
COLECCIÓN ENTRECRUZADOS
TURING
HERENCIAS
ENIGMAS
DE LOS NÚMEROS
COMPUTABLES
AL CELULAR
EDA CESARATTO,
MARCELA FALSETTI
Y LUCAS ROZENMACHER
(COMPILADORES)
EDICIONES UNGS
Universidad Nacional de General Sarmiento

Turing, herencias y enigmas
De los números computables al celular

Turing, herencias y enigmas : de los números computables al celular / Verónica Becher ... [et al.] ; Compilación de Eda Cesaratto ; Marcela Falsetti ; Lucas Rozenmacher. - 1a ed - Los Polvorines : Universidad Nacional de General Sarmiento, 2024.
Libro digital, PDF - (Entrecruzados ; 3)

Archivo Digital: descarga y online
ISBN 978-987-630-736-9

1. Criptografía. 2. Matemática. 3. Tecnologías. I. Becher, Verónica II. Cesaratto, Eda , comp. III. Falsetti, Marcela, comp. IV. Rozenmacher, Lucas, comp.
CDD 652.8

EDICIONES **UNGS**

J. M. Gutiérrez 1150, Los Polvorines (B1613GSX)
Prov. de Buenos Aires, Argentina
Tel.: (54 11) 4469-7507
ediciones@ungs.edu.ar
www.ungs.edu.ar/ediciones

Diseño de tapa: Daniel Vidable
Diagramación: Eda Cesaratto
Corrección: María Inés Castaño

Libro
Universitario
Argentino

COLECCIÓN **ENTRECRUZADOS**

Turing, herencias y enigmas

De los números computables al celular

Eda Cesaratto, Marcela Falsetti y Lucas Rozenmacher
(compiladores)

Verónica Becher
Daniel Lvovich
Ana Maffei
César Mónaco
Gustavo Piñeiro
Lucas Rozenmacher
Nicolás Sirolli
Tomás Tetzlaff
Ariel Waissbein

EDICIONES **UNGS**
Universidad Nacional de General Sarmiento

Índice general

Prólogo

Lucas Rozenmacher

"Alzo la copa en honor a Alan Turing,
que nació en un tiempo más oscuro, más duro
quien pensó por fuera del cuadrado
y amó por fuera de los límites
y por eso se rompió el código de los descifradores
y nos lamentamos.
Ahora, que ya dijimos que nos lamentamos
que la conciencia oficial se ha despertado
- con un guión muy cuidado, pero al menos desencriptado
la historia sugiere
una segunda parte del Test de Turing:
1. ¿Pueden las máquinas comportarse humanamente?
2. ¿Podemos nosotros?"
Matt Harvey[*]

1. Desde dónde parte este libro

El libro al que estamos ingresando refleja parte de lo elaborado y discutido durante las *Jornadas Ecos de la figura y la obra de Turing en Argentina*[†], en donde, a partir de una serie de encuentros, charlas, intercambios y conferencias fueron delineándose las vinculaciones entre cuestiones relacionadas con los avances e investigaciones en computación y criptografía y otras derivaciones propuestas en torno a la ciencia y la sociedad y el impacto de sus desarrollos innovadores, y también sobre la influencia de la figura de Turing en la sociedad contemporánea.

Este entrelazamiento entre textos y reflexiones matemáticas, desarrollo de trayectorias biográficas y análisis del contexto histórico terminaron por darle cuerpo a los textos que se encuentran incluidos en este volumen.

[*]Traducción: Sofía Cazeres.

[†]Estas jornadas se realizaron en el Instituto del Desarrollo Humano de la Universidad Nacional de General Sarmiento entre el 7 y el 10 de noviembre de 2017.

2. ¿Por qué Turing?

Cuando con las editoras e impulsoras de este libro (Marcela Falsetti y Eda Cesaratto), armamos el esquema de esta compilación de presentaciones y trabajos, en unas anotaciones al margen, Cesaratto deslizó (en modo de brújula e intriga) la pregunta de ¿por qué Turing? y en esa misma pregunta comenzamos a tejer una serie de temas y problemas que trazan algunas de las dimensiones y abordajes que pueden establecerse a través de la figura de Alan Turing, en cuanto a su relevancia e impacto desde antes de mediados del siglo veinte hasta nuestros días.

En este hombre-nombre (además), se albergan historias de intrigas, distopías, revoluciones comunicacionales, espionaje y referencia fundamental para la reivindicación de derechos civiles.

Para ello proponemos realizar un recorrido en tres pasos, uno que aborda la relación de Turing con la creación de lo que se denomina "La Máquina de Turing", el otro que trabaja sobre la relación entre Turing y su papel en la Segunda Guerra Mundial como matemático encargado de descifrar los códigos encriptados del Eje, también en un esquema de recorrido tripartito y una última sección que pone de relevancia la vinculación histórica de Turing con su época y la relación que este tiene con la sociedad contemporánea que a modo de coda funciona como una estructura de un solo relato.

3. Primer momento o el ingreso a la discusión pública

El volumen abre con una imbricación entre la historia de Turing como ser humano interviniente en su comunidad y el proceso en el que fue desarrollando la denominada "Máquina de Turing", una computadora con capacidad de intervenir en la vida humana y modificar esa relación para siempre.

Verónica Becher despliega un recorrido breve pero intenso sobre la vida de Alan Turing y su relación con la creación de su máquina computacional, con sus pares y hasta con su madre.

En este recorrido veremos cómo fue desarrollándose la vinculación de Turing con la búsqueda del conocimiento y el impacto que provocó la aparición de una máquina que pudiera actuar sin poder despejarse la duda de si la acción era llevada por un hombre o por una mujer y a ello le sumó un paso más y planteó un juego en el que había que descifrar si esta acción (de juego) era realizada por una persona o una máquina a partir de un dispositivo que no nos permitiera identificar quién estaba detrás de ello.

Este artículo nos centra en el trabajo que Turing desplegó para avanzar sobre una inteligencia que funcionara no como inteligencia corporal sino como una reductora del tiempo. De este descubrimiento y propuesta inquietante podemos encontrar correlatos en dos escritores contemporáneos a don Turing, uno de ellos es Isaac Asimov y el otro Ray Bradbury y desde allí

una larga lista de libros, poemas, obras teatrales, series y películas que abordan distintas perspectivas sobre los límites entre lo humano y lo maquinal que hacen convivir a este experimento con una tríada de leyes denominadas como leyes de la robótica que luego sería incorporada a producciones posteriores en cuanto a los límites de las relaciones entre humanos y máquinas a partir de lo propuesto por Asimov.

Entre los ejemplos posteriores que dan cuenta del temor abierto frente a la relación humano-máquina con respecto a la Inteligencia Artificial, se puede citar el poema espacial escrito por Harry Martinson llamado *Aniara*. El panorama del ser humano en el espacio y el tiempo en donde una máquina toma la decisión de qué es lo mejor para los humanos y con ello lleva a la extinción misma de la humanidad. Algo parecido ocurre con *Blade Runner* (la interpretación cinematográfica de "¿Sueñan los androides con ovejas eléctricas?") en el que se vuelve indistinguible el reconocimiento de lo humano y lo maquinal, en el que las reglas y los temores a lo maquínico se hace carne y el mismísimo caminante que se encarga de despachar a las "réplicas" duda sobre el grado propio de su humanidad.

Otros ejemplos fueron disparándose a lo largo de este poco menos de un siglo de temores, elucubraciones y despliegue poético sobre lo que Turing dio en llamar el "Juego de la Imitación" con la tetralogía de *Matrix* y el proponer un mundo en el que los humanos son dominados por las máquinas y viven para alimentar a estas últimas a partir del trastocamiento de tiempo y materia.

En ese mismo camino Ana Maffei y Tomás Tetzlaff nos cuentan en qué consiste "la máquina de Turing" y cuáles son los alcances de lo computable. Los autores despliegan los detalles en los que se establecen las posibilidades de una máquina como esta y los propios límites.

Como culminación de este segmento de la publicación Gustavo Piñeiro propone un viaje a la búsqueda de sobre qué hombros estaba parada la "máquina de Turing" llevándonos a comienzos del siglo veinte a la conferencia de David Hilbert en la apertura del II Congreso Internacional de Matemática en París y a partir de esta los lazos hasta llegar a la relación entre el algoritmo y la "máquina de Turing".

En esta primera sección contamos con un terceto de artículos que trabajan y desarrollan distintos aspectos sobre una de las elaboraciones centrales de Alan Turing y de la que nos nutrimos de manera material hasta el día de hoy que es la de la computadora tal como la conocemos en la actualidad y del desarrollo posible de la inteligencia artificial.

4. Segundo momento

A través de los recovecos que surcan los mensajes encriptados que retiemblan a punto de explotar en las manos de quien intente descifrarlos, solo

sobre esto pueden pensarse un libro o una serie de volúmenes completos hablando acerca de los aspectos técnicos de la criptografía y con ello zambullirse en aquellas tramas que atravesarán científicos y agentes secretos en el intento de desenmascarar planes mortales, cargados de intrigas, engaños y cócteles exquisitos.

Litros y litros de tinta (material y digital) y kilómetros interminables de fílmico (y también digital) desde hace más de medio siglo van reencauzando los disparadores que se producen alrededor del mensaje encriptado y de la forma en que fue desenmascarada la máquina Enigma, una acción que permitió despejar al fascismo como sector triunfante de la Segunda Guerra Mundial y que abrió también todo un camino a la literatura y la cinematografía contrafácticas y en algunos casos hasta ucrónicas.

Turing debió descifrar un sofisticado encriptamiento, alejado de la tinta limón utilizada en los espías ideados por Graham Greene en sus novelas, o en los intercambios epistolares que fueron establecidos entre Juan Domingo Perón y John Willam Cooke durante parte del exilio y proscripción del peronismo entre finales de la década de 1950 y comienzos de la siguiente y, de algún modo, acercándose a las formas en que Julio Verne contactaba a sus lectores en cada una de sus novelas.

Sobre este tema, como ya dijimos podrían desarrollarse capítulos y libros enteros, pero en esta ocasión la selección de este libro se ciñe a la participación de Daniel Lvovich y un análisis histórico-social sobre la situación en la que Turing debió desarrollar su trabajo criptográfico.

El siguiente texto pertenece a Ariel Waissbein y nos dará una introducción acerca de las características de la criptografía mecánica y el modo en que Alan Turing debió atacar a la máquina Enigma para descifrar los pasos que seguirían los Nazis y de ese modo anticiparse a los movimientos por parte de los aliados.

Terminada la Segunda Guerra, el mundo se encontró en medio de una guerra fría y sin contar con protocolos criptográfico seguros. Gabriel García Marquez relata en "Playa Girón y el escritor que se adelantó a la CIA" cómo el escritor y periodista argentino Rodolfo Walsh logró descifrar un mensaje en el que se comunicaba que los Estados Unidos iban a invadir a Cuba por Playa Girón en abril de 1961. Walsh realizó esta tarea con la única ayuda de un libro de criptografía recreativa sin haberlo hecho nunca, sin ningún entrenamiento en la materia y solo con papel y lápiz. La hazaña intelectual de Walsh es indiscutible. Sin embargo, este hecho también demuestra la fragilidad del criptosistema usado por los Estados Unidos en la época.

Entonces, ¿cómo es que ahora enviamos miles de millones de "mensajes encriptados de extremo a extremo" y hacemos transacciones bancarias "seguras"? Para cerrar esta sección del libro, nos encontramos con el trabajo de Nicolás Sirolli que nos instruye sobre las claves matemáticas de la revolu-

ción que implicó la criptografía de "clave pública" concebida a principios de los años ochenta y que, junto con la disponibilidad de dispositivos digitales, incluyó a la criptografía en nuestra vida cotidiana.

5. Sobre las derivas o momento final

En este esquema de tríada, que ya adelantamos, encontramos a lo largo del capítulo 7, a cargo de César Mónaco, un trabajo que nos ubica en una serie de cuestiones históricas y sociales que nos dan las herramientas necesarias para ubicar a la figura de Alan Turing y a su obra en nuestra contemporaneidad desde una perspectiva histórica, política y cultural.

Por un lado, retoma un análisis de la situación que Europa y particularmente el Reino Unido transitaron durante los años de la Guerra y el papel del estado en la recuperación de países devastados por la guerra, la miseria y la crueldad humana, a partir de políticas activas de lo que se denominó como el Estado de Bienestar.

A este punto, el autor le sumó la nueva realidad sociopolítica en la que se vio involucrado el mundo y que fue la denominada como guerra fría con el crecimiento del comunismo como modo de gobierno en distintas partes del planeta y las tensiones contenidas y a punto de explotar entre modelos políticos, económicos, sociales y culturales.

Por último Mónaco analiza los cambios que fueron dándose a nivel mundial con respecto a la persecución de la homosexualidad, dado que Inglaterra, en el momento de ser impartida la pena a Turing por sodomía y su posterior suicidio, contaba con una ley que perseguía la sodomía datada en 1533 a manos de Enrique VIII y que recién en los años de la década de 1960 fue derogada y llamada "Ley Alan Turing".

En este recorrido el autor hace un hallazgo sobre las marcas de época en el propio historiador Eric Hobsbawm en cuanto al relato que este realiza sobre el modo en el que es detenido Turing al describirlo como una persona desconectada del mundo real describiéndolo de la siguiente manera: "Solo un hombre que, como él, desconocía el mundo en el que vivían los demás podía ocurrírsele ir a denunciar el robo cometido en su casa por un amigo íntimo (temporal), dando así a la policía la oportunidad de detener a dos delincuentes a la vez".

Finalmente podemos entender en esta tríada una serie de puntas disparadoras que nos permiten avisorar nuevos abordajes que partirán de este trabajo que proponemos.

6. Agradecimientos

En primer lugar, queremos agradecer a nuestro colega Cristian Conde del Instituto de Ciencias por su involucramiento en la organización de las *Jor-*

nadas Ecos de la Figura y la obra de Turing en Argentina. Sin duda, este libro no hubiera sido posible sin la colaboración de Paula Albarracín, Gastón Bidart Gauna, Rodrigo Moreno y Melina Sarni en la organización y grabación de las Jornadas. En ese momento eran estudiantes del Profesorado Universitario en Educación Superior en Matemáticas de la UNGS y ahora son docentes de nuestra universidad. Es el momento de agradecer la apertura que nos ofreció Paola Miceli, en ese entonces Secretaria de Investigación de la UNGS, la mesa redonda organizada por Luciano Grippo (UNGS) y Jorge Kamlofsky (UTN), y finalmente, al artista visual Juan Miceli quien nos brindó una charla sobre la influencia de la programación en su obra.

El trabajo de Albarracín desgrabando y revisando los capítulos 1, 4 y 5 fue clave en la preparación de este texto.

Nuestro agradecimiento para Ángel Jara cuyos dibujos ilustran el concepto de cardinal de un conjunto vía biyecciones del capítulo 3.

Finalmente, queremos agradecer el trabajo de los evaluadores de este libro por sus miradas, aportes, comentarios y el compromiso con el que nos efectuaron las devoluciones.

Bibliografía consultada

[1] I. Asimov. *Yo robot*. Buenos Aires: Editorial Sudamericana, 2014.

[2] R. Bradbury. *Fantasmas de lo nuevo*. Madrid: Minotauro, 2021.

[3] K. Dick Philip. *Sueñan los androides con ovejas eléctricas*. Madrid: Minotauro, 2020.

[4] G. García Marquez. *Playa Girón y el escritor que se le adelantó a la CIA*. `https://lacasaeditora.org/playa-giron-y-el-escritor-que-se-adelanto-a-la-cia/`. Consultada el 1 de febrero de 2023.

[5] M. Harvey. *Quotes*. `https://bukrate.com/author/matt-harvey`. 2012.

[6] H. Martinson. *Aniara. Un panorama del hombre en el tiempo y el espacio*. España: Gallo negro, 2015.

Parte I

La automatización del pensamiento

Capítulo 1
Alan Turing, padre de la computación

Verónica Becher[*]

1. Vida personal

Les voy a contar por qué pienso que Alan Turing es considerado el padre de la computación. Es mi visión personal acerca de esta historia. Me crucé con esta pregunta la primera vez que oí su nombre mientras cursaba la carrera de computación. El profesor dijo en clase "máquina de Turing" y mientras él anotaba el nombre en el pizarrón, me preguntaba ¿cómo se escribirá, Tou, Tuu?, ¿cómo es? No tenía ni idea de qué se trataba esa máquina. Para mí era un objeto teórico que servía para resolver algunos problemas de computación de las materias de tercer año. Un tiempo después, cuando todavía era estudiante, me crucé con su historia.

1.1. La familia

Turing nació en 1912 cerca de Londres, en un lugar llamado Maida Vale. Siempre lo imaginé comiendo comida india porque su padre trabajaba para el servicio civil de la India por lo que viajaba allí con cierta frecuencia. Además, su abuelo materno era ingeniero en la compañía de trenes de ese país.

La relación entre Turing y su madre, Ethel Sara, era fluida. De hecho, ella escribió la biografía de su hijo; este es uno de los pocos casos de la historia de la ciencia en los que se da esta situación. La segunda edición de este libro [6] se publicó en el año 2012 para conmemorar los 100 años del nacimiento de Turing. Ella es de las pocas personas que siguió defendiendo la tesis de que Turing no se suicidó, sino que su muerte se debió a un accidente mientras él

[*] N. de la R.: este artículo está basado en la charla de la autora durante las jornadas "Ecos de la figura y de la obra de Alan Turing" (UNGS, 2017). Desgrabación: Paula Albarracín.

trabajaba con sustancias químicas. Otra de las razones que hacen pensar en una muy buena relación es la carta que Turing le envió cuando el servicio del gobierno británico lo contactó en 1938 para formar parte del proyecto de descifrado de la máquina Enigma. En esta carta le escribió a su madre: "No sé qué hacer, tengo que ver si acepto esta convocatoria o la dejo pasar". No le escribió al padre, le escribió a la madre.

Turing no era ni comunista ni criptólogo. De la edición de 1983 de la biografía que escribió Andrew Hodges sobre Alan Turing (veáse [3]) podemos concluir que estas son las causas que lo hicieron reflexionar acerca de su participación. Finalmente, concluyó que sus conocimientos eran lo suficientemente valiosos como para contribuir en la tarea encomendada y aceptó participar.

1.2. La niñez

Me gustó saber cómo era Turing de niño, es la parte de su biografía en la que más me enfoqué porque está relacionada con mi investigación acerca de su manuscrito sobre números normales [1].

Como mencioné anteriormente, sus padres viajaban ida y vuelta a la India por lo que él se quedaba junto a su hermano en la casa de una pareja de militares retirados. Se educó en una escuela pública y a sus 10 años comenzó la preparatoria Sherborn, instancia que parece haberlo marcado mucho según la comprometida y extensa biografía de Turing publicada por Andrew Hodges [3]. Me llamó mucho la atención de esta biografía la descripción de Turing como un desaliñado; le chorreaba siempre la lapicera y era incompetente con las exigencias de la escuela pública. No es que era rebelde, era incompetente y no le salía bien. Alan tenía su propia agenda y estaba preocupado por otros temas sobre los que le interesaba leer. Allí conoció a su compañero Christopher Morcom que era mayor que él.

Hodges cuenta que en la preparatoria comenzó esta historia de amor que lo marcó fuertemente. Me pregunto, ¿por qué creer en esta versión de Hodges? Es una interpretación. Podemos ver que los escritos de Turing parecen estar redactados como si estuviese explicándole las ideas a alguien quien seguramente era la figura de Morcom. Christopher, que era un excelente alumno, se daba cuenta de todas esas habilidades que Turing no había logrado aprender de la escuela. "Sos un desprolijo, no te entiendo cuando me explicás tus ideas de matemática" imagino que le decía Morcom a Turing frente a sus soluciones alternativas de los problemas. Las soluciones de Alan no eran aceptadas por su compañero quien le reclamaba que tenía que poner ordenadamente las fórmulas. Esto le implicó a Alan todo un esfuerzo científico.

Andrew Hodges nos da a entender que entre Morcom y Turing hubo una amistad muy fuerte pero que el amor que sentía Turing por Morcom no

era completamente correspondido. En 1930, Turing, con 16 años, quedó muy afectado por el fallecimiento de Christopher que estaba enfermo de tuberculosis. Siguió en contacto con su madre, y, de hecho, en una oportunidad se quedó en la habitación de la casa materna de su amigo. Es muy posible que todas las ideas posteriores las haya escrito recordando aquellas preguntas que Christopher le hacía para comprender sus explicaciones.

1.3. La universidad

A pesar de ser un desaliñado y no tener un buen rendimiento escolar, lograba aprobar. Tuvo que dar dos veces el ingreso a la universidad. No le resultó fácil. Su voz de pito era un aspecto que resultaba desagradable para mucha gente. Finalmente logró ingresar al King's College Cambridge en donde estudió desde los 18 a los 22 años. Allí se encontró en un lugar donde pudo expandirse intelectualmente. Fue alumno de Godfrey Hardy, un matemático reconocido, una de las grandes figuras de la teoría de números. Sin embargo, él nunca reconoció el trabajo de Alan Turing como un trabajo importante. Turing no fue valorado dentro del entorno académico en ese momento. Su trabajo era considerado como un "trabajito" irrelevante, un artículo más.

También allí conoció a David Champernowne. Ellos compartieron clases de matemática y unos quince años después crearon un analizador de jugadas de ajedrez. Champernowne es conocido por sus trabajos en economía y por la resolución de un problema sobre números normales, tema sobre el cual trabajó también Turing y que es una de mis líneas principales de investigación.

Cuando Turing terminó su carrera de matemático pasó a ser "Fellow", es decir, un miembro de la Universidad de Cambridge. En ese entonces, en Europa, no hacía falta estar doctorado para formar parte de la universidad. Alcanzaba con haber terminado la carrera de grado. En ese momento desarrolló el gran trabajo del que voy a hablar hoy y que es el mismo que lo llevó a Princeton a estudiar con Alonzo Church.

2. Aportes científicos

2.1. La máquina de Turing y los números computables. El artículo de 1936

En su artículo "Los números computables, con una aplicación al *Entscheidungsproblem*" [5][*], Turing formalizó la computadora como objeto matemático. Este es el trabajo que le valió el reconocimiento como "padre de la computación". Cuando uno investiga, lo hace en relación con una pregunta y no porque surge una idea de la nada. Por ese motivo, intenté encontrar

[*] N. de la R.: el artículo fue enviado y aceptado en 1936, pero publicado en 1937.

cuál era la pregunta que se hizo Turing. Mi interpretación es que la respuesta está en el título del artículo: "una aplicación al *Entscheidungsproblem*". En este trabajo define una clase particular de números y los llama computables. Estos se pueden describir como aquellos números cuya expansión decimal, es decir, los dígitos que vienen después de la coma, se pueden describir mediante métodos finitos. Turing no tuvo miedo al delirio, y describió estos métodos finitos mediante un objeto imaginario al que llamó "máquina" y finalmente consideró que un número es computable si su parte decimal se puede describir mediante esa máquina.

2.2. Métodos finitos, problemas imposibles

Ya hemos definido los números computables como aquellos cuya parte decimal se describe mediante métodos finitos y los métodos finitos los entendemos como aquellos que pueden ser implementados en una máquina que es un dispositivo mecánico. La máquina ejecuta instrucciones que fueron escritas antes de ser ejecutadas y nos arroja un resultado, una salida, que en general es un número escrito en binario, por ejemplo: 010101. ¡Y es una máquina! Una máquina conformada por piezas simples, como podrían ser algunas varillas y ruedas, dispuestas de tal forma que al funcionar construya tablas de datos. A partir de esta concepción, uno podría dibujar un esquema de un artefacto que transforme energía en tiras de ceros y unos, y definirlo como "máquina finita". Estos ceros y unos combinados conforman el lenguaje binario. El hecho de elegir ceros y unos es irrelevante, se podrían reemplazar por cualquier otro conjunto de símbolos.

El trabajo de Turing es el que presenta la idea de la existencia de una máquina capaz de hacer el trabajo de cualquier otra máquina que trabaje con símbolos, que pueden ser "0" y "1". A estas máquinas las denomina "máquinas universales". Esta idea no era nueva en la matemática. El concepto de equivalencia y cociente ya muestra esta concepción de que, a veces, existe un representante de cierta clase que puede hacer lo que haría cualquier otro elemento de su misma clase. Al leer cómo se construyen estas máquinas universales nos encontramos con la sorpresa de que cada una de las computadoras que tenemos hoy en día en nuestros escritorios es una versión moderna de una máquina universal de Turing.

Me pregunto lo siguiente: ¿qué significa ser universal y por qué es importante? ¿Por qué es necesario inventar una máquina universal? En esa época, dentro de los ámbitos matemáticos, se buscaba responder preguntas acerca de la (im)posibilidad de resolver ciertos problemas: ¿cuáles problemas tienen solución y cuáles problemas no? Uno se puede remontar a los griegos que se preguntaban cuáles son los objetos geométricos que se pueden construir con regla no graduada y compás. En este contexto, la posibilidad de construir un objeto es equivalente a la posibilidad de trazar un segmento

de una cierta longitud "fundamental" a partir del segmento unidad, siempre limitándose al uso de regla y compás.

Uno de los problemas griegos más conocidos es el problema de duplicar un cubo que consiste en construir otro que tenga el doble de volumen del inicial. La respuesta recién fue hallada en 1837 por el matemático francés P. Wantzel quien demostró que no es posible hacer esa construcción con regla no graduada y compás. Otro problema famoso es la llamada "cuadratura del círculo". A partir de un círculo se pretende construir un cuadrado que tenga la misma superficie que tiene el círculo; es decir, si para pintar el círculo se usa una cierta cantidad de tinta, se quiere dibujar un cuadrado que requiera la misma cantidad de tinta para ser pintado. Resulta que dibujar un tal cuadrado con regla no graduada y compás es imposible. Esto se supo en 1882, y es consecuencia de relacionar este problema con un número muy importante, el número π. Los griegos plantearon problemas cuya respuesta no pudieron encontrar. La conclusión actual es que no pudieron hacerlo porque tales construcciones son imposibles.

Turing se hizo una pregunta análoga. En vez de regla y compás consideró métodos finitos. Por ejemplo, cuáles cálculos se pueden hacer con un ábaco, cuáles son las operaciones lógicas que se pueden hacer finitamente. Estas preguntas acompañan al teorema de Kurt Gödel, fundamental dentro de la lógica matemática, quien había demostrado que dentro de la aritmética, formalizada en un lenguaje lógico, llamado lenguaje de Primer Orden (porque se cuantifica exclusivamente sobre elementos del dominio y no sobre conjuntos de elementos), hay ciertas verdades que no se pueden demostrar. Hay proposiciones de las cuales no se puede decidir si son verdaderas o falsas a partir de un sistema de axiomas y reglas de inferencia. Este es otro resultado que habla de ciertas imposibilidades, en este caso de la lógica matemática. Turing y sus colegas de la época conocían estos resultados. La formulación de Turing de esta pregunta es la siguiente: ¿dado un enunciado matemático, se puede determinar por métodos finitos si es demostrable? Determinar finitamente se puede tomar como equivalente a exhibir un algoritmo que decida si un enunciado es verdadero.

Un ejemplo de enunciado matemático es el siguiente: "El número natural x es primo". [*] Otro enunciado clásico de la aritmética es el que sigue: "Dados dos números x e y, el número x es divisible por y". La pregunta es, entonces, ¿para cualquier enunciado (de ese tipo) es posible determinar por métodos finitos si es demostrable? ¿O hay verdades que requieren una matemática infinita? El problema disparado por esta pregunta fue llamado *Entscheidungsproblem*.

[*] Los números primos son aquellos números naturales distintos de la unidad y que solamente se pueden dividir por sí mismos o por la unidad.

2.3. De los algoritmos a la invención de la computadora

Abu Abdallah Muhammad Ibn Musa Al Juarismi (ca. 780 - ca. 850) fue un matemático árabe a quien se le atribuye la noción de algoritmo. Entre sus obras, se destaca su tratado de álgebra *Compendio sobre el cálculo de al jabr y al muqabala*, que ha influenciado a toda la ciencia occidental de la Edad Media.

Intentemos responder la siguiente pregunta: ¿qué es un algoritmo? Es una receta, una serie de pasos para resolver un problema y esos pasos pueden incluir sumas, restas y otro par de operaciones muy simples que están disponibles para su uso. Hay un listado de tales operaciones, que no discutiremos en este texto pues requiere de una discusión más extensa. Al-Khwarizmi dio una noción informal de algoritmo. Esta noción podría incluir una receta para hacer pasta frola: en vez de sumas usamos "batir", y en vez de productos usamos "hornear"; con los ingredientes y ese tipo de procedimientos simples, se prepara una pasta frola. Esta analogía, en el contexto del trabajo de Turing, nos permite pensar en un algoritmo como una cantidad finita de pasos simples (receta) que a partir de ciertas entradas (ingredientes) produce una salida (pasta frola).

Turing buscaba responder al *Entscheidungsproblem* que era una pregunta abierta. Para eso definió la noción de algoritmo, es decir, dio una descripción matemática de qué es un algoritmo. Sin embargo, su definición de algoritmo es completamente intuitiva, pero esa búsqueda de formalización lo llevó a la invención de la computadora.

Es importante observar que la "computadora" de Turing no fue inicialmente un artefacto físico, real; fue un artefacto mental cuya existencia estaba plasmada en el texto del artículo en cuestión.

2.4. Problemas indecidibles

En 1936, Turing prueba que hay enunciados matemáticos tales que ningún algoritmo puede determinar si son demostrables. Casualmente, al mismo tiempo en Estados Unidos, Alonzo Church (1903-1995) proporcionó otra demostración de este resultado.

Al enterarse de la existencia del trabajo de Church, Turing sintió malestar y temor por el hecho de que su trabajo no sea aceptado, ya que él no era prestigioso mientras que Alonzo Church era muy reconocido.

Finalmente, ambos trabajos fueron aceptados, y fue Turing quien se encargó de demostrar la equivalencia de ambos puntos de vista. El hecho de que las dos soluciones sean equivalentes, como puede imaginar el lector, es muy importante porque utilizan dos definiciones distintas de algoritmo: una mediante una máquina; otra sobre la base de la noción de cálculo lambda, que no vamos a discutir en este libro (una referencia para este tema es el libro [2]). Cuando se demuestra que dos formulaciones son equivalentes,

la idea de que deben ser formulaciones correctas para esa noción intuitiva se ve reforzada, en este caso la noción de algoritmo.

Como ya se adelantó, Turing demostró que hay enunciados verdaderos para los cuales no existen algoritmos que puedan demostrar que lo son. Seguramente, a todos alguna vez se nos colgó la computadora. En el sistema operativo Windows suele aparecer en la pantalla un cartelito que dice "Su programa no responde. ¿Qué desea hacer?". Podemos cuestionarnos: ¿por qué me pregunta a mí? Es obvio que deseo seguir trabajando en mi archivo Word. ¿Es porque Windows es un sistema operativo malo? Sí, es malo. ¿Hay una razón de fondo por la cual me pregunta eso? ¿Por qué no toma la decisión solo? ¿Hay una limitación? Resulta ser que sí. Hay una limitación matemática. Imaginemos que no hay limitaciones, ¿cuáles son las funciones que podría hacer la máquina? Podría hacer varias cosas a la vez. Mientras ejecuta Word, otra porción de la máquina le consulta por qué se colgó o no el programa y analiza el código escrito por el programador. Se espera que responda si es un programa que se cuelga o es uno que no se cuelga. Sin embargo, la máquina no lo puede responder, pues no es posible para ningún algoritmo determinar si hay otro algoritmo que se va a colgar o no. Esa es la cuestión que Turing logró determinar: no se puede decidir algorítmicamente si un programa está colgado o no. En otras palabras, ese fenómeno de la vida real y práctica que experimentamos con bastante padecimiento es precisamente el descubrimiento de Alan Turing quien probó que no tiene solución. Este problema es más conocido como el *halting problem* y sobre él, indagaremos un poco más en capítulos 2 y 3. Turing, que inventó las computadoras, encontró su debilidad.

Turing formalizó el cómputo como objeto matemático y esto le permitió predecir lo que una computadora puede hacer o no hacer. Con estas ideas pudo dar su solución al *Entscheidungsproblem* sobre la pregunta: ¿para cada pregunta matemática que se pueda responder por sí o no (solo se aceptan esas opciones) hay un algoritmo capaz de encontrar la respuesta? Esta solución plantea que no hay ningún algoritmo capaz de determinar si otro algoritmo entró en un ciclo infinito, es decir si está colgado, por lo que existen preguntas matemáticas que un algoritmo no puede responder. Gracias a ese trabajo y esa comprensión profunda de la noción de "computar" está institucionalizado en el campo de las ciencias de la computación el premio Turing (A.M Turing Award en inglés) que es entregado todos los años por la Association Computing Machinery desde 1966.

3. Turing y los juegos

A Turing también le llamaban la atención los juegos. De hecho, inventó algunos y estudió otros. Entre ellos está el juego del 15 que tal vez conozcan y hayan jugado cuando eran chicos. El segundo juego, el de la imitación, se

cuenta entre uno de los primeros intentos de definir la noción de inteligencia artificial.

3.1. El juego del 15

El juego del 15 se juega con un tablero que tiene 16 posiciones de fichas del 1 al 15 y un casillero vacío. Tal vez hayan recibido alguno en un cumpleaños. El objetivo es mover las fichas solamente en las posiciones de libertad del tablero para acomodarlas en orden. Turing se preguntó si tiene solución y en caso de tenerla cuántos movimientos hacen falta para dejar el tablero en condiciones. Si algún niño travieso levanta alguna ficha e invierte un solo par de fichas, el juego no tiene solución. Si invierte una cantidad par de fichas, siempre tiene solución. Entonces, como venía de fábrica tiene solución, con un par de fichas invertidas no tiene solución, con dos pares de fichas invertidas vuelve a tener solución. Así que el hecho de tener solución depende de la configuración inicial.

3.2. El juego de la imitación

¿Puede una computadora pensar, puede ser inteligente? [*] Turing no le tenía miedo al ridículo, lo desafió constantemente, no lo alteró decir que estaba pensando en esta pregunta que en la época parecía algo delirante, y a medida que pasaban los años era más y más provocador. El delirio no lo afectó. Ideó un test, el test de Turing, que dice que una computadora es inteligente si puede mentir. Esa mentira se llama el juego de la imitación y consiste en que la computadora tiene que hacerse pasar por un ser humano, y convencer que es humana. Aquí Turing pone en juego su homosexualidad y su humor inglés[†]. Hace un juego en donde un hombre se hace pasar por mujer, y la mujer por mujer y luego repite este juego entre computadora y hombre. El hombre se hace pasar por hombre, es decir, no miente y la computadora se tiene que hacer pasar por hombre. Ese es el juego de la imitación en el cual la computadora es declarada inteligente si convence al que está haciendo las preguntas de que es un hombre, la misma cantidad de veces que en el juego de la imitación hombre-mujer, el hombre hubiera convencido de que es mujer. Y aquí introduce otra innovación, la definición de inteligencia artificial que propone es probabilística.

4. El legado de Turing

Uno puede preguntarse finalmente ¿qué hacen las computadoras de Turing? En su trabajo describe cómo armar mecánicamente un aparato, un

[*] N. de la R.: con esta pregunta, Turing abre su artículo "Computer machinery and intelligence" en el cual introduce el juego de la imitación y la idea de inteligencia artificial [4].

[†] N. de la R.: en el capítulo 7 se analiza este aspecto de la vida de Turing en el contexto de su época.

dispositivo que produce una salida que es una tira de ceros y unos. En la concepción de Turing, esos ceros y unos representan la parte fraccionaria del desarrollo en base dos de números reales. No expresa cuantos números hay que escribir, son los que haga falta, arbitrariamente muchos y eventualmente todos.

Esta fila de números es infinita, ¿es infinita como internet?, ¿en qué sentido es infinito internet? Es infinito en el sentido de que alguien manda un mail, este mail es respondido y se genera una interacción que puede perdurar eternamente. La concepción de internet es infinita en el sentido que empezamos una comunicación vía mail que no tiene un fin previsto. No existe un programa que nos diga "el 31 de diciembre de 2020 usted tiene que cambiar el mail". Quizás lo tengamos que cambiar por otros motivos, pero los programas están concebidos para que sigan por siempre. No saben cuál será el último mensaje. Las máquinas de Turing se prestan a eso, máquinas que mientras haya cosas para decir, mientras haya números en su desarrollo en base dos, seguirán en funcionamiento. Por eso se puede decir que las máquinas de Turing son las de internet, ya que concibe a la computadora como un dispositivo que opera sobre la base de una configuración finita, con un funcionamiento eterno y este es el modelo de cómputos que tenemos en la actualidad. El modelo de Turing es el modelo de la computación que hoy concebimos en la vida diaria y modela todas las máquinas personales. Cabe entonces preguntarse cuáles son las capacidades de las computadoras actuales que no se inducen de las lecturas de las obras de Turing. Dicho de otra manera, ¿sobre cuáles aspectos de la computación Turing no pensó?

Para responder esta pregunta, aquí se presenta un listado, incompleto, de las áreas temáticas sobre las que sí pensó: teorema central del límite, números normales, ordinales, función zeta de Riemann, criptografía, morfogénesis (mancha de animales, forma de hojas), inteligencia artificial y programa de ajedrez Turochamp.

Retomando la pregunta y pensando en lo que él no pudo ver, me llamó la atención que no formalizó la dificultad del cómputo. Turing resolvió problemas de criptografía cuyo desafío principal era lograr romper una cifra en poco tiempo. Si abordaba el problema usando fuerza bruta[*], aunque encontrara la solución, esta hubiera llegado tarde, cuando ya no servía. Turing se enfrentó con esta dificultad de la complejidad computacional y sin embargo no formalizó esta teoría. De todas formas concibe que la computadora sirve para reducir el tiempo que lleva resolver un problema.

El modelo de la noción de inteligencia que Turing propone deja explícitamente de lado al cuerpo y al movimiento. Considera el tipo de inteligencia que hace falta para ganar al ajedrez, para mentir, pero no la que tiene que

[*] Se dice que un problema se resuelve por "fuerza bruta" cuando la solución es encontrada realizando una inspección sistemática de casos.

ver con lo corporal. No pone en juego el espacio. Él pensó en la computadora como un reductor de tiempo, lo espacial lo dejó de lado. Por esta razón considero que quizás para él sería una sorpresa encontrar que en la actualidad su máquina también sea un artefacto que nos permite reducir el espacio, ya que internet nos permite comunicarnos casi instantáneamente con cualquier parte del mundo.

Breve biografía de la autora

Verónica Becher es profesora titular del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires e investigadora principal del CONICET. De su vasta trayectoria, nos interesa destacar la relación de su trabajo con la obra de Alan Turing. Esta comenzó cuando recuperó un manuscrito no publicado de Turing, cuyos resultados algunos creían incorrectos y, junto con Santiago Figueira y Rafael Pichi, "descifró" la letra y las ideas de este matemático. Finalmente, demostraron que el trabajo de Turing era correcto y que contenía valiosas ideas para generar bits aleatorios. Los aspectos matemáticos de esta historia se encuentran en el artículo [1].

Referencias

[1] V. Becher, S. Figueira y R. Picchi. "Turing's unpublished algorithm for normal numbers". En: *Theoretical Computer Science* 377 (2007), págs. 126-138.

[2] M. Davis. *Computability and Unsolvability*. Dover Publications, 1985.

[3] A. Hodges. *Alan Turing: The Enigma*. Simon y Schuster, 1983.

[4] A. M. Turing. "Computing Machinery and Intelligence". En: *Mind* 49 (1950), págs. 433-460.

[5] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". En: *Proceedings of the London Mathematical Society* S2-42.1 (1937), págs. 230-265.

[6] S. Turing. *Alan M. Turing: Centenary Edition*. Cambridge University Press, 2012.

Capítulo 2
En las máquinas de Turing está el poder y los límites de la computación

Ana Maffei
Tomás Tetzlaff

1. ¿Qué es una máquina de Turing?

En 1936, Alan Turing envió para su publicación su artículo titulado "Los números computables, con una aplicación al *Entscheidungsproblem*" [3], en el que daba a conocer la descripción de su máquina. De manera simplificada, podemos decir que se compone de una cinta de longitud no acotada, dividida en celdas, en las que se pueden escribir símbolos, por ejemplo ceros y unos, y consta de un cabezal capaz de moverse por la cinta a izquierda y derecha, leyendo y escribiendo sobre las celdas. En cada paso la máquina de Turing se encuentra en cierto estado, lee un símbolo, puede escribir un nuevo símbolo, cambiar de estado y moverse a la celda de la izquierda o a la celda de la derecha. Por ejemplo la instrucción $(A, 1, 0, I, B)$ establece que si la máquina está en el estado A y lee un 1, escribe un 0, se mueve a la izquierda y pasa al estado B. Con combinaciones de instrucciones de este tipo se pueden hacer operaciones aritméticas y lógicas elementales y también implementar algoritmos complejos. Además una máquina de Turing se puede construir físicamente, excepto por la cinta infinita, que correspondería a una disponibilidad de memoria no acotada para una computadora moderna, algo no demasiado irreal hoy en día con todas las posibilidades de almacenamiento que existen. El problema que tiene programar con una máquina de Turing es que el número de instrucciones y la dificultad para combinarlas, aún para una tarea bastante sencilla, es mucho mayor que con un lenguaje de computación moderno.

Como pequeño ejemplo, veamos una máquina en la que el cabezal recorre la cinta hacia la derecha cambiando todos los ceros por unos hasta que encuentra el primer 1 y allí se detiene (y si no encuentra un 1 no se detiene). Las dos instrucciones que mostramos a continuación definen completamente esta máquina, que comienza en estado A:

Instrucción $(A, 0, 1, D, A)$: en el estado A y leyendo un 0, se escribe sobre él un 1, se mueve el cabezal a la derecha y se sigue en estado A.

Instrucción $(A, 1, 1, D, H)$: en el estado A y leyendo un 1, se mantiene el 1, se mueve el cabezal a la derecha y se pasa al estado H, que significa detención.

Con su formalización Turing hizo preciso el concepto de número computable, aquel para el cual existe una máquina de Turing que devuelve el $n-$ésimo dígito de su desarrollo decimal en un número finito de pasos. Pero su invento tiene una aplicación mucho más general gracias a su ingeniosa exploración de los límites que aparecen cuando se estudia la posibilidad de que un algoritmo ejecute otro algoritmo como parte de sus instrucciones. Esto último es una parte fundamental de la práctica de la programación. Se presenta por ejemplo cuando una aplicación que muestra una lista de nombres en la pantalla llama a su vez a un subprograma para que los ordene alfabéticamente, o cuando un ciclo va recorriendo los números menores que un número n para ver si alguno es divisor de n. De hecho un ciclo es un algoritmo que se llama a sí mismo en la última instrucción, usualmente hasta que se cumple alguna condición. Veremos que aún sin los variados ejemplos que provee la programación actual sobre la posibilidad de que un procedimiento llame a otro o incluso a sí mismo, Turing encontró límites siempre vigentes para el poder de las computadoras.

2. Computabilidad

La idea de computabilidad de un problema es la de ser resoluble con una computadora o aplicando un algoritmo, sin recurrir por ejemplo a la opinión de una persona o a una información influenciada por el azar. Como no se encontró ningún algoritmo o programa que no se pueda ejecutar con una máquina de Turing, se puede decir que un problema es computable si se resuelve con una de estas máquinas. De hecho las operaciones elementales que realiza una computadora, que combinadas consiguen todas las operaciones más complejas, se pueden llevar a cabo con una máquina de Turing. Las computadoras modernas son más rápidas pero no pueden hacer más que una máquina de Turing. Todo esto se expresa en la llamada Tesis de Church y Turing:

Tesis de Alonzo Church y Alan Turing: cualquier cuestión computable puede ser resuelta con una máquina de Turing.

Esta tesis es un supuesto de la computación teórica que se sostiene so-

bre la base de que se ha fundamentado que una máquina de Turing puede calcular lo mismo o más que cualquier otro modelo matemático de computación. Algunas de las bases de esta afirmación se discuten en la sección 7 del capítulo 3. A su vez, una máquina de Turing se puede simular mediante un programa escrito en un lenguaje de programación, suponiendo infinita disponibilidad de memoria. O sea que por cada máquina de Turing hay un programa de computadora que hace lo mismo que esa máquina de Turing y viceversa. Así resulta que con sus máquinas, Turing definió todo lo computable. Podemos observar que es una manera de definir parecida a la que se suele citar del psicólogo Binet (1857-1911) [1], quien diseñó un test de inteligencia y ante la pregunta sobre qué es la inteligencia respondió "inteligencia es lo que mide mi test". Una definición de este tipo puede no ser exitosa si más adelante aparece un test más satisfactorio para medir lo que se entiende informalmente por inteligencia. Pero la Tesis de Church y Turing resultó exitosa porque no se encontró un problema que corresponda a la idea de computable, que no se pueda resolver con una máquina de Turing.

En el estudio de las posibilidades y limitaciones de la computabilidad se suele usar la máquina de Turing como referencia, pero la equivalencia con los programas de computadora hace que nos podamos imaginar más fácilmente hoy en día qué significa que un problema sea computable, qué es una información de entrada o de salida, y también qué significa que la ejecución de un algoritmo termine en error o ingrese en un ciclo infinito.

2.1. Límites de la computabilidad

¿Cuál es el máximo común divisor de 45 y 75? ¿Cuál es el décimo dígito en la expresión decimal del número π? Son problemas computables porque una máquina de Turing, o equivalentemente, un programa de computadora, puede dar la respuesta en un número finito de pasos. La máquina que resuelve un problema computable no necesariamente se detiene sino que tal vez ejecuta un ciclo infinito que da la solución en cierto paso o en una sucesión de pasos. Es tal como lo imaginamos en una computadora moderna, pero recordando que asumimos infinita disponibilidad de memoria. Ahora bien, ¿hay problemas no computables? Seguramente sí, en temas sentimentales o en ciencias no exactas. Pero ¿hay problemas bien definidos matemáticamente cuya respuesta no puede conseguirse con un programa de computadora? La respuesta es sí, como lo demostró el mismo Turing. De manera que tanto dio una definición satisfactoria de computabilidad como mostró sus limitaciones.

2.2. Problemas de decisión y decidibilidad

Para ver las limitaciones de la computabilidad vamos a definir un caso particular de computabilidad, que es la decidibilidad. Decimos que un proble-

ma de decisión es un problema que puede tener una información de entrada (también llamada input o instancia del problema) y plantea una pregunta cuya respuesta posible es solamente sí o no. Un problema de decisión se dice decidible si es computable, es decir, si se puede llegar a la respuesta correcta (sí o no) mediante una máquina de Turing o mediante un programa en una computadora moderna. Por ejemplo, un problema de decisión es determinar si el n-ésimo dígito de π es 7 o no es 7, y debido a que existe un algoritmo que, para cualquier input n, devuelve el n-ésimo dígito de π, el problema es decidible. Comparando este dígito con el 7, se puede saber si la respuesta es sí o no. El llamado *halting problem*, que vamos a analizar, es un problema de decisión no computable, es decir, un problema indecidible.

3. El problema de detención (halting problem)

Notemos que si un programa no se detiene después de cierta cantidad de pasos, pueden ocurrir dos cosas:

1) que todavía no haya llegado a un resultado (o a una situación de error en la que se detendrá),

2) que nunca vaya a detenerse.

Una cuestión importante sobre estas dos posibilidades es que no siempre podemos determinar cuál es el caso en un tiempo finito dado. Si en cierto momento un programa está ejecutándose, puede ser que se vaya a detener más adelante o que nunca se detenga, y habría que conocer muy bien el programa para decir cuál va a ser el caso. Con algunos programas esto es sencillo. Por ejemplo, puede escribirse un programa que tome un número x y calcule $2x + 1$. Podríamos demostrar que cuando lo ejecutamos en cierta computadora se va a detener en un número finito de pasos. También podemos determinar si hay o no detención en el caso de un programa en el que si el input x es 1 se entra en un ciclo infinito (por ejemplo mandando a ejecutar la instrucción número 10, que consiste en mandar a ejecutar la instrucción número 10) mientras que si el input es un valor x distinto de 1, el programa devuelve el resultado de $2x + 1$. En este caso tendremos un programa del que sabemos para cualquier input si se va a detener o no. Pero no siempre es tan sencillo. Por algo las computadoras quedan a veces atrapadas en ciclos muy largos o infinitos (lo que se suele llamar “colgadas") sin que esa sea la intención del fabricante ni del usuario. Programando con prolijidad se puede prever con bastante éxito que la computadora va a entrar en un ciclo infinito, pero esto no significa resolver un problema más general, el problema de detención o *halting problem*, que consiste en determinar, ante cualquier programa y con cualquier input si va a haber detención o no. Veremos que es relevante y demostraremos que es indecidible.

3.1. Programas que procesan programas, máquinas que analizan máquinas

El *halting problem* plantea si hay posibilidad o no de que un programa sea capaz de analizar cualquier programa y cualquier input para este programa y responder si habrá detención. Notemos que no es poco usual que un programa sea analizado por otro, es decir, que un programa sea el input de otro. En efecto, los programas llamados compiladores traducen programas de un lenguaje de computación a otro y pueden detectar errores. Tampoco es imposible que un programa sea input de sí mismo. Por ejemplo un programa podría tomar una copia de sí mismo y encontrar definiciones de variables que no cumplen función alguna, con el fin de eliminarlas. Como otro ejemplo, podemos imaginarnos un manual escrito en español para traducir del español al francés, el cual incluiría reglas sintácticas y un diccionario. Siguiendo sus instrucciones podríamos traducir ese mismo manual del español al francés.

Además, en el mundo de las computadoras, toda información (datos e instrucciones) puede traducirse a números, incluso se puede codificar solamente con cadenas de ceros y unos en las máquinas de Turing y en las computadoras digitales modernas. Dado que los dígitos del 0 al 9, las letras y los símbolos pueden codificarse con cadenas de ceros y unos, tanto las instrucciones que dimos como ejemplo en la sección 1 como cualquier máquina de Turing se pueden representar mediante cadenas de ceros y unos.

Para el funcionamiento de una de estas máquinas solo hace falta su conjunto de instrucciones, el pequeño número de operaciones que propuso Turing, común a todas sus máquinas, y cierta información binaria de entrada en la cinta. Ese conjunto de instrucciones que definen una máquina en particular puede a su vez estar escrito en la cinta de otra máquina que lo procesa, tal como una función de un lenguaje de programación invoca dentro de sí misma a otra función con el fin de utilizarla para un cálculo auxiliar o quizá para analizarla con el fin de detectarle errores o traducirla a otro lenguaje, por ejemplo. De manera que tanto datos como máquinas o programas pueden representarse como cadenas de ceros y unos, y por eso no hay problema en incorporarlos como inputs.

Turing vislumbró esta posibilidad sin llevarla a la práctica mediante la construcción de máquinas, pero dedujo sus consecuencias. En las computadoras modernas se lleva a cabo el procesamiento en binario tanto de datos como de programas pero los ceros y unos no quedan a la vista del usuario. Cero y uno son dos estados posibles, electrónicamente diferentes que ocupan cada sitio de la memoria de las computadoras digitales representando toda la información que manejan. Lo importante es la posibilidad de

representar datos e instrucciones con cadenas de caracteres, es decir, que si las computadoras utilizaran un lenguaje de tres o de más signos en lugar de dos, sus posibilidades y limitaciones serían las mismas, de hecho se podría demostrar su equivalencia con las computadoras binarias mediante una traducción de un sistema a otro.

3.2. Indecidibilidad del problema de detención

Basados en la demostración de Turing, vamos a probar a continuación que el *halting problem* es indecidible. Lo enunciamos en su versión para máquinas de Turing. Para programas en computadoras actuales el enunciado y la demostración son análogos en virtud de la equivalencia. "Máquina de Turing" se puede reemplazar por "algoritmo", "programa", "subrutina", o "función" de un lenguaje de programación.

Problema de detención (halting problem): ¿existe una máquina de Turing que pueda decidir ante una máquina de Turing cualquiera T y cualquier input x si la ejecución de T con input x se va a detener?

Como demostraremos que la respuesta es no, tendremos que el *halting problem* es indecidible. Es decir, que no existe tal máquina que pueda contestar para toda máquina y todo input si habrá detención ni tampoco existe un gran programa de computadora que dado cualquier programa y cualquier input pueda determinar si el programa que se está analizando se va a detener.

Supongamos que existe una máquina de Turing que recibe como entradas una máquina de Turing y un input, y decide si la ejecución de dicha máquina con el input dado se detiene o no. A esta máquina que resolvería el *halting problem* la llamamos H. Si el programa se detiene, H devuelve un SI, de lo contrario devuelve un NO (véase la figura 2.1).

Para realizar la demostración supondremos que existe H y llegaremos a una contradicción.

Antes de seguir es interesante señalar una demostración de inexistencia por el absurdo similar, la de la paradoja del barbero, que fue utilizada por Bertrand Russell (1872-1970) [2] en temas de fundamentos de la matemática. Esta paradoja consiste en considerar que en un pueblo hay un barbero que afeita a todos aquellos que no se afeitan a sí mismos. Lo que hay que notar es que este barbero no puede existir, porque si se afeitara a sí mismo no cumpliría la condición de afeitar a todos los que no se afeitan a sí mismos y si no se afeitara a sí mismo tampoco la cumpliría.

Ahora llegaremos a una contradicción basados en la existencia de la máquina H. Para eso construimos una máquina más grande que llamamos A, la cual nos va a llevar a un absurdo. Esta máquina A consiste en H precedida por una máquina duplicadora D y sucedida por una máquina contrariante (u opositora) C en la salida. La máquina D simplemente duplica el in-

put que recibe. Si recibe x como entrada, devolverá dos x como salida (véase la figura 2.1). La máquina C no es un simple negador lógico (que cambia SI por NO y NO por SI) sino que si recibe SI ("se detiene") como input ejecuta un ciclo infinito y si recibe NO ("no se detiene") como input, entonces se detiene y devuelve un cero u otro número (es irrelevante cuál es la salida, lo importante es que se detiene) (véase la figura 2.1).

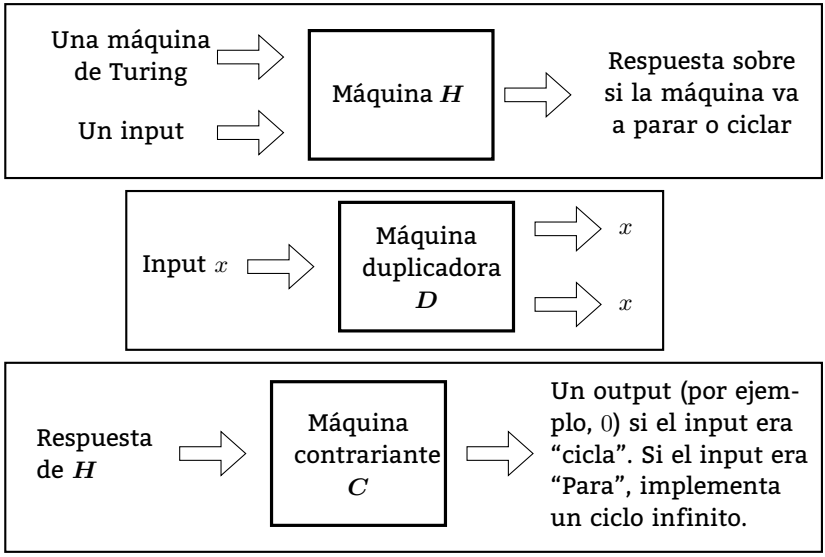


Figura 2.1. Máquinas H, D y C.

Construimos A acoplando las máquinas D, H y C (véase la figura 2.2):

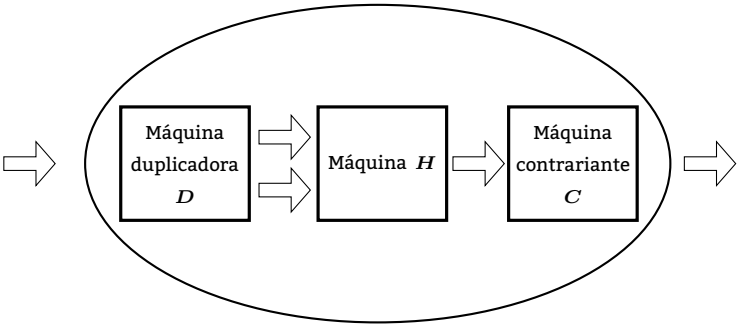


Figura 2.2. Máquina A.

Ahora supongamos que a la máquina A le introducimos A como input. Si la máquina A se detiene, entonces H debe devolver un SI al analizar A con

input A y hacer que la máquina C implemente un ciclo infinito, contradiciendo el hecho de que la máquina A se detiene. Por otro lado, si la máquina A no se detiene, entonces H debe devolver un NO, y por lo tanto, la máquina C devuelve un cero haciendo que la máquina A termine deteniéndose, contradiciendo que A no se detiene. Ocurrió entonces que suponiendo que A se detiene, llegamos a que no se detiene y suponiendo que no se detiene, llegamos a que se detiene. Es un absurdo, similar al de la paradoja del barbero, que provino de suponer que existe una máquina con la capacidad de H, es decir, provino de suponer que el *halting problem* se puede resolver con una máquina de Turing.

Esta demostración, basada en la original de Turing, es sorprendente porque no incursiona en detalles de las instrucciones de los programas o de las posibles combinaciones entre ellas. Turing no hizo una demostración analizando el funcionamiento de las máquinas y tratando de vislumbrar con ese análisis sus limitaciones, sino que dio una demostración por el absurdo.

El *halting problem* puede parecer de poca aplicabilidad o muy restringido al mundo de las computadoras. Uno puede preguntarse: ¿existen otros problemas indecidibles importantes? Sí, por ejemplo algunos de gran interés matemático como el problema de correspondencia de Post (sobre la posibilidad de alineamiento de secuencias) o el décimo problema de Hilbert (sobre la existencia de soluciones enteras de ecuaciones con coeficientes enteros). Sobre este último, se puede consultar la sección 4 del capítulo 3. Se puede demostrar que son indecidibles, que la ilusión de resolverlos con un programa de computadora o equivalentemente con un algoritmo formal (que eso es un programa de computadora) debe ser lógicamente desechada. Ante estos temas surge a menudo la inquietud: ¿no podrán resolverse estos problemas algún día con computadoras más poderosas? Como en el caso del *halting problem*, se puede demostrar que no. Cuando existe un algoritmo, éste puede ser más sencillo o más complejo pero es capaz de dar la respuesta. Para problemas no computables, no existe un algoritmo que se pueda implementar en un lenguaje de programación.

Breves biografías de los autores

Ana Maffei es doctora en Ciencia y Tecnología (UNGS) y profesora universitaria en Matemática (UNGS). Actualmente se desempeña como profesora adjunta, dedicación exclusiva, en la Universidad Nacional de Luján (UNLU). Participa en los proyectos de investigación "Procesos estocásticos, grafos aleatorios y aplicaciones" (UNLu) y "Probabilidad, estadística y minería de datos" (UNGS).

Tomás Tetzlaff es doctor en Computación (UBA), especialista en Estadística (UBA), licenciado en Biología (UBA) y licenciado en Matemática (UNLP).

Actualmente se desempeña como investigador docente del Instituto de Ciencias de la Universidad Nacional de General Sarmiento. Su tarea de investigación es interdisciplinaria con una producción científica que aborda problemas propios de los procesos estocásticos, el estudio de plagas, la genética de la hemofilia, árboles aleatorios y aplicaciones de teoría de grafos. En este sentido, Tomás Tetzlaff comparte con la obra de Turing el interés por los temas matemáticos y de biología (Turing, hacia el final de su vida, hizo aportes fundamentales a la comprensión del origen de las formas periódicas y recurrentes en biología). Tetzlaff sostiene que "los ciclos aparecen en la naturaleza y son de gran utilidad teórica y práctica en matemática y programación. Tienen un valor muy especial porque con una definición finita capturan la idea de infinito".

Referencias

[1] A. Binet, T. Simon y E. S. Kite. *The Development of Intelligence in Children: (the Binet-Simon Scale)*. Classics in psychology. Williams & Wilkins, 1916.

[2] J. van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Source books in the history of the sciences. Harvard University Press, 1995.

[3] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". En: *Proceedings of the London Mathematical Society* S2-42.1 (1937), págs. 230-265.

Capítulo 3
El sueño de Hilbert, las máquinas de Turing y los teoremas de Gödel

Gustavo Piñeiro

Un viejo matemático francés decía: "No puede considerarse completa una teoría matemática hasta que no se haya hecho tan clara que pueda explicarse al primer hombre que encontremos por la calle".
(David Hilbert, conferencia inaugural del
II Congreso Internacional de Matemática, París, 1900)

1. Introducción

Los avances en física y matemática a lo largo de los siglos xviii y xix fueron cimentando, dentro de la comunidad científica, un optimismo ilimitado en el poder de la mente humana. La mecánica de Newton, por ejemplo, dio paso a un conocimiento sin precedentes en materia de física y de astronomía. En matemática, la creación y la fundamentación del análisis, así como la creación del álgebra abstracta, abrieron el camino a nuevas y espectaculares investigaciones. En esos años, muchos problemas matemáticos que llevaban siglos sin solución (como la cuadratura del círculo o la resolubilidad de las ecuaciones de quinto grado) fueron finalmente resueltos.

La capacidad de saberlo todo, de responder todas las preguntas, parecía al alcance de la mano. Tanto es así, que Pierre-Simon de Laplace (1749–1827) escribió en 1814 que, si conociéramos la posición y la velocidad de todas las partículas del universo, y poseyéramos la suficiente capacidad de cálculo, podríamos conocer totalmente la historia pasada y futura del cosmos (véase [11]).

En ese contexto, a principios del siglo xx, el gran matemático alemán David Hilbert (1862–1943) se permitió proponer la búsqueda de un método claro, concreto y objetivo que permitiera resolver todos los problemas matemáticos que pudieran llegar a ser planteados alguna vez; un método que

asegurara que ningún problema quedaría jamás sin respuesta. Nuestra historia trata sobre esta propuesta de Hilbert, sobre las ideas que la motivaron, y sobre cómo fue respondida.

Este relato comienza con la primera formulación del sueño de Hilbert, en su famosa conferencia de París, en el año 1900.

2. La conferencia de Hilbert

Durante el siglo xix y hasta principios del xx las dos grandes potencias en la investigación matemática fueron Francia y Alemania. El predominio actual de los estadounidenses comenzó después de la Segunda Guerra Mundial y es consecuencia, por un lado, de su aplastante triunfo militar, político y económico, pero sobre todo se debe a que durante la década de 1930 grandes mentes alemanas y francesas huyeron a Estados Unidos, donde formaron a las generaciones posteriores de científicos.

No es sorprendente, entonces, que el I Congreso Internacional de Matemática, realizado en Zúrich en 1897, fuera organizado por los alemanes, y que el II Congreso Internacional de Matemática (París, 1900) fuera organizado por los franceses. De hecho, el tercer congreso, ocurrido en Heidelberg en 1904, también fue organizado por los alemanes.

En un congreso científico, y mucho más en un congreso internacional, un momento muy destacado es la conferencia inaugural. Solo las personalidades más relevantes de la comunidad científica fueron invitadas a ocupar ese lugar de honor. En 1897, como gesto de buena voluntad, los organizadores alemanes invitaron a dictar la conferencia inaugural del I Congreso Internacional de Matemática a Henri Poincaré (1854–1912), el matemático francés más importante de ese tiempo, y uno de los más notables de toda la historia. En una devolución de gentilezas, en 1900 los franceses invitaron a dictar la conferencia inaugural a David Hilbert, el más destacado de los matemáticos alemanes de ese tiempo, y también uno de los más importantes de la historia.

Dicen los comentarios de la época que la conferencia de Poincaré no fue tan brillante como se esperaba. Seguramente en boca de cualquier persona normal hubiera sido considerada una disertación extraordinaria, pero, en comparación con lo que se esperaba del gran genio, resultó un poco decepcionante. Al recibir la invitación de París, David Hilbert decidió que su conferencia superaría a la de Poincaré (porque franceses y alemanes intercambiaban gentilezas, pero también eran rivales), y que expondría ideas que serían estudiadas durante muchos años. Puede decirse que Hilbert logró ampliamente su objetivo porque, tal como él deseaba, su conferencia influyó en el pensamiento matemático a lo largo de las décadas siguientes, y todavía hoy es motivo de estudio y análisis.

3. El décimo problema

Cuando un matemático investiga siempre tiene en mente un problema que desea resolver. Los métodos que crea y los conceptos que define siempre están al servicio de un problema concreto. La investigación matemática tiene necesidad de buenos problemas. Por ese motivo los grandes matemáticos no son solo aquellos que pueden resolverlos, sino también aquellos que son capaces de plantear problemas interesantes.

En su conferencia inaugural del II Congreso Internacional de Matemáticas, David Hilbert propuso nada menos que veintitrés problemas centrales en distintas áreas de la matemática. Estos problemas, él entendía, guiarían buena parte de la investigación matemática a lo largo del siglo xx. Dos de ellos no despertaron (hasta ahora, al menos) tanto interés como Hilbert esperaba, pero todos los demás, algunos de los cuales permanecen todavía sin resolver, fueron muy influyentes a lo largo de todo el siglo pasado, y hasta el día de hoy "resolver un problema de Hilbert" es sinónimo de ganarse un lugar en la historia de la matemática. (La lista completa de los problemas, con comentarios sobre ellos, así como la traducción de la conferencia de Hilbert pueden verse en [7]). De los veintitrés problemas de Hilbert nos interesa especialmente el décimo, que dice así:

Dada una ecuación diofántica con cualquier número de incógnitas y con coeficientes enteros, idear un procedimiento de acuerdo con el cual pueda determinarse, en un número finito de operaciones, si la ecuación tiene soluciones enteras.

¿Qué significan los términos de este enunciado? Una ecuación diofántica (el nombre se debe a Diofanto de Alejandría, matemático griego del siglo iii quien fue el primero en estudiarlas) es, técnicamente, una ecuación polinómica con coeficientes enteros. En otras palabras, es una ecuación en la cual las incógnitas aparecen sumadas, restadas, multiplicadas entre sí, o elevadas a potencias enteras positivas, y en la que solo figuran coeficientes enteros. Por otra parte, como dice el enunciado del problema, en una ecuación diofántica solamente nos interesan las soluciones dadas por números enteros. Veamos dos ejemplos:

$$2xy = 4x + 1, \quad x^3 + y^3 + z^3 = k,\ k \in \mathbb{Z}.$$

Con la restricción de que las incógnitas solo pueden representar números enteros, no es difícil comprobar que la primera ecuación no admite soluciones. Porque si x e y son enteros, entonces $2xy$ es un número par, mientras que $4x + 1$ es impar, por lo que es imposible que $2xy$ y $4x + 1$ sean iguales.

La segunda expresión representa, en realidad, toda una familia de ecuaciones: una ecuación diferente para cada valor entero de k. En 1954, un grupo de matemáticos de la Universidad de Cambridge planteó el desafío de

determinar si la ecuación correspondiente a cada valor de k entre 1 y 100 tiene, o no, solución. En algunos casos la respuesta es evidente. Por ejemplo, es claro que $x^3 + y^3 + z^3 = 1$ sí tiene solución. Un poco menos obvio es el hecho de que $x^3 + y^3 + z^3 = 41$ no admite soluciones (para demostrarlo se puede analizar los posibles restos al dividir por 9 de los números involucrados).

Con el correr de los años, se fue determinando para qué valores de k había solución, y para cuáles no (con demostraciones, a veces, de una gran densidad teórica). Hasta hace muy poco tiempo el último valor que todavía se resistía era 42. Finalmente, en septiembre de 2019 (véase [10] y [13]), Andrew Sutherland (del MIT) y Andrew Booker (de la Universidad de Bristol), mediante un trabajo que llevó millones de horas de cálculos computacionales, hallaron la siguiente solución para $x^3 + y^3 + z^3 = 42$:

$$\begin{aligned} x &= -80.538.738.812.075.974 \\ y &= 80.435.758.145.817.515 \\ z &= 12.602.123.297.335.631\,. \end{aligned}$$

4. Primera versión del Entscheidungsproblem

Dada una ecuación diofántica cualquiera, el décimo problema de Hilbert pide "idear un procedimiento de acuerdo con el cual pueda determinarse, en un número finito de operaciones, si la ecuación tiene soluciones enteras". ¿Qué entiende Hilbert por un "procedimiento"?

Hilbert llama "procedimiento" al concepto que hoy en día se conoce como un "procedimiento efectivo" o, más comúnmente, como un "algoritmo". ¿Qué es, entonces, un algoritmo? Intuitivamente, es una receta que puede seguirse mecánicamente paso a paso sin necesidad de pensar. Los programas de computadora, o las aplicaciones de los celulares, son ejemplos bien conocidos de algoritmos. Por este motivo, podemos traducir el décimo problema de Hilbert al lenguaje del siglo xxi de la siguiente manera: ¿es posible programar una computadora de tal modo que, dada una ecuación diofántica cualquiera, determine en una cantidad finita de pasos (o una cantidad finita de tiempo) si la ecuación tiene, o no tiene, solución?

Este problema es, históricamente, la primera versión del famoso *Entscheidungsproblem*, o problema de la decisión, de Hilbert. Como se dijo en el capítulo 2, en su forma más general, el *Entscheidungsproblem* propone la siguiente cuestión. Tomemos una teoría matemática cualquiera y consideremos, dentro de esa teoría, una familia amplia de preguntas que pueden ser respondidas por "sí" o por "no". El *Entscheidungsproblem* plantea si existe un algoritmo que permita responder, en una cantidad finita de pasos, cualquiera de esas preguntas. En el caso particular del décimo problema, tene-

mos una pregunta para cada ecuación diofántica, que es, obviamente, si la ecuación tiene solución.

En una forma más rigurosa, el *Entscheidungsproblem* plantea, dado un sistema de axiomas y un enunciado matemático P, si existe un algoritmo que determine, en una cantidad finita de pasos, si P es demostrable a partir de esos axiomas. Esta formulación equivale a la que comentamos en el párrafo anterior si entendemos que los axiomas son la base de alguna teoría matemática y el enunciado P se ve como una conjetura de la que quiere saberse si es verdadera o falsa.

Volvamos ahora al décimo problema. Una propuesta para resolverlo podría consistir en una búsqueda por "fuerza bruta". Para entender la idea, tomemos, a modo de ejemplo, la ecuación $x^3 + y^3 + z^3 = 42$. Como hay tres incógnitas, cada solución posible puede escribirse como una terna de números enteros (x, y, z). En una búsqueda por "fuerza bruta" se prueba una por una cada terna posible hasta hallar una solución.

Para sistematizar la búsqueda, el algoritmo puede comenzar con todas las ternas en las cuales la suma de los valores absolutos de los números es 0. Hay solo una: $(0, 0, 0)$. A continuación, verifica si esta terna es solución de la ecuación. En este caso, no lo es. Como no encontró una solución, el algoritmo pasa a analizar todas las ternas en la cuales la suma de los valores absolutos es 1. Hay seis en total:

$$(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1).$$

Verifica si cada una de ellas es solución, y comprueba que ninguna lo es. Pasa, entonces, a analizar las 18 ternas en las cuales la suma de los valores absolutos es 2. El algoritmo sigue así hasta que encuentra una terna que sea solución. Cuando esto último sucede imprime "sí" como respuesta y se detiene. En el ejemplo de la ecuación $x^3 + y^3 + z^3 = 42$ el algoritmo se detendrá cuando llegue a la terna:

$$\begin{aligned} x &= -80.538.738.812.075.974 \\ y &= 80.435.758.145.817.515 \\ z &= 12.602.123.297.335.631\,. \end{aligned}$$

Es interesante observar que el algoritmo que usaron Sutherland y Booker para hallar esta solución no consistió en una búsqueda por "fuerza bruta", sino en una búsqueda más sutil, que aprovechaba atajos teóricos muy potentes. Cualquier computadora que implemente el algoritmo de búsqueda por "fuerza bruta" tardaría millones de años en llegar hasta esa terna por lo que carecería de toda utilidad en un sentido práctico. Sin embargo, Hilbert quería, en principio, una respuesta teórica al problema, y no necesariamente una solución práctica. En otras palabras, el décimo problema pregunta si

existe un algoritmo que nos dé siempre una respuesta en algún lapso de tiempo finito, sin que importe si ese lapso es enormemente largo en comparación con una vida humana.

De todos modos, más allá de esta consideración, la verdad es que la búsqueda por "fuerza bruta" no resuelve el décimo problema de Hilbert. Para entender por qué, supongamos que ingresamos a la computadora una ecuación que no tiene solución (aunque nosotros no lo sabemos, porque justamente queremos determinar si la solución existe o no). En ese caso, la búsqueda por "fuerza bruta" no nos dará nunca una respuesta; pasarán los días, los meses, los siglos y los milenios sin obtener una solución, y sin saber si alguna vez la obtendremos. La búsqueda por "fuerza bruta", entonces, contrariamente a lo que pide el problema de Hilbert, no nos garantiza que obtendremos una respuesta en una cantidad finita de tiempo.

5. ¿Qué es un algoritmo?

La ecuación $x^3 + y^3 + z^3 = 41$ y la ecuación $x^3 + y^3 + z^3 = 42$ son, a primera vista, muy similares, sin embargo, como acabamos de ver, están en condiciones bien diferentes, la primera no tiene solución, mientras que la segunda tiene una solución enorme que la comunidad matemática tardó décadas en hallar. Por otra parte, el razonamiento que prueba que $x^3 + y^3 + z^3 = 41$ no tiene solución es distinto del que prueba que $2xy = 4x + 1$ no la tiene (otras demostraciones de inexistencia de solución son todavía más diversas y complejas).

Estas situaciones tan diferentes sugieren que es difícil que haya un algoritmo que pueda determinar siempre, en una cantidad finita de pasos, si una ecuación diofántica tiene, o no tiene, solución. Pero Hilbert, claro está, no buscaba una respuesta intuitiva al problema; él pedía, o bien una demostración matemática de que existe un algoritmo como el planteado (una forma de hacerlo sería mostrar explícitamente un algoritmo así), o bien una demostración matemática de que tal algoritmo no puede existir.

Ahora bien, para hacer una demostración matemática referida a algún concepto en particular se necesita tener, en primer lugar, una definición rigurosa de ese concepto. El décimo problema de Hilbert, y el *Entscheidungsproblem* en general, llevaron, entonces, a la necesidad de definir en términos matemáticamente precisos qué es un algoritmo. Como ya dijimos, un algoritmo es, informalmente hablando, una serie de instrucciones que pueden seguirse mecánicamente "sin necesidad de pensar". El nuevo desafío consistía, entonces, en traducir esta idea informal a términos matemáticamente exactos. Vamos a concentrarnos ahora en este nuevo problema, más adelante volveremos a las soluciones del *Entscheidungsproblem* y del décimo problema de Hilbert.

Los primeros en dar una definición rigurosa del concepto de algoritmo fueron, independientemente uno del otro, Alonzo Church (1903–1995) en 1936, y Alan Turing (1912–1954) en 1937 (véase [3, 2, 19]). Church publicó su trabajo algunos meses antes, cuando Turing todavía no había terminado de escribir su artículo, pero Turing decidió no leerlo para no verse influido por otras ideas.

Aunque las dos definiciones son equivalentes, en el sentido de que son maneras distintas de expresar el mismo concepto, la concepción de Turing no solo es intuitivamente más satisfactoria, sino que además fue el puntapié inicial para el diseño de las primeras computadoras electrónicas. Por estos motivos, nos dedicaremos a estudiar solamente la definición de Turing. Los lectores interesados en la definición de Church pueden consultar [3].

6. La máquina de Turing

Imaginemos que queremos usar un algoritmo para resolver un problema o para realizar un cálculo. Un problema podría ser, por ejemplo, dado un número entero positivo, determinar si es primo, o no. En este tipo de situación siempre entran en juego tres elementos.

El primer elemento es la entrada o el input del algoritmo, que está constituida por los datos iniciales. Por ejemplo, en el problema de determinar si un número es primo, la entrada sería el número en cuestión. Cuando trabajamos con una calculadora y tecleamos los números y las operaciones que queremos realizar, estamos introduciendo el input en el algoritmo que está programado en el aparato.

El segundo elemento que entra en juego es la salida o el output, que es la respuesta del problema, o el resultado del cálculo. En el caso del ejemplo de determinar si un número es primo, la salida es "Sí" si el número realmente es primo, y "No" en caso contrario. En el ejemplo de la calculadora, la salida es el resultado que nos muestra el visor cuando apretamos la tecla "=".

El tercer elemento, finalmente, está formado por las instrucciones que constituyen al algoritmo en sí, y que indican qué acciones se deben realizar a partir de la entrada para obtener, en una cantidad finita de pasos, la salida correcta.

Para expresar de manera rigurosa estos conceptos Turing creó la noción hoy conocida como *máquina de Turing*[*]. Expliquemos en qué consiste esta idea fundamental.

Para comenzar, podemos pensar a la máquina de Turing como una computadora abstracta, la cual, como cualquier otra computadora, tiene un *hardware* y un *software*. El *hardware*, en este caso, consiste en una cinta que se extiende infinitamente hacia la derecha y hacia la izquierda, y que está

[*] Veáse también el capítulo 2.

dividida en celdas o casillas, todas iguales entre sí (véase la figura 3.1). Es interesante observar que un objeto así no puede existir en la realidad física (puede haber cintas de longitudes finitas muy grandes, pero nunca una de longitud infinita), y es por ese motivo que hablamos de una computadora teórica o abstracta.

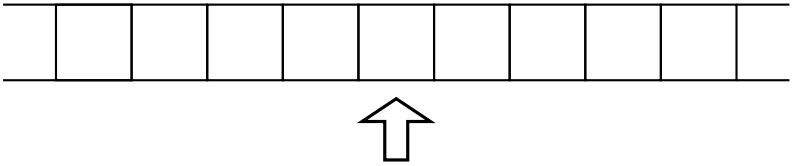


Figura 3.1. Cinta infinita con celdas y cursor.

Además de la cinta en sí, el hardware se completa con un cursor, que está representado en la imagen por una flecha. Este cursor lee el contenido de una casilla por vez, y puede, como veremos en breve, moverse sucesivamente una casilla hacia la derecha o una casilla hacia la izquierda.

Cada celda, por otra parte, puede contener una X, o bien puede estar vacía. De este modo, el contenido total de la cinta, si no está completamente vacía, tendrá siempre la forma de una cantidad finita de secuencias de símbolos X, separadas entre sí por una o más casillas en blanco (en la figura 3.2 se muestra un ejemplo).

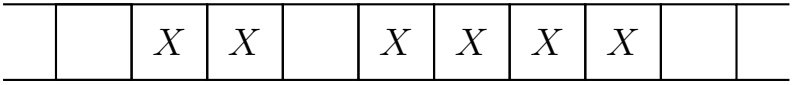


Figura 3.2. Par $(2, 4)$ representado en las celdas usando el símbolo X.

En la cinta se escribirá tanto la entrada, como la salida del algoritmo, y también servirá para realizar todas las operaciones intermedias que sean necesarias. En otras palabras, la cinta funciona como pantalla (o visor) de la máquina y también como su memoria de trabajo. Para que esto sea posible necesitamos codificar toda la información necesaria usando solamente secuencias finitas de X. Una manera estándar de hacerlo es convenir en que una X representa al número 1; XX representa al número 2; XXX representa el número 3; y así sucesivamente. Dos secuencias de X separadas por una casilla en blanco representarán un par ordenado de números; tres secuencias de X separadas entre sí por una casilla en blanco representarán una terna ordenada de números; y así sucesivamente. De este modo, por ejemplo, el contenido de la cinta de la figura 3.2 representa el par $(2, 4)$.

Si el problema que queremos resolver hiciera necesario representar en la cinta otro tipo de símbolos (por ejemplo, si se quisiera anotar como entrada

una ecuación diofántica) habría que agregar alguna convención que permitiera expresar esos símbolos mediante secuencias de números. No es indispensable entrar en detalles acerca de cómo puede hacerse esto, digamos solamente que para lograrlo podría usarse el código ASCII, u otro similar.

Una pregunta que podría surgir en este punto es por qué nos limitamos a un único símbolo X. ¿La cinta no podría contener otros símbolos? La respuesta es que sí. *A priori*, nada limita los símbolos que podemos usar; de hecho, la máquina que define Turing en su artículo de 1937 admite en su cinta ceros y unos, además de otros símbolos auxiliares. Tampoco estamos forzados a limitarnos a una única cinta; podemos trabajar con máquinas que usen dos o más cintas, cada una con su respectivo cursor, o inclusive máquinas que trabajen sobre un cuadriculado bidimensional infinito en lugar de una cinta unidimensional.

El hecho relevante, sin embargo, es que estas máquinas más "complejas" (con más símbolos o más cintas) pueden realizar las mismas tareas que las máquinas "sencillas" que estamos describiendo (con una cinta y un único símbolo X). En otras palabras, las máquinas "complejas" y las "sencillas" pueden resolver exactamente los mismos problemas y realizar exactamente los mismos cálculos. La única diferencia es que las máquinas "sencillas" lo harán más trabajosamente, y seguramente invirtiendo más tiempo. De todos modos, como ya dijimos, no nos interesan aquí las cuestiones prácticas, sino el análisis teórico de si cualquier problema puede, o no puede, ser resuelto mediante un algoritmo, y en este aspecto es equivalente trabajar con una máquina "compleja" o con una "sencilla". Finalmente, optaremos por trabajar con máquinas "sencillas" porque esto simplificará algunos de los razonamientos que haremos al estudiar la solución del *Entscheidungsproblem*.

7. La definición de Turing

Todas las máquinas de Turing que estudiaremos tienen exactamente el mismo hardware (una cinta con un cursor); la característica que diferencia una máquina de otra es su software, es decir, el programa que le dice a la máquina qué operaciones debe realizar. Mostremos un ejemplo y expliquemos a continuación su significado:

	0	X
q_1	$0Dq_1$	$0Dq_2$
q_2	$XQ(\mathsf{Fin})$	XDq_2

Un programa consta de una cantidad finita de estados (cada estado es una fila de la tabla), a los que indicaremos como $q_1, q_2, q_3, q_4 \ldots$ Cada estado consta, a su vez, de dos instrucciones. La primera instrucción nos dice qué hacer

si en ese momento el cursor está viendo una casilla vacía ("0" significa en este caso "casilla vacía"), la segunda instrucción nos dice qué hacer si la celda que está viendo el cursor contiene una X.

Cada una de estas instrucciones tiene, a su vez, tres partes. La primera nos dice qué contenido debe dejarse en la casilla que está observando el cursor ("0" si la casilla debe quedar vacía, "X" si debe contener una "X"), la segunda parte nos dice hacia dónde debe moverse el cursor ("D" si debe moverse una casilla hacia la derecha, "I" si debe moverse una casilla hacia la izquierda, "Q" si debe quedar en el mismo lugar). La tercera parte nos dice a qué estado debe pasar la máquina a continuación. Si la tercera parte dice "(Fin)", esto indica que el cómputo terminó; en ese caso, el contenido escrito en la cinta es la salida.

Por ejemplo, en la tabla que acabamos de mostrar, si la máquina está en el estado q_1, y el cursor está viendo una casilla vacía, el programa nos dice que debemos ejecutar $0Dq_1$; es decir, la casilla debe seguir vacía, el cursor debe moverse un lugar hacia la derecha, y la máquina debe seguir en el estado q_1.

Para completar la descripción solo falta decir que, por convención, cuando comienza su trabajo la máquina se encuentra siempre en el estado q_1, la cinta contiene la entrada (traducida a secuencias de X), y el cursor está observando la primera X de la izquierda.

Imaginemos, ahora, por ejemplo, que nuestra máquina comienza a operar con la entrada $(2, 4)$. Inicialmente la cinta se verá como se muestra en la figura 3.3.

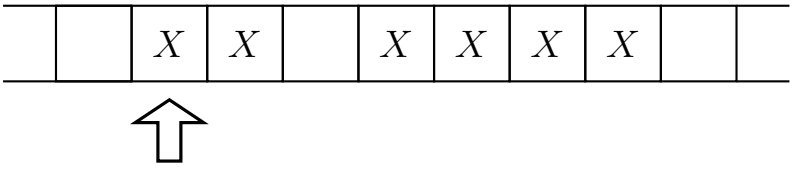


Figura 3.3. Máquina lista para correr el programa que suma, con la entrada $(2, 4)$.

Los lectores pueden verificar que, si se ejecutan mecánicamente todas las instrucciones de la tabla, la salida será la que se muestra en la figura 3.4. Es decir, si la entrada es $(2, 4)$ entonces la salida es 6. Si repiten la operación con otros ejemplos, tomando siempre como entrada un par ordenado de números naturales, se convencerán de que, si la entrada es el par (n, m), entonces la salida es el número $n + m$. En otras palabras, nuestra máquina de Turing calcula la suma de dos números naturales. (En este trabajo adoptamos la convención de considerar como naturales a los números $1, 2, 3, 4, 5, \ldots$ Una convención alternativa, elegida por muchos autores, consistiría en incluir también el 0. Ambas convenciones son válidas.)

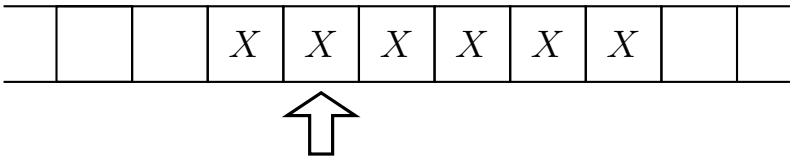


Figura 3.4. La cinta muestra el resultado $6 = 2 + 4$ después de correr el programa.

Todos sabemos que existen algoritmos para calcular la suma de dos números naturales, de hecho, los aprendemos en la escuela primaria. Nuestro ejemplo de máquina de Turing realiza exactamente la misma tarea, en el sentido de que, si se toma la misma entrada, entonces en ambos casos (tanto en el algoritmo escolar como la máquina de Turing) se obtiene como resultado la misma salida.

La llamada tesis de Turing (también llamada, como se dijo en el Capítulo 2, tesis de Church-Turing) dice que esta situación es totalmente general: dado un algoritmo (o sea, una serie de instrucciones que intuitivamente entenderíamos como un algoritmo) existe siempre una máquina de Turing que representa ese algoritmo: si a la máquina se le introduce la misma entrada que al algoritmo, entonces ambos entregan la misma salida.

La tesis de Turing no puede ser demostrada matemáticamente, porque para demostrar que todo algoritmo equivale a una máquina de Turing se necesitaría previamente una definición rigurosa del concepto de algoritmo. Pero justamente es la tesis de Turing la que provee esa definición. Como máximo, podemos aspirar (como hace Turing en su artículo de 1937) a dar argumentos que nos convenzan de que la tesis es plausible. Mencionemos tres de ellos. El primer argumento, que Turing incluye en su artículo, es que todas las definiciones de algoritmo que se han dado desde 1937 hasta ahora son equivalentes entre sí. Esto nos sugiere que la máquina de Turing abarca todos los procesos algorítmicos posibles.

El segundo argumento, que también desarrolla Turing, es que, cuando aplicamos un algoritmo para realizar un cálculo, procedemos esencialmente como lo hace una máquina de Turing: vamos operando con un dígito por vez según nos lo indica una cantidad finita de instrucciones. El tercer argumento, que agregamos aquí, es que, hasta ahora, a pesar de las décadas de familiaridad con programas de computadoras o aplicaciones de celulares (que son todos algoritmos) nadie ha podido ni siquiera imaginar un procedimiento algorítmico que no pudiera ser representado mediante una máquina de Turing. La tesis de Turing, en definitiva, suele aceptarse como verdadera, en la forma de una suerte de postulado de la informática teórica. (Por otra parte, una pregunta filosófica que planteó el trabajo de Turing y que todavía es motivo de intenso debate es si la actividad del cerebro humano equivale a una máquina de Turing. Dado que ese tema excede los objetivos del presente trabajo, no lo trataremos aquí.)

8. Entradas válidas

Nuestra intención, en las próximas secciones, es explicar la solución del *Entscheidungsproblem* de Hilbert, pero antes necesitamos exponer algunos conceptos previos. Para comenzar, tomemos, a modo de ejemplo, la siguiente máquina de Turing.

	0	X
q_1	$0Dq_1$	$0Dq_2$
q_2	$0Qq_2$	XQ(Fin)

Los lectores pueden comprobar que, si la entrada es un número natural n (expresado como una secuencia formada por símbolos X), entonces la salida es el número $n-1$. En otras palabras, si $n \neq 1$ esta máquina calcula el predecesor de n. ¿Qué sucede si la entrada es el número 1, es decir, una sola X? No es difícil verificar que en ese caso la máquina llega a la primera instrucción de la segunda fila, $0Qq_2$, y allí se queda para siempre. El cómputo nunca alcanza la instrucción (Fin), por lo que la máquina nunca se detiene ni entrega una salida.

Basados en este ejemplo, diremos que un número natural n es una *entrada válida* para una máquina de Turing si, al recibir ese número como entrada, entonces la máquina, al cabo de una cantidad finita de pasos, llega a una instrucción que dice (Fin) y se detiene. En el ejemplo que hemos mostrado, todo número natural n es una entrada válida, mientras que $n = 1$ no lo es.

Los lectores pueden también comprobar que, para la máquina siguiente, todos los números pares son entradas válidas, mientras que todos los impares son no válidas. En otros términos, el conjunto de todas las entradas válidas de esta máquina de Turing es el conjunto de los números pares.

	0	X
q_1	$0QQ$(Fin)	XDq_2
q_2	$0Dq_2$	XDq_1

Además, a modo de ejercicio, pueden pensar cómo diseñar una máquina de Turing que acepte como entradas válidas solamente los números que son múltiplos de 3.

9. Conjuntos recursivos

Como es bien sabido, y ya hemos mencionado, existe un algoritmo que determina en una cantidad finita de pasos si un número natural n es primo, o no. El algoritmo más sencillo, aunque no el mejor en un sentido práctico, consiste en verificar si algún número entre 2 y $n-1$ es divisor de n (un algoritmo mejor consiste en verificar si algún número entre 2 y $\sqrt{n}$ es divisor de n).

La tesis de Turing nos dice que existe, entonces, una máquina de Turing capaz de reconocer en una cantidad finita de pasos si un número es primo, o no. Más exactamente, existe una máquina de Turing tal que, si recibe como entrada un número natural , entonces, al cabo de una cantidad finita de pasos, entrega como salida un "Sí" (que podemos codificar como X) si el número es primo, y un "No" (que podemos codificar como XX) si no es primo. Nótese que, en particular, para esa máquina todo número es una entrada válida y la salida siempre es X o XX.

La máquina de Turing que vemos más abajo nos muestra un ejemplo más concreto de la misma idea. En este caso, tenemos una máquina que da como salida X si la entrada es un número par, y XX en caso contrario.

	0	X
q_1	$XQ(\mathsf{Fin})$	$0Dq_2$
q_2	XDq_3	$0Dq_1$
q_3	$XQ(\mathsf{Fin})$	$XQ(\mathsf{Fin})$

Estos ejemplos sugieren, a su vez, la siguiente definición, que fue introducida por Emil Post (1897–1954) en 1944 (véase [16]). La definición dice que un conjunto formado por números naturales (o sea, un subconjunto de $\mathbb{N}$) es recursivo si y solo si existe una máquina de Turing tal que, cuando recibe como entrada un número n, entonces, al cabo de una cantidad finita de pasos, entrega como salida una X si n pertenece al conjunto en cuestión, y XX si n no pertenece al conjunto. Es decir, un subconjunto de $\mathbb{N}$ es recursivo si y solo si existe una máquina de Turing (o un algoritmo, o un programa de computadora) que es capaz de determinar en una cantidad finita de pasos si un número natural n cualquiera pertenece, o no pertenece, al subconjunto.

Casi cualquier subconjunto de $\mathbb{N}$ que uno pueda imaginar seguramente será recursivo. Por ejemplo, son recursivos, entre otros, el conjunto de todos los números primos, el de los pares, los impares, los que tienen resto 7 al dividir por 988, los números capicúas, los que son capicúas si se los escribe en binario, y también el conjunto de todos los números pares que pueden escribirse como suma de dos primos. ¿Habrá algún conjunto que no sea recursivo? En otras palabras, ¿hay algún subconjunto de $\mathbb{N}$ tal que sea imposible diseñar un algoritmo (o un programa de computadora) capaz de reconocer si un número pertenece, o no, a ese subconjunto? La respuesta es que sí existe un conjunto así; explicaremos por qué en las secciones siguientes.

10. Una numeración de las máquinas de Turing

Como hemos visto, cada instrucción de una máquina de Turing se escribe usando tres símbolos de esta lista: 0, X, D, I, Q, (Fin), $q_1, q_2, q_3, \ldots$ En este contexto podemos considerar a "(Fin)" como un único símbolo (podríamos

haber puesto simplemente F), y lo mismo sucede con q_1, con q_2, con q_3, y así sucesivamente. Estos símbolos, podemos decir, constituyen el *alfabeto* en el que se escriben las máquinas de Turing.

Observemos, además, que estas máquinas, que hasta ahora hemos descripto mediante tablas de dos entradas, pueden describirse también usando una única "palabra" escrita en este nuevo alfabeto. Por ejemplo, la máquina que se muestra a continuación puede definirse usando la palabra $0Dq_1 0Dq_2 XQ(\text{Fin})XIq_2$, en la que aparecen, una a continuación de la otra, las dos instrucciones de la primera fila y luego las dos instrucciones de la segunda.

	0	X
q_1	$0Dq_1$	$0Dq_2$
q_2	$XQ(\text{Fin})$	XIq_2

¿Para qué nos sirven estas reflexiones? Porque, así como el orden usual de las letras $a, b, c, d, e, f, \ldots$ nos permite ordenar alfabéticamente las palabras de nuestro idioma, de manera similar, el orden de los símbolos $0, X, D, I, Q, (\text{Fin}), q_1, q_2, q_3, \ldots$ nos permite establecer un orden "alfabético" entre las máquinas de Turing. Para establecer este ordenamiento, convenimos en que, si una palabra es más corta que otra, entonces la más corta viene siempre antes que la más larga. Por ejemplo, cualquier palabra que corresponda a una máquina de un solo estado (que tendrá seis símbolos, ya que consta de dos instrucciones) vendrá antes que cualquier palabra que corresponda a una máquina de dos estados (que tiene doce símbolos). Esto no sucede en castellano, donde "abastecer" viene antes que "bote". Por otra parte, si dos palabras tienen la misma longitud, entonces, ahora sí, tal como se hace en castellano, se deberán comparar los símbolos. Por ejemplo, a igual longitud, una palabra que comience como 0 vendrá antes que una que comience con X.

Los lectores pueden verificar que, según este ordenamiento, la primera máquina de Turing del diccionario es $0D(\text{Fin})0D(\text{Fin})$, que corresponde a la tabla que se muestra más abajo. Después vienen todas las demás máquinas de Turing de un estado (ordenadas adecuadamente), luego todas las máquinas de dos estados, y así sucesivamente.

	0	X
q_1	$0D(\text{Fin})$	$0D(\text{Fin})$

¿Cuántas máquinas de un solo estado existen? Las palabras que corresponden a las máquinas de un solo estado usan solamente los símbolos $0, X, D,\ I,\ Q, (\text{Fin}), q_1$ ya que nunca aparecen $q_2, q_3, \ldots$ Cada palabra, a su vez, tiene seis símbolos. El primer símbolo puede ser 0 o X (es decir,

hay 2 elecciones posibles); el segundo símbolo puede ser D, IoQ (o sea, 3 elecciones posibles); el tercero puede ser (Fin) o q_1 (es decir, 2 elecciones). Para el cuarto símbolo hay 2 elecciones, para el quinto hay 3 elecciones, y para el sexto, finalmente, hay 2. En conclusión, la combinatoria nos dice que la cantidad total de máquinas de Turing de un estado es: $2 \cdot 3 \cdot 2 \cdot 2 \cdot 3 \cdot 2 = 144$. En nuestro ordenamiento la primera máquina de dos estados, que es $0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})$, ocupa el lugar 145. Los lectores pueden calcular qué posición ocupa la primera de las máquinas de tres estados.

El ordenamiento establece, entonces, una asociación entre máquinas de Turing y números naturales, el cual resultará de gran utilidad. En esta asociación vamos a llamar $\boldsymbol{T}_1$ a la máquina de Turing que ocupa el primer lugar del diccionario, que es, como ya vimos, $0D(\text{Fin})0D(\text{Fin})$. La siguiente, que es $0D(\text{Fin})0D$, será $\boldsymbol{T}_2$, la siguiente será $\boldsymbol{T}_3$, y así sucesivamente. La primera máquina de dos estados, que ya habíamos visto que se codifica como $0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})$, es $\boldsymbol{T}_{145}$.

Es importante notar que, dada una máquina de Turing, el procedimiento que calcula su número es puramente mecánico, es decir, algorítmico. En otras palabras, existe un algoritmo (y, por lo tanto, una máquina de Turing) que, dada una máquina de Turing, calcula el número que le corresponde. Por ejemplo, ante la entrada $0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})0D(\text{Fin})$, adecuadamente codificada, el algoritmo nos dirá que se trata de la máquina 145. Recíprocamente, dado un número n, el algoritmo también nos devuelve el programa de la máquina $\boldsymbol{T}_n$.

Esta observación, a su vez, permite introducir un concepto muy potente, la llamada *máquina universal de Turing*; una máquina que, como su nombre sugiere, contiene en sí misma a todas las demás. Concretamente, la máquina universal es una máquina de Turing que, cuando recibe como entrada el par (n, m), escribe el programa de $\boldsymbol{T}_n$ y ejecuta lo que éste haría al recibir como entrada el número m. Es decir, la máquina universal reproduce lo que haría cualquier máquina al recibir una entrada cualquiera, por lo que ella sola es capaz de ejecutar cualquier procedimiento algorítmico. En términos informáticos, la máquina universal es un programa que contiene la descripción de todos los programas posibles, tanto los que han sido escritos, como los que se escribirán en el futuro; desde el algoritmo del buscador de Google, por ejemplo, hasta el programa que controlará la alarma de oxígeno del hábitat de los futuros colonizadores de Marte.

11. Una mirada al infinito

Nuestra intención, como ya adelantamos, es demostrar que existen conjuntos que no son recursivos. Para ello, debemos volver aún más en el tiempo, hasta las últimas décadas del siglo xix. En aquella época, aproximadamente

entre 1872 y 1897, Georg Cantor (1845–1918) desarrolló la teoría de conjuntos, y muy especialmente, la teoría que estudia los conjuntos infinitos, uno de los mayores logros intelectuales de la historia de la Humanidad.

Un punto fundamental de la teoría de Cantor es la idea de que hay "infinitos más grandes que otros". En otras palabras, Cantor demostró que los conjuntos infinitos no son todos equivalentes entre sí, sino que, en un sentido muy preciso, hay conjuntos infinitos que tienen "una mayor cantidad de elementos" que otros. Para comenzar a entender esta idea, imaginemos un salón con una cantidad muy grande de personas, y una cantidad también muy grande de sillas (véase la figura 3.5).

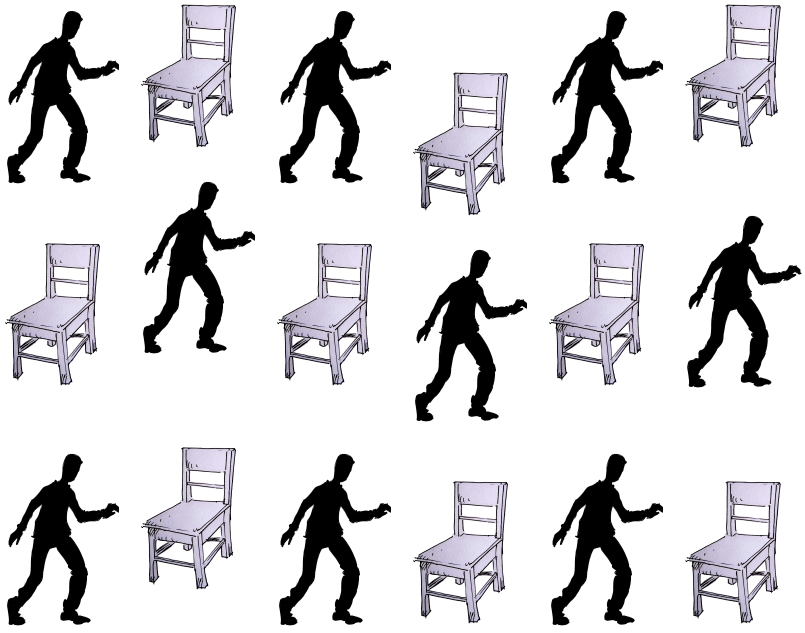


Figura 3.5. Sillas y personas.*

Es importante aclarar que estamos suponiendo que, tanto la cantidad de sillas como la de personas, aunque grandes, son ambas finitas. La palabra "finita" significa "que se termina": si comenzamos a contar una por una, las personas o las sillas, en los dos casos ese conteo terminará en algún momento (aunque pueda demorar miles de años). Por el contrario, si intentáramos contar los elementos de un conjunto infinito (palabra que quiere decir "que

*Ilustración: Ángel Jara.

no se termina"), ese conteo no finalizará nunca. Aquí la palabra "nunca" debe entenderse en el sentido más literal del término, el conteo no terminará jamás, aunque nos dediquemos a contar durante miles de millones de años.

Volvamos ahora a la situación del salón con personas y sillas, y planteemos la siguiente pregunta: ¿cómo podríamos saber si la cantidad de personas es igual a la de sillas, o si hay más de unas que de las otras? Una primera idea sería, simplemente, contar cuántas sillas y cuántas personas hay en el salón, y luego comparar los dos números. Pero este método, obviamente correcto para cantidades finitas, no podría aplicarse en el caso de conjuntos infinitos porque, como dijimos, el conteo no terminaría nunca.

Una manera más ingeniosa y más adecuada de responder la pregunta sería pedirles a todas las personas que se sienten. Si nadie queda de pie y ninguna silla queda vacía, entonces podemos concluir que había la misma cantidad de ambas. En otras palabras, las cantidades son iguales si y solo si podemos establecer, entre personas y sillas, una correspondencia uno-a-uno (véase la figura 3.6).

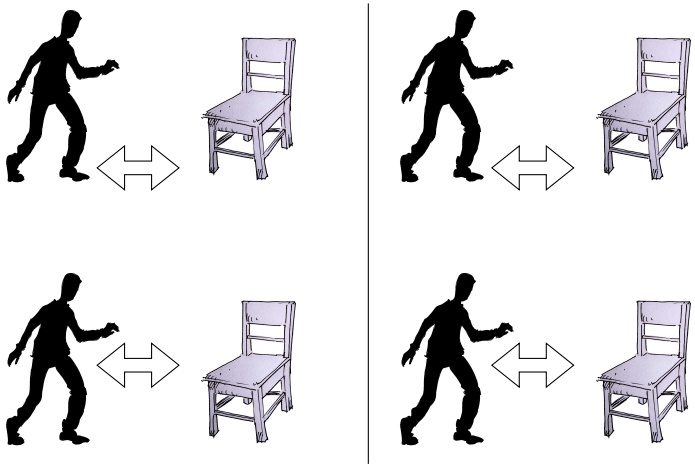


Figura 3.6. Correspondencia uno-a-uno entre personas y sillas*

La primera definición de la teoría de Cantor consiste en extender esta idea a los conjuntos infinitos. Cantor dice que dos conjuntos tienen el mismo cardinal ("cardinal" es esencialmente sinónimo de "cantidad de elementos") si y solo si es posible establecer una correspondencia uno-a-uno (es decir, un emparejamiento perfecto) entre los elementos de uno y otro conjunto. Técnicamente se usa la palabra "cardinal" en lugar de "cantidad de

*Ilustración: Ángel Jara.

elementos" porque, dado que los elementos de un conjunto infinito no pueden ser contados, entonces no se puede hablar propiamente de una "cantidad" definida de elementos. Sin embargo, en lo que sigue continuaremos apelando a la frase, más intuitiva y esencialmente correcta, de "misma cantidad".

12. Algunos ejemplos

Como primer ejemplo de la definición de Cantor, diremos que el conjunto de los números enteros $\mathbb{Z}$ tiene el mismo cardinal que el de los números naturales $\mathbb{N}$, en otras palabras, que hay la misma "cantidad" de unos y otros. A primera vista esto suena contradictorio porque $\mathbb{Z}$ contiene a todos los elementos de $\mathbb{N}$, más sus opuestos aditivos, más el 0, por lo que parecería que los enteros deberían ser más que el doble de los naturales. Sin embargo, como hemos dicho, ambos conjuntos tienen el mismo cardinal. Para los conjuntos infinitos no se aplica la regla aristotélica de que el todo es siempre mayor que la parte.

Para demostrar la equivalencia entre $\mathbb{Z}$ y $\mathbb{N}$ basta con observar la figura 3.7, donde se muestra un emparejamiento entre ambos conjuntos.

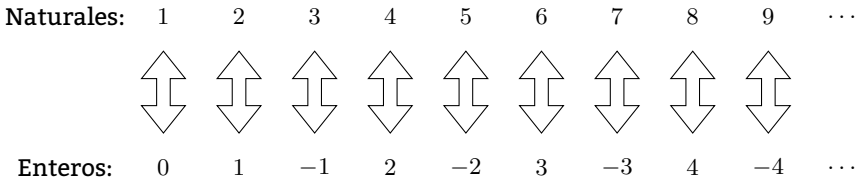


Figura 3.7. Emparejamiento uno-a-uno entre los naturales y los enteros.

En un segundo ejemplo, podemos demostrar que el conjunto de los números naturales $\mathbb{N}$ tiene el mismo cardinal que el de los racionales $\mathbb{Q}$. Para ello, comenzamos haciendo una lista de los números racionales positivos. Escribimos primero la única fracción cuyo numerador y denominador suman 2, que es $\frac{1}{1}$. Luego, escribimos las fracciones donde esa suma es 3, que son $\frac{1}{2}$ y $\frac{2}{1}$. A continuación, aquellas donde la suma es 4, que son $\frac{1}{3}$, $\frac{2}{2}$ y $\frac{3}{1}$. Pero, como $\frac{2}{2}$ representa el mismo número racional que $\frac{1}{1}$, que ya había sido anotado, la quitamos. La lista, entonces, comienza así:

$$\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{1}, \frac{1}{4}, \frac{2}{3}, \frac{3}{2}, \frac{4}{1}, \frac{1}{5}, \frac{5}{1}, \dots$$

Colocamos delante el 0, y vamos insertamos el inverso aditivo de cada número:

$$0, \frac{1}{1}, -\frac{1}{1}, \frac{1}{2}, -\frac{1}{2}, \frac{2}{1}, -\frac{2}{1}, \frac{1}{3}, -\frac{1}{3}, \frac{3}{1}, -\frac{3}{1}, \frac{1}{4}, -\frac{1}{4}, \frac{2}{3}, -\frac{2}{3}, \frac{3}{2}, -\frac{3}{2}, \frac{4}{1}, -\frac{4}{1}, \dots$$

que también podemos escribir de la siguiente manera

$$0,\ 1,\ -1,\ \frac{1}{2},\ -\frac{1}{2},\ 2,\ -2,\ \frac{1}{3},\ -\frac{1}{3},\ 3,\ -3,\ \frac{1}{4},\ -\frac{1}{4},\ \frac{2}{3},\ -\frac{2}{3},\ \frac{3}{2},\ -\frac{3}{2},\ 4,\ -4,\cdots$$

No es difícil convencerse de que en esta sucesión aparece, una vez cada uno, todos los números racionales. Finalmente, emparejamos el primer número de la lista con el 1, el segundo con el 2, el siguiente con el 3, y así sucesivamente. Queda establecida así una correspondencia uno-a-uno entre $\mathbb{N}$ y $\mathbb{Q}$, por lo que ambos conjuntos tienen el mismo cardinal (hay la misma cantidad de números naturales que de números racionales).

Uno de los descubrimientos fundamentales de Cantor es que, por el contrario, el conjunto de los números reales $\mathbb{R}$ tiene un cardinal mayor que el de los naturales; o sea, hay una "mayor cantidad" de números reales que de naturales. Cuando vamos ascendiendo por los conjuntos numéricos, al pasar de los naturales a los enteros seguimos en el "mismo nivel de infinitud", al pasar a los racionales seguimos aún en el mismo nivel, pero al pasar a los reales subimos a un nivel de infinitud superior.

Para demostrarlo, imaginemos que intentamos establecer una correspondencia uno-a-uno entre $\mathbb{N}$ y $\mathbb{R}$. En la figura 3.8 se muestra un ejemplo. En la columna de la derecha, convengamos en que, si un número real tiene en su escritura una cantidad finita de decimales, entonces completamos la expresión con infinitos ceros. Convenimos también en nunca usar una escritura con "9 periódico". Por ejemplo, el número $0,5$ se escribirá como $0,5000\ldots$ y no como $0,49999\ldots$ (que es otra forma de escribir $0,5$).

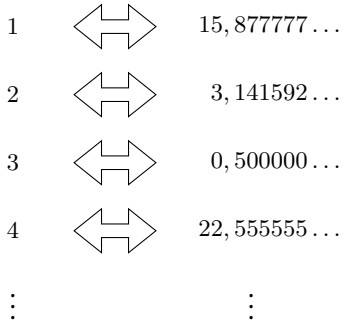


Figura 3.8. Un ejemplo de intento de emparejamiento entre $\mathbb{N}$ y $\mathbb{R}$.

Definimos ahora un número b de la manera que vamos a explicar a continuación. Para comenzar, tomamos, como se muestra en la figura 3.9, la primera cifra decimal del primer número real, la segunda cifra decimal del segundo número real, y así sucesivamente, descendiendo siempre hacia la derecha en diagonal (motivo por el cual, esta demostración es conocida como el *argumento diagonal de Cantor*).

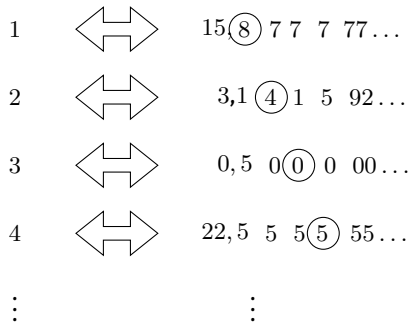


Figura 3.9. Argumento diagonal de Cantor.

El número b que queremos definir tendrá parte entera igual a 0. A continuación, nos fijamos en la primera cifra de la diagonal, que en nuestro ejemplo es un 8. La primera cifra decimal de b será, entonces, un dígito cualquiera distinto de 8 y de 9 (esto último tiene solo la intención de evitar que se genere una expresión con "9 periódico"). La primera cifra decimal de b puede ser un 7.

Miramos ahora la segunda cifra de la diagonal, que en nuestro ejemplo es un 4. La segunda cifra decimal de b será un dígito diferente de 4 y de 9, podemos tomar 5. Siguiendo la misma idea, como tercera cifra podemos tomar un 1, y como cuarta, un 6. El número comenzará entonces con $0,7516\ldots$ (figura 3.10).

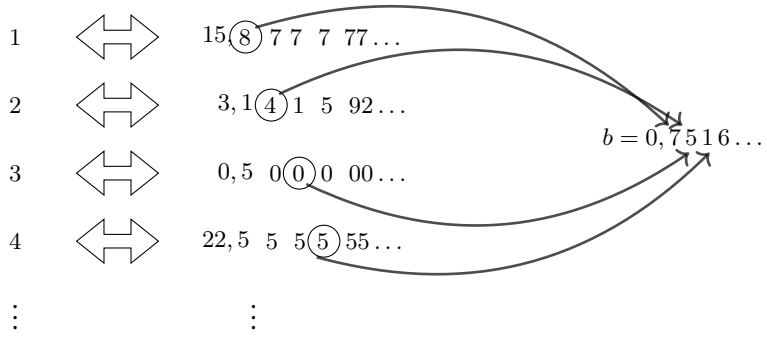


Figura 3.10. Argumento diagonal de Cantor. Construcción de un número que no se encuentra en la enumeración.

Observemos ahora que el número b no puede ser el que está emparejado con el natural 1, ya que ambos difieren en la primera cifra decimal. Y esto es válido no solo en el ejemplo que hemos mostrado, sino en cualquier

otro ejemplo que se plantee, porque, por construcción, la primera cifra decimal de b siempre será diferente de la primera cifra del número emparejado con el 1. (La figura 3.10 nos muestra, entonces, un *ejemplo generalizador*, es decir, un ejemplo con características generales y que, por lo tanto, tiene la misma fuerza argumentativa que una demostración abstracta.)

De la misma forma, b no puede ser el número emparejado con el 2, porque ambos difieren en la segunda cifra decimal. Tampoco puede ser el que está emparejado con el 3, porque difieren en la tercera cifra decimal; y así sucesivamente. Concluimos que, no importa cuál sea la correspondencia que se intente, siempre habrá un número b que no aparece en la lista de la derecha.

Volvamos a la idea de las personas y las sillas, e imaginemos que los números naturales son las personas, y que los reales son las sillas. No importa cómo intenten sentarse los naturales, la existencia del número b nos muestra que siempre habrá una silla vacía. Concluimos así que hay más "sillas" que "personas", es decir, que hay una mayor cantidad de números reales que de naturales o, en términos más correctos, que el cardinal de $\mathbb{R}$ es mayor que el de $\mathbb{N}$.

Aunque no hemos visto más que un pequeño atisbo de la teoría de Cantor, dejaremos aquí su desarrollo, ya que la parte que hemos mostrado será suficiente para demostrar la existencia de conjuntos no recursivos. (Los lectores interesados en profundizar en el tema pueden leer, por ejemplo, [14].)

13. Conjuntos no recursivos

¿Cuántos conjuntos recursivos existen? Como primera respuesta podemos decir que hay infinitos; pero, con más precisión, ¿cuál es ese nivel de infinitud? Por ejemplo, ¿hay tantos conjuntos recursivos como números reales? Como veremos enseguida, la respuesta es que hay la misma cantidad de conjuntos recursivos como de números naturales, es decir, que es posible establecer una correspondencia uno-a-uno que a cada conjunto recursivo le asigne un número natural.

Para explicar por qué, recordemos que cada conjunto recursivo está asociado a una máquina de Turing que reconoce si un elemento pertenece, o no pertenece, a ese conjunto. Las máquinas que realizan este tipo de reconocimiento tienen dos características: por un lado, aceptan a todo número natural como entrada válida, por otro, entregan siempre como salida X o XX.

Tomemos, entonces, la numeración de las máquinas de Turing que describimos en la sección 10: $\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \boldsymbol{T}_4, \ldots$ A continuación, recorremos la lista hasta encontrar la primera máquina de Turing que cumpla las dos características mencionadas. A esa primera máquina le asignamos el número 1, que es como asignar el número 1 al primer conjunto recursivo.

Seguimos avanzando por la numeración de máquinas de Turing hasta en-

contrar la segunda máquina que cumpla las dos características: si está asociada al mismo conjunto que la anterior (porque un mismo conjunto recursivo puede estar asociado a dos o más máquinas de Turing), la descartamos; si no, le asignamos el número 2. Y así seguimos: vamos identificando las máquinas que cumplen las dos condiciones y cada vez que llegamos a una de ellas, si no reconoce un conjunto recursivo ya considerado, le asignamos un número natural nuevo.

En resumen, cada conjunto recursivo está asociado con una máquina de Turing que cumple las dos condiciones, y cada una de estas máquinas, a su vez, queda emparejada con un número natural. Por transitividad, entonces, cada conjunto recursivo queda emparejado con un número natural. Deducimos así que hay tantos conjuntos recursivos como números naturales.

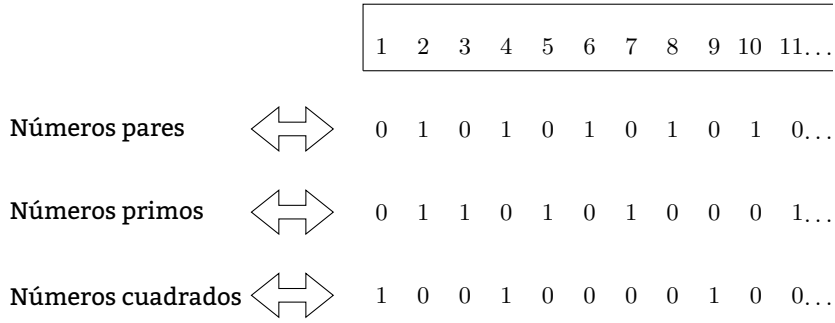


Figura 3.11. Emparejamiento entre conjuntos y sucesiones de ceros y unos.

Consideremos ahora todos subconjuntos de $\mathbb{N}$, tanto los recursivos, como los no recursivos (si los hubiere). Como se muestra en la figura 3.11, cada uno de estos conjuntos puede emparejarse con una sucesión formada por ceros y unos.

Para formar la sucesión que corresponde a un subconjunto dado, colocamos debajo de cada número natural un 1 si el número pertenece al subconjunto, y un 0 si no pertenece. No es difícil convencerse de que queda así establecida una correspondencia uno-a-uno entre subconjuntos de $\mathbb{N}$ y sucesiones de ceros y unos, y que, en consecuencia, hay tantos conjuntos formados por números naturales como sucesiones de ese tipo.

Finalmente, el mismo argumento diagonal de Cantor que nos permitió asegurar que hay más números reales que naturales nos permite ahora demostrar que hay más sucesiones de ceros y unos que números naturales. La figura 3.12 muestra la parte esencial de ese razonamiento; los lectores pueden completar los detalles de cómo esa imagen demuestra que, si los números naturales fueran personas y las sucesiones sillas, entonces siempre quedaría alguna silla vacía.

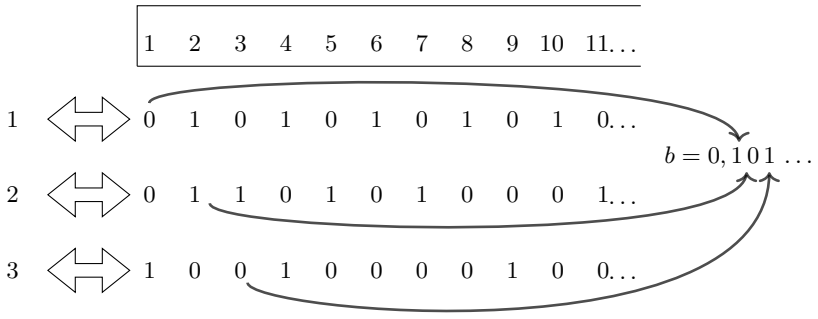


Figura 3.12. El argumento diagonal de Cantor demuestra que hay más sucesiones de ceros y unos que números naturales.

En resumen, la cantidad de conjuntos recursivos es igual a la cantidad de todos los números naturales, que es menor que la cantidad de todas las sucesiones de ceros y unos, que es igual a la cantidad de todos los subconjuntos de $\mathbb{N}$. Más brevemente, si Card significa cardinal,

$$\begin{aligned}\text{Card}(\text{conjuntos recursivos}) &= \text{Card}(\mathbb{N}) \\ &< \text{Card}(\text{sucesiones de ceros y unos}) \\ &= \text{Card}(\text{todos los subconjuntos de } \mathbb{N}).\end{aligned}$$

Concluimos así que hay en total más subconjuntos de $\mathbb{N}$ que subconjuntos que sean recursivos. Por lo tanto, como queríamos demostrar, existe algún subconjunto de $\mathbb{N}$ que no es recursivo. Observemos que la demostración que hemos hecho es de *existencia pura*, es decir, demuestra que cierto objeto matemático existe, pero no exhibe ningún ejemplo, ni muestra tampoco un modo de calcularlo. Esta observación será de interés en la sección siguiente.

14. ¿Una respuesta al Entscheidungsproblem?

La existencia de conjuntos no recursivos parece dar una respuesta al *Entscheidungsproblem*, el cual, recordemos, pregunta si todo problema claramente planteado dentro de alguna teoría matemática es resoluble algorítmicamente. La respuesta sería que no, porque si A es un subconjunto no recursivo de $\mathbb{N}$ entonces, por definición, el problema de determinar si un número n pertenece a A no es resoluble algorítmicamente.

Sin embargo, si profundizamos un poco, veremos que esta respuesta no es completamente satisfactoria. Para entender por qué, preguntémonos primero ¿cómo se define un subconjunto de $\mathbb{N}$? En la sección 9 hemos mencionado algunos subconjuntos, como el de los números primos o el los números que son capicúas al escribirlos en binario. Pero esas definiciones están

expresadas en el lenguaje natural, adecuado para un texto de divulgación como éste, pero muchas veces impreciso para las necesidades del trabajo matemático. ¿Qué características debería tener un lenguaje más preciso?

Todos los subconjuntos de $\mathbb{N}$ que hemos mencionado están definidos a partir de propiedades estudiadas por la aritmética, que es la teoría matemática que trata de los números naturales y sus operaciones básicas (suma y producto). Un lenguaje capaz de definir esos conjuntos debe tener, entonces, variables que se refieran a los números naturales (como $n, m, k \ldots$), el símbolo de la igualdad, los paréntesis, símbolos para las operaciones numéricas (suma y producto), las constantes 0 y 1, y símbolos para las operaciones lógicas ($\forall$, que es "para todo"; $\exists$, que es "existe"; $\neg$, que representa la negación; $\Rightarrow$, que representa la implicación; además de $\wedge$, $\vee$, que significan "y", "o", respectivamente). En este lenguaje, la frase "n es primo" puede expresarse así:

$$\neg(n = 1) \wedge \forall m \forall k (n = m \cdot k \Rightarrow (m = 1 \vee k = 1)) \, .$$

La expresión anterior, en consecuencia, es una definición en el lenguaje formal del conjunto de los números primos:

$$\text{Números primos} = \{n : \neg(n = 1) \wedge \forall m \forall k (n = m \cdot k \Rightarrow (m = 1 \vee k = 1))\} \, .$$

El punto interesante es que, así como antes establecimos un orden "alfabético" para las máquinas de Turing, de manera análoga podemos establecer un orden alfabético para las expresiones del lenguaje formal. Por lo tanto, tal como hicimos con las máquinas de Turing asociadas a los conjuntos recursivos, podríamos ahora emparejar un número natural con cada expresión formal que define un subconjunto de $\mathbb{N}$. Concluimos, entonces, que hay tantos subconjuntos de $\mathbb{N}$ que son definibles en el lenguaje formal como números naturales. Pero el cardinal de todos los subconjuntos de $\mathbb{N}$ es mayor; en consecuencia, existen infinitos conjuntos *inefables*, o sea, conjuntos que, no importa qué lenguaje formal se elija, no pueden ser definidos de ninguna manera.

¿Qué nos dice esta observación acerca de nuestra respuesta al *Entscheidungsproblem*? Hemos probado que existe un conjunto A tal que el problema de determinar si un número pertenece a él no es resoluble algorítmicamente (porque A no es recursivo). Pero ¿qué sucedería si ese conjunto resulta ser *inefable*? En ese caso, la pregunta "¿n pertenece a A?" no puede ser expresada en el lenguaje formal de la matemática, por lo que difícilmente pueda considerarse como un problema matemático claramente planteado.

Por ese motivo es que hemos dicho que nuestra respuesta al *Entscheidungsproblem* resulta ser poco satisfactoria el problema supuestamente no resoluble: quizás, en realidad, no sea ni siquiera expresable. Una solución más adecuada consistiría en mostrar un ejemplo explícito de un

conjunto A que no sea recursivo, pero que sí sea expresable en el lenguaje formal. En las siguientes secciones nos dedicaremos a mostrar un ejemplo así.

15. Conjuntos recursivamente numerables

Antes de mostrar un ejemplo concreto de un subconjunto no recursivo de $\mathbb{N}$, debemos proponer un nuevo concepto (también introducido por Emil Post en 1944); se trata de la idea de conjunto recursivamente numerable. Para entender de qué se trata, consideremos primero, a modo de ejemplo, el conjunto A formado por todos los números naturales que pueden escribirse como resta de dos primos consecutivos. Por ejemplo, el 1 pertenece al conjunto ya que $1 = 3 - 2$; también pertenece el 2, porque $2 = 5 - 3$; y el 4, porque $4 = 11 - 7$. Notemos que excepto el 1, todos los demás elementos de A son números pares.

Imaginemos ahora una computadora teórica, a la que llamaremos la "computadora permanente", que está preparada para funcionar por tiempo indefinido (que es un modo de decir que funcionará durante un lapso infinito, de ahí que sea teórica). Suponemos, además, que esta computadora tiene una capacidad de memoria ilimitada, pero que su programa consta siempre de una cantidad finita de instrucciones. Imaginemos también que la computadora está programada para ir imprimiendo sucesivamente los elementos de A. Es decir, el programa recorre la sucesión de los números naturales y, cuando encuentra dos primos consecutivos, calcula su resta e imprime el resultado obtenido (si el número ya había sido impreso, el programa lo saltea).

Rápidamente, por ejemplo, el programa imprimirá los números $1, 2, 4, 6, 8$. Seguirá después con $14, 10, 12, 18, \ldots$ El 16 aparecerá cuando la computadora permanente calcule la resta $1847 - 1831$.

¿Cómo podemos saber si un número par n pertenece, o no, al conjunto? Si solo contáramos con el programa que acabamos de describir, el único modo sería sentarnos a esperar a que la computadora imprima el número n. Sin embargo, esto no nos asegura que tendremos una respuesta en una cantidad finita de pasos. Porque, si resulta que n no pertenece al conjunto, entonces nunca lo sabremos (pasarán miles de millones de años sin que n aparezca impreso, pero sin que sepamos si acaso aparecerá mañana). Vemos aquí un fenómeno similar al que mostramos en la sección 4 cuando hablamos de la búsqueda por "fuerza bruta" de una solución para una ecuación diofántica: si la ecuación no tiene solución, nunca podremos saberlo. (Desde luego, podría haber un modo más ingenioso de determinar si n es, o no, la resta de dos primos consecutivos. De hecho, una conjetura no demostrada afirma que todo número par es la resta de dos primos. Pero esta observación, aunque válida, no es relevante para nuestra exposición.)

Llegamos ahora a la definición propuesta por Emil Post. Un conjunto A es *recursivamente numerable* si y solo si podemos programar la computadora permanente de tal modo que se cumplan las dos condiciones siguientes:

a) Todo número que imprime la computadora pertenece al conjunto A.

b) Todo número que pertenezca al conjunto A será impreso al cabo de una cantidad finita de operaciones. (Aun cuando esto pueda demorar miles de millones de años.)

Aunque no será necesaria para nuestra exposición, podemos dar una versión más rigurosa de esta idea, basada en la noción de máquina de Turing. En esta versión (que es la que realmente propuso Post), la definición dice que un conjunto A es recursivamente numerable si y solo si existe una máquina de Turing $\boldsymbol{T}$ que cumple estas dos condiciones: 1) Si n es una entrada válida para $\boldsymbol{T}$, entonces la salida es un número natural m (aunque no necesariamente toda entrada es válida); 2) Un número m pertenece al conjunto A si y solo si es la salida que entrega $\boldsymbol{T}$ a partir de alguna entrada válida.

En otras palabras, si llamamos $\boldsymbol{T}(n)$ a la salida que entrega $\boldsymbol{T}$ al recibir la entrada n (si es que es válida); entonces, A es recursivamente numerable si y solo si existe una máquina de Turing $\boldsymbol{T}$ tal que $A = \{\boldsymbol{T}(n) : n \text{ es una entrada válida para } \boldsymbol{T}\}$. Para esta segunda versión de la definición, en el ejemplo de los números que son resta de dos primos consecutivos, si la entrada es el número n, la máquina de Turing correspondiente imprimiría el n-ésimo término de la sucesión $1, 2, 4, 6, 8, 14, 10, 12, 18, \ldots$

16. Dos propiedades

¿Qué relación hay entre los conjuntos recursivamente numerables y los conjuntos recursivos? En esta sección y en la siguiente no solo responderemos a esta pregunta, sino que también demostraremos algunas propiedades que serán necesarias para responder el *Entscheidungsproblem*.

Para comenzar, recordemos que un conjunto es recursivo si, dado un número natural n cualquiera, es posible programar una computadora de modo que determine (en una cantidad finita de pasos) si pertenece, o no pertenece al conjunto. Por otra parte, un conjunto es recursivamente numerable si es posible programar a la computadora permanente de modo que vaya imprimiendo, no importa en qué orden, los números que lo forman.

Vamos a demostrar en primer lugar que todo conjunto recursivo es también recursivamente numerable. En otras palabras:

Si A es recursivo, entonces A es recursivamente numerable.

Para demostrarlo, hay que probar que, si existe un algoritmo que reconoce en una cantidad finita de pasos si un número n pertenece, o no, al conjunto A, entonces es posible programar la computadora permanente de tal modo

que vaya imprimiendo los elementos de A. En efecto, el modo de programar la computadora permanente es bastante claro; la computadora recorre los sucesivos números naturales, $1, 2, 3, 4, 5, 6, \ldots$ y en cada caso aplica el algoritmo que determina si el número pertenece al conjunto A. Si la respuesta es que sí pertenece, entonces la computadora imprime el número. Si la respuesta es que no pertenece, entonces no lo imprime. En consecuencia, todo número que pertenezca al conjunto A será impreso al cabo de una cantidad finita de pasos.

¿Vale la recíproca? ¿Podemos decir que es recursivo si y solo si es recursivamente numerable? La respuesta es que no, pero dejaremos la explicación para más adelante. Demostremos por ahora la siguiente propiedad:

Un conjunto A es recursivo si y solo si A y su complemento son ambos recursivamente numerables.

(El complemento de A, al que llamaremos A^c, es el conjunto de todos los números naturales que no pertenecen a A. Por ejemplo, el complemento del conjunto de los números pares es el conjunto de los números impares.)

Tenemos que demostrar dos afirmaciones; la primera es que si A es recursivo entonces A y su complemento son ambos recursivamente numerables. Es decir, tenemos ahora dos pantallas en la computadora permanente y tenemos que programarla de tal modo que en la pantalla 1 se impriman los elementos de A, y en la 2, los elementos de A^c. El modo de lograrlo es similar al que explicamos más arriba; la computadora recorre los sucesivos números naturales, y a cada número n le aplica el algoritmo que determina si n pertenece a A. Si la respuesta es que sí, entonces imprime al número n en la pantalla 1, si la respuesta es no, lo imprime en la pantalla 2.

La segunda afirmación que tenemos que demostrar es que si A y su complemento son ambos recursivamente numerables entonces A es recursivo. Para ello, suponemos que la computadora permanente está programada para imprimir, en una pantalla 1 los elementos de A, y en una pantalla 2, los elementos de A^c. Tenemos que probar que, dado un número n, es posible determinar en una cantidad finita de pasos si n pertenece, o no pertenece, a A. El modo de hacerlo es el siguiente: esperamos a que la computadora imprima el número n. Si se imprime en la pantalla 1, la respuesta es que pertenece a A, si se imprime en la 2, la respuesta es que no pertenece. Como todo número necesariamente pertenece, o bien a un conjunto, o bien a su complemento (a uno y solo uno de los dos), entonces tenemos la certeza de que obtendremos una respuesta al cabo de una cantidad finita de pasos.

17. Una tercera propiedad

En esta sección demostraremos la última propiedad que necesitamos antes de dar una respuesta satisfactoria al *Entscheidungsproblem*. Para comen-

zar, observemos que, a cada máquina de Turing $\boldsymbol{T}$ podemos asociarle el subconjunto de $\mathbb{N}$ formado por todas sus entradas válidas, conjunto al que llamaremos $V(\boldsymbol{T})$.

La propiedad que queremos demostrar dice que un conjunto A es recursivamente numerable si y solo si es el conjunto de las entradas válidas de alguna máquina de Turing $\boldsymbol{T}$. En otras palabras, A es recursivamente numerable si y solo si existe alguna máquina de Turing $\boldsymbol{T}$ tal que $A = V(\boldsymbol{T})$.

Para demostrarlo, tenemos que probar primero que, si A es recursivamente numerable, entonces A es el conjunto de entradas válidas de alguna máquina de Turing $\boldsymbol{T}$. Veamos la demostración.

Si A es recursivamente numerable, podemos programar la computadora permanente de tal modo que vaya imprimiendo los elementos de A. Tomemos, entonces, la máquina de Turing $\boldsymbol{T}$ que, dada la entrada n, ejecuta el programa de la computadora permanente hasta que éste imprime el número n. Cuando lo imprime, la máquina $\boldsymbol{T}$ se detiene. Si el número n nunca es impreso, la máquina nunca se detendrá. Es claro que la máquina $\boldsymbol{T}$ se detendrá si y solo si la entrada n es un elemento de $V(\boldsymbol{T})$. En otras palabras, como queríamos demostrar, $V(\boldsymbol{T}) = A$.

La segunda afirmación que tenemos que probar es que, si A es el conjunto de las entradas válidas de una máquina $\boldsymbol{T}$ cualquiera, entonces A es recursivamente numerable. Es decir, dada una máquina de Turing $\boldsymbol{T}$, tenemos que probar que se puede programar la computadora permanente de tal modo que vaya imprimiendo todos los números que son entradas válidas de $\boldsymbol{T}$, y solo esos números.

Para demostrarlo, tenemos que programar a la computadora permanente de tal modo que, a la larga, ejecute las instrucciones de la máquina $\boldsymbol{T}$ correspondientes a *todas* las entradas posibles, sean válidas o no. Un intento podría consistir en que la máquina ejecute primero todas las instrucciones que corresponden a la entrada $n = 1$; cuando termina, pasa a ejecutar todas las instrucciones que corresponden a la entrada $n = 2$; y así sucesivamente. Sin embargo, esta idea tiene un problema: si un valor de n es una entrada no válida, entonces el programa *nunca* pasará al valor siguiente. Para lograr nuestro objetivo de una manera adecuada, llamemos $i_{1,1}$ a la primera instrucción que ejecuta la máquina $\boldsymbol{T}$ cuando trabaja con la entrada $n = 1$; $i_{1,2}$ será la segunda instrucción que ejecuta la máquina $\boldsymbol{T}$ cuando trabaja con la entrada $n = 1$; $i_{1,3}$ será la tercera instrucción que ejecuta la máquina $\boldsymbol{T}$ cuando trabaja con la entrada $n = 1$; y así sucesivamente. De manera similar, $i_{2,1}$ será la primera instrucción que ejecuta la máquina $\boldsymbol{T}$ para la entrada $n = 2$; $i_{2,2}$ será la segunda instrucción que ejecuta la máquina $\boldsymbol{T}$ para la entrada $n = 2$; y así sucesivamente (véase la figura 3.13, izquierda).

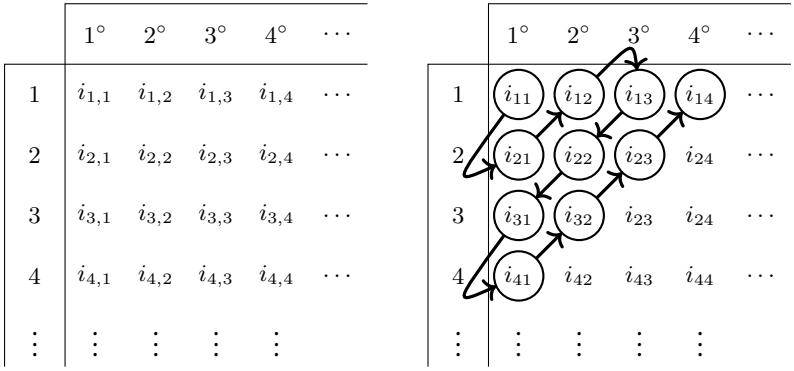


Figura 3.13. Un camino que recorre la matriz.

La parte derecha de la figura 3.13 nos muestra el comienzo de un camino que va recorriendo totalmente la matriz. Este camino pasa, en orden, por $i_{11}, i_{21}, i_{12}, i_{13}, i_{22}, i_{31}, i_{41}, \ldots$ (Nótese la similitud con ideas previas: tomamos primero los subíndices que suman 2; luego, los que suman 3; luego, los que suman 4; y así sucesivamente). Teniendo en cuenta este ordenamiento de las instrucciones, programamos la computadora permanente de la siguiente manera:

1. Ejecuta la instrucción i_{11} (la primera instrucción de $\boldsymbol{T}$ cuando la entrada es $n = 1$).
2. Ejecuta la instrucción i_{21}.
3. Ejecuta la instrucción i_{12}.
4. Ejecuta la instrucción i_{13}.

Y así sucesivamente. Nótese que toda instrucción posible será ejecutada al cabo de una cantidad finita de tiempo. Por otra parte, si al ejecutar una de estas instrucciones la máquina $\boldsymbol{T}$ se detiene, entonces el programa imprime el valor de n y ya no vuelve a ejecutar las instrucciones que correspondan a esa entrada. Si una entrada, por el contrario, no es válida, entonces la máquina $\boldsymbol{T}$ nunca se detendrá al trabajar con ella y la computadora permanente nunca la imprimirá. En resumen, el programa de la computadora permanente imprimirá exactamente las entradas válidas de $\boldsymbol{T}$. Hemos probado así que $V(\boldsymbol{T})$ es recursivamente numerable.

18. El ejemplo concreto

Finalmente, estamos en condiciones de mostrar un ejemplo concreto de un conjunto no recursivo. Para comenzar, recordemos la numeración de las

máquinas de Turing $\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \boldsymbol{T}_4 \ldots$ que hemos mostrado en la sección 10, y recordemos también la máquina universal U, que puede escribir el programa de cualquier máquina $\boldsymbol{T}_n$ y ejecutarlo para cualquier entrada m.

Definimos ahora el conjunto S_0 de la siguiente manera: S_0 es el conjunto de todos los números n tales que n es una entrada válida para $\boldsymbol{T}_n$. Más formalmente:

$$S_0 = \{n : n \in V(\boldsymbol{T}_n)\}.$$

Vamos a demostrar primero que S_0 es recursivamente numerable; es decir, que podemos programar la computadora permanente de tal modo que vaya imprimiendo sus elementos. Para ello, apelamos una vez más a la figura 3.13 y al esquema $i_{11}, i_{21}, i_{12}, i_{13}, i_{22}, i_{31} \ldots$

1. Ejecuta la instrucción i_{11}.
2. Ejecuta la instrucción i_{21}.
3. Ejecuta la instrucción i_{12}.
4. Ejecuta la instrucción i_{13}.

Y así sucesivamente. Solo que ahora i_{nm} representa la instrucción número m que ejecuta la máquina $\boldsymbol{T}_n$ si la entrada es n. Como antes, si al ejecutar una de estas instrucciones la máquina $\boldsymbol{T}_n$ se detiene, entonces el programa imprime el valor de n y ya no vuelve a trabajar con esa entrada. Si una entrada n no es válida para $\boldsymbol{T}_n$ la computadora permanente nunca lo imprimirá. Vemos así que el programa imprime exactamente los números n que son entradas válidas de $\boldsymbol{T}_n$, de donde concluimos que S_0 es recursivamente numerable.

A continuación, vamos a demostrar que, por el contrario, el complemento de S_0 no es recursivamente numerable. Para hacerlo, razonaremos por el absurdo, es decir, vamos a suponer que S_0^c sí es recursivamente numerable y vamos a llegar a una contradicción.

Observemos primero que el complemento de S_0 es el conjunto:

$$S_0^c = \{n : n \notin V(\boldsymbol{T}_n)\}.$$

Si S_0^c es recursivamente numerable entonces, por lo que probamos en la sección 17, S_0^c es el conjunto de entradas válidas de alguna máquina de Turing. Es decir, existe algún número natural M tal que $S_0^c = V(\boldsymbol{T}_M)$. La pregunta que debemos hacernos es: ¿M pertenece, o no pertenece, a S_0^c?

Si $M \in S_0^c$, entonces M cumple la condición que define a S_0^c, es decir, $M \notin V(\boldsymbol{T}_M)$. Pero $V(\boldsymbol{T}_M) = S_0^c$. Luego, $M \notin S_0^c$. Deducimos así que si $M \in S_0^c$ entonces $M \notin S_0^c$. Esto es absurdo, y en consecuencia no puede ser cierto que $M \in S_0^c$.

Pero, si $M \notin S_0^c$, entonces M no cumple la definición que define a S_0^c, es decir, $M \in V(\boldsymbol{T}_M)$. Sin embargo, una vez más, $V(\boldsymbol{T}_M) = S_0^c$. O sea, $M \in S_0^c$. Deducimos así que si $M \notin S_0^c$ entonces $M \in S_0^c$. Esto también es absurdo, y entonces tampoco puede ser cierto que $M \notin S_0^c$.

En resumen, si M existe, entonces es falso que pertenezca a S_0^c, y también es falso que no pertenezca. Esto es imposible, por lo que M no puede existir, y entonces tampoco existe $\boldsymbol{T}_M$. En consecuencia, S_0^c no es recursivamente numerable.

En conclusión, S_0 es recursivamente numerable, pero su complemento no lo es. Por lo probado en la sección 16, podemos afirmar que S_0 no es recursivo. En particular, esto demuestra que, aunque es cierto que todo conjunto recursivo es recursivamente numerable, la recíproca, sin embargo, es falsa.

19. El Halting Problem y el Entscheidungsproblem

El conjunto S_0 es recursivamente numerable, pero no es recursivo. Este hecho tiene algunas consecuencias muy importantes, analizaremos a continuación las dos primeras (las otras consecuencias, como veremos después, se relacionan con los teoremas de incompletitud de Gödel).

Recordemos, en primer lugar, que S_0 es el conjunto de todos los números naturales n tales que n es una entrada válida para $\boldsymbol{T}_n$. Luego, determinar si n pertenece a S_0 equivale, obviamente, a determinar si n es una entrada válida para $\boldsymbol{T}_n$. Por lo tanto, si no existe un algoritmo que determine en una cantidad finita de pasos si n pertenece a S_0, entonces tampoco existe un algoritmo que determine en una cantidad finita de pasos si n es una entrada válida para $\boldsymbol{T}_n$.

El *halting problem* (el *problema de la detención*, o el *problema de la parada*, ya planteado en el capítulo 2) es el problema, planteado por Turing en su artículo de 1937, que pide, dado un número natural n y una máquina de Turing $\boldsymbol{T}$, determinar si n es, o no, una entrada válida para $\boldsymbol{T}$. El hecho de que S_0 no sea recursivo implica que el *halting problem* no es resoluble algorítmicamente. Esta es, ahora sí, la respuesta que da Turing al *Entscheidungsproblem*: en la teoría de algoritmos existe al menos un problema (el *halting problem*) que no es resoluble por ninguna computadora.

Más aun, se puede demostrar, aunque la demostración excede los alcances de este trabajo, que la definición de cualquier conjunto recursivamente numerable puede ser expresada en el lenguaje formal de la aritmética que describimos en la sección 14 (la demostración se basa en el hecho de que las operaciones que realiza una máquina de Turing con las secuencias de símbolos X puede traducirse a operaciones aritméticas con números naturales). En particular, la afirmación "n pertenece a S_0", análogamente a lo que sucede con "n es primo", puede expresarse en ese lenguaje formal. Por lo tanto, el problema "¿n pertenece a S_0?" es, podemos afirmar ahora,

un problema aritmético. Deducimos así que existen problemas *aritméticos* bien definidos que no pueden resolverse algorítmicamente.

Destacamos el hecho de que el problema sea aritmético porque la aritmética, que estudia los números más "sencillos" con sus operaciones más "básicas", es la más "simple" de las teorías clásicas de la matemática. Si en la aritmética existen problemas no resolubles algorítmicamente, entonces es casi una certeza que también tienen que existir en cualquier otra teoría matemática más "compleja". Los problemas no resolubles algorítmicamente abundan en la matemática.

Pero ¿qué significa que un problema no sea resoluble algorítmicamente? Por ejemplo ¿qué significa que el *halting problem* no lo sea? ¿Significa que, dados n y $\boldsymbol{T}$, no hay modo de decidir si n es, o no es, una entrada válida para $\boldsymbol{T}$? Claramente no es eso lo que significa, porque hay máquinas de Turing para las cuales siempre es posible responder si un n cualquiera es una entrada válida para ella. Por ejemplo, en la sección 8 vimos una máquina de Turing de la que podemos saber que acepta como entradas válidas a todos los números mayores que 1, y otra que acepta a los números pares y solo a ellos. Por otra parte, si en una máquina de Turing no aparece la instrucción (Fin), como sucede en $0Dq_1 0Dq_1$, entonces ningún n será una entrada válida.

Que un problema no sea resoluble algorítmicamente significa, en realidad, que no puede existir un criterio objetivo y uniforme que, en una cantidad finita de pasos, nos dé siempre la respuesta correcta. Podemos proponer algoritmos que se apliquen a ciertos casos específicos, pero nunca sucederá que un mismo algoritmo se aplique exitosamente a todas las máquinas y a todas las entradas. Por ejemplo, podemos escribir un algoritmo que determine si alguna instrucción de $\boldsymbol{T}$ contiene la indicación (Fin); si la respuesta es que no, entonces el algoritmo nos dirá, correctamente, que ningún n es una entrada válida, pero si (Fin) aparece entonces el algoritmo no nos dirá nada.

Una observación

La exposición que acabamos de hacer se basó principalmente en los conceptos de conjunto recursivo y de conjunto recursivamente numerable, los cuales, como ya dijimos, fueron introducidos por Emil Post en 1944. Por lo tanto, obviamente, no son estas las ideas que usó Turing en 1937 para resolver el *Entscheidungsproblem*. Turing, en realidad, apeló a la idea de número real computable, que explicaremos brevemente a continuación.

Para comenzar digamos que, como es bien sabido, todo número natural puede escribirse en base 2, o en binario; una escritura en la que cada dígito es el coeficiente de una potencia de 2 con exponente entero no negativo. Por

ejemplo:

$$(11001)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^1 + 1 \cdot 2^0 = 25.$$

Pero la escritura binaria no se restringe a los números naturales, sino que se extiende, de hecho, a todo número real. En el caso de los números reales, los dígitos detrás de la coma (es decir, los dígitos de la *mantisa*) son también coeficientes de potencias de 2, pero con exponentes negativos. Por ejemplo:

$$(0,011)_2 = 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0,375.$$

También podemos escribir este número agregando a la mantisa infinitos ceros: $(0,011000\ldots)_2$. Asimismo, puede suceder, tal como ocurre en la escritura decimal usual, que la mantisa en binario de un número real contenga infinitas cifras no nulas. Por ejemplo:

$$(0,1010101\ldots)_2 = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + \cdots = 0,666\ldots$$

En la última igualdad se aplica una fórmula bien conocida para calcular la suma de una serie geométrica. Esta fórmula dice que, si $|r| < 1$ y $a \in \mathbb{R}$, entonces

$$a + ar + ar^2 + ar^3 + \cdots = \frac{a}{1-r}.$$

En el caso del ejemplo, $a = \frac{1}{2}$ y $r = \frac{1}{4}$.

De esta forma, cada número real genera una sucesión infinita de ceros y unos: la sucesión de los dígitos de su mantisa en la escritura binaria. Por ejemplo, $(0,011000\ldots)_2$ genera $011000\ldots$ y $(0,1010101\ldots)_2$ genera $1010101\ldots$ (Para evitar ambigüedades, por el resto de esta sección, y sin necesidad de aclararlo cada vez, solo trabajaremos con números reales con parte entera igual a 0.)

En su artículo de 1937 Turing define que una sucesión de ceros y unos es *computable* si y solo si es posible programar la computadora permanente de tal modo que vaya imprimiendo sus dígitos, uno por uno, en orden (se sobreentiende que cada dígito se calcula en una cantidad finita de pasos). A partir de esta idea, Turing dice que un número real es computable si y solo si su mantisa en binario forma una sucesión computable.

Para entender cómo se relaciona esta definición con las ideas de Post, recordemos, para comenzar, que en la sección 13 hemos mostrado que hay una correspondencia uno-a-uno entre los subconjuntos de $\mathbb{N}$ y las sucesiones de ceros y unos. En esta correspondencia, cada sucesión queda asociada con el conjunto de las posiciones ocupadas por los números 1. Por ejemplo, la sucesión $01100000\ldots$ está asociada con el conjunto $\{2,3\}$, ya que $01100000\ldots$ tiene un 1 en la posición 2 y otro 1 en la posición 3. A la sucesión $1010101\ldots$, por su parte, le corresponde el conjunto de los números impares.

Finalmente, la relación entre números computables y conjuntos recursivos es la siguiente: una sucesión de ceros y unos es computable si y solo si el conjunto que le corresponde es recursivo. En otras palabras, un número real es computable si y solo si a su mantisa le corresponde un conjunto recursivo. Veamos por qué.

Para demostrar la afirmación, probemos primero que, si una sucesión de ceros y unos es computable entonces su conjunto correspondiente, llamémoslo A, es recursivo. Tenemos que demostrar que hay un algoritmo que, dado un número natural n, determina en una cantidad finita de pasos si n pertenece a A. Tal algoritmo, en efecto, consiste en calcular la sucesión de ceros y unos hasta llegar al lugar n; si a esa posición le corresponde un 1, la respuesta es que pertenece a A, si le corresponde un 0, la respuesta es que no pertenece.

Recíprocamente, hay que demostrar que si A es recursivo, entonces la sucesión es computable. Para programar la computadora permanente, procedemos así: para calcular el n- dígito de la sucesión, la computadora aplica el algoritmo que determina si n pertenece a A. Si la respuesta es sí, imprime un 1, si la respuesta es no, imprime un 0.

La pregunta clave es ¿qué tipo de sucesión (y, en consecuencia, qué tipo de número real) le corresponde al conjunto S_0 de la sección 18?

Como S_0 no es recursivo, la sucesión de ceros y unos que le corresponde no es computable. Es posible programar la computadora permanente para que vaya imprimiendo, aunque no en orden, las posiciones que ocupan los números 1; pero, dada una posición n, no hay forma de determinar algorítmicamente, en una cantidad finita de pasos, si esa posición estará ocupada por un 1 o por un 0.

Si programamos a la computadora para que se detenga cuando determine si un lugar específico de la sucesión hay un 0, no hay forma de saber (algorítmicamente, en una cantidad finita de pasos) si se detendrá o no. Desde un punto de vista histórico esta es, esencialmente, la solución de Turing al *Entscheidungsproblem*. Nuestra elección de presentar el tema a través de las ideas de Post se debe a que éstas nos permiten introducir fácilmente al otro gran protagonista de nuestro relato: Kurt Gödel. Hablaremos de él en las secciones siguientes.

Antes de dejar esta sección, es interesante comentar que los conceptos que hemos desarrollado en estos últimos párrafos permiten demostrar que existe una correspondencia uno-a-uno entre los números reales y los subconjuntos de $\mathbb{N}$. Ya mostramos la idea fundamental, que consiste en asociar cada número real con un subconjunto de $\mathbb{N}$ por medio de una sucesión de ceros y unos. Definimos, por ejemplo, el número real $0,375$; lo escribimos en binario, $(0,011000\ldots)_2$; esta escritura genera la sucesión $011000\ldots$, que se corresponde con el conjunto $\{2,3\}$. Luego, en el emparejamiento uno-a-

uno, el número $0,375$ se asocia con el subconjunto $\{2,3\}$. Del mismo modo, $0,666\cdots = (0,101010\ldots)_2$ se asocia con el conjunto de los números impares. Hay algunas cuestiones técnicas que resolver, como qué ocurre si la parte entera del número real no es 0, o cómo se maneja el hecho de que, así como $0,042 = 0,0419999\ldots$, del mismo modo $(0,011000\ldots)_2 = (0,0101111\ldots)_2$, por lo que al número $0,375$ le corresponderían, en principio dos sucesiones diferentes. No nos detendremos aquí en esos detalles. El hecho importante es que la existencia de esta correspondencia uno-a-uno demuestra que hay tantos números reales como subconjuntos de $\mathbb{N}$, por lo que, así como hay subconjuntos de $\mathbb{N}$ que no son definibles en el lenguaje formal, lo mismo sucede con los números reales: existen números reales *inefables*.

20. La paradoja de Russell

La teoría de conjuntos de Cantor, conocida como *teoría intuitiva de conjuntos*, no solamente dio inicio al estudio de los cardinales infinitos, sino que, más cotidianamente para el trabajo matemático, introdujo todo un lenguaje y una forma de pensar que, hacia fines del siglo xix, y tras mucha resistencia inicial, llegó a transformarse en la base del razonamiento matemático. Hoy en día, de hecho, la matemática está totalmente adaptada al uso del lenguaje y de los conceptos conjuntistas. Nótese, por ejemplo, cuántas veces hemos usado en este texto las palabras "conjunto", "subconjunto" o "pertenece"; o cómo hemos podido introducir, sin necesidad de mayores explicaciones, la notación $S_0 = \{n : n \in V(\boldsymbol{T}_n)\}$.

Sin embargo, en 1902 Bertrand Russell (1872–1970) descubrió que la teoría intuitiva de conjuntos contenía un error fundamental. En [17] se transcribe la carta en la que Russell anuncia por primera vez su hallazgo; la misma está dirigida a Gottlob Frege (1848–1925), lógico alemán que había intentado formalizar la teoría de Cantor usando un lenguaje matemático riguroso.

Para entender el descubrimiento de Russell, comencemos diciendo que uno de los axiomas de la teoría intuitiva de conjuntos es el llamado *axioma de comprehensión*, que afirma esencialmente que, dada cualquier propiedad, existe siempre el conjunto de todos los objetos que cumplen esa propiedad. Este axioma es usado tan frecuentemente y con tanta naturalidad en la práctica matemática que, en general, ni siquiera somos conscientes de estar aplicando un axioma. Lo hemos usado, por ejemplo, al hablar del conjunto de los números primos, o al definir el conjunto S_0.

Russell descubrió que el axioma de comprehensión, tan usado y aparentemente tan inocuo, es, no obstante, autocontradictorio. En otras palabras, demostró que el axioma lleva por sí mismo a una contradicción, a la llamada *paradoja de Russell*. Para ver por qué, consideremos la propiedad "Es un conjunto que no es elemento de sí mismo". De acuerdo con el axioma de

comprehensión, existe el conjunto, llamémoslo H, formado por todos los conjuntos que no son elementos de sí mismos:

$$H = \{A : A \notin A\}.$$

La pregunta que debemos hacernos es: ¿H es elemento de sí mismo? Si H es elemento de sí mismo, o sea si $H \in H$, entonces H no cumple la propiedad que lo define. Pero si H no cumple la propiedad que lo define entonces H no es elemento de H. En resumen, si suponemos que H es elemento de sí mismo entonces deducimos que no lo es. Esto es absurdo, luego no puede ser cierto que $H \in H$.

Por otra parte, si H no es elemento de sí mismo, o sea si $H \notin H$, entonces H cumple la propiedad que lo define. Si H cumple la propiedad que lo define entonces H es elemento de H. Deducimos así que, si suponemos que es H no es elemento de sí mismo entonces sí lo es. Esto también es absurdo, entonces tampoco puede ser cierto que $H \notin H$.

Pero si H existe entonces, o bien es verdad que $H \in H$, o bien que $H \notin H$. Si ambas afirmaciones son falsas, entonces H no existe. Sin embargo, el axioma nos dice que sí existe; luego, el axioma es contradictorio. (Nótese la similitud de estructura con la demostración de que S_0 no es recursivo.)

Antes de continuar, conviene hacer dos aclaraciones. La primera es: ¿existen conjuntos que no sean elementos de sí mismos? En realidad, la mayoría de los conjuntos que uno pueda imaginarse no lo son. Por ejemplo, el conjunto de los números reales $\mathbb{R}$ está formado por números, no por conjuntos, luego $\mathbb{R}$ no es un elemento de sí mismo. Lo mismo sucede con cualquier otro conjunto formado por números, vectores o funciones. Russell ejemplificaba esta idea diciendo que "Un conjunto de caballos no es un caballo" (es decir, un conjunto de caballos no es uno de sus elementos). Los conjuntos "raros" son aquellos que son elementos de sí mismos. Para encontrar ejemplos debemos elevarnos a conjuntos altamente abstractos y abarcadores, como el conjunto de todos los conjuntos (que es un conjunto y, por ende, elemento de sí mismo), o el conjunto de todos los conceptos abstractos.

La segunda aclaración es que Russell dio posteriormente una versión más intuitiva de su paradoja, habitualmente conocida como la *paradoja de barbero*. Esta versión, que tiene la forma de un acertijo, dice así: en un pueblo hay un único barbero, que afeita a todos los hombres que no se afeitan a sí mismos, ¿se afeita el barbero a sí mismo? La respuesta es que el barbero no puede afeitarse a sí mismo, pero tampoco puede no afeitarse, porque en ambos casos se llega a una contradicción. Es decir, si consideramos el conjunto de todos los hombres que se afeitan a sí mismos, el barbero no puede pertenecer a él, pero tampoco puede no pertenecer. (En una tercera versión, más humorística, Groucho Marx dijo alguna vez que jamás sería miembro de un club que aceptara miembros como él.)

21. La crisis de los fundamentos

La paradoja de Russell no solo demostró que la teoría que la matemática había elegido como base era contradictoria, sino que, peor aún, esa contradicción provenía de un axioma de cuya validez nadie hubiera dudado. Esta situación provocó una desconfianza generalizada hacia todas las demostraciones, que eran muchas, que hacían uso de las nociones conjuntistas, especialmente si involucraban conjuntos infinitos. A esta situación de duda general, que llevó incluso a que se cuestionara la naturaleza misma de la matemática, se la conoció como la *crisis de los fundamentos*. Las consecuencias de esta crisis, como veremos, se sienten todavía en la actualidad.

En los primeros años del siglo xx hubo dos propuestas diferentes para superar esta situación. La primera, presentada por L.E.J. Brouwer (1881–1966), dio origen a la llamada escuela intuicionista o constructivista. En una muy apretada síntesis (y por lo tanto un poco injusta), el intuicionismo sostenía que las "rarezas" de la teoría de Cantor (como el hecho de que haya tantos números naturales como enteros o racionales) demostraban que el concepto de conjunto infinito es contradictorio en sí mismo. El infinito, para esta escuela, solo es aceptable como cantidad finita que crece tanto como se quiera, pero que nunca deja de ser finita (como sucede, por ejemplo, en la definición clásica de límite).

Los intuicionistas, por otra parte, rechazaban la validez de las demostraciones por el absurdo (véase [1]), así como la validez de las demostraciones de existencia pura. Para ellos, una demostración de existencia solo podía ser constructiva. La idea de "conjunto inefable", por ejemplo, así como la justificación de su existencia que hemos explicado, les habrían parecido un sinsentido completo.

Brouwer sostenía que había que abandonar toda la matemática que hacía uso de la teoría de Cantor y que había que reconstruirla sobre la base de los principios intuicionistas. Sin embargo, aunque varios matemáticos veían las ideas de Brouwer con simpatía, una reconstrucción de ese tipo, que implicaba abandonar décadas de investigación matemática y adquirir nuevas (y más limitadas) reglas de razonamiento, era totalmente inviable en la práctica. La propuesta de Brouwer, entonces, no logró imponerse. (De todos modos, algunos de los preceptos del intuicionismo siguen en cierto modo vigentes. Por ejemplo, en los casos en que resulta posible elegir entre una demostración de existencia pura y una demostración constructiva, aunque las primeras se aceptan como válidas, se prefiere siempre la demostración constructiva.)

La segunda propuesta para superar la crisis provino del propio Russell y dio origen a la *escuela logicista*. Para el logicismo, los axiomas de la lógica, junto con el lenguaje de la teoría de conjuntos, bastaban para funda-

mentar rigurosamente todo el trabajo matemático (la matemática sería, en cierto modo, una parte de la lógica). Para evitar las paradojas Russell proponía hacer una modificación profunda en el lenguaje de la matemática, con el objetivo de evitar las autorreferencias. Es decir, la estructura misma del lenguaje haría imposible que un enunciado hablara de sí mismo, como en el lenguaje natural sí sucede con "Este enunciado es falso" o "El conjunto no cumple este enunciado, que es el que lo define".

Según Russell, la autorreferencia era el origen de todas las paradojas; si el lenguaje impedía las autorreferencias entonces las paradojas desaparecerían por sí mismas. Lamentablemente, el sistema lingüístico de Russell, que implicaba la creación de infinitos niveles de metalenguaje, adoleció desde el principio de muchos problemas técnicos, que nunca pudieron ser resueltos satisfactoriamente (véase [18]). Finalmente, al cabo de algunos años, Russell desistió de su propuesta. (Sin embargo, como en el caso del intuicionismo, algunas de sus ideas sobreviven hasta hoy; por ejemplo, el entender a cada número racional como una clase de equivalencia de fracciones.)

Algún tiempo después, a lo largo de una serie de artículos publicados durante la década de 1920 (los más importantes pueden leerse en [9]), David Hilbert fue dándole forma a la propuesta que más nos interesa analizar: *el programa formalista o programa de Hilbert*. Hablaremos de él en la sección siguiente.

22. El programa de Hilbert

Aunque Hilbert apoyaba totalmente el estudio de la teoría de conjuntos, entendía, sin embargo, que la base de la matemática tenía que ser la aritmética. El motivo es que ésta, como ya hemos dicho, es "más simple" y, por ende, menos propensa a generar contradicciones o controversias. Si se tuviera una fundamentación rigurosa para la aritmética, decía Hilbert, se tendría un punto de partida sólido para toda la matemática, y se terminarían todas las polémicas desencadenadas por la crisis. Fiel a las ideas que venimos exponiendo, el programa de Hilbert proponía que esta fundamentación rigurosa de la aritmética se apoyase en la idea de algoritmo.

Toda teoría matemática, decía Hilbert, y en especial la aritmética, debía basarse en axiomas a partir de los cuales se irían demostrando los sucesivos teoremas de la teoría. El punto central de la propuesta de Hilbert es que debía existir un algoritmo capaz de verificar en una cantidad finita de pasos que esas demostraciones fuesen correctas. En otras palabras, para terminar con las discusiones acerca de la validez de ciertos métodos de demostración, Hilbert proponía establecer un criterio claro, objetivo y mecánico que permitiera resolver cualquier controversia. (En realidad, la propuesta de dar un fundamento axiomático riguroso para la aritmética aparecía ya en la conferencia de Hilbert de 1900. Era, de hecho, el segundo problema

de la lista. Aunque en esa primera versión no había ninguna referencia a la exigencia de verificación algorítmica. Esta idea es incorporada en la década de 1920.)

Es importante aclarar la idea de "axioma" que planteaba Hilbert. Según el concepto clásico, un axioma es una afirmación verdadera que es "evidente por sí misma". Para Hilbert, en cambio, los axiomas eran simplemente afirmaciones matemáticas que se elegían, por convención y por el consenso de la comunidad matemática, como puntos de partida de las demostraciones matemáticas. No se pedía, de ninguna manera, que fueran "evidentes".

Una demostración es, para Hilbert, una sucesión finita de enunciados tales que cada uno de ellos, o bien es un axioma, o bien se deduce de enunciados anteriores por aplicación de las reglas de la lógica (las llamadas *reglas de inferencia*). El último enunciado de la sucesión es el teorema que ha sido demostrado. (Técnicamente, una sucesión formada por un único axioma es una demostración que prueba, precisamente, ese axioma. Es decir, en esta presentación formal, los axiomas son, ellos mismos, teoremas.)

Para verificar algorítmicamente la validez de una demostración, ésta primero debe traducirse a un lenguaje formal previamente establecido. A continuación, el algoritmo recorrería, uno por uno, los enunciados que la forman y verificaría si se cumple alguna de las dos condiciones mencionadas: si el enunciado en cuestión es un axioma o si se deduce de manera inmediata de enunciados previos por aplicación de las reglas de inferencia. Esto implica a su vez, dos condiciones. En primer lugar, que los axiomas tienen que formar un conjunto recursivo (existe un procedimiento algorítmico que reconozca en una cantidad finita de pasos si un enunciado es, o no es, un axioma).

En segundo lugar, implica que las reglas de inferencia también deben ser recursivas: debe existir un algoritmo que, dadas ciertas premisas $P_1, P_2, P_3, \ldots, P_n$ y un enunciado Q, reconozca en una cantidad finita de pasos si Q es una consecuencia inmediata de las premisas.

Según la concepción de Hilbert, las demostraciones se basan en los axiomas específicos de la teoría matemática en la que se enmarcan. Pero también se basan en las reglas de inferencia y en los axiomas de la lógica. Los axiomas de la lógica, comunes a todas las teorías matemáticas, son enunciados universalmente válidos elegidos convenientemente. Estos enunciados "universalmente válidos" son verdaderos no importa el significado que se les atribuya a los símbolos y representan "reglas generales del pensamiento". Un enunciado universalmente válido es, por ejemplo, "P o $\neg P$". Por su parte, las reglas de inferencia tienen esta forma general: de los enunciados $P_1, P_2, \ldots, P_n$, se deduce Q. En la lógica matemática hay muchas reglas de inferencia, casi todas ellas heredadas de la lógica aristotélica clásica. Una regla de inferencia es el *modus ponens*, que dice que de "$P \Rightarrow Q$" y de P, se

deduce Q. Otra regla es el *modus tollendo tollens*, que dice que de "$P \Rightarrow Q$" y de $\neg Q$, se deduce $\neg P$. Ahora bien, si los axiomas de la lógica se eligen adecuadamente, entonces puede trabajarse con una única regla de inferencia: el *modus ponens* (lo cual reduce la dificultad del análisis lógico de las demostraciones). A modo de ejemplo, digamos que entre los axiomas de la lógica incluimos "$(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$" y que trabajamos con el *modus ponens* como única regla de inferencia. Demostremos que a partir de estos elementos podemos demostrar que de "$P \Rightarrow Q$" y de $\neg Q$ se deduce $\neg P$. Es decir, demostremos que no es necesario tomar al *tollendo tollens* como regla en sí misma. La demostración formal es la siguiente:

1. $P \Rightarrow Q$. (Premisa)
2. $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$. (Axioma de la lógica)
3. $\neg Q \Rightarrow \neg P$. (De 1 y 2 por *modus ponens*)
4. $\neg Q$. (Premisa)
5. $\neg P$. (De 3 y 4 por *modus ponens*)

El programa de Hilbert, además, pedía que los axiomas elegidos para la aritmética cumplieran las siguiente dos condiciones.

La primera es conocida como la condición de *completitud*, y dice que, dado cualquier enunciado aritmético P, o bien su negación $\neg P$, tiene que ser demostrable a partir de esos axiomas. En otros términos, dada cualquier conjetura no resuelta (como que todo número par sea la resta de dos primos consecutivos), los axiomas tienen que poder demostrarla o refutarla. Todo problema aritmético, tarde o temprano, será resuelto. Esta idea no era nueva en el pensamiento de Hilbert quien, en su conferencia de 1900, la formuló de la siguiente manera: "Debemos saber, sabremos" ("*Wir müssen wissen, wir werden wissen*", en alemán). Tan importante era esta frase para Hilbert que años más tarde pidió que se inscribiera en su lápida.

La segunda condición que debían cumplir los axiomas es la *consistencia*. Un sistema de axiomas es consistente si no existe un enunciado P, tal que él y su negación sean ambos demostrables. En otras palabras, un sistema de axiomas es consistente si nunca llevará a una paradoja similar a la de Russell. (Esta condición es fundamental porque, si una teoría es inconsistente, entonces todos los enunciados, tanto los verdaderos como los falsos, son demostrable a partir de ella. En otras palabras, una teoría inconsistente carece de toda posibilidad de distinguir la verdad de la falsedad.)

El programa de Hilbert indicaba también que debía existir un algoritmo que, una vez elegido un sistema de axiomas para la aritmética, pudiera verificar en una cantidad finita de pasos que ese sistema es consistente: la con-

sistencia de la aritmética debía ser verificable algorítmicamente. La consistencia de otros sistemas de axiomas, propuestos para teorías más complejas, se demostraría, o bien algorítmicamente, o bien tomando como base la consistencia de la aritmética.

A diferencia del intuicionismo, que propiciaba un cambio profundo en todos los métodos de demostración matemática, el programa de Hilbert proponía criterios de validez que se veían como razonables, a la vez que permitían que la práctica matemática continuara sin grandes modificaciones. Se ha dicho muchas veces que la intención de Hilbert era reemplazar el razonamiento matemático por operaciones mecánicas. Sin embargo, esta interpretación no es correcta. Hilbert dice explícitamente que los matemáticos seguirían trabajando como siempre lo han hecho, y que se crearía una ciencia paralela, que él llamaba metamatemática, que se ocuparía de la verificación mecánica de la validez de las demostraciones.

No solo el programa de Hilbert parecía muy razonable, sino que, además hacia fines de la década de 1920 comenzó a recibir buenas noticias. Por ejemplo, en 1929, en su tesis doctoral, Kurt Gödel (1906–1978) demostró que existe un conjunto recursivo de axiomas lógicos y de reglas de inferencia que permite representar todos los razonamientos matemáticos usuales. Este teorema de Gödel, no tan famoso como el que estudiaremos en la próxima sección, es conocido como su teorema de *completitud*. (En ese momento todavía no se contaba con una definición rigurosa del concepto de algoritmo, pero basándose solamente sobre la idea intuitiva estaba claro que los axiomas lógicos y las reglas de inferencia de Gödel eran recursivas. De forma similar, no se necesita una definición rigurosa del concepto de algoritmo para aceptar que el conjunto de los números primos es recursivo.)

También en 1929, el matemático polaco Mojesz Presburger (1904–1943) demostró que, para una versión restringida de la aritmética, hoy conocida como la aritmética de Presburger, puede darse un conjunto recursivo, consistente y completo de axiomas. (En la aritmética de Presburger solo existe la operación de suma, la cual puede aplicarse una vez, dos veces, tres veces, pero no n veces con un n arbitrario, por lo que en esa teoría la suma no permite definir la multiplicación.)

La impresión general era que solo faltaba extender el trabajo de Presburger a la aritmética usual. El triunfo del programa de Hilbert parecía estar a la vuelta de la esquina.

En septiembre de 1930, en Königsberg, la ciudad natal de Hilbert (el dato no es casual), justo la víspera del día en que el parlamento de Königsberg lo nombraría ciudadano ilustre (el dato tampoco es casual) se cerraba un congreso internacional dedicado a debatir el problema de los fundamentos de la matemática. El clima general era de felicitación a Hilbert, quien no asistió en persona por problemas de salud, aunque sí estaba presente su ilustre

discípulo, John von Neumann (1903–1957). La conferencia de cierre estuvo a cargo de Arendt Heyting (1898–1980), el más importante de los representantes del intuicionismo presentes. Los intuicionistas habían sido los más férreos opositores al programa de Hilbert, y el ofrecimiento de la conferencia de cierre era, en este caso, una oportunidad para rendirse con honor. Heyting, con estas palabras, aceptó la derrota: "Si se completa el programa de Hilbert hasta los intuicionistas abrazarán el infinito".

Loas, medalla y felicitación para Hilbert.

Pero, dicen los relatos, justo en ese momento un joven austríaco casi desconocido, la personificación del estereotipo del matemático delgado, tímido, pálido y de anteojos, que dos días antes había dado una conferencia explicando su tesis doctoral, levantó la mano y dijo que tenía algo importante que anunciar. Ese joven era Kurt Gödel, y lo que tenía que anunciar era que había demostrado dos teoremas que probaban que el programa de Hilbert era irrealizable: los célebres teoremas de incompletitud de Gödel. Hablaremos de ellos en las secciones que siguen. (El artículo con los teoremas de Gödel, que puede leerse en [6], recién se publicó a principios de 1931, porque Gödel se tomó algunos meses para revisar que su demostración no contuviera errores, y que estuviera claramente explicada. Fueron, para él, meses de mucha tensión. Tanto fue así que después de publicado el artículo sufrió un colapso nervioso que lo dejó postrado en cama casi medio año.)

23. El primer teorema de Gödel

El primer teorema de incompletitud de Gödel dice que, si se elige para la aritmética un sistema de axiomas que sea recursivo y consistente, y si las reglas de inferencia lógica también son recursivas, entonces, contrariamente a lo que pide el programa de Hilbert, el sistema de axiomas será necesariamente incompleto. Es decir, existirá un enunciado aritmético P tal que ni él, ni su negación, son demostrables a partir de los axiomas (se dice que ese enunciado P es *indecidible*). Dicho de otro modo, si los axiomas son recursivos y consistentes, y las reglas lógicas son también recursivas, entonces habrá un problema que no podrá ser resuelto aplicando esos axiomas y reglas lógicas, porque la conjetura P no podrá ser nunca demostrada ni refutada.

En lo que sigue, vamos a apelar a los conceptos de conjunto recursivo y recursivamente numerable para dar una demostración de este teorema. Obviamente, no se trata de la demostración original de Gödel, ya que éste demostró su teorema en 1930, mientras que Post introdujo sus ideas recién en 1944. Por otra parte, nuestra demostración será por el absurdo, mientras que la demostración original de Gödel es constructiva: Gödel exhibe un algoritmo que, dado los axiomas y las reglas de inferencia, permite escribir un enunciado P que es indecidible. El artículo original de Gödel puede leerse en [6] y su prueba constructiva también se encuentra en [12] y en [15].

(Una observación acerca de la terminología. La palabra "recursivo", como sinónimo de "algorítmico", fue usada por primera vez por Gödel en el artículo de 1931. Post retoma el término en 1944 al hablar de conjuntos recursivos o recursivamente numerables.)

Supongamos, entonces, por el absurdo, que es posible dar un sistema de axiomas para la aritmética que sea recursivo, consistente y completo. Comencemos diciendo en primer lugar que, dado que el sistema de axiomas es recursivo, entonces el conjunto de todos los enunciados demostrables a partir de los axiomas es recursivamente numerable. Es decir, es posible programar la computadora permanente de tal modo que vaya imprimiendo, uno por uno, todos los enunciados demostrables a partir de los axiomas.

Para ello, numeramos los axiomas como $1, 2, 3, 4, 5, \ldots$ Después, programamos la computadora permanente de manera que imprima, en orden, todas las demostraciones de longitud 1 (la longitud es la cantidad de enunciados que forman la demostración) basadas en el axioma 1. Luego, todas las demostraciones de longitud a lo sumo 2, basadas en algunos de los axiomas 1 y 2. Luego, todas las demostraciones de longitud a lo sumo 3, basadas en algunos de los axiomas 1, 2 y 3. Y así sucesivamente.

Si el sistema es, como estamos suponiendo, completo, entonces, dado cualquier enunciado aritmético P, o bien su negación $\neg P$ (uno y solo uno de ambos) será impreso, tarde o temprano, en una cantidad finita de pasos.

Tomemos ahora el conjunto S_0. Como dijimos en la sección 19, el enunciado "$n \in S_0$" es aritmético (puede traducirse al lenguaje formal de la aritmética). Por lo tanto, tarde o temprano, la computadora permanente demostrará e imprimirá el enunciado $n \in S_0$, o bien demostrará e imprimirá su negación, $n \notin S_0$. Pero entonces tenemos un algoritmo para determinar en una cantidad finita de pasos si un número n pertenece, o no, a S_0: basta con esperar hasta que se imprima alguno de los dos enunciados. En conclusión, si hubiera un sistema de axiomas para la aritmética que sea recursivo, consistente y completo, el conjunto S_0 sería recursivo. Pero esto es un absurdo porque, como hemos visto en la sección 18, en realidad no lo es.

Concluimos así que si el sistema de axiomas es recursivo y consistente, entonces no puede ser completo porque existe algún número natural M tal que ni el enunciado $M \in S_0$, ni su negación, son demostrables. Hemos probado, entonces, el primer teorema de incompletitud de Gödel.

24. El segundo teorema de Gödel

El segundo teorema de incompletitud dice, en principio, que, el problema de determinar si un sistema de axiomas para la aritmética es consistente no es resoluble algorítmicamente. En una versión más amplia, el teorema dice que la consistencia de un sistema axiomático solo puede ser demostrada si se supone previamente la consistencia de un sistema más complejo (y,

por lo tanto, cuya consistencia es aún más dudosa). Vamos a comentar las ideas principales de la demostración de la primera versión del teorema; la segunda, lamentablemente, excede los alcances de este texto.

Observemos, en primer lugar, que, si $A_1, A_2, A_3, \ldots, A_k$ es un sistema consistente de axiomas y P es un enunciado cualquiera, entonces:

1) Si P se demuestra a partir de los axiomas, entonces el sistema $A_1, A_2, A_3, \ldots, A_k, P$ que agrega como nuevo axioma, sigue siendo consistente; pero, por el contrario, $A_1, A_2, A_3, \ldots, A_k, \neg P$ es inconsistente.

   (¿Es lícito agregar P, que es un teorema, como nuevo axioma? Como dijimos antes, la respuesta es que sí, porque los axiomas son solo los puntos de partida de las demostraciones que son elegidos convencionalmente. Por otra parte, es claro que si agregamos como axioma un teorema que ya había sido demostrado entonces estamos agregando una afirmación redundante y la naturaleza del sistema axiomático no cambia. Si, por el contrario, agregamos la negación de un teorema, por ejemplo, si en la aritmética tomamos como axioma $2 + 2 = 5$, el sistema se vuelve inconsistente.)

2) Si $\neg P$ se demuestra a partir de los axiomas, entonces el sistema $A_1, A_2, A_3, \ldots, A_k, \neg P$ es consistente, mientras que $A_1, A_2, A_3, \ldots, A_k, P$ no lo es. (Análogo al caso anterior, pero intercambiando P con $\neg P$.)

3) Finalmente, si P es indecidible con respecto a los axiomas, entonces tanto $A_1, A_2, A_3, \ldots, A_k, P$ como $A_1, A_2, A_3, \ldots, A_k, \neg P$ son ambos consistentes.

En resumen, si $A_1, A_2, A_3, \ldots, A_k$ es consistente, y si existiera un algoritmo que determina en una cantidad finita de pasos si un sistema de axiomas cualquiera es consistente o inconsistente, ese mismo algoritmo nos permitiría determinar si un enunciado P se demuestra, es refutado, o es indecidible con respecto a esos axiomas. Para ello, bastaría aplicar el algoritmo de verificación de la consistencia al sistema $A_1, A_2, A_3, \ldots, A_k, \neg P$ y al sistema $A_1, A_2, A_3, \ldots, A_k, P$. De este modo se puede determinar si se cumple la condición 1), la 2) o la 3) respectivamente.

Tomemos ahora como $A_1, A_2, A_3, \ldots, A_k$ los llamados *axiomas de Peano*, que pueden verse en [12], y que son considerados los axiomas estándar de la aritmética. Este sistema de axiomas es recursivo. Aceptaremos también, como se hace habitualmente, que es consistente (hecho que solo puede demostrarse rigurosamente sobre la base de un sistema más complejo). Por otra parte, dado que S_0 es recursivamente numerable, puede demostrarse que los axiomas de Peano $A_1, A_2, A_3, \ldots, A_k$ en cierto modo "generan" S_0. Esto significa que todo enunciado de la forma $n \in S_0$, si es verdadero, entonces

es demostrable a partir de los axiomas de Peano. También serán demostrables algunos enunciados verdaderos de la forma $n \notin S_0$.

Conviene hacer aquí una observación. Decir que "todo enunciado de la forma $n \in S_0$, si es verdadero, es demostrable" parece contradecir lo que afirmamos en la sección anterior. ¿No habíamos dicho que existe algún número natural M tal que ni $M \in S_0$, ni $M \notin S_0$, son demostrables? Parece contradictorio, pero no lo es porque los casos en que ni $M \in S_0$, ni $M \notin S_0$, son demostrables solamente ocurrirán cuando $M \in S_0$ sea falso, es decir, solo pueden suceder para valores de M que no pertenezcan a S_0.

En resumen, si tomamos los axiomas de Peano entonces, para cada n natural se puede dar una y solo una de las siguientes tres situaciones:

1. $n \in S_0$ es demostrable a partir de $A_1, A_2, A_3, \ldots, A_k$. Esto implicaría que n pertenece a S_0.

2. $n \notin S_0$ es demostrable a partir de $A_1, A_2, A_3, \ldots, A_k$. Esto implicaría que n no pertenece a S_0.

3. $n \in S_0$ es indecidible con respecto a $A_1, A_2, A_3, \ldots, A_k$. Esto implicaría que n no pertenece a S_0.

Como decíamos más arriba, si existiera un algoritmo capaz de determinar si un sistema de axiomas es consistente o inconsistente, ese mismo algoritmo nos diría, para cada n, cuál de las tres situaciones que acabamos de enumerar sucede. Luego, habría un algoritmo capaz de determinar si n pertenece, o no pertenece a S_0; sin embargo, esto es absurdo porque S_0 no es recursivo. El absurdo proviene de suponer que existe un algoritmo que determina si un sistema de axiomas es consistente o inconsistente; ese algoritmo, entonces, no puede existir. De este modo queda demostrado el segundo teorema de incompletitud de Gödel, en su primera versión.

25. ¿Se resolvió la crisis?

Hubo tres respuestas a la crisis de los fundamentos; el logicismo, que no logró superar sus propios problemas técnicos; el intuicionismo, que no pudo imponer su punto de vista; y el programa de Hilbert, o formalismo, que, vemos ahora, resultó ser irrealizable. ¿Qué sucedió entonces?

Poco tiempo después de que Gödel publicara sus resultados, el nazismo llegó al poder en Alemania y algunos años más tarde comenzó la Segunda Guerra Mundial. Muchos de los involucrados en el debate sobre los fundamentos, entre ellos el propio Gödel, debieron huir a Estados Unidos; otros, como von Neumann, además de huir, quedaron involucrados en tareas relacionadas con la guerra. Turing comenzó a trabajar para el ejército inglés, descifrando los códigos secretos alemanes. Presburger, que se quedó en Polonia, murió en un campo de concentración. Frente a una situación tan te-

rrible y oscura, el problema de los fundamentos de la matemática pasó a un segundo plano.

La discusión sobre la crisis fue retomada por Nicolás Bourbaki, que no es una persona, sino el nombre de un grupo de matemáticos franceses creado en 1935, aunque comenzó realmente su actividad diez años más tarde, después de terminada la guerra. El grupo Bourbaki que, con miembros renovados sigue activo hasta hoy, domina la concepción actual que los matemáticos tienen sobre los fundamentos de su ciencia.

La filosofía de Bourbaki retoma esencialmente las ideas de Hilbert, aunque sin la exigencia de la verificación algorítmica de las demostraciones. Para Bourbaki, como de hecho sucede, toda teoría matemática debe estar basada en axiomas, a partir de los cuales se demuestran los sucesivos teoremas. No se pide que los axiomas sean "evidentes", sino que, como decía Hilbert, simplemente son elegidos convencionalmente por el consenso de la comunidad matemática, a veces meramente por razones de conveniencia. La única exigencia es que formen un sistema consistente.

El segundo teorema de incompletitud implica que no hay forma de garantizar más allá de toda duda que los axiomas de cualquier teoría matemática relevante son consistentes. Pero los matemáticos toman esta situación con bastante tranquilidad. Aunque tienen que convivir con la idea de que tal vez algún día un nuevo Russell encuentre alguna inconsistencia, aceptan este riesgo sin mayores preocupaciones. Dado que los axiomas solo son convencionales, si surgiera una contradicción, entienden que ésta se resolvería cambiando adecuadamente el sistema de axiomas.

En la práctica, el método axiomático de Bourbaki funciona perfectamente. Los matemáticos pueden hacer demostraciones y resolver problemas sin necesidad de debatir sobre cuestiones tales como, por ejemplo, si la matemática es solo una creación mental humana, o si los objetos matemáticos existen en una realidad platónica abstracta. Bourbaki considera irrelevante este tipo de discusiones, que, por el contrario, preocupaban centralmente a personas como Russell y Brouwer.

Sin embargo, nos permitimos estar en desacuerdo con esta actitud displicente hacia las preguntas de la filosofía de la matemática. Entendemos, por el contrario, que cuestiones como la relevancia de ciertos problemas matemáticos, o sobre la validez de ciertos métodos, o sobre preguntas relacionadas con la enseñanza de la matemática, solo pueden ser planteadas razonablemente sobre la base de un debate filosófico como el que Bourbaki descarta.

26. Una pregunta sobre historia

Antes de seguir adelante, hay dos preguntas que parece necesario discutir y que se refieren al desarrollo histórico de los temas que estamos tratando.

La primera pregunta es: ¿cómo se resolvió el problema de la inconsistencia de la teoría de conjuntos (de la que hablamos en la sección 17)?

En los primeros años del siglo xx muchos matemáticos, Cantor entre ellos, se convencieron de que todos los problemas de la teoría de conjuntos (de los cuales la paradoja de Russell fue el más grave, pero no el único) surgen de considerar conjuntos "demasiado abarcadores", como el conjunto de todos los conjuntos. Con esta idea en mente, en 1908 Ernst Zermelo (1871–1953) propuso un sistema de axiomas para la teoría de conjuntos. Este sistema estaba diseñado de tal modo que impedía demostrar la existencia de esos conjuntos abarcadores, por lo que, se esperaba, estaría libre de paradojas. El sistema de Zermelo fue perfeccionado en 1922 por Abraham Fraenkel (1891–1965); los axiomas resultantes forman la llamada *teoría de Zermelo-Fraenkel* (abreviada ZF).

Posteriormente se propusieron otras teorías axiomáticas de conjuntos, como NBG (por von Neumann, Gödel y Bernays) y MK (por Morse, Kelley); en estas sí existen los conjuntos muy abarcadores, solo que tienen propiedades más restringidas (los conjuntos muy abarcadores son llamados "clases propias" y, por ejemplo, no pueden ser elementos de conjuntos más grandes).

El grupo Bourbaki descarta por irrelevante la cuestión de cuál de las tres, ZF, NBG o MK, es la "verdadera" teoría de conjuntos. Si todas son consistentes entonces todas son válidas. Sin embargo, suele tomarse, simplemente porque es la más sencilla, a la teoría de Zermelo-Fraenkel como la teoría de conjuntos estándar.

Sin embargo, es interesante señalar que en su práctica diaria los matemáticos (con la sola excepción de los lógicos y de los expertos en teoría de conjuntos) siguen usando la teoría intuitiva de Cantor, precisamente porque es más intuitiva. El modo que tienen de evitar las paradojas consiste en fijar un marco de referencia, como $\mathbb{R}$ o el conjunto de todos los subconjuntos de $\mathbb{N}$, y nunca excederlo; de este modo se evita trabajar conjuntos muy abarcadores.

La segunda pregunta que es interesante plantear es: ¿cómo definió Gödel, en 1931, el concepto de "recursivo", si la definición rigurosa de algoritmo recién fue presentada en 1937?

Gödel da una definición propia de la noción de algoritmo, que no detallamos aquí porque es demasiado técnica y no hace a nuestra discusión. Sin embargo, era consciente de que su definición podía ser limitada, en el sentido de que podrían llegar a existir procedimientos algorítmicos que no estuvieran contemplados por esta. Tanto es así, que Gödel acota, por ejemplo, que el segundo teorema de incompletitud puede no estar completamente demostrado, porque podrían existir procedimientos algorítmicos de verificación de las demostraciones que no son abarcados por su definición.

Posteriormente se demostró que, en efecto, existen algoritmos (en el sentido de máquinas de Turing) que la definición de Gödel no incluye. Sin embargo, también se comprobó que la demostración de los teoremas de incompletitud puede adaptarse para contemplar esta situación. De hecho, en 1963 Gödel agregó a su artículo la siguiente posdata: "Como consecuencia de avances posteriores, en particular del hecho de que gracias a los trabajos de A. M. Turing ahora disponemos de una definición precisa e indudablemente adecuada de la noción de sistema formal [aquél donde las demostraciones son verificables algorítmicamente], ahora es posible dar una versión completamente general de los teoremas VI y XI [que, en la enumeración del artículo, son respectivamente el primero y el segundo teorema de incompletitud]".

Podemos hacer un tercer comentario, que parte de esta pregunta: ¿existe un conjunto que tenga un cardinal mayor que el de $\mathbb{N}$, pero menor que el de $\mathbb{R}$? En 1877 Georg Cantor conjeturó que tal conjunto no existía, es decir, que no había un nivel de infinitud intermedio entre el de $\mathbb{N}$ y el de $\mathbb{R}$. A esta conjetura se la conoce como la hipótesis del continuo, y durante muchos años Cantor intentó demostrarla, aunque sin éxito. En su conferencia de 1900, Hilbert puso el problema de la hipótesis del continuo en el primer lugar de su lista. ¿Fue resuelto este problema?

En 1939 Gödel probó que, si la teoría de conjuntos de Zermelo-Fraenkel es consistente, entonces la hipótesis del continuo no puede ser refutada a partir de ella. Pero, por otra parte, en 1963 el estadounidense Paul Cohen (1934–2007) demostró que tampoco puede ser demostrada. Es decir, la hipótesis del continuo es indecidible con respecto a la teoría estándar de conjuntos.

Entonces, ¿es verdadera o es falsa? Para la filosofía del grupo Bourbaki la pregunta es irrelevante. De hecho, así como existen geometrías no euclidianas, perfectamente válidas, que niegan el quinto postulado de Euclides; Bourbaki acepta la existencia de dos teorías de conjuntos válidas. Una teoría "cantoriana" que incorpora la hipótesis del continuo como nuevo axioma, y una teoría "no cantoriana" que incorpora como axioma su negación.

27. La respuesta al décimo problema

Quedaba pendiente la respuesta al décimo problema de Hilbert: ¿existe un algoritmo que, dada una ecuación diofántica cualquiera, permita determinar en una cantidad finita de pasos si esa ecuación tiene, o no, solución? Este problema fue resuelto, en 1970, por la matemática estadounidense Julia Robinson (1919–1985) y a esta altura del desarrollo de nuestro tema los lectores no se sorprenderán al saber que la respuesta es que no existe un algoritmo así. Es decir, el problema de determinar si una ecuación diofántica tiene solución no es resoluble algorítmicamente. (En realidad, Julia Robin-

son completó casi toda la demostración de la solución del décimo problema, le faltó solamente un pequeño detalle. Este último punto fue completado por el matemático ruso, en ese tiempo soviético, Yuri Matiyasévich, nacido en 1947. Sin embargo, en un gesto de machismo flagrante, el resultado es conocido como el teorema de Matiyasévich, y no como teorema de Robinson.)

La solución del décimo problema (que puede leerse en [4]) puede desarrollarse mediante ideas muy similares a las que hemos mostrado para probar que el *halting problem* es irresoluble algorítmicamente. Para comenzar, se demuestra que, así como hay una numeración de las máquinas de Turing, existe asimismo una numeración de todos los polinomios, en una o varias variables, con coeficientes enteros. Digamos que esa numeración es: $P_1, P_2, P_3, P_4, \ldots$

A continuación, de manera similar a lo hecho para S_0, definimos ahora el siguiente conjunto:

$$D_0 = \{n : \text{ la ecuación } P_n = n \text{ no tiene solución } \}.$$

Puede demostrarse que D_0 es recursivamente numerable. Para probarlo, recordemos primero que, según vimos en la sección 12, los números enteros (posibles soluciones de una ecuación de una sola incógnita) pueden ponerse en correspondencia uno-a-uno con $\mathbb{N}$. En la sección 4 vimos que lo mismo sucede con las ternas de números enteros (posibles soluciones de una ecuación de tres incógnitas). Análogamente para los pares de números enteros, y así sucesivamente.

A continuación, programamos la computadora permanente de tal modo que haga, en paralelo, una búsqueda por "fuerza bruta" de las posibles soluciones de todas las ecuaciones de la forma $P_n = n$. Para ello, seguimos una vez más el esquema $i_{11}, i_{21}, i_{12}, i_{13}, i_{22}, i_{31}$ de la figura 3.13, solo que ahora i_{nm} indica que hay que testear la m-ésima solución posible de la ecuación $P_n = n$. En otras palabras:

1. La computadora toma la primera solución posible (el primer entero, o el primer par, etc. dependiendo de la cantidad de incógnitas) para $P_1 = 1$, y comprueba si es, o no, realmente solución de la ecuación.

2. Hace lo mismo para la primera solución posible para $P_2 = 2$.

3. Prueba la segunda solución posible para $P_1 = 1$.

Y así sucesivamente. Cuando encuentra una solución, imprime el valor correspondiente de n y ya no vuelve a trabajar con esa ecuación. De este modo, el programa irá imprimiendo todos los elementos de D_0 que, entonces, forman un conjunto recursivamente numerable.

Sin embargo, puede probarse que D_0^c, el complemento de D_0, no es recursivamente numerable (la demostración, sin embargo, excede los alcances de este texto). Deducimos así que D_0 no es recursivo. Es decir, no existe un algoritmo que, dado el número n, determine en una cantidad finita de pasos si $P_n = n$ tiene solución. En consecuencia, el problema de determinar si una ecuación diofántica tiene, o no, solución es irresoluble algorítmicamente.

28. Reflexiones finales

El primer teorema de incompletitud de Gödel dice que, fijados los axiomas y las reglas de inferencia de alguna teoría matemática clásica (como el análisis, la aritmética o la teoría de conjuntos), siempre existirán, en esa teoría, problemas que son irresolubles. Pero irresolubles a partir de esos axiomas, nada impide que puedan resolverse si se agregan nuevos axiomas, o se crean nuevos métodos, o se toman los axiomas de alguna otra teoría. Lejos de ser una observación puramente hipotética, hay muchísimos casos que ejemplifican esta situación. Comentemos, a modo de ejemplo, los problemas que mencionamos en la introducción: la cuadratura del círculo y la resolubilidad de la ecuación de quinto grado.

La cuadratura del círculo es el problema que pide, dado un círculo cualquiera, construir con regla no graduada y compás un cuadrado con la misma área que el círculo dado. Este problema geométrico, planteado en Grecia en el siglo v a. C., fue resuelto recién en 1882. La respuesta es que la construcción es irrealizable, pero la respuesta no se basa en los métodos de la geometría, sino del álgebra y del análisis.

El segundo problema fue planteado por algebristas italianos del siglo xvi. La pregunta es si existe una fórmula resolvente para la ecuación de quinto grado, similar a la bien conocida para la ecuación cuadrática (en el siglo xvi se habían encontrado fórmulas para las ecuaciones de tercer y cuarto grado). Este problema fue resuelto en la década de 1820, cuando se demostró que tal fórmula no existe. La demostración, sin embargo, requirió la creación del álgebra abstracta y, en particular, de la teoría de grupos.

Un tercer ejemplo, más reciente, es el caso del último teorema de Fermat, una conjetura planteada por Pierre de Fermat (1601–1665) en 1637. La conjetura dice que, si $n > 2$ entonces la ecuación diofántica $x^n + y^n = z^n$ no tiene soluciones enteras positivas. Se trata, evidentemente, de un problema formulado en el campo de la aritmética; sin embargo, su demostración, hallada por Andrew Wiles en 1995, emplea métodos que combinan el álgebra abstracta con las funciones de variable compleja.

En resumen, podría pensarse a primera vista que la palabra incompletitud que se asocia a los teoremas de Gödel tiene una connotación negativa, porque incompleto es aquello a lo que le falta algo, que carece de algo. Sin embargo, en este caso la idea de incompletitud nos habla de un aspecto po-

sitivo de la matemática. Nos dice que la matemática nunca se terminará, que siempre habrá problemas que nos obliguen a crear nuevos métodos o nuevas teorías. Nos dice que la matemática, como construcción colectiva humana, es un camino infinito.

Breve biografía del autor

Gustavo Piñeiro es licenciado en Ciencias Matemáticas (FCEN-UBA) y doctor en Filosofía (FFyL-UBA). Desde hace más de treinta años se dedica a la docencia, principalmente en institutos de formación docente, y también a la divulgación de la matemática. Como divulgador ha publicado numerosos artículos y libros de divulgación matemática, entre ellos biografías de Georg Cantor, Kurt Gödel, Bernhard Riemann, Luca Pacioli y Jakob Bernoulli (publicadas por RBA y distribuidas por el diario *La Nación* en 2017/2018), "Gödel para Todos" (en colaboración con Guillermo Martínez), "El infinito en matemáticas". Actualmente, escribe el blog "El Topo Lógico" `https://eltopologico.blogspot.com/` y publica videos en el canal de YouTube "Gustavo Piñeiro Matemática" (usuario: gpineiro2002).

Referencias

[1] L.E.J. Brouwer (1923). "On the significance of the principle of excluded middlein mathematics, especially in function theory". En: *Intuitionistic reflections on formalism*. Ed. por van Heijenoort. 1976, págs. 334-345.

[2] A. M. Church (1936). "A Note on the *Entscheidungsproblem*". En: *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Ed. por M. Davis. 1993.

[3] A. M. Church (1936). "An Unsolvable Problem of Elementary Number Theory". En: *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Ed. por M. Davis. 1993, págs. 89-107.

[4] M. Davis. *Computability and Unsolvability*. Dover, 1982.

[5] M. Davis (Comp). *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Dover, 1993.

[6] K. Gödel. *Obras completas*. Alianza Editorial.

[7] J. Gray. *El reto de Hilbert (Los 23 problemas que desafiaron a la matemática)*. Crítica, 2000.

[8] J. van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Source books in the history of the sciences. Harvard University Press, 1995.

[9] D. Hilbert. *Fundamentos de las Matemáticas*. Facultad de Ciencias, Univ. Nac. Autóm. de México.

[10] R. Houston. *42 is the answer to the question: what is* $(-805387388\ 12075974)^3 + 80435758145817515^3 + 126012329733563 1^3$*?* `https://aperiodical.com/2019/09/`. Consultada el 1 de febrero de 2023.

[11] P. S. de Laplace. *Ensayo filosófico sobre las probabilidades*. Altaya.

[12] G. Martínez y G. Piñeiro. *Gödel (Para Todos)*. Seix Barral, 2009.

[13] S. Miller. *The answer to life, the universe, and everything*. `http://news.mit.edu/2019/answer-life-universe-and-everything-sum-three-cubes-mathematics-0910`. Consultada el 1 de febrero de 2023.

[14] G. Piñeiro. *Georg Cantor (El infinito matemático)*. Barcelona: RBA, 2013.

[15] G. Piñeiro. *Kurt Gödel (Los teoremas de incompletitud)*. Barcelona: RBA, 2012.

[16] E. Post. "Recursively Enumerable Sets and their Decision Problems". En: *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Ed. por M. Davis. 1993, págs. 305-337.

[17] B. Russell. *Introducción a la Filosofía*. Paidós, 1988.

[18] B. Russell (1902). "Letter to Frege". En: *Intuitionistic reflections on formalism*. Ed. por van Heijenoort. 1976, págs. 124-125.

[19] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". En: *Proceedings of the London Mathematical Society* S2-42.1 (1937), págs. 230-265.

Parte II

Turing, la Segunda Guerra, enigmas y secuelas

Capítulo 4
El contexto de la obra de Alan Turing: la Segunda Guerra Mundial

Daniel Lvovich[*]

Las dos guerras mundiales pueden ser estudiadas como resultado de una crisis general que abarcó buena parte de la primera mitad del siglo xx, como dos componentes íntimamente vinculadas de un mismo proceso histórico. Entre las causas estructurales que explican la Primera Guerra Mundial se encuentra el modo en que las principales potencias se posicionaron en relación con el reparto del poder y la riqueza del mundo tras el fin de la época del imperialismo, es decir, el modo en que países que lograron alcanzar un desarrollo económico tardío en relación con las potencias que lo habían hecho desde la primera mitad del siglo xix buscan posicionarse cuando ya una parte del mundo estaba repartido en términos políticos y económicos.

La Primera Guerra Mundial fue un conflicto largo y enormemente destructivo, que se peleó fundamentalmente en territorios europeos. La Segunda Guerra Mundial en este contexto muy general se considera normalmente como uno de los resultados del modo en el que se tramitó la paz en la Primera Guerra Mundial, poniendo condiciones políticas y económicas sobre los países derrotados, en particular sobre Alemania, que contribuyeron no solo a debilitar la naciente democracia en ese país - en la llamada República de Weimar - sino a generar unas condiciones económicas que a la vez explican en parte los orígenes de la crisis de 1930. Esta crisis provocó a su vez conmociones sociales a nivel global, y la quiebra de la democracia en buena parte de Europa y de América Latina.

Visto más de cerca, mientras que para la Primera Guerra Mundial hay mucha discusión sobre las responsabilidades, que no son las causas estructura-

[*] N. de la R.: desgrabación de la charla original (UNGS, 2017): Paula Albarracín.

les, sino sobre las decisiones políticas que derivaron en el estallido del conflicto en 1914, en el caso de la Segunda Guerra Mundial no hay dudas: está claro que el régimen nazi y el militarismo japonés toman la iniciativa y son los responsables por haber desarrollado la guerra. Son regímenes basados en el expansionismo, en el rechazo de la completa herencia de la revolución francesa con su idea de igualdad entre los hombres, en la concepción de la supremacía natural de sus propias naciones.

Las potencias, que serían luego los aliados en la guerra, no hicieron nada para evitar una serie de movimientos que hoy sabemos que anticipaban la guerra. Entre ellos podemos reseñar la invasión japonesa de Manchuria, la invasión italiana de Etiopía en 1935, la intervención italiana y alemana en la guerra civil española del lado del franquismo, frente a la cual ni Inglaterra ni Francia intervienen, mientras la Unión Soviética lo hace tarde y de un modo que a la larga se reveló desastroso. Algo similar ocurrió ante la anexión de Austria por Alemania, seguida por la anexión de casi todo el territorio de Checoslovaquia por el Tercer Reich. Hay una política de apaciguamiento fracasada por parte de Francia e Inglaterra ante el rearme y el expansionismo alemán. La información sobre la Segunda Guerra Mundial que exponemos en este trabajo se basa en dos textos fundamentales: Eric Hobsbawm, *Historia del Siglo* xx, 1914- 1991, Barcelona, Crítica, 1995 y Enzo Traverso, *A sangre y fuego: de la guerra civil europea 1914-1945*, Buenos Aires, Prometeo, 2009.

La guerra comenzó el primero de septiembre de 1939 cuando Alemania invadió Polonia, tras un previo acuerdo entre Stalin y Hitler que implicó el desmembramiento de aquel país. Ante este evento Francia e Inglaterra declararon la guerra a Alemania. En pocas semanas Alemania derrotó y ocupó Noruega, Dinamarca, los Países Bajos, Bélgica y Francia. Italia se sumó a la guerra y avanzó sobre Yugoslavia. Es un proceso muy rápido, a mediados del año 40 Inglaterra se encuentra sola en la lucha contra Alemania, que además controla los mares con sus submarinos. El control del Océano Atlántico era muy importante porque a través del mar llegaban los abastecimientos militares enviados desde Estados Unidos y los alimentos que venían de otros lugares del mundo, como la Argentina, cuya neutralidad era una condición necesaria para poder abastecer a Inglaterra. La guerra en esta etapa se libra sobre todo en el mar y a través de ataques aéreos, ya que Alemania y sus aliados controlan prácticamente todo el territorio europeo.

La guerra entra en una segunda fase cuando el 22 de junio de 1941, Alemania invade la Unión Soviética. Esta es una guerra con una característica distinta a la guerra anterior, porque la guerra del frente oriental y sobre todo en Rusia fue una guerra de exterminio. El proyecto nazi para Europa era un predominio alemán, un predominio ario sobre el conjunto de Europa, en el cual los europeos occidentales podrían integrarse, no en condiciones de

igualdad, pero recibiendo un trato relativamente bueno. Para Europa oriental, pueblos eslavos, los nazis solo ofrecían muerte, servidumbre y esclavitud. De hecho, en Polonia el nazismo extermina físicamente al grueso de la clase dirigente, intelectual, eclesiástica, política en el periodo que va de 1939 a 1941. Las dotaciones de alimentos, las calorías que se suministraban a la población polaca no llegaban al nivel de subsistencia, y eran muy inferiores a las que recibían los franceses, belgas u otras naciones ocupadas por la Alemania Nazi. Cuando empieza la guerra en Rusia hay una orden de fusilar a todos los comisarios políticos del partido comunista y a todos los judíos. Esta es una primera fase del exterminio, se estima que fueron 1,5 millones de judíos fusilados allí y alrededor de 1 millón de dirigentes del partido comunista y del estado soviético.

Se trató de una guerra de exterminio y esclavitud, que buscaba, siguiendo la idea de Hitler, dominar todo el enorme territorio de la Unión Soviética en tres meses. Ese objetivo no se logró por la enorme resistencia rusa. Hay ciudades como Leningrado que estuvieron sitiadas por años, se calcula que mueren por hambre en estos sitios casi un millón de personas. También en el año 1941 la guerra se internacionaliza cuando Japón ataca la base militar estadounidense de Pearl Harbor. Con ello se comienza a desarrollar la guerra en el Océano Pacifico y a la vez la Alemania nazi le declara la guerra a Estados Unidos. Ya antes, a partir del año 1940, también la guerra había llegado a África, donde las tropas inglesas se enfrentan a los ejércitos de Alemania e Italia.

Desde 1942 Estados Unidos logra revertir a muy alto costo la situación en el Océano Pacifico comenzando su avance hacia Japón, mientras la URSS tras detener el avance alemán en Stalingrado comienza una larga contraofensiva. Tras los desembarcos aliados en Sicilia en 1943 y en Normandía en 1944 comienza un movimiento desde ambos frentes que culminaría con la derrota alemana y de sus aliados europeos.

Entre los años 1941 y 1945, los únicos países que no participaron de la guerra son España, Portugal, Suecia, Suiza y Turquía en Europa, Afganistán en Asia y la mayor parte de los países latinoamericanos, aunque hay algunos que participan directamente con tropas como Brasil y algunos países centroamericanos. Fue una guerra de dimensiones colosales, nunca antes vistas. Algunas cifras dan cuenta de esta magnitud de la destrucción. En la Segunda Guerra Mundial muere del 10 al 20 % de la población total de Polonia y de Yugoslavia, del 4 al 6 % de la población total de Alemania, Italia, Austria, Hungría, China y Japón, 1 % de la población de Francia y de Gran Bretaña. Se estima que mueren unos 24 millones de rusos y un millón y medio de yugoslavos. Una aterradora cifra muestra las características de la guerra en el Frente Oriental. De 5.700.000 prisioneros rusos, mueren 3.300.000 en los campos de prisioneros alemanes de hambre y de frío.

Se destruye un 25 % de toda la capacidad de producción de la Unión Soviética, 13 % de la de Alemania, 8 % de la de Italia, 7 % de la de Francia. En la URSS quedan destruidos 70 mil pueblos y 7 mil ciudades, Yugoslavia pierde el 50 % de su ganado, el 60 % de sus rutas, el 5 % de sus puentes, el 20 % de sus casas.

A todo esto, hay que agregarle el genocidio. La propia palabra genocidio se comienza a aplicar después de la guerra para definir el proceso que en los campos de concentración y de exterminio nazis llevó a la muerte a casi seis millones de judíos, junto a centenares de miles de gitanos y de otros grupos étnicos, religiosos y políticos.

Por supuesto, la guerra termina en Europa en mayo de 1945 cuando las tropas soviéticas toman Berlín, y en Asia en agosto de ese año, tras la explosión de las bombas atómicas norteamericanas sobre Hiroshima y Nagasaki. Desde las guerras napoleónicas - cuando tras la revolución francesa, los ejércitos de Napoleón salen a conquistar toda Europa - la mayor parte de la población masculina joven, en edad de combatir, es incorporada a las Fuerzas Armadas. Comparadas las guerras modernas con las de la antigüedad, se llega a proporciones muy altas de la población directamente involucradas en la actividad bélica.

Se estima que en la Segunda Guerra Mundial un 20 % de la población activa se enrola en los ejércitos. La novedad del siglo xx es que la guerra involucra a todos los ciudadanos. ¿Qué quiere decir que involucra a todos los ciudadanos? En un sentido quiere decir que una parte importante de la población es movilizada como soldados, y otra parte importante de la población es movilizada para el esfuerzo de guerra, la industria se orienta a la producción bélica, los estados orientan los recursos colectivos para sostener el esfuerzo bélico. En este tipo de guerras, la mayor parte de la población se dedica al esfuerzo bélico. En Alemania e Inglaterra, para quienes la guerra dura seis años, toda la población se encuentra trabajando en las fábricas de armamentos, o en el servicio de rescate, o en la producción de textiles para los ejércitos. Esto solo puede ocurrir en sociedades industriales, y en la Primera y la Segunda Guerra Mundial van a resultar vencedores aquellos que tengan más capacidad para producir armas, municiones, combustibles, transportes.

Pero también la población civil pasa a ser un objetivo directo de los ataques multiplicando las víctimas. Cuando Alemania invadió Bélgica en la primera Guerra Mundial mueren seis mil civiles, lo cual genera un escándalo universal. A partir de allí asistimos a un progresivo acostumbramiento a la barbarie de la destrucción masiva de vidas civiles. Las cifras de civiles muertos en la segunda Guerra fueron escalofriantes. Por ejemplo, para el caso de la Unión Soviética, se estima que del total de las víctimas, 18 millones fueron civiles. Los bombardeos de zonas industriales y los de las zonas

pobladas se hicieron la regla dejando centenares de miles de muertos en las ciudades inglesas, alemanas y de otros países, pese a que se demostró que lejos de desmoralizar a la población, los bombardeos reforzaron su determinación.

Los conflictos bélicos del siglo xx fueron guerras industriales que requirieron cantidades increíbles de materiales. Por ejemplo, en la batalla de Jena en 1806 –solo 135 años antes del estallido de la Segunda Guerra Mundial– Napoleón derrotó a los prusianos con 1500 disparos de artillería. En contraste Francia al final de la Primera Guerra producía 200 mil proyectiles de artillería diarios y solo en las últimas dos semanas de la batalla de Berlín se dispararon cuarenta mil toneladas de bombas. Esto exige, por lo tanto, un estado que planifique la economía. Cuando hay una economía de guerra, el estado planifica la economía, sea como propietario directo de las fábricas de armas u orientando toda la producción hacia el esfuerzo bélico. No hay guerra moderna que se pueda ganar de otra manera.

Esto tuvo en ocasiones resultados paradójicos. Gran Bretaña termina la guerra muy endeudada por los acuerdos de arriendo de armas con Estados Unidos y con los países que, como Argentina, la proveían de materias primas y alimentos Pero como parte de esta planificación de la economía la población inglesa va a ser más sana, va a tener menos mortalidad infantil y va a estar mejor alimentada que antes de la guerra, porque va a haber un tratamiento más igualitario en una sociedad que era sumamente desigual. Por supuesto que esto fue posible también porque la guerra no se libró en territorio británico, pero el estado de bienestar británico, el famoso plan Beverigde, también es un resultado de la guerra.

Este carácter de la guerra implicó cambios en la gestión de la producción, se produce de manera distinta porque todo se ve orientado por la gestión estatal hacia esta producción, y es curioso porque la gestión estatal británica y norteamericana de la guerra fueron más exitosas que la soviética y que la de la propia Alemania nazi que ya eran economías total o altamente estatizadas antes de la guerra.

Entonces podemos entender que el conflicto mundial es también un conflicto por acceder a nuevas tecnologías, por desarrollarlas más rápido que el enemigo, por robarle sus secretos tecnológicos por todos los medios posibles.

Podemos comparar al caso de Alan Turing y su equipo con el proyecto Manhattan, desarrollado a partir de la advertencia de unos científicos nucleares húngaros refugiados y del propio Albert Einstein, quienes le informan al presidente Roosevelt en agosto de 1939, poco antes de que estalle la guerra, que Alemania estaba desarrollando un proyecto de armas nucleares. Con esta carta se inicia el proyecto Manhattan, que es el nombre clave del proyecto nuclear de Estados Unidos que terminaría con las bombas so-

bre Hiroshima y Nagasaki. Otros países tuvieron iniciativas similares, como el proyecto Uranio en Alemania y el proyecto Borodino en la Unión Soviética. De hecho, en la batalla de Berlín, uno de los objetivos del Ejército Rojo fue apropiarse de las instalaciones donde se estaba llevando a cabo el desarrollo central del proyecto Uranio.

En el proyecto Manhattan participaron además de algunas personas célebres, como Einstein y Oppenheimer, unas 130.000 personas, lo que supone la existencia de todo un estado haciendo inversiones de capital y humanas orientando el conjunto de la economía de una manera decisiva para alcanzar este objetivo.

En el caso de la organización en que se inserta Turing fueron diez mil las personas involucradas, lo que implica un esfuerzo económico y organizativo de vasto alcance. Alan Turing estaba trabajando desde antes de su doctorado en temas similares, pero ya para septiembre de 1939 se había incorporado al servicio de criptografía de la inteligencia británica. Tenía 26 años en ese momento.

No voy a desarrollar los modos en que logra romper los códigos de Enigma, el sistema que los alemanes usaban para encriptar sus mensajes, pues de ello se ocupa el capítulo 5. Pero puedo brindar algunos datos para medir la efectividad del trabajo de Turing para desencriptar los códigos alemanes. En el año 1940, son hundidos 520 buques aliados que iban hacia Inglaterra, en 1941 pasan a ser 456, en 1942 son 1155 las embarcaciones que se hundieron, pero cuando se descifra al año siguiente el código alemán, la cifra se redujo nuevamente a 452 hundimientos. Tanto en el desarrollo de la guerra en el Mediterráneo y en el norte de África como en el frente ruso la información aportada por el Servicio Británico de Descifrado fue fundamental. Se sabe que unos días antes del desembarco de Normandía, los aliados sabían que los alemanes creían que el desembarco iba a ser en el paso de Calais y no en Normandía, a través del descifrado de los textos encriptados por las máquinas Enigma. Por eso se estima que la contribución de Turing a la guerra fue fundamental y que logró evitar millones de muertes y reorientar decisivamente el curso de la guerra.

La guerra mundial transformó al mundo, fue el comienzo del mundo bipolar de la guerra fría, y también porque fue la cuna de algunos de los grandes saltos tecnológicos que llegan desde la energía nuclear hasta los orígenes de la computación. Por eso siempre hay que pensar en la cultura y la tecnología como resultados de procesos humanos maravillosos, de cooperación, de conocimiento, de transmisión de ideas y como resultados también de procesos de barbarie inauditos a los que lamentablemente nos venimos acostumbrando en los últimos 100 años.

El destino personal de Alan Turing no fue acompañado por la gloria que sí obtuvieron los generales victoriosos. Todo su trabajo permaneció en se-

creto hasta la década de 1970, y en 1952 fue procesado por homosexualidad, condición que era penada por la ley en Inglaterra hasta 1967. Turing murió en la cárcel, dos años después de su condena, en circunstancias nunca del todo aclaradas. Recién en 2013 se exoneró a Turing y se anularon los cargos en su contra.

Breve biografía del autor

Daniel Lvovich es profesor asociado del Área de Historia del Instituto del Desarrollo Humano de la UNGS e investigador principal del CONICET. Realizó sus estudios de grado en la Universidad Nacional del Litoral y su doctorado en la Universidad Nacional de La Plata. Se dedica a la investigación de la historia política y social argentina. En particular ha desarrollado investigaciones sobre la historia de las derechas, las políticas sociales, y los trabajadores.

Referencias

[1] E. Hobsbawm. *Historia del Siglo xx, 1914–1991*. Barcelona: Crítica, 1995.

[2] E. Traverso. *A sangre y fuego: de la guerra civil europea 1914–1945*. Buenos Aires: Prometeo, 2009.

Capítulo 5
La criptografía mecánica de la Segunda Guerra

Ariel Waissbein *

1. El trabajo de Turing como criptógrafo

Turing se hizo famoso por haber "atacado" la máquina Enigma usada por los alemanes durante la segunda guerra mundial, pero su trabajo como criptógrafo no se circunscribió a esta única hazaña. Además de descifrar la Enigma, Turing hizo dos grandes contribuciones más: el descifrado de la máquina Lorenz y el desarrollo de la máquina Delilah. Los alemanes venían usando sistemas automáticos de cifrado desde antes de la primera guerra y contaban con numerosas variantes de máquinas para esta tarea, entre ellas, la máquina Lorenz, diferente de Enigma, cuya existencia fue descubierta por los aliados hacia el final de la guerra. Por otro lado, cuando Turing viajó a Estados Unidos, hacia el final de la segunda guerra, se enteró que Churchill y Roosvelt usaban una máquina que les encriptaba las conversaciones telefónicas. Su versión mejorada de esa máquina, Delilah, hacía que se oyera mejor y aceleraba el proceso. Pero aquí nos queremos ocupar de Enigma, así que dejamos a Lorenz y a Delilah.

2. El funcionamiento de Enigma

Antes de empezar la descripción de Enigma, hay que imaginar el mundo en esa época. Situémosnos en 1910. La comunicación era muy básica; se usaban telégrafos, algunas máquinas de comunicación inalámbrica o radio para mandar mensajes. No había correo electrónico, ni Telegram, ni ninguno de estos medios digitales con los que contamos actualmente. Esos medios que se usaban no tenían capacidad para digitalizar la voz, pensemos en una radio. Delilah digitalizaba voz, pero es bastante posterior. Para transmitir

* N. de la R.: este artículo está basado en la charla del autor durante las jornadas "Ecos de la figura y de la obra de Alan Turing" (UNGS, 2017). Desgrabación: Paula Albarracín.

un mensaje, este se escribía en papel, se cifraba de alguna manera, el emisor lo transmitía en código Morse por algún medio de comunicación, el receptor lo transcribía a mano, lo descifraba y, finalmente, se lo mostraba al destinatario.

Estos procesos eran lentos y no sacaban provecho de la nueva tecnología del recién inciado siglo xx. La empresa alemana Scherbius y Ritter, fundada en 1918, se concentró en el proyecto de sustituir los sistemas de encriptado con "lápiz y papel" utilizados durante la primera guerra mundial por alguna máquina que sacara provecho de los nuevos desarrollos científicos y tecnológicos. Como resultado de este trabajo, Scherbius diseñó la máquina Enigma.

Figura 5.1. Máquina Enigma, fotografía tomada por Thomas Guest [*].

El aspecto exterior de la Enigma es como el de una máquina de escribir antigua un poco aparatosa (ver figura 5.1). Desde afuera, se puede apreciar que además del teclado en el que se presionan las letras para escribir el texto, tiene un tablero con las veintiséis letras del alfabeto (los alemanes no tienen "ñ"), cada letra con una lamparita pequeña, un clavijero y unas ruedas dentadas que son los rotores (tres o más, depende de la versión). Cada rotor tiene al lado un pequeño indicador que muestra su posición, como las perillas de la cocina, y tiene 26 posiciones posibles, indicadas con una letra.

Como en los sistemas actuales, para comenzar hay que introducir una clave en la máquina. Supongamos que tenemos una máquina de tres rotores. La clave tendrá entonces tres letras. El operador, la persona que usa la máquina, posiciona cada rotor según la letra de su clave; toma el texto "llano"[†] que tiene anotado en un papel y teclea en el teclado la primera letra; en ese

[*] Foto disponible en `https://wordpress.org/openverse/image/dd870ee8-cc70-434a-8831-f06294dc3751/` con licencia de Creative Commons BY 2.0.

[†] Se llama texto llano a un texto inteligible por una persona que conoce el idioma en el que está escrito.

momento, una corriente eléctrica pasa por el cableado interno e ilumina la letra correspondiente del texto encriptado en el tablero. El operador anota la letra que salió y continúa con la segunda letra de su texto. Toda la operatoria se hacía manualmente. Una vez encriptado el mensaje, se lo pasaba por telégrafo al destinatario del mensaje. Ambos, destinatario y emisor conocían la clave secreta (cómo se habían puesto de acuerdo es otro tema). El destinatario acomodaba los rotores según la clave, tecleaba el texto encriptado y obtenía el texto llano.

Ahora tratemos de entender el funcionamiento interno. Esta explicación se basa en el excelente libro de Simon Singh, *Los códigos secretos* [4]. La versión más simple de la máquina tiene tres rotores. Cada rotor es un cilindro chato con dos tapas de metal y una goma interna llena de cables, como una especie de alfajor. No se puede abrir. Cada rotor tiene tantos cables como letras del alfabeto y cada cable va desde una tapa a la otra uniendo una entrada y una salida. En la figura 5.2 se muestra un ejemplo de un rotor con 6 cables, para que sea más claro.

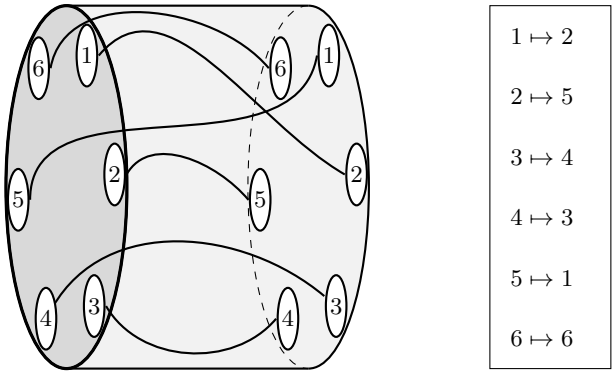


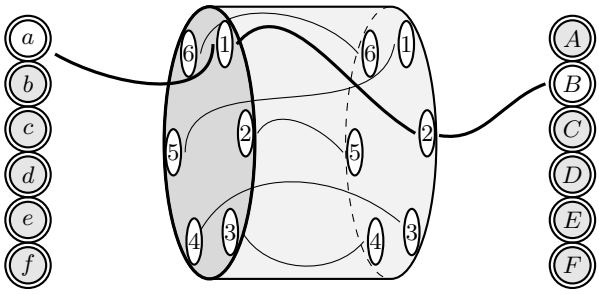
Figura 5.2. Esquema ilustrativo de un rotor con el cableado interno.[*]

Vamos a empezar explicando como funcionaría una Enigma de un rotor para un alfabeto de 6 letras, de la "a" a la "f". Al presionar una tecla del teclado, ocurren dos acciones: en primer lugar, una corriente eléctrica pasa por los cables uniendo la tecla presionada con uno de los cables del rotor y a este con una luz del tablero (ver figura 5.3); en segundo lugar, el rotor gira una posición modificando la forma en que se vinculan las teclas de entrada con las salidas en el tablero. En las figuras 5.3 y 5.4 se busca representar esquemáticamente la situación en la posición inicial y luego de un giro usando el rotor de la figura 5.2.

Como se muestra en la figura 5.3, la letra "a" del teclado se conecta con la entrada 1 del rotor cuando este se halla en la posición inicial, la letra "b" se conecta con la entrada 2, la letra "c" con la 3 y así siguiendo. De la misma

[*] Ilustración: Eda Cesaratto.

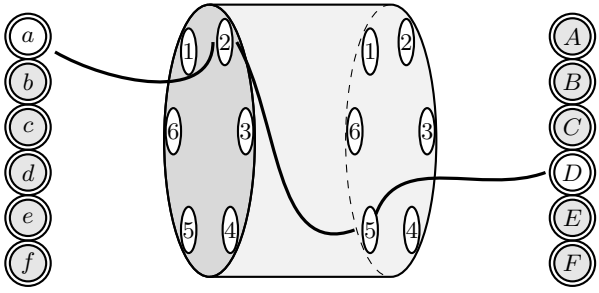
forma, la letra "A" del tablero de salida se conecta con la salida 1 del rotor, la "B" con la 2, etc. En la figura 5.3, se muestra que la letra "B" se conecta con la salida 2. De esta forma, con el rotor de la figura 5.2, resulta que el camino que sigue el pulso eléctrico es "a"$\to 1 \to 2 \to$"B" (porque en nuestro ejemplo de rotor, el 1 está conectado con el 2). El rotor gira. El cable que va del teclado al rotor *no se mueve* por lo que resulta que la letra "a" queda conectada con la entrada 2 del rotor. Lo mismo pasa con los cables del tablero. Estos no se mueven. Entonces la salida 5 del rotor en la posición 2 se conecta con la D, pues en la posición inicial del rotor estaba la salida 4. Así, después de un giro, resulta que si pulsamos la tecla "a" se produce el siguiente recorrido "a"$\to 2 \to 5 \to$"D", ya que el 2 está conectado al 5 en nuestro rotor. Resulta que el texto plano "aa" queda cifrado como "BD".



Se presiona la letra a del teclado

Se enciende la tecla B del tablero

Figura 5.3. El sistema en la posición inicial. La letra "a" se cifra como "B".



Se presiona la letra a del teclado.

Se enciende la tecla D del tablero.

Figura 5.4. El sistema en la posición 2. Recordemos que los cables que van del teclado (izquierda) al rotor y del rotor al tablero (derecha) no se mueven respecto de la posición inicial. Solamente gira el rotor. La letra "a" queda cifrada como "D"*.

*Ilustraciones: Eda Cesaratto.

En una máquina de un rotor y para una alfabeto de 6 letras, hay 6 diferentes opciones de cifrar un alfabeto de 6 letras. En una máquina de 2 rotores, una vez que el primer rotor completó una vuelta, comienza a girar el segundo rotor. De esta forma, en una máquina de dos rotores y para un alfabeto de 6 letras, hay 6^2 formas diferentes de cifrar el alfabeto, para tres rotores hay 6^3.

Las máquinas que usaban los alemanes tenían tres rotores para un alfabeto de 26 letras. Esto daba en total $26^3 = 17576$ posibilidades. Posteriormente, se le agregó un rotor más y un deflector que duplicaba las posibilidades de cifrado.

Al momento de escribir estas notas, además de una excelente descripción de la Enigma, el sitio `https://piotte13.github.io/enigma-cipher/` propone un emulador. Un detalle importante es que el emulador solo funciona con los navegadores Mozilla Firefox y Opera.

3. Los avances polacos en el descifrado de Enigma

Entre 1929 y 1932, bastante antes de que Turing dejara la universidad y fuera enrolado en el servicio de inteligencia inglés, los polacos, enemigos históricos de los alemanes, observaron que los métodos de desencriptado de principios del siglo xx ya no funcionaban pues los métodos automatizados de cifrado, como Enigma, habían entrado en escena. Tanto los aspectos históricos como los técnicos pueden ampliarse recurriendo al libro *Los códigos secretos* de S. Singh [4], del cual está basada buena parte de esta explicación.

En 1932 el gobierno polaco enroló a tres matemáticos, Rejewski, Zygalski y Rozycki, para trabajar sobre las primeras versiones de Enigma. Ellos disponían de una máquina porque Scherbius, su creador, hizo una versión comercial, que era ligeramente distinta de la militar, pero que se podía conseguir más fácilmente.

El primer paso de Rejewski fue entender cómo funcionaba la Enigma militar. No sabía cuál era el cableado interno de los rotores porque habían cambiado buena parte de la configuración de la máquina comercial de la que él disponía. Rejewski tenía algunas claves y algunas cribas (al par formado por un mensaje de texto común, sin encriptar, y el correspondiente texto encriptado se lo suele llamar criba). Sobre la base de este material, concibió un método que le permitió descubrir cómo estaban asociadas las letras del mensaje cifrado con las del mensaje sin cifrar. En otras palabras, a partir de una criba podía determinar que la configuración del cableado de rotores de esa máquina mandaba tal letra a tal otra, por ejemplo, que la "a" estaba asociada con la "g", la "f" con la "p", etc. Usando esta información y el hecho de que el segundo rotor giraba después de 26 giros del primer rotor, y el tercero después del 26 giros del segundo, es decir, después de 26×26 del primero, Rejewski lograba establecer cómo estaba cableado el primer rotor. En ese momento, los alemanes cambiaban cada tanto el orden de los rotores, maniobra que le dio suficiente información para establecer el cableado del segundo rotor; y sabiendo el cableado de estos dos, apelaba al ingenio para

calcular el tercero. Así que, aun cuando los alemanes cambiaban el orden cada 3 meses, y un poco más adelante, cada un mes, él podía establecer el cableado.

Estos desarrollos les permitieron duplicar las máquinas alemanas y de hecho mandaron a construir una máquina igual a las que funcionaban, aproximadamente, en 1934. Aun con la máquina replicada no es obvio cómo desencriptar los mensajes interceptados. Para ello hay que saber la clave con la que se cifraron y diferentes operadores usaban claves distintas. Para lograr este objetivo los matemáticos polacos aprovecharon que la clave se transmitía dos veces. Rejewski se las ingenió para encontrar un sistema que le permitía descartar claves de forma muy eficiente. Ese sistema funcionaba a partir de un catálogo con las $26^3 = 17576$ posibilidades de claves. Para hacerlo, se construía una máquina que era una suerte de enigma de seis letras, y encriptando de a seis comenzaban a construir un catálogo de combinaciones posibles o imposibles con el cual armaban unas hojas perforadas con agujeros pequeños, una por rotor. El número 6 viene de la cantidad de formas que se puede combinar los 3 rotores, que es en efecto $3! = 6$. Estas hojas, conocidas como hojas de Zygalski, eran puestas a trasluz. La alineación de algún agujerito en las tres hojas daba información de la posición inicial y la clave.

Los polacos construyen también una máquina que es conocida como bomba kryptologiczna que, según Wikipedia, es llamada bomba por su postre favorito, el bombón suizo. Esa bomba era una máquina que, usando fuerza bruta[*], probaba todas las claves, pero para que fuera eficiente (diera una respuesta en un tiempo razonable) era necesario darle como entrada la información que se obtenía de las hojas de Zygalski.

Como vemos, estos matemáticos y criptólogos polacos habían avanzado muchísimo en la comprensión de la máquina Enigma.

4. Inglaterra toma la posta

Alrededor de 1938 los polacos se dieron cuenta de que los alemanes estaban cambiando la forma de usar Enigma y que ya no podían descifrar los mensajes como antes y que, además, el caudal de mensajes había aumentado significativamente. Estos datos les dio la pauta de que algo pasaba. Es simplemente estrategia, inteligencia de señales. Sabían que la guerra era inminente. Entonces decidieron aliarse a los ingleses y franceses. En una reunión en Varsovia, les informaron todo lo que sabían, les dieron unas copias de las Enigmas que habían construido y ellos mismos destruyeron toda evidencia de este trabajo en Polonia. Un mes después Polonia era invadida y los criptólogos fueron enviados a Francia.

Estando todavía los polacos trabajando en Francia, los alemanes agregaron los rotores 4 y 5. El procedimiento que ellos usaban dependía de la cantidad de formas de ubicar los rotores. Con 3 rotores, hay 6 posibilidades. Con

[*]Se dice que un problema se resuelve por "fuerza bruta" cuando la solución es encontrada realizando una inspección sistemática de casos.

5 rotores hay 120, entonces los polacos tenían que hacer 120 de estos catálogos de combinación de claves, y 120 de esas hojas de Zygalski. No podían porque estaban en Francia y sin recursos. Así que acordaron con los ingleses el intercambio de las dos máquinas enigmas que habían construido por la confección de las hojas de Zygalski.

Al comienzo de la guerra los alemanes cambiaban los protocolos todo el tiempo porque se daban cuenta de que los estaban desencriptando. Para comprobar que así era, hacían pruebas, decían por la radio: "este es un mensaje secreto, tal submarino tiene una tal ruta" y sacrificaban el submarino enviándolo por esa ruta. Si al día siguiente los ingleses lo interceptaban, se podía conjeturar que, con alta probabilidad, los británicos habían leído el mensaje. Los ingleses también hacían contrainteligencia y dejaban que les hundan barcos a propósito para no dar señales de que sabían leer mensajes. Sin embargo, iban a dejar de hundir un barco pero no 10. Con un uso simple de la estadística, si les mandaban 10 barcos y 7 eran atacados es que les estaban leyendo los mensajes. Además usaban agentes dobles y muchas otras técnicas. Alrededor de 1940, los alemanes dejan de usar este procedimiento del indicador y el ataque de los polacos dejó de servir, lo que hizo que los ingleses no pudieran descifrar más mensajes.

Uno o dos días después de que Inglaterra le declaró la guerra a Alemania, comenzaron a trabajar Turing, Welsman y algunos más en un lugar de Inglaterra conocido como Bletchley Park. Allí se podían encontrar ajedrecistas, especialistas en sopas de letras, etc., pues se esperaba que fueran eficientes para observar patrones de coincidencia, además de lidiar con sistemas eléctricos y mecánicos. La electrónica no existía aún. En ese momento, Turing conduce el diseño y el armado del procedimiento, la máquina y la bomba inglesa para desencriptar mensajes.

Para trabajar con la bomba necesitaban una criba que es un mensaje de texto común, sin encriptar, y el correspondiente texto encriptado. Si habían interceptado 20 mensajes de un barco, sabían que alguno era el reporte meteorológico por la hora de emisión (todos los barcos debían emitir el reporte a las 6 a. m.), conocían el barco que emitía los mensajes y tenían noción de su ubicación. Más aún, muchas veces identificaban a los operadores que generaban los mensajes y hasta les ponían apodos, ya que no sabían sus verdaderos nombres. Esta información recolectada todos los días les permitía generar estadísticas sobre el uso de ciertas palabras o giros idiomáticos de cada operador de cada barco que podían interceptar (por ejemplo, hace calor o está caluroso). Con todos estos trucos, se armaban una criba.

Una vez que tenían la criba, el problema era decidir cómo estaban asociadas las letras del texto encriptado y las del texto plano. Para ello deslizaban ambos mensajes uno encima del otro. Si cuando los alineaban, una letra se correspondía a sí misma descartaban esa posibilidad pues ninguna letra puede encriptarse consigo misma. De esta forma descartaban posibilidades. Con esta información se construía un "menú". Este menú tenía dos columnas. En la primera, se ponían las letras y en la segunda, con qué letra estaban encriptadas.

En definitiva, la máquina que hizo fabricar Turing, la bomba, aceleraba este proceso probando claves, una después de la otra muy rápido (no hay que confundirla con una "máquina de Turing" que es una noción de carácter teórico como se explica en los capítulos 1 y 2). Cada vez que una de estas posiciones, clave, etc. no funcionaba, la eliminaba y cada vez que daba vuelta a todo el ciclo, alguien anotaba que pasó el test. Así descartaban la mayoría de los posibles órdenes de rotores, de clave, etc. Las pocas posibilidades que quedaban eran probadas por las "redes", que era un grupo de mujeres que trabajaba en Bletchey Park con unas máquinas enigma modificadas para que pudieran imprimir. Para probar las claves que no habían sido descartadas por la bomba, ellas desencriptaban el mensaje cifrado de la criba con esas claves y decidían si las coincidencias con el sin cifrar eran suficientes como para suponer que era la correcta. No buscaban encontrar el cableado exacto porque hacerlo es muy difícil, pero intentaban encontrar una solución aproximada. De hecho, estas mujeres eran contratadas por tener "ojo de ajedrecista" y, por lo tanto, eran capaces de determinar rápidamente inversiones de letras, ciclos, etc. Por ejemplo, si en lugar de WETTER, que significa clima en alemán, se encontraban con "TEWWER" se esperaba que se dieran cuenta de que se había invertido la T y la W.

5. La criptografía en la actualidad

Después de leer sobre el esfuerzo de los polacos, y de los ingleses de Bletchley Park es natural preguntarse sobre cómo funciona y para qué se usa la criptografía hoy en día, que contamos con medios digitales para imitar a Turing con sus cribas. Esta sección intenta dar algunas ideas sobre estas cuestiones.

5.1. Claves

En la actualidad usamos criptografía cada vez que escribimos una clave para entrar en nuestra cuenta bancaria, nuestra cuenta de correo, nuestro teléfono, etc. Podemos empezar por una pregunta muy práctica: ¿cómo elegir nuestras claves? La elección usual es utilizar alguna información personal como, por ejemplo, nombres y fechas de cumpleaños, a los que se les cambia alguna minúscula por una mayúscula, se le reemplaza el cero por la O, etc. Finalmente, estas claves son difíciles de recordar pues son muy raras y no tienen significado para el usuario.

Para medir la seguridad de una clave se suele usar el concepto de "entropía" en el sentido que esta palabra tiene en la teoría de la información [*]. Se estima que la entropía de las claves mencionadas en el párrafo anterior es del orden de 28 bits y que pueden ser descubiertas en 3 días. Por el contrario, se sabe que una clave creada a partir de la elección al azar de 4 palabras

[*] N. de la R.: la entropía, en teoría de la información, se entiende como una medida de la cantidad de información que produce cada "símbolo de un lenguaje" y es uno de los conceptos básicos introducidos por C. Shannon en el artículo [3] que dio origen a este área de la informática.

del diccionario, por ejemplo, "correcto, caballo, batería y sello", tiene una entropía del orden de 44 bits. En este caso, serían necesarios más de 30 años para descubrir la combinación correcta de palabras. Así que una forma de crear claves seguras es elegir palabras del diccionario al azar.

5.2. Ataques

Aunque parezca extraño, muchos ataques para adivinar nuestras claves se hacen usando "fuerza bruta". Un ataque usual por fuerza bruta para descifrar passwords hace uso de cribas. Las páginas webs usan muchos trucos para evitar esos ataques. A veces hacen que cada prueba tarde 2 segundos y si un usuario prueba varias veces por el mismo IP, lo bloquea. Hay algunas aplicaciones con passwords que son atacables con cribas y hay muchos softwares cuyo objetivo es realizar tales ataques. Un ejemplo de fuerza bruta es el uso de la criba de Eratóstenes para atacar algunos esquemas criptográficos que se sostienen sobre la supuesta imposibilidad de factorizar números enteros. Es muy común encontrarse con estos tipos de ataques y es difícil evitarlos.

Los ataques que se llaman Side Channels son súper interesantes. Cuando la computadora hace operaciones para encriptar está regalando información. Hace unos años nos dimos cuenta de que un programa de uso corriente desencripta midiendo el tiempo en el que uno tarda en entrar los caracteres de la clave. Por ejemplo, comparemos el tiempo que se tarda en teclear la W después del 4 en una clave como 1234W, y el tiempo que se tarda en teclear L en la clave 1234L. Obviamente, tardaremos mucho más en teclear la L pues está más lejos del 4. Este programa mide esta diferencia de tiempos, hace estadísticas y aunque la clave esté encriptada, es posible deducir partes de ella o, incluso, deducir la clave entera. Muchos ataques usan información estadística sobre cómo el usuario teclea la clave o las operaciones que hace la computadora para encriptar. Así, el cifrado de la clave no garantiza seguridad. También se pueden hacer ataques acústicos: ciertos ruidos dan idea acerca del lugar del teclado en el que están las manos cuando se introduce la clave. Otro ataque es medir el consumo de energía, porque el disco rígido gasta diferentes cantidades de energía para diferentes operaciones matemáticas. La investigación en este tema es muy activa por los intereses económicos y políticos que están detrás. De la seguridad informática dependen las elecciones, los bitcoins, y las sumas de dinero extraordinarias que están flotando en internet. El interés por vulnerar los sistemas de protección de la información es muy grande.

Un tipo de ataque usual es el llamado "ataque de diseño". Por ejemplo, quiero proteger mi casa y refuerzo la puerta que da a la calle porque pienso que es el único lugar vulnerable. Por el contrario, resulta que los ladrones saltan por los fondos y entran por la puerta trasera o la puerta trampa, la "trap door" en inglés. Se han dado casos de este tipo de problemas de diseño en la criptografía moderna, por ejemplo, el protocolo Data Encryption Standard, usado por el gobierno de los Estados Unidos entre 1976 y 2002, que fue acusado por la National Security Agency de tener "trap doors" (veáse [2]).

Para atacar los protocolos de "clave pública" también se usan ataques a los canales, "Side Channels", y "ataques de diseño". Este tipo de protocolos están basados en la clase de esquemas criptográficos de criptografía asimétrica concebida por Whitfield Diffie y Martin Hellman. Ellos demostraron, exhibiendo un esquema en particular, que dos personas pueden compartir un secreto transmitiéndose información por canales inseguros sin necesidad de encontrarse (veáse [1]). Este trabajo abrió las puertas a la introducción de la criptografía en la vida civil, en la vida cotidiana de la gente común. El hecho de que las personas puedan intercambiar claves en forma segura sin necesidad de encontrarse físicamente es lo que permite el uso de internet para transacciones comerciales, trámites bancarios, transmisión de información confidencial, etc. Una explicación detallada del trabajo de Diffie y Hellman y sus fundamentos matemáticos se encuentra en el capítulo 6.

Un protocolo de clave pública permite el intercambio seguro de una clave K entre dos personas, "A" y "B", por un canal inseguro que es monitoreado por un atacante "M". Para entender el principio de este funcionamiento imaginemos que "A" tiene un candado público del cual guarda celosamente su llave y quiere compartir con "B" una llave K distinta de la de su candado. Para intercambiar esa llave compartida K con "B", "A" le manda a "B" una caja con su candado, "B" mete una llave K, de la cual tiene una copia que se guarda, cierra la caja con el candado de "A" y se la reenvía a "A" que es la única persona que puede abrir esa caja. Ahora "A" y "B" pueden compartir mensajes protegidos por la llave K de extremo a extremo, lo que significa que ni "M" ni terceros pueden leerlos. En términos informáticos, este sistema tiene como principio que cada persona tenga dos claves una pública (usuario) y otra privada (palabra clave).

Por el momento, no se conocen formas sistemáticas de ataque a estos protocolos de clave pública. Sin embargo, hay ataques que aprovechan debilidades de los canales de comunicación, como por ejemplo, el atacante "M" puede meterse en el canal de comunicación y hacerle creer a "A" que es "B" y viceversa. Entonces "A" y "B" le mandan su clave K a "M" y este puede leer todos los mensajes que se mandan entre ellos. "M" se aprovecha de un mal diseño del protocolo, no del esquema criptográfico, puesto que "A" y "B" supusieron que se comunicaban entre ellos. Hay formas de determinar la indentidad del emisor de un mensaje para evitar este tipo de ataques.

Un ataque conocido al protocolo SSH V1 se aprovechaba de una debilidad de su sistema de generación de claves. En esta primera versión, el usuario negociaba con el protocolo la elección de la clave: métodos para generarla, su longitud en bits, etc. Como la versión 1 no contaba con un test de calidad de las claves, el programa permitía usar claves de 1 o 2 bits, con lo cual una elección poisible era el número 1. Como se puede ver en el capítulo 6, una de las operaciones matemáticas que usan los protocolos para encriptar es la potenciación discreta con una de las claves como exponente. Pero elevar a la 1 no modifica el número original y el mensaje original, en realidad, no resultaba cifrado. En el trabajo como diseñador y programador, uno va notando estos errores y vulnerabilidades de los sistemas con el uso.

Otra fuente importante de problemas son los "bugs" de implementación. Llaman a una mamá desde el colegio de su hijo y le preguntan: ¿su hijo se llama "Roberto´); drop table students;"? Obviamente no es la maestra la que cargó ese dato. La pregunta es por qué pasa esto. Son simplemente bugs en la programación de la página web en donde el colegio carga los datos de los estudiantes. La clave de datos está mal armada. Ejemplos de este estilo de errores con programas que implementan protocolos criptográficos son mas difíciles de explicar, pero abundan. Por eso, las mejores bibliotecas (o librerías) de programas son las públicas pues son usadas por mucha gente. Es muy desaconsejable usar programas propios que solamente fueron verificados por uno mismo porque nadie los corrige. De todas formas hay que desconfiar e investigar antes de usar, los errores abundan y los atacantes aprovechan.

Breve biografía del autor

Ariel Waissbein se graduó como licenciado en Ciencias Matemáticas y doctor en Matemática en Facultad de Ciencias Exactas y Naturales de la UBA de la cual ha sido docente e investigador. Actualmente, se desempeña en el ámbito profesional técnico como consultor experto en proyectos de ciencia aplicada al tratamiento de datos, seguridad computacional y criptografía (aka, crypto currencies). Es co-fundador de la empresa BetterAgro de servicios para la producción agropecuaria.

Referencias

[1] W. Diffie y M. E. Hellman. "New Directions in Cryptography". En: *IEEE Transactions on Information Theory* 22 (1976), págs. 644-654.

[2] J. von zur Gathen. *CryptoSchool*. 1st. Springer Publishing Company, Incorporated, 2015. isbn: 3662484234, 9783662484234.

[3] C. E. Shannon. "A Mathematical Theory of Communication". En: *Bell System Technical Journal* 27.3 (1948), págs. 379-423.

[4] S. Singh. *Los códigos secretos*. Debate, 2000.

Capítulo 6
Revoluciones en la criptografía post Turing

Nicolás Sirolli

¿Qué cambió en la criptografía después de que Turing *rompió* la máquina Enigma?

Gracias al desarrollo de la tecnología, en los últimos tiempos el intercambio de información se ha vuelto considerablemente más rápido y fácil, y también más necesario en nuestra vida cotidiana. Eso supuso nuevos desafíos para quienes necesitaran comunicarse de manera segura.

La idea que revolucionó la criptografía consistió en mostrar que dos personas pueden usar una clave pública para comunicarse de manera segura, sin necesidad de encontrarse previamente en privado. Esta idea aparece por primera vez en el protocolo de Diffie-Hellman, publicado en 1976, gracias al cual los autores recibieron recientemente el premio Turing.

En estas notas ilustraremos este gran salto con ejemplos sencillos de protocolos de criptografía basados en claves privadas y públicas.

1. Criptografía de clave privada

Los protocolos clásicos de la criptografía están basados en la utilización de una clave privada, que deben compartir las dos partes que desean comunicarse; los llamaremos Alicia y Beto. A grandes rasgos, el esquema consiste en:

- Un conjunto $\{k\}$ de claves.
- Para cada clave k, un par de funciones e_k (de encriptado) y d_k (de desencriptado), que satisfacen

$$d_k(e_k(m)) = m \qquad \text{para todo mensaje } m. \tag{1.1}$$

Es deseable que evaluar las funciones d_k y e_k sea sencillo y rápido.

Para comunicarse Alicia y Beto deben primero acordar, en privado, una clave k. Una vez hecho esto, el protocolo a seguir es el siguiente:

- Alicia encripta el mensaje m, calculando $e = e_k(m)$.
- Alicia envía a Beto el mensaje encriptado e.
- Beto desencripta el mensaje, calculando $d_k(e)$.

Gracias a (1.1) se tiene que $d_k(e) = m$. Así, Beto obtiene el mensaje m.

Asumiremos el peor escenario posible: que Miranda, quien está interesada en conocer el mensaje m, pudo interceptar la comunicación y conoce el mensaje encriptado e. Más aún, asumimos que Miranda sabe cómo utilizar la función de desencriptado d_k si conoce la clave k. Por eso, el protocolo será seguro solamente si es difícil obtener m o k a partir de $e = e_k(m)$.

A continuación daremos un ejemplo clásico de un protocolo de criptografía de clave privada.

1.1. Cifrado de Vigènere

Este cifrado, que data del siglo xvi, se destaca por su simplicidad, y por haber sido considerado *irrompible* por muchísimos años. Para formularlo, aunque no es estrictamente necesario, utilizaremos aritmética modular sencilla.

La clave que comparten secretamente Alicia y Beto es una palabra. Por ejemplo, la palabra `clave`.

Tanto para encriptar como para desencriptar, numeraremos las letras del abecedario, haciendo corresponder la `A` con el 0, la `B` con el 1, y así siguiendo.

La función de encriptado consiste en, una vez hecha esta identificación, ir sumando, letra por letra, las letras del mensaje con las letras de la clave; volviendo a repetir la clave una vez que esta se termina en caso de ser necesario.

Por ejemplo, supongamos que el mensaje que quiere enviar Alicia es "Ganó Boca". Para simplificar, omitiremos espacios y caracteres especiales, y consideraremos $m = \texttt{GANOBOCA}$. Para encriptar, Alicia realiza la suma:

$$\begin{array}{r} \texttt{GANOBOCA} \\ \texttt{CLAVECLA} \\ \hline \texttt{ILNJFQNA} \end{array}$$

Que en términos de la correspondencia entre letras y números es la suma:

$$\begin{array}{r} 6\ \ 0\ 13\ 14\ 1\ 14\ \ 2\ 0 \\ 2\ 11\ \ 0\ 21\ 4\ \ 2\ 11\ 0 \\ \hline 8\ 11\ 13\ \ 9\ 5\ 16\ 13\ 0 \end{array}$$

Notemos que no está sumando enteros, sino que está realizando la suma módulo 26, que es la cantidad de letras del abecedario. Por ejemplo, al sumar `O` y `V`, es decir al sumar 14 y 21, el resultado es `J`, es decir 9, que es congruente a $14 + 21 = 35$ módulo 26.

Para desencriptar, Beto tiene que simplemente restar la palabra CLAVE al mensaje encriptado $e =$ ILNJFQNA que recibió; teniendo en cuenta también que las restas se deben realizar módulo 26.

$$\begin{array}{r} \texttt{ILNJFQNA} \\ \texttt{CLAVECLA} \\ \hline \texttt{GANOBOCA} \end{array}$$

Que en términos de la correspondencia entre letras y números es la resta:

$$\begin{array}{rrrrrrrrr} 8 & 11 & 13 & 9 & 5 & 16 & 13 & 0 \\ 2 & 11 & 0 & 21 & 4 & 2 & 11 & 0 \\ \hline 6 & 0 & 13 & 14 & 1 & 14 & 2 & 0 \end{array}$$

La ventaja de este método se basa en que una misma letra del mensaje, por ejemplo la O, aparece cifrada de dos maneras distintas: como J y como Q. Gracias a eso, Miranda no puede hacer un análisis de frecuencia de la aparición de las letras en el mensaje cifrado para intentar averiguar la clave y/o el mensaje.

Pese a eso, en el siglo xix, mucho tiempo después de su aparición, este cifrado fue roto por Kasiski (véase [1], sección 4.2).

1.2. Enigma y después

La máquina Enigma, utilizada por los alemanes durante la segunda guerra mundial, utilizaba un conjunto de claves más complejas que simples palabras, dadas por configuraciones de rotores y *plugboards*, y utilizaba para cifrar y descifrar los mensajes un dispositivo notablemente más complicado que la aritmética módulo 26.

Más allá de que el cifrado también haya sido vulnerado, esta vez más rápidamente gracias a la genialidad de Turing y su equipo, este protocolo adolece de algo en común con el de Vigenère: en ambos Alicia y Beto necesitan encontrarse secretamente para acordar una clave con la que comunicarse de manera segura después. Esto, en medio de una guerra, es desde ya nada deseable, pero no inviable.

En la actualidad, en nuestra vida cotidiana nos encontramos constantemente comunicándonos por canales inseguros, y necesitamos poder encriptar nuestra información sin necesidad de encontrarnos previamente con la otra parte, lo cual sí nos resulta inviable. Este problema es resuelto gracias a las ideas que desarrollamos a continuación.

2. Criptografía de clave pública

Alicia y Beto desean compartir un secreto sin encontrarse, utilizando un canal de comunicación que es *a priori* inseguro: Miranda podría ver toda la información que se transmiten. A primera vista su objetivo parece difícil de cumplir, pero Diffie y Hellman mostraron que es posible. A grandes rasgos, el esquema consiste en:

- Un conjunto $\{g\}$ de claves *públicas*.
- Un conjunto $\{a, b, \dots\}$ de claves *privadas*.
- Para cada par de claves privadas a, b, una par de funciones e_a, e_b (de encriptado) que satisfacen

$$e_b(e_a(g)) = e_a(e_b(g)) \qquad \text{para toda clave pública } g. \tag{2.2}$$

Para comunicarse Alicia y Beto deben primero acordar una clave pública g. Una vez hecho esto, el protocolo a seguir es el siguiente:

- Alicia elige una clave privada a, calcula $A = e_a(g)$ y se lo envía a Beto.
- Beto elige una clave privada b, calcula $B = e_b(g)$ y se lo envía a Alicia.
- Alicia calcula, en privado, $e_a(B)$.
- Beto calcula, en privado, $e_b(A)$.

Gracias a (2.2) se tiene que $e_a(B) = e_b(A)$. De esta manera, Alicia y Beto comparten este secreto, que llamaremos C.

Como antes, asumimos que Miranda pudo interceptar la comunicación y que conoce los mensajes encriptados A y B. Además conoce g, que es público, y sabe cómo utilizar las funciones de encriptado si conoce las claves privadas a o b; por lo tanto, en tal caso podría obtener C. Por eso, al igual que antes, el protocolo será seguro solamente si es difícil obtener a o b a partir de $A = e_a(g)$ o $B = e_b(g)$.

Remarcamos que no necesariamente este tipo de protocolos vienen a reemplazar los de clave privada descriptos en la sección anterior, sino que pueden utilizarse complementariamente: por ejemplo, utilizando el secreto compartido C que se obtuvo como clave privada para un protocolo como el de Vigenère.

2.1. El protocolo de Diffie-Hellman

Tal como el cifrado de Vigenère, el protocolo de Diffie-Hellman está basado en la aritmética modular. Pero en lugar de realizar simples sumas y restas, la operación utilizada para encriptar es la exponenciación.

Concretamente, Alicia y Beto comienzan eligiendo un primo p; todos los cálculos que harán serán módulo p. Hecho esto, eligen un entero g módulo p, que no sea divisible por p. Para ser precisos, la clave pública que usarán será entonces el par (p, g). Sus claves privadas serán enteros a y b, y las funciones de encriptado las dadas por

$$e_a(x) = x^a \pmod p, \quad e_b(x) = x^b \pmod p.$$

Notemos que la propiedad (2.2) se verifica gracias a que

$$(g^a)^b \equiv (g^b)^a \pmod p.$$

Más aún, por el pequeño teorema de Fermat, que afirma que $x^{p-1} \equiv 1 \mod p$ siempre que x no sea divisible por p, a efectos de los cálculos a realizar los enteros a y b pueden ser considerados módulo $p-1$.

Por ejemplo, tomemos $p = 101$. Elegimos, por razones que se explicarán en breve, $g = 2$. Si los exponentes secretos son $a = 20$ y $b = 17$, entonces:

- Alicia calcula $A \equiv g^a \equiv 2^{20} \equiv 95 \mod 101$, y se lo envía a Beto.
- Beto calcula $B \equiv g^b \equiv 2^{17} \equiv 75 \mod 101$, y se lo envía a Alicia.
- Alicia calcula $B^a \equiv 75^{20} \equiv 36 \mod 101$.
- Beto calcula $A^b \equiv 95^{17} \equiv 36 \mod 101$.

De esta manera, Alicia y Beto comparten el secreto $C \equiv 36 \mod 101$.

2.2. El problema del logaritmo discreto

Miranda conoce el primo p. Por eso, para empezar, necesitamos que p sea *grande*. Si por el contrario, por ejemplo en el caso que consideramos arriba, p es chico, habrá solo p posibilidades para el secreto C, y Miranda puede probar uno por uno hasta dar con la clave. Por ejemplo, un primo p de 26 dígitos es grande: si Miranda puede analizar un millón de secretos C por segundo, le llevaría más de 10^{13} años analizar los p secretos posibles. Eso es demasiado tiempo: la edad del universo está estimada en 10^{10} años.

Paréntesis. Como remarcamos, es deseable que evaluar las funciones involucradas en nuestros protocolos sea sencillo y rápido. En este caso, si p es grande y a es del orden de p, entonces para calcular $A = g^a \mod p$ Alicia debe realizar del orden de p multiplicaciones. Por eso, para el cálculo sea viable, no se puede exponenciar ingenuamente; se utiliza, por ejemplo, el algoritmo de *exponenciación rápida*, que permite calcular $g^a \mod p$ realizando del orden de $\log a$ multiplicaciones (véase [1], sección 1.3.2).

Siguiendo, necesitamos que g sea tal que los valores posibles de la función $x \mapsto g^x \mod p$ sean muchos. De lo contrario, tendríamos como recién un conjunto acotado de secretos C posibles, lo cual no es deseable. Una de las bondades de la aritmética modular es que para todo p siempre existirá un g tal que el conjunto de valores es tan grande como es posible: tiene $p-1$ elementos. En otras palabras, el grupo de unidades de los enteros no nulos módulo p es *cíclico*.

Supongamos entonces que elegimos p y g "correctamente". Miranda conoce g, A y B, y sabe que $g^a \equiv A \mod p$. Por lo tanto, para obtener $C \equiv B^a \mod p$ le basta con despejar a de la ecuación $A \equiv g^a \mod p$. Es decir, tiene que calcular el logaritmo de A en base g. Tratándose de cálculos módulo p, lo llamamos *logaritmo discreto*.

Por lo observado arriba, utilizar la *fuerza bruta*, esto es, recorrer los $p-1$ valores posibles de a hasta obtener uno que satisfaga $g^a \equiv A \mod p$, no es viable si p es grande.

Tal como hicieran Kasiski con el cifrado de Vigenère y Turing con la máquina Enigma, Shanks mostró que el problema del algoritmo discreto se

puede atacar con algo mejor que la fuerza bruta. Su algoritmo para calcular el logaritmo de A en base g, llamado *Pasos de bebé, pasos de gigante* (véase [1], sección 2.7), consiste en:

- Tomar $n = \lfloor 1 + \sqrt{p-1} \rfloor$.
- *Pasos de bebé*: calcular la lista

$$1, g, g^2, g^3, \ldots, g^n \pmod{p}.$$

- *Pasos de gigante*: calcular la lista

$$A, g^{-n}A, g^{-2n}A, g^{-3n}A, \ldots, g^{-n^2}A \pmod{p}.$$

- Hallar una *colisión* entre ambas listas: esto es, un par de valores i, j tales que $g^i = g^{-jn}A$.
- Devolver $a = i + jn$.

Notemos que para calcular ambas listas necesitamos n multiplicaciones, por lo que en total serán necesarias del orden de $2\sqrt{p}$ multiplicaciones: muchas menos que las p que necesita la fuerza bruta.

En el algoritmo está implícita la afirmación de que existe una colisión entre ambas listas: ese fue el descubrimiento de Shanks. La demostración de este hecho excede los límites del presente trabajo.

Epílogo. El problema del logaritmo discreto excede a la aritmética modular: puede ser planteado en un grupo abeliano G cualquiera. El algoritmo de Shanks, como puede observarse, funciona en este contexto general.

Si bien mejora al de fuerza bruta, la cantidad de operaciones que necesita el algoritmo de Shanks sigue siendo *exponencial* en el logaritmo de p. Hay otros algoritmos, exclusivos de la aritmética modular, que son sensiblemente más rápidos: por ejemplo, el *cálculo de índices* resulta subexponencial (véase [1], sección 3.8). Por eso resultan de interés para la criptografía las *curvas elípticas* sobre cuerpos finitos, ya que son grupos abelianos cuya estructura es lo suficientemente complicada como para que hasta ahora, eligiendo los parámetros de manera adecuada, el algoritmo de Shanks sea el ataque más rápido que se conoce.

Breve biografía del autor

Nicolás Sirolli se graduó como licenciado en Ciencias Matemáticas y como doctor en Matemática por la Facultad de Ciencias Exactas y Naturales de la UBA. Actualmente reviste como profesor adjunto de la misma facultad y como investigador CONICET del Instituto IMAS. Se dedica a la investigación en teoría de números y algunas de sus conexiones con la criptografía.

Referencias

[1] J. Hoffstein, J. Pipher y J. H. Silverman. *An introduction to mathematical cryptography*. 2nd. Undergraduate Texts in Mathematics. Springer, 2014.

Parte III

Derivas

Capítulo 7
La posguerra de Turing
César Mónaco

1. Una idea general de la posguerra a partir de dos imágenes

A varias décadas de distancia de finalizada, se han tornado cada vez más evidentes los efectos en el largo plazo de la Segunda Guerra Mundial sobre el mundo occidental y más allá de este. El conflicto, su desarrollo y finalización marcaron a fuego el siglo que los contuvo, y sus repercusiones siguen aún manifestándose sobre la vida cotidiana de buena parte del planeta. Pues, luego de 1945, se gestaron e impulsaron una serie de transformaciones económicas, sociales, políticas, tecnológicas y culturales de enorme trascendencia (dentro de la amplia y variada bibliografía, véase [5] y [3]). Como ha sostenido Tony Judt en su obra notable [4], el cierre de la contienda bélica abrió el comienzo de una nueva era.

No obstante, esa imagen actual se contrapone con fuerza a la de los años inmediatos al cese de la contienda. Lo que primó en ellos fue "una perspectiva de total miseria y desolación" [4, p. 35]. Nada sería igual luego de seis años de destrucciones y masacres. Ante todo, la guerra había sido una "experiencia civil": entre 50 y 70 millones de víctimas fatales, más otro tanto –muy superior aún– de damnificados más o menos directos. Pues como se ha sostenido, si la Primera Guerra Mundial había traído consigo un destacado movimiento de fronteras políticas, la Segunda conllevó el movimiento de extensas poblaciones: bombardeos, limpieza étnica y el traslado de millares de personas [4, p. 35]. A esto le continuaron, ya terminada la conflagración, nuevos reacomodamientos poblacionales, todos ellos sobre un escenario signado por la pobreza y el hambre, la devastación y la permanente amenaza de epidemias y enfermedades contagiosas. Para aquellos que habían logrado sobrevivir a la guerra –afirma Judt– le siguió un no menos duro sobrevivir a la paz [4, p. 45].

1.1. La situación británica

Al igual que la beligerante Alemania, Gran Bretaña participó los seis años que duró el conflicto. Más aún, dentro de las principales potencias aliadas,

fue esta la única que transitó en extenso la etapa. De esto no se induce, desde ya, que su territorio haya sido el blanco permanente de las fuerzas del Eje, y menos aún que haya estado exento de los ataques.

Como sabemos, la guerra se desarrolló en varios escenarios geográficos a la vez, aunque con una importante centralidad en la Europa continental. No obstante, luego de la aplastante invasión de Francia por parte del ejército alemán, y su consecuente capitulación en junio de 1940, Hitler decidió avanzar sobre Inglaterra. Entre julio de 1940 y mayo de 1941, este sector de las islas británicas padeció constantes y feroces ataques aéreos. Consideremos, además, que para ese entonces el gobierno insular era la única potencia enfrentada, en términos bélicos, a los nazis (Francia se encontraba ocupada; la Unión Soviética y los Estados Unidos, por diversos motivos, ingresarían al conflicto en junio y diciembre de 1941). Y si bien los "verdaderos horrores", en comparación, se vivirían en el continente (con avances terrestres, desplazamientos de personas, etc.), en Inglaterra, los costos sociales y económicos fueron altísimos [4, p. 43]. Durante meses, la Luftwaffe y su *Blitzkrieg* asediaron el país. En Londres, varios distritos del East End fueron arrasados, y la vieja *City* (donde se había asentado la antigua ciudad) desapareció entre las llamas.

Algunas referencias: en términos de vidas, fueron cerca de 400000 las víctimas británicas –entre militares y civiles–. Esto es, 1 cada 125 ingleses murieron a causa de la guerra. Si queremos establecer una comparación con algunas de las otras potencias: en Francia la relación fue de 1 cada 77 habitantes; en Alemania fue de 1 cada 15; en la URSS fue de 1 entre 11.

Respecto al esfuerzo económico, mientras que una parte importante de los recursos para la guerra Alemania los obtuvo de la explotación de los pueblos conquistados, Gran Bretaña llegó a dedicar más de la mitad de su Producto Bruto Interno (PBI) a sus esfuerzos bélicos. Como era de esperar, las restricciones y la escasez precipitaron el nivel de vida de la población (en el caso alemán, esto fue percibido por su pueblo recién en los meses finales de la guerra).

Al igual que buena parte de Europa, concluidas las hostilidades en 1945, la economía insular comenzó rápidamente su recomposición. Para esto, necesitó de los flujos financieros de los Estados Unidos. Sin embargo, la restauración no se daría en los mismos términos. De inmediato, la presencia del Estado, en los planos económico y social, se volvió más marcada. Fue el inicio del Estado benefactor británico. Una reestructuración impulsada por el laborismo inglés y apoyada por los sectores liberales, ahora inclinados hacia la reforma.[*]

En efecto, la Segunda Guerra Mundial "transformó tanto el papel del Estado moderno como lo que se esperaba de él" [4, p. 164]. Y esta transforma-

[*] En su reconocido informe de 1942, el economista liberal William Beveridge "estableció cuatro supuestos acerca de la dotación de servicios de bienestar para la postguerra, todos los cuales serían incorporados a la política británica durante la siguiente generación: la necesidad de que hubiera un servicio nacional de salud, una pensión estatal adecuada, ayudas familiares y cuasi pleno empleo" [4, p. 126]'.

ción tuvo su paradigma en el modelo británico. Un Estado fuerte, presente y con una marcada política social. En el primer lustro posterior a 1945, se nacionalizaron y extendieron los servicios públicos (electricidad y gas), el ferrocarril, empresas de producciones o extracciones consideradas estratégicas (hierro, acero, carbón), la aviación civil y la radio. Se subvencionó el transporte público y se volcaron recursos para las artes y la cultura. Y sobre todo, se estructuró una prestación de servicios sociales basados en la construcción de viviendas, la ampliación del sistema educativo (los estudios secundarios pasaron a ser gratuitos) y un Sistema Nacional de Salud de amplios alcances (el renombrado NHS). Surgió así un Estado proveedor de seguros contra la enfermedad, pero también frente al desempleo, los accidentes y la vejez. Hacia 1949, el 17 % del gasto público se dirigía hacia este sistema de seguridad social (un incremento de un 50 % si nos situamos diez años antes).

En términos políticos, para la estructuración de este proceso fue clave la llegada del Partido Laborista al poder. A partir de 1945, y luego de ganar con el 48 % de los votos, Clement Atlee asumió como primer ministro. Comenzaba así esta era de reformas con impacto en múltiples dimensiones de la vida de los británicos.

1.2. La guerra fría

Como trasfondo de esta recomposición económica y despliegue del Estado de Bienestar europeo, y también de otras tantas transformaciones en la sociedad y la cultura, se desplegó una inusitada situación en la geopolítica internacional: la Guerra Fría. Esta consistió en el enfrentamiento sostenido (por lo que restó del siglo xx) entre las dos superpotencias que surgieron luego de finalizada la Segunda Guerra Mundial: por un lado, los Estados Unidos al frente de lo que se conoció como bloque occidental capitalista; por el otro, la Unión Soviética, en tanto nación hegemónica en el bloque socialista (o experiencias del "socialismo real").

Salvo en lo bélico (entre sus dos protagonistas, al menos), la Guerra Fría implicó una confrontación en términos políticos, económicos, científicos, sociales y culturales. Y fue tal su relevancia para el orden internacional, tanto como en la cotidianeidad interna de la mayoría de los países del mundo, que terminó dándole el nombre a un período histórico concreto.

En lo concerniente a los fines de este capítulo –que busca dar un marco general del contexto en el que debemos ubicar a Alan Turing, luego de 1945– importa señalar que, más allá de todo lo dicho, esta fue también una etapa de persecución política e ideológica. En sus memorias, al recordar su vida académica durante esos años, Eric Hobsbawm ha señalado en recurrentes ocasiones los cambios provocados entre sus pares, dentro de las instituciones y en la misma comunidad que los contenía, a raíz de la "guerra fría".

Una nueva mirada y una nueva actitud (o renovadas, ya que no eran del todo nuevas) se comenzaron a esparcir sobre aquellos que adscribían al comunismo (de forma pública o dentro de la intimidad), sospechosos en todo

momento de participar en los servicios secretos del "enemigo"[*].

2. Una mención a Turing, un signo de época

Claro que esta etapa de evidentes transformaciones en clave reformista, que preanunciaba otras que con las décadas se irían desplegando en el cuerpo de las sociedades[†], se daba de bruces con el arraigado conservadurismo inglés.[‡] En verdad, el particularismo británico perdía mucha de su singularidad si se lo comparaba con otras experiencias dentro y fuera del continente europeo. Un conservadurismo social extendido, que atravesaba ideologías, generaciones, países y se sostenía —aunque pronto perdería fuerzas— mientras los gobiernos intervenían de modo progresivo por medio de la asistencia y la redistribución de la riqueza.

Esta particularidad, que como acabamos de decir no era solo británica, se veía reforzada en países de tradiciones arraigadas como Inglaterra. Yendo al caso de Turing, y lo que será su trágico final, hay que señalar que en la Gran Bretaña de los años cincuenta la homosexualidad se encontraba aún perseguida y penada. Entendida como una "enfermedad", su punición solía ir acompañada de acciones directas (tratamientos) con el objeto de "corregir" tal "desviación".

En uno de sus libros más relevantes, Hobsbawm, historiador, ciudadano británico y un completo hombre del siglo xx —se podría decir—, describió a Alan Turing:

> [Como ese] genio de tez pálida y aspecto desmañado, que era por aquel entonces un becario aficionado al *jogging* y que se convirtió póstumamente en una especie de ídolo para los homosexuales, no era una figura destacada ni siquiera en su propia facultad universitaria, o al menos yo no lo recuerdo como tal.

Para agregar en una nota a pie de página:

> Turing se suicidó en 1954, tras haber sido condenado por comportamiento homosexual, que por aquel entonces se consideraba un delito y también una patología que podía curarse mediante un tratamiento médico y psicológico. Turing no pudo soportar la "cura" que le impusieron. No fue tanto una víctima de la criminalización de la homosexualidad (masculina) en Gran Bretaña antes de los años sesenta, como de su propia incapacidad para asumirla.

[*] No olvidemos que la Europa de posguerra trajo consigo un avance sustancial del comunismo y su presencia destacada en el sistema de partidos de muchos de sus países.

[†] Pensemos en los vecinos años sesenta y el protagonismo de la juventud, la liberación sexual y su crítica a las costumbres oprimentes, la mayor presencia de las mujeres en la esfera pública (cada vez menos excepcional), la rebeldía revolucionaria, el consumismo iconoclasta expresado en el rock. O en el avance y aceptación que desde la década del 80 hasta la actualidad han tenido las minorías sexuales.

[‡] Al punto que, respecto a su estructura de clase: "En una sociedad como la inglesa de la primera mitad del siglo pasado [s. xx], pasar de un entorno social a otro constituía una forma de emigración" ([3, p. 101].)

> Sus inclinaciones sexuales no provocaron ningún problema en el King's College de Cambridge, ni entre el notable conjunto de personas raras y excéntricas que durante la guerra se dedicaron a descifrar códigos en Bletchley, donde Turing vivió antes de trasladarse a Manchester, una vez terminada la guerra.

Y finaliza:

> Solo un hombre que, como él, desconocía el mundo en el que vivían los demás podía ocurrírsele ir a denunciar el robo cometido en su casa por un amigo íntimo (temporal), dando así a la policía la oportunidad de detener a dos delincuentes a la vez. [3, p. 250].

Estas palabras de Hobsbawm fueron escritas a mediados de los años noventa, cuatro décadas después de la muerte de Turing, y desde mi punto de vista expresan evidentes reminiscencias epocales de los años cincuenta. En ellas, lo llamativo es la ausencia de condena o mera mención a una situación represiva que, desde nuestro presente, no deja de parecernos aberrante. Visto desde otro ángulo, lo llamativo, luego de tantas décadas transcurridas, es la implícita aceptación de la criminalización de prácticas sexuales "fuera de la norma"[*].

2.1. Acerca de la legislación antihomosexual, no solamente inglesa

Desde tiempos remotos, la legislación contra la sodomía ha venido persiguiendo y penalizando a aquellas actividades sexuales consideradas indecentes e inmorales, y por tales, tipificadas como delito. Preferentemente, esta ha sido aplicada contra la homosexualidad masculina. Una situación que pervive aún en muchos países. Según un informe de una asociación especializada, hacia 2020, sesenta y ocho estados continuaban criminalizando "los actos sexuales consensuales entre personas adultas del mismo sexo" (véase, [6]). Las penas, de acuerdo a cada uno de estos países, pueden ir de la mera criminalización de facto, la prisión entre 8 y 10 años, a una posible o efectiva pena de muerte.

Este mapa actual, desde ya, no es un invento moderno. Las leyes de sodomía perviven desde la antigüedad. La introducción del cristianismo implicó un endurecimiento de las persecuciones, expresado sobre todo en la punición de los "pecados nefandos" por parte de la Santa Inquisición medieval. Más allá de occidente, y de funesta persistencia, vale mencionar a la sharia en el amplio mundo musulmán.

En los Estados Unidos, hubo que esperar hasta los años setenta del siglo xx para que se fueran derogando las sanciones. Antes de eso, solo el estado de Illinois, en 1962, dejó de penar los actos sexuales consentidos entre personas del mismo sexo. Al ser un asunto de incumbencia estatal (y no federal),

[*] N. de la R.: en el capítulo 1, sección 3.2, se propone una relación entre estas vivencias de Turing y su propuesta de definición de inteligencia artificial que originalmente propuso en el artículo [7].

la evolución del proceso de descriminalización dentro del país fue variado, y en no pocas ocasiones se suscitaron obstáculos político-institucionales que hicieron muy difícil su superación. En el año 2003, fue el propio Tribunal Supremo –la instancia judicial más elevada en esa nación– quien dispuso avanzar sobre el tema y normalizar la situación en una serie de estados (14 en total) que seguían sosteniendo las puniciones.

En la Unión Soviética, a partir del ascenso de Stalin al poder a comienzos de los años treinta, se restablecieron, y con mayor dureza, las sanciones sobre las prácticas homosexuales que habían sido derogadas bajo el gobierno bolchevique. Estas permanecieron, aunque con cierto grado de flexibilización conforme fueron avanzