que es solamente el primer paso[2]– tener una teoría bien elaborada de la complejidad del tamaño de programas para cómputos que nunca se detienen, por ejemplo, de conjuntos recursivamente enumerables. Esta es la forma en que yo traté de encararlo, pero otra posibilidad sería una teoría de complejidad de funciones computables.

¿Y cuál es la aplicación de la *TAI*?

Bueno, ya he descrito las distintas etapas en la evolución de mi teoría, TAI_{60}, TAI_{70}, TAI_{90}. Hay quienes han tomado esta teoría con miras a aplicaciones prácticas que a mí no me interesan. Para mí lo más interesante de esta teoría es el hecho de que es una teoría impráctica, porque la complejidad no se puede calcular. Para mí esto es fascinante e indica los límites del razonamiento. Pero hay quienes tienen interés en hacer cosas en el mundo real, y algunos de ellos ya no dicen que sus ideas tienen algo que ver con mi teoría, pero si uno mira sus primeras publicaciones, citaron trabajos míos o de otra gente en este campo, y después se fueron en otra dirección.

Conozco dos casos así: un caso en la estadística, y el nombre de Jorma Rissanen es un buen nombre para mencionar en relación con aplicaciones a la estadística de conceptos de este tipo[3]. Pero en la estadística no se puede trabajar con mi complejidad porque si se usa la *TAI* como es, no se puede calcular nada, no se puede calcular la complejidad de nada. Por esta razón, Rissanen hizo algo así como una versión de juguete de mi teoría. Él la llama "MDL", *minimum description length.* Se aplica a la estadística no paramétrica, que creo que es cuando se tiene que escoger entre distintas clases de distribuciones. MDL, *minimum description length,* es algo muy parecido al largo mínimo de programas.[4]

Otra versión práctica de la *TAI* se hizo para tener una manera de comprimir. Gente práctica quería comprimir información en la computadora. Hay un método de compresión que se usa bastante en la práctica que se conoce como

LZW

Es de Lempel y Ziv[5] y, creo, Walsh.

[2]G. Chaitin, Algorithmic entropy of sets, Computers and Mathematics with Applications 2, pp. 233-245, 1976.

[3]J. Rissanen, Stochastic Complexity in Statistical Inquiry, World Scientific, 1989.

[4]También está la teoría MML, *minimum message length,* del grupo de Chris Wallace en la Universidad de Monash en Australia (ver C.S. Wallace, D.L. Dowe, Minimum message length and Kolmogorov complexity, The Computer Journal 42 (4), pp. 270-283, 1999). Parece que MML es semejante a MDL. Wallace concibió MML en forma independiente; no conocia previamente la *TAI* ni MDL.

[5]J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Transactions on Information Theory IT-23, pp. 337-343, 1977.

El primer trabajo de Lempel y Ziv dice, bueno, Chaitin y Kolmogorov hablaron de autómatas infinitos, pero qué pasa si estudiamos un largo de programa en el cual se usan solamente autómatas finitos, de un número finito de estados. Su método de compresión empezó así: Ellos leyeron mi teoría y LZW es una versión de autómata finito de compresión, porque el programa óptimo, de tamaño mínimo para hacer algo es, en cierto sentido, la mejor compresión de la cosa que calcula. Entonces ellos dijeron ¿qué pasa si se usa un autómata finito para descomprimir, para recuperar el mensaje original? Ellos dijeron ¿qué pasa si usamos un autómata finito en lugar de un autómata infinito, como se hace en la *TAI*? Y así comenzó el método LZW de compresión.

Estos trabajos que cito indican que a veces las cosas más teóricas pueden sugerir cosas prácticas. Pero, a mi juicio ¿cuál es la aplicación más importante de todo esto? Primero quiero señalar que hay diferentes versiones de la historia de este campo, y aquí estoy dando mi versión.

Kolmogorov y yo dijimos independientemente, bueno, sería interesante una teoría de la falta de estructura, del azar, basada en esta medida de complejidad.

Yo creo que esto es cierto, pero no da una versión nueva de la teoría de probabilidades. Es casi lo mismo. Cuando mi teoría habla de esto, es muy semejante a la teoría tradicional. Pero creo que esta no es la aplicación más interesante. No reemplaza la teoría clásica de probabilidades. Kolmogorov no se dio cuenta de que la definición original de complejidad estaba mal, pero más importante aún es que no se dio cuenta de que su aplicación más significativa era los límites de la matemática, era comprender mejor el fenómeno de incompletitud que descubrió Gödel. Yo creo que allí está la aplicación más importante. Y mi ejemplo que indica esto más claramente es la probabilidad de detención, el número

Este número cumple con mi definición de incompresibilidad, es aleatorio. Es incompresible y también es incomprensible; las palabras son muy semejantes.

Es decir que escapa al poder del raciocinio. Este número se define fácilmente como la probabilidad de que un programa generado echando una moneda al aire, se detiene finalmente o no. No es una cosa del otro mundo, tiene una definición muy simple, pero es una definición no constructiva.

No solamente no se puede calcular este número, sino que nunca se puede saber cuales son sus bits, porque esta información es información matemática incomprensible . . . porque no se puede comprimir.

Es decir que la única forma de deducir los valores de bits individuales de este número es esencialmente poniéndolos como axiomas. Es un caso en el cual uno no gana nada con razonar y todo está dado directamente en los axiomas. Es un caso en el cual el razonamiento no ayuda para nada. Es decir que para poder

deducir los valores de los primeros n bits de Ω necesito una teoría de n bits de complejidad, una teoría matemática de la misma complejidad que el fenómeno estudiado.

Pero esto significa que no gano nada razonando y que aquí la razón no me da nada, porque los primeros n bits de Ω son n bits de complejidad, y yo podría tranquilamente tomar estos bits como mis axiomas; es lo mejor que puedo hacer.

La idea es que básicamente nada ayuda a conocer estos bits. Para conocer n bits de Ω, para deducirlos, simplemente tengo que poner esta información en los axiomas en forma directa. Es decir que aquí el razonamiento no sirve para nada. Este es un campo de la matemática, los bits de Ω, en el cual la verdad matemática no tiene absolutamente ninguna estructura.

Otra forma de decirlo es que normalmente se piensa que si algo es cierto, lo es por una razón, que hay una razón por la cual es cierto. En la matemática la razón por la cual algo es cierto se llama una prueba, y el trabajo del matemático es encontrar pruebas, es hacer demostraciones. Pero los bits de Ω son hechos matemáticos al azar, accidentales. Es decir que, está tan delicadamente balanceado si un bit determinado de este número es cero o uno, que nunca vamos a saber qué es. Tiene que ser uno u otro, pero no hay ninguna razón para que sea uno u otro. Así que simplemente la única forma de deducir cuál es el valor de un bit determinado en la expansión binaria de este número es agregando este hecho como un nuevo axioma. Pero uno puede demostrar cualquier cosa si uno lo agrega como un nuevo axioma. Lo extraño de este número es que en este caso es la única forma de ir adelante.

En otras palabras, los bits de Ω son el caso extremo de hechos matemáticos irreducibles que no tienen ninguna estructura. Aquí sabemos que no es culpa de uno no poder demostrar cuáles son sus bits ni poder encontrar su estructura, porque no hay ninguna estructura. Este es mi ejemplo del azar en la matemática, el azar que los físicos conocen muy bien y con el cual se sienten cómodos. Este es mi ejemplo de un caso en el cual se encuentra el azar en la matemática pura.

Ahora bien, el próximo paso que di fue tomar este número y convertir la cuestión de cuáles son sus bits en una cuestión de la Aritmética. Con un programa, elaboré una ecuación de

200 páginas

y es una ecuación diofántica, lo que quiere decir que es una ecuación algebraica, siguiendo al griego Diofantos de Alejandría de hace 2000 años, es una ecuación algebraica en la cual todos los números tienen que ser enteros. Es decir que las incógnitas tienen que ser números enteros y todos los coeficientes son números enteros.

Es una ecuación algebraica que tiene 200 páginas de largo, una sola ecuación, más o menos 100 páginas a la izquierda y 100 páginas a la derecha, tiene más o menos 20.000 variables o incógnitas. Una de las variables es un parámetro. Entonces fijo este parámetro en cero, luego lo fijo igual a uno, lo fijo igual a dos,

$$i = 0$$
$$i = 1$$
$$i = 2$$

y sustituyo este valor cada vez que ocurre el parámetro dentro de la ecuación. Esto me da una serie de ecuaciones, una primera, una segunda, una tercera, etcétera.

Para cada una de estas ecuaciones pregunto ¿tiene un número finito o infinito de soluciones? (en números enteros).

Una solución es una tupla de 20.000 números enteros que cuando sustituyo se cumple la ecuación, el valor del lado izquierdo es igual al valor del lado derecho.

Esta es mi ecuación, y está hecha con cuidado para convertir el problema de cuál es un bit determinado, si es 0 ó 1, en Ω, para convertirlo en la cuestión de si una ecuación diofántica tiene un número finito o infinito de soluciones. Si éste número, Ω, es aleatorio e imposible de conocer, también esto se encuentra en la Aritmética, en la teoría elemental de los números.

Y fue en este momento que el mundo científico se interesó en este campo, porque esto realmente sorprendió a mucha gente.

Esto lo publiqué en el año 1987, en mi libro de Cambridge University Press. Y el resultado fue al año siguiente en un artículo en *Scientific American,* después un artículo en *New Scientist,* después un artículo en *La Recherche,* todo seguido. Y el título del artículo en *Scientific American,* que fue el mejor, el más sorprendente, fue "Randomness in Arithmetic", "El azar en la aritmética". Porque normalmente se piensa que la Aritmética es algo totalmente claro, blanco y negro.

Ahora bien, yo quiero que me entiendan. Fíjense que la BBC me entrevistó y alguna gente piensa que yo afirmé que dos más dos a veces no son cuatro...

$$2 + 2 = 4$$

Bueno, esto no tiene nada de aleatorio. Es decir, sabemos que es 4, nunca 5, nunca 3.

El primer paso es tomar una ecuación algebraica y preguntar si tiene solución o no, en números enteros. Este es el 10º problema de Hilbert, del año 1900. Hilbert pidió que se buscara un método para determinar si una ecuación algebraica arbitraria tiene una solución en números enteros. Y alguna gente que yo conozco trabajó en esto.

Martin Davis, Julia Robinson, Hilary Putnam...

Finalmente Yuri Matiyasevich terminó la demostración, creo que más o menos hace 30 años, cuando era un estudiante en San Petersburgo. Pero él se basó en el trabajo de esos otros matemáticos, que estaban en los Estados Unidos.

A Davis lo conozco personalmente hace mucho, nos conocimos en la década del 60 en Nueva York. Y una vez le estreché la mano a Raphael Robinson, el

marido de Julia Robinson. Fue en Berkeley, California, después del fallecimiento de su mujer. Y me crucé con Matiyasevich una vez en Toronto, Canadá. Él demostró que no hay un algoritmo para determinar si una ecuación algebraica tiene solución en números enteros o no. Demostró que la dificultad de saberlo es equivalente al problema de Turing, de determinar si un programa se detiene eventualmente o no.

Pero esto no da el azar en la aritmética, esta ecuación diofántica que construyó Yuri Matiyasevich. Los distintos casos del 10º problema de Hilbert no son independientes, no pueden serlo.

Pero distintos casos de mi cuestión, que es si una ecuación diofántica tiene un número finito o infinito de soluciones en números enteros, sí pueden ser independientes, y en el caso de mi ecuación lo son.

¿Por qué no pueden ser independientes las respuestas si uno se pregunta si cada ecuación diofántica en un conjunto de ecuaciones diofánticas, tiene solución en números enteros o no? Bueno, la razón es muy simple. Esos casos nunca pueden ser casos independientes; casos diferentes del 10º problema de Hilbert nunca son independientes. ¿Por qué? Si tengo n ecuaciones algebraicas, o mejor dicho, n ecuaciones diofánticas, y quiero saber para cada una si tiene una solución en números enteros o no, lo peor sería si necesito conocer n bits de información, porque hay n casos, n ecuaciones.

Pero resulta que solamente hay $\log_2 n$ bits de información. ¿Por qué? En otras palabras, se comprime mucho, exponencialmente. ¿Por qué se puede comprimir tanto? Porque si yo sé cuantas ecuaciones tienen solución, puedo encontrar cuáles tienen solución y cuáles no. Si Dios o un oráculo (Turing hablaba de oráculos), si un oráculo me dice cuántas de esas ecuaciones tienen solución, es un número de $\log_2 n$ bits, que me dice cuántas ecuaciones diofánticas en este conjunto de n tienen solución en números enteros. Es decir que si sé cuántas tienen solución, me pongo sistemáticamente a tantear, a ensayar, sustituyendo números enteros en forma sistemática, hasta que encuentro la cantidad debida de ecuaciones con solución. En este momento sé que ya encontré todas y que las otras ecuaciones diofánticas no tienen solución en números enteros.

Por lo tanto, n casos del 10º problema de Hilbert son solamente $\log_2 n$ bits de información, no son n bits, que sería lo que se necesita para un ejemplo de aleatoriedad.

Pero n casos de preguntar si tienen un número finito o infinito de soluciones en números enteros, no si tienen solución o no, eso sí puede dar la máxima cantidad, n bits de información. Esto ocurre con mi ecuación, mi ecuación con un número finito o infinito de soluciones según el valor del parámetro. ¿Por qué? Porque esto resulta ser igual a determinar los bits de Ω, del número Ω de la probabilidad de detención.

Así que aquí finalmente encontramos no solamente algo no computable dentro de la Aritmética, porque eso ya lo hizo el 10º problema de Hilbert. Encon-

tramos algo aleatorio en la Aritmética, en la teoría elemental de los números. Es un paso bastante grande más allá de lo que hicieron Julia Robinson, Martin Davis y Yuri Matiyasevich cuando ellos resolvieron el 10º problema de Hilbert.

En otras palabras, esta versión del 10º problema de Hilbert en la cual se pregunta si tiene un número finito o infinito de soluciones enteras, no si tiene solución o no, crea un problema, demuestra que tenemos problemas serios en la matemática.

Ahora bien, hay quienes pueden decir "¿Qué me importa todo esto? Yo voy a continuar haciendo matemática en la forma tradicional".

La otra razón por la que creé esta teoría es que cuando era un jovencito, leía ensayos escritos por Hermann Weyl y John von Neumann, en los cuales ellos reaccionaron al descubrimiento de Gödel en la década del treinta. Ellos estaban asombradísimos, realmente se veía que todo su mundo se les venía abajo, pues ellos pensaban que la noción convencional de la matemática se derrumbaba, y que la matemática de seguridad absoluta caía en ruinas. Yo leí estos ensayos y me impresionaron, pero después vi que todo el mundo se había olvidado ya, cuando yo los leí hace unos 40 años, al fin de la década del cincuenta o comienzos de la década del sesenta. Los matemáticos en la práctica seguían trabajando exactamente en la forma tradicional y empezaban a oirse voces que decían, bueno, en la práctica el teorema de Gödel realmente no tiene ningun impacto.

Por ejemplo, el grupo de Bourbaki en Francia; ellos lo dijeron de una forma muy terminante. Ellos dijeron que en la práctica el teorema de Gödel no tiene ningún significado para la matemática. Que solamente es un resultado que interesa a los lógicos y los filosófos pero no a los matemáticos.

Entonces yo me pregunté si esto podía ser cierto. Mi sospecha era que no, que algo tan importante como lo de Gödel, algo tan extraño, debería tener mucho significado. Y fue por esto que hice todo este desarrollo.

* * *

Creo que es muy interesante mirar todo esto desde arriba, desde un punto de vista histórico. La historia de la lógica en este siglo es realmente apasionante, es tremendamente interesante. Algunas personas pueden pensar que la lógica fracasó, y en cierto sentido fué así. La lógica es el esfuerzo por formalizar la matemática hasta el máximo grado posible. Y para mí, Hilbert es quien mejor explica la meta de la lógica clásica. Por lo menos para mí fue Hilbert quien lo hizo mejor; no soy un historiador.

Hilbert dijo que buscaba una formalización completa de la matemática. Esta sería un sistema axiomático formal o SAF, que según él es un sistema deductivo en el cual existe una regla mecánica, es decir, un algoritmo, para verificar si una prueba es correcta o no, si es cierta o no. Y como la lógica es el formalismo llevado al máximo, el SAF de Hilbert tenía que expresarse usando un idioma artificial. Porque los idiomas naturales nos encantan, son hermosos, son poéticos,

¡pero son ambiguos!

Entonces el SAF de Hilbert tiene que venir con un idioma completamente artificial y formal, con una gramática muy estructurada. Las reglas de lógica que este SAF utiliza son reglas de estructura, no hablan del significado de los objetos. Son muy claras y no involucran la intuición, es decir, son estructurales.

Creo que este SAF no se debe solamente a Hilbert; Hilbert es la culminación de una trayectoria que empieza en la Grecia antigua y sigue con Leibniz, Boole, Frege, Peano, Russell y Whitehead. Hay muchos otros nombres que debería mencionar. Pero creo que Hilbert fue quien agudizó el análisis. El hablaba de un sistema axiomático formal que abarcara toda la matemática y acerca del que todos pudiéramos ponernos de acuerdo. Esta era su meta.

Ahora bien, si ese plan hubiese funcionado, sería muy hermoso. Inclusive el *Entscheidungsproblem* estaría resuelto. *Entscheidungsproblem,* que también lo mencionó Hilbert, significa "problema de decisión" en alemán. Hilbert observó—así lo creo—que si hay una forma mecánica de verificar si una demostración cumple las reglas o no, esto tiene el corolario un poco extraño, de que se puede generar una lista de todos los hechos verdaderos. E inclusive, si el sistema axiomático formal es consistente y completo, se podría resolver algorítmicamente cualquier cuestión matemática. ¿Cómo?

Si tengo un método mecánico para ver si una demostración es correcta, podría examinar todas las demostraciones posibles en orden de tamaño creciente, y en orden alfabético dentro del mismo largo. (El idioma artificial con que trabajo está basado en un alfabeto finito.) Entonces, miro todas las cadenas de un carácter de largo, 2, 3, 4, 5 caracteres, un renglón, una página, dos páginas. Lo hago así sucesivamente. En la práctica no se puede hacer, pero en teoría se podría. Me fijo si cumple las reglas de una demostración y si las cumple termina con un teorema.

En otras palabras, si tengo un sistema axiomático formal en el estilo de Hilbert, como lo propuso Hilbert, hay un algoritmo, hay un proceso mecánico para generar todos los teoremas. Es teórico, en la práctica no se puede hacer, pero teóricamente sí, y esto generaría todos los teoremas en el orden del largo de sus demostraciones.

Y entonces si el sistema axiomático formal es bueno—bueno significa que dice toda la verdad y sólo la verdad—esto me daría una forma de resolver cualquier cuestión. Porque si tengo una cuestión matemática que tiene que ver con los conceptos de esta teoría—y Hilbert hablaba de una teoría que tenía que abarcar toda la matemática—si tengo un problema matemático, una cuestión que me interesa, me pongo a recorrer todas las posibles demostraciones hasta que encuentro un teorema que afirma lo que me interesa o afirma lo contrario. Y de esta forma ¡puedo resolver cualquier cuestión matemática!

Creo que Hilbert no pensó que en la práctica esto eliminaría la matemática—¡este no era su propósito!—pero como meta filosófica es hermosa. ¿Por qué es

hermosa la lógica?

Ustedes pueden pensar que la lógica es muy pedante y seca. Creo que no, que el esfuerzo histórico ha sido desmenuzar las demostraciones y romperlas en pedacitos cada vez más chicos, para entender exactamente qué es lo que camina y qué es lo que no camina en la matemática. Creo que la lógica nunca quiso que nosotros tuviésemos que trabajar dentro de un sistema axiomático formal. Porque si ustedes miran las demostraciones completamente formales, son terriblemente largas y difíciles de entender, y además están hechas en un idioma artificial.

Creo que el propósito de Hilbert no era que nos convertiésemos en máquinas. A mi juicio, la meta no era imponer un formalismo y un idioma artificial a todo el mundo. Él quiso hacer todo esto por razones filosóficas, para comprender mejor los alcances de la razón.

Bueno, de todos modos, este intento fracasó. La lógica fracasó, este intento de formalización fracasó. Pero la formalización no fracasó del todo, porque con computadoras ustedes pueden tener el gusto de trabajar con idiomas artificiales y de preocuparse por detalles sumamente minuciosos.

El intento fracasó porque Gödel demostró que cualquier SAF es incompleto, y que no es posible formalizar toda la matemática.[6] Uno puede pensar que este esfuerzo de formalización fue un desastre. Pero quiero sostener la tesis contraria, que es que realmente fue el éxito más grande del siglo XX, pero como tecnología, no como filosofía.

En este momento todos estamos trabajando con sistemas formales. ¿Qué clase de sistemas formales? ¡Idiomas de programación! Los idiomas de programación son el éxito tecnológico más grande de este siglo. Por supuesto también hace falta algo físico, es decir, las máquinas que comprenden estos idiomas de programación y trabajan con ellos.

Y no se puede negar que estos idiomas de programación son bien formales, de hecho más formales que los formalismos que usan los matemáticos. Es una observación aguda que hizo el matemático William Thurston. Él comentó que la atención al detalle que se requiere para hacer andar un programa es mucho mayor que la que se requiere para demostraciones matemáticas, ¿y acaso no es cierto? Porque una demostración matemática uno tiene que explicarla a sus colegas, pero cuando uno hace un programa tiene que explicarlo a una máquina que no entiende nada, y el esfuerzo es mucho mayor. Si cualquier detalle está

[6] A pesar de esto, Jacob Schwartz, del Instituto Courant, sigue tratando de elaborar un sistema axiomático formal basado en la teoría de conjuntos y suficientemente poderoso para expresar todas las demostraciones de los teoremas de Analisis I en una forma concisa. Vendría con un programa verificador que conoce bastante de teoría de conjuntos para permitirlo. Schwartz subraya que el teorema de Gödel no es un impedimento para su proyecto y creo que ¡tiene mucha razón! El sistema lógico que él esta desarollando tiene la particularidad de que su propósito es reducir al mínimo no el número de principios básicos, sino el tamaño de las demostraciones. En otras palabras, él intenta hacer una versión actualizada de *Principia Mathematica* de Russell y Whitehead, que pueda ser verificada mecánicamente.

mal, la computadora sigue adelante y se mete en líos. Mientras que en la matemática, cuando se publica un trabajo, a pesar de que los referees lo leyeron y el autor también, muchas veces hay cosas que están mal. Normalmente la gente se da cuenta, leyendo el trabajo, y puede hacer algunas correcciones. A veces son faltas de ortografía, a veces un símbolo se omite o se reemplaza por otro, pero en general esto no destruye una demostración. Pero sí, definitivamente destruye un programa.

En otras palabras ¿cuál fué el éxito del formalismo? Los idiomas artificiales son un éxito tremendo, pero no son para razonar, son para expresar algoritmos, para expresar procesos de cómputo. Y antes de la existencia de computadoras, había muchos trabajos de lógica en los cuales se empleaba lo que hoy en día se llamaría un idioma de programación. Uno lo ve si mira las colecciones de trabajos de lógica escogidos por van Heijenoort y por Davis. Van Heijenoort tiene un libro gordo de trabajos importantes escritos por lógicos, y Martin Davis publicó otra colección de este tipo.[7]

Por ejemplo, el cálculo λ de Church es un idioma de programación. Y si uno mira el trabajo original de Gödel, hay un idioma de programación de alto nivel. Tiene funciones recursivas, y a mi juicio es muy parecido a LISP. Por supuesto el trabajo de Turing contiene un idioma de programación del tipo de lenguaje de máquina. ¡Y esto lo hicieron lógicos cuando no había computadoras!

De modo que vemos que los idiomas artificiales son un éxito tremendo, pero son para computar, no para razonar.

También está el hecho de que el teorema de incompletitud de Gödel dice que ningún conjunto finito de axiomas es completo y da toda la verdad. Ningún SAF permite expresar todas las demostraciones posibles. Pero ocurre lo contrario con los idiomas de programación. Con la máquina universal de Turing no hay incompletitud pues es una computadora que puede simular cualquier otra computadora, cualquier otra máquina.

En otras palabras, los idiomas de programación que se usan son universales, en el sentido de que puede realizarse cualquier proceso de cómputo. En la práctica el idioma de programación es importante, pero en teoría no, en teoría cualquier idioma de programación tiene el mismo alcance que otro, porque cualquier algoritmo se podría expresar en cualquiera de ellos.

En otras palabras, los sistemas formales para expresar un proceso de cómputo, los idiomas de programación, son universales, son completos, no sufren de la incompletitud de los sistemas formales deductivos para razonar.

Existe un libro, que no sé si sostiene exactamente esta tesis, pero dice algo semejante. Este libro lo escribió el geofísico Douglas Robertson y se llama *El Nuevo Renacimiento,* y es una reacción al libro *El Fin de la Ciencia* de John

[7]Jean van Heijenoort (editor), From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931, Harvard Univ. Press, 1967.
M. Davis, The Undecidable, Raven Press, 1965.

Horgan. *El Nuevo Renacimiento* es un libro muy optimista que dice que la computadora no solamente en la práctica, sino también en la teoría ha sido el comienzo de una nueva época. ¡Prefiero los libros optimistas a los libros pesimistas! No sé si Robertson dijo exactamente lo que acabo de escribir acerca del éxito de la formalización. Creo recordar que leyendo su libro esto se aclaró para mí, pero no sé si lo tomé de él o lo pensé yo.

Ahora bien, si Hilbert pudiera vernos ahora, ¿cómo reaccionaría? Creo que por un lado, mal. Él se enteró del trabajo de Gödel cuando ya era viejo y no le gustó para nada. Creo que hubiera sido diferente si hubiese sido joven, pero ya era viejo y famoso y además ya venía el desastre de Alemania, estaban ya empezando los disturbios. Pero a Hilbert le hubiese encantado el tremendo éxito de los idiomas de programación, y el hecho de que hay sistemas formales por todos lados y mucha gente los utiliza, aunque sean sistemas formales de cómputo y no de razonamiento.

Para seguir un poco más lejos con este análisis, tengo un conocido, un físico matemático que se llama Stephen Wolfram y es el fundador de una empresa que vende software que se llama Mathematica. Mathematica es un idioma de programación de muy alto nivel para cómputos simbólicos. Este sistema de programación conoce mucha matemática y matemática física, es como un ayudante. Por ejemplo, puede hacer integrales y derivadas, puede hacer los deberes matemáticos. La empresa de Wolfram vende un producto para alumnos de Analisis I que tiene inconvenientes porque si los alumnos lo utilizan, hacen todo automáticamente y no aprenden. Pero también estimula a los profesores a dejar la parte rutinaria y poner mayor énfasis en los conceptos. Utilizar Mathematica es como tener un ayudante, porque puede hacer derivadas, puede resolver ecuaciones diferenciales, y es casi como una inteligencia artificial, dentro de límites, por supuesto. Además a Wolfram no le gusta TEX y LATEX, ni HTML ni Java. ¡Mathematica hace todo solo! Trabaja con algo que se llama un *Mathematica Notebook,* que es una especie de libro electrónico. Una carpeta de Mathematica incluye texto—y además hay formas de hacer un texto muy lindo—y también incluye fórmulas y programas y diagramas y colores y más.

En otras palabras, es un libro electrónico que no es pasivo, que mezcla texto, que se puede leer, con fórmulas y con programas que se pueden correr en el acto, como parte del libro. Por ejemplo, uno hace un clickeo y el programa empieza a andar y a arrojar dibujos. Además, es un idioma de programación muy poderoso. Al principio me entusiasmé mucho y en mi libro sobre los límites de la matemática tengo un intérprete LISP que hice en Mathematica. Francamente, mirándolo ahora, dos o tres años después de que lo escribí, temo que no es tan fácil comprenderlo. Pensé que como es un idioma de muy alto nivel y los programas son bastante cortos, sería más fácil comprenderlo que si estuviera escrito en C. Además, mi intérprete es 1.000 renglones en C y 300 renglones en Mathematica y estaba seguro de que no quería incluir 1.000 renglones de C en

mi libro; ocupa demasiado espacio.

De todos modos, para publicar algoritmos matemáticos, creo que convendría usar el idioma de más alto nivel posible, siempre y cuando se pueda correr, sino no sirve para nada. Además está el problema del tiempo de cómputo. Si se requiere 100.000 veces más tiempo para correrlo en Mathematica que para correrlo en otro idioma, nadie va hacerlo. Así que hay que tomar en cuenta simultáneamente el poder del lenguaje y su tiempo de cómputo.[8]

Me entusiasmé cuando Wolfram vino una vez a la Universidad de Columbia e hizo una demostración de la versión de su sistema que se llama Mathematica versión 3. Su demostración vino en un *Mathematica Notebook.* Él vino con su laptop y lo proyectó directamente, y la demostración que hizo fue texto mezclado con fórmulas matemáticas, y las manipulaba y las resolvía. Además se podía cambiar el formato de todo fácilmente. Cambiaba con un clickeo una presentación frente a un gran público, a un formato adecuado para imprimirlo como un libro, y lo hacía todo automáticamente, lo que me pareció bastante bueno. ¡Y no soy accionista de su empresa!

A lo que quiero llegar, y fue algo que dije en la Universidad de Columbia frente a todo el mundo, es que en cierto sentido Wolfram logró el sueño de Hilbert. Ahora bien, este no es el único producto de este tipo. Tengo amigos a quienes les gusta mucho Maple, que también es un idioma de programación de alto nivel para cómputos simbólicos. Estos programas lograron el sueño de Hilbert. ¿Por qué digo esto? Porque aquí tenemos un sistema formal, y es un sistema formal que abarca más o menos toda la matemática, por lo menos la que se usa en la física. Porque se ve que Wolfram es un físico matemático y su sistema contiene cosas que a un matemático puro le pueden parecer un poco extrañas.

Pero hizo un buen trabajo de ingeniería, es un hombre muy capaz, y su sistema integra bien muchos elementos disímiles. Es una mezcolanza, trae cosas de muchos lugares, y también hay muchos algoritmos matemáticos que vienen suministrados por el sistema. Y Wolfram tuvo que encontrar cómo hacer andar todo junto, porque a veces diferentes modalidades de trabajo chocan entre sí y no se combinan bien.

¿Cuál es el resultado? Que tenemos en Mathematica una formalización de toda la matemática, pero es para computar, no es para razonar, no es para encontrar demostraciones.

Computar es fácil y razonar es difícil. Porque un algoritmo de razonamiento tiene un árbol de posibilidades que crece exponencialmente, si uno mira todos los pasos posibles de la demostración, mientras que en un cómputo está siempre el próximo paso, y el tiempo no crece tan velozmente. Es decir, aproximadamente,

[8]En su momento también me gustaron mucho el SETL de Jacob Schwartz, que trabaja con conjuntos, y el APL, que trabaja con vectores y matrices. A su vez, LISP trabaja con listas. Mathematica tiene todas estas modalidades de trabajo y también incluye "pattern matching", o sea, substitución dirigida por patrones.

que algo es un cómputo cuando existe un algoritmo veloz. Si no hay uno veloz, si el tiempo crece exponencialmente, estamos razonando.

Me asombré al pensar que en cierto sentido, el sueño de Hilbert de la formalización de toda la matemática se había realizado. Inclusive se puede preparar un libro directamente en Mathematica, hacerlo en la computadora, y después imprimirlo como un libro. Al principio pensé en hacer mi libro sobre los límites de la matemática así, pero finalmente lo hice en HTML y Java, porque todo el mundo tiene browser para HTML y puede correr applets de Java.

Por supuesto un idioma vale más si todo el mundo lo entiende y todo el mundo lo puede correr—si hay muchos idiomas diferentes que compiten entre sí, todo se fragmenta. En fin, creo que desde el punto de vista filosófico, todo este desarrollo tecnológico es muy significativo. La lógica y la formalización son un gran éxito, a pesar de mis resultados que dicen que los SAF tienen limitaciones bastante serias. Creo que la tecnología de la computación es a la vez un gran éxito tecnológico y conceptual.

Ahora quiero aclarar un poco la relación entre mis resultados sobre los límites de la razón y los de Gödel.

Para empezar, mis resultados surgen de otra fuente de incompletitud. Yo no puedo deducir el resultado de Gödel tal cual lo hizo Gödel, con mis métodos. Son resultados diferentes. Mis resultados de incompletitud están en el espíritu de los de Gödel, pero los métodos son diferentes y los resultados a los cuales se llega son diferentes.

¿Cuál es la diferencia más grande entre el resultado de Gödel y los míos, a grandes rasgos? La diferencia principal es que mis métodos se basan en un concepto de complejidad de información. En mi enfoque sobre la incompletitud, lo que hago es considerar un sistema axiomático formal, que contiene un conjunto finito de

axiomas

y tambien incluye reglas de inferencia, de lógica:

$$\boxed{\text{axiomas} \quad \longrightarrow_{\text{reglas de inferencia}} \quad \text{teoremas}}$$

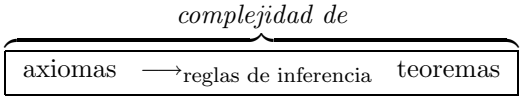
Y mido la complejidad de este SAF en bits. ¿Cuál va a ser su complejidad?

Para dar un ejemplo más concreto de mis métodos, uno puede medir la complejidad de un SAF en caracteres de programación en LISP, en lugar de utilizar bits de información.

¿Cuál es la complejidad de un SAF en la *TAI*? Bueno, para empezar uno puede copiar los axiomas de un libro de lógica y ver cuál es su largo total en caracteres. Esto da un idea aproximada. Algunos SAF son cortos, por ejemplo, los axiomas de Peano; la teoría de conjuntos de Zermelo y Fraenkel es más complicada. Uno ve por el tamaño de la presentación en un libro que la complejidad de cada una de estas teorías es diferente.

Por lo tanto, el largo de su presentación en un libro de lógica nos da una idea aproximada de la complejidad de un SAF, y nos ayuda a desarrollar nuestra intuición al respecto. Pero esto no nos brinda una definición formal y precisa de su complejidad. Para lograrla, hay que tomar en cuenta la complejidad del programa que verifica si una demostración es correcta o no. Recuerden que Hilbert decía que la característica primordial de un SAF reside en la existencia de un algoritmo para decidir si una demostración cumple las reglas o no, y que la rechaza o la acepta.

Más precisamente, la complejidad del algoritmo verificador es importante, pero también es necesario poder generar todas las demostraciones posibles, a partir de la gramática y el alfabeto que usamos. Mi definición final es que la complejidad de un sistema axiomático formal es el largo del programa más conciso que genera todos sus teoremas.



Ahora bien, hay que reconocer que esta definicion de la complejidad de un SAF es un poco abstracta. El grado de abstracción de esta definición es semejante al de la definición de SAF que creo que dió Post por primera vez, que dice que un sistema axiomático formal es un conjunto recursivamente enumerable de enunciados. Ambas definiciones son bastante abstractas, pero creo que van al corazón del asunto. Porque puedo demostrar resultados de incompletitud sin saber qué es lo que pasa dentro de esta caja negra que es el sistema axiomático formal. Para mí es suficiente mirarlo desde afuera, sin conocer sus mecanismos internos.

Inclusive se podría decir que mi enfoque es un poco como el de la termodinámica. La termodinámica es el campo de la física en el cual se estudian máquinas térmicas (máquinas a vapor) mediante el concepto de temperatura, que todos conocemos, y el concepto de entropía, que es mucho más misterioso. En este campo muchas veces los argumentos se encaran como *gedanken experiments* o sea experimentos conceptuales, pensados, no realizados en el laboratorio. Y a veces estos experimentos conceptuales tienen que realizarse infinitamente despacio para que sean reversibles. Por lo tanto, la termodinámica es una teoría bastante teórica en la cual el tiempo no importa demasiado. Fíjense que en estos aspectos la termodinámica se asemeja bastante a la *TAI.* Porque en mi teoría tampoco me interesa el tiempo de cómputo, y además la entropía, o desorden, que es el concepto clave de la termodinámica, está muy cerca del largo de programas, que es mi concepto clave.

En mi teoría sobre los límites de sistemas axiomáticos formales considerados como cajas negras, uso la complejidad del largo de programas para analizar "máquinas" que son sistemas axiomáticos formales, y esto es bastante análogo al uso de entropía en la termodinámica para estudiar máquinas térmicas.

Los vínculos entre la *TAI* y la termodinámica son aún mas estrechos que esto. No es solamente una mera analogía; inclusive hay algunos físicos como Wojciech Zurek, que toman mi teoría del largo de programas, y la aplican en la física a cuestiones como el demonio de Maxwell.[9] ¿Qué es el demonio de Maxwell? Sería una forma de extraer energía del gas que está dentro de una habitación cerrada que está en equilibrio térmico, lo que es imposible según la termodinámica. Sin embargo, Maxwell observó que si separo la habitación en dos con una pared, y hago un pequeño agujero que permite pasar, o no, una sola molécula de gas, y pongo allí un pequeño demonio como agente de tránsito, él puede hacer lo siguiente: Cuando ve una molécula que va a atravesar el agujero, el demonio abre o cierra el agujero para permitir que solamente las más veloces que el promedio crucen en una dirección, y las menos veloces que el promedio crucen en la otra dirección.

Y así eventualmente tengo todas las moléculas veloces en una mitad de la habitación, y todas las menos veloces en la otra, y ésta es una forma de extraer energía de un sistema en equilibrio térmico. ¿Sería ésta una máquina de movimiento perpetuo? No, del primer tipo no, porque no crea energía de la nada, pero sí del segundo tipo, porque rompe la ley de que la entropía de un sistema no puede bajar si uno no invierte energía. Por lo tanto los físicos saben que esto no camina, que el demonio de Maxwell no puede funcionar, pero ¡les ha llevado más de cien años entender por qué!

Szilard jugó un papel importante en esto. En la década del veinte escribió un trabajo sobre el demonio de Maxwell,[10] en el cual trata de demostrar que el demonio tiene que confundirse mediante un análisis de su estado mental, de la información que contiene, información que Szilard midió como en la teoría de información de Shannon (de la década del cuarenta). Como veremos, el concepto de información de Shannon es sumamente cercano a la entropía en la termodinámica y en la física estadística.

¿Pero qué pasa si remplazamos al demonio por una computadora? Esto fue analizado en los últimos veinte años por algunos físicos como Charles Bennett y Carlton Caves y Murray Gell-Mann, aplicando mi concepto de información algorítmica en lugar del de Shannon. Así que el tamaño de programas y la entropía de Shannon pueden ser considerados como variantes de la entropía termodinámica; ambos sirven para demostrar que el demonio se confunde, sea ente pensante o computadora.

Ya que estamos en esto, voy a explicar en más detalle porqué no funciona el demonio de Maxwell, tal como me lo explicó mi colega Charles Bennett, que profundizó el análisis de Rolf Landauer, también de mi laboratorio.

Recuerden que el demonio es un pequeño ente pensante o una computadora,

[9] Ver, por ejemplo, M. Gell-Mann, The Quark and the Jaguar, W. H. Freeman, 1994.

[10] El artículo de Szilard aparece reimpreso en: H.S. Leff, A.F. Rex, Maxwell's Demon, Princeton University Press, 1990.

que tiene que tomar la decisión de permitir, o no, que una molécula atraviese el agujero que divide una habitación en dos. Resulta que el problema con el demonio de Maxwell es que tiene que olvidar, y esto gasta energia. Resulta que desde el punto de vista de la termodinámica, en la computación lo que cuesta es olvidar cosas, porque esto es irreversible. Un proceso termodinámico reversible no incrementa la entropía y, si se hace lentamente, no requiere inversión de energía. Pero el problema es que el demonio tiene que borrar su estado mental, y por lo tanto se va a calentar mucho.

Si el demonio no se olvida de nada, tiene que grabar una memoria infinita, y va a terminar siendo la parte más importante del sistema físico en lugar de ser algo microscópico. Y si le hacemos olvidar lo no necesario, hay que gastar energía, porque olvidar información es irreversible. Y el demonio calienta todo a su alrededor y tampoco puede funcionar. En ambos casos el demonio es refutado.

Y este es más o menos el análisis que demuestra que el demonio no puede funcionar. ¿Qué tiene que ver esto con la *TAI*? Bueno, si el demonio se considera un ordenador, una computadora muy pequeña, resulta que es natural aplicar en el análisis del demonio mi tipo de entropía del largo de programas.

La entropía normalmente indica en la física si está bien distribuida la probabilidad sobre todo el espacio de posibilidades, que son los "microestados" del sistema físico. La entropía aumenta al máximo cuando la distribución de probabilidades es uniforme. La entropía es dada por la fórmula de Boltzmann y Shannon

$$H(p_1, p_2, p_3, \ldots) = \sum_i -p_i \log_2 p_i,$$

donde p_i es la probabilidad del i-ésimo microestado. Pero mi teoría, la *TAI,* da una forma de definir la entropía de un microestado individual: es el largo en bits del programa más conciso que produce una descripción del microestado.

En la termodinámica clásica no existe la noción de entropía de un microestado individual, pero ahora, gracias a la *TAI,* la tenemos.

Ahora volvamos a Gödel y a la incompletitud. ¿Cuál es la diferencia entre su enfoque y el mío?

Mi enfoque es aproximadamente como una "termodinámica del razonamiento", y tengo algunos resultados muy generales que dicen cuál es la eficiencia máxima que puede tener un SAF. Son como los resultados en la termodinámica que dan la eficiencia máxima que puede tener una máquina térmica en función de la diferencia de temperatura de sus partes.

Yo mido la complejidad, el contenido de información, como el largo en bits del programa más conciso que produce todos los teoremas:

$$\overbrace{\text{axiomas} \longrightarrow_{\text{reglas de inferencia}} \text{teoremas}}^{\textit{complejidad de}}$$

Pero en la práctica ¿qué va a hacer este programa? Este programa va a generar todas las demostraciones posibles y verificar cuales son correctas y cuales no. De modo que la parte principal de esta medida de complejidad es el largo del programa del que hablaba Hilbert, el programa que mecánicamente verifica, o no, que una prueba que alguien nos ofrece cumple las reglas del juego.

Pero técnicamente, la mejor forma que encontré de encarar esto es hablando de un proceso de cómputo que nunca termina, que como les he dicho genera todos estos teoremas, y realmente no importa de dónde vienen. No necesito saber qué es lo que está pasando dentro de un sistema axiomático formal. Lo miro como una caja negra que de vez en cuando arroja un teorema. Lo único que importa es que tiene que arrojar todos los teoremas que son consecuencias de los axiomas y las reglas de inferencia de este sistema. Al SAF lo miro de bien arriba y no me interesa su estructura interna. Sólo me interesa saber cuántos bits de información contiene, o sea, cuál es el largo del programa más conciso que genera todos estos teoremas.

Es decir que esta sería la versión más concisa del sistema axiomático formal. En la práctica no conozco la versión más concisa. Podría descubrir axiomas más concisos que generan un mismo conjunto de teoremas. Pero programando un conjunto de axiomas con sus reglas de juego, que generan una teoría matemática, obtengo una cota de la complejidad, que sería en efecto los axiomas más concisos que me dan exactamente este conjunto de teoremas. Así que esta es mi definición, y de esta manera siempre puedo medir la complejidad de un sistema axiomático formal. A veces, puedo tomar un

$$\underline{\text{teorema}},$$

un enunciado matemático que alguien desea demostrar a partir de estos axiomas

$$\boxed{\text{axiomas} \quad \longrightarrow_{\text{reglas de inferencia}} \quad \text{teoremas}},$$

y también puedo medir la complejidad de este teorema en bits. A veces, no siempre. Esto es muy importante subrayarlo.

Repito, siempre puedo medir la complejidad del SAF:

$$\boxed{\text{axiomas} \quad \longrightarrow_{\text{reglas de inferencia}} \quad \text{teoremas}}$$

Mis métodos también se aplican a ciertos tipos de teoremas y a veces puedo medir la complejidad en bits del enunciado que se desea demostrar:

$$\underline{\text{teorema}}$$

Y en ese caso, cuando puedo hacerlo, si la complejidad del enunciado es mayor que la del SAF, no puede andar, este enunciado no puede ser consecuencia de estos axiomas. Esto es el tipo de resultado de incompletitud que obtengo

con mi teoría. En otras palabras, no puede andar si el teorema matemático es demasiado complicado para poder deducirse a partir de esos axiomas. No puede andar porque los axiomas son demasiado simples para este teorema complicado.

Un ejemplo concreto es tratar de demostrar que un programa es "elegante". ¿Qué es un programa elegante? Defino un programa como elegante si ningún programa más conciso, más pequeño, escrito en el mismo lenguaje de programación, produce exactamente el mismo resultado. El tiempo de cómputo no me interesa. Me interesa solamente cuán conciso es, el largo del programa en bits o en caracteres. Si uno quiere medirlo en caracteres, de LISP por ejemplo, mi método también anda así.

Para cualquier proceso de cómputo, cualquier resultado, evidentemente hay una gran cantidad de programas que van a hallar el mismo resultado y algunos de ellos tienen que ser más concisos, más pequeños que los otros. Puede haber muchos, puede haber varios programas, puede haber un empate. Podría haber tres programas elegantes para una tarea determinada, pero por lo menos hay uno, para cualquier tarea.

Por lo tanto hay una infinidad de programas elegantes, pero ¿qué pasa si quiero estar seguro de que tengo un programa elegante? ¿Qué pasa si quiero estar seguro de que un programa en particular es elegante?

Bueno, resulta que no se puede estar seguro. Yo no puedo demostrar que un programa es elegante si es más grande que los axiomas que se emplean, si es más complicado que los axiomas que estamos usando.

Más precisamente, si estamos estudiando programas elegantes en LISP, por ejemplo, se tiene que medir la complejidad de los axiomas mediante el mismo idioma de programación, LISP. Es decir que hay que medir el largo del programa más conciso en LISP que genera todos los teoremas de mi sistema axiomático formal en el cual se habla de programas elegantes. Y si hacen falta 50.000 caracteres en LISP para generar todos estos teoremas, y si tengo un programa en LISP que creo que es elegante pero es más grande que 50.000 caracteres, es decir, es más complicado que mis axiomas, no se puede demostrar que este programa es elegante a partir de estos axiomas.

Nótese que aquí estoy midiendo el grado de complejidad usando el mismo idioma de programación en las dos partes, para los axiomas y para el teorema. Es decir que si el programa es más grande que los axiomas, si el programa que quiero demostrar es el más conciso posible, que es elegante, si este programa es más grande que los axiomas, no se puede hacer. Es un resultado del tipo de paradoja de Berry, no del tipo de paradoja del mentiroso, que usó Gödel.

Esto termina este resumen del campo de información y azar. Para más información, fíjese en mis libros citados en la bibliografía.

Agradecimiento: El autor desea expresar su gratitud a la Dra. Verónica Becher por toda su ayuda.

Referencias

[1] G. Chaitin, The Limits of Mathematics, Springer-Velag, 1998.

[2] G. Chaitin, The Unknowable, Springer-Verlag, 1999.

[3] G. Chaitin, Exploring Randomness, Springer-Verlag, 2001.

[4] G. Chaitin, Conversations with a Mathematician, Springer-Verlag, 2002.

Gregory Chaitin
IBM Research, USA
chaitin@us.ibm.com
http://www.cs.umaine.edu/∼chaitin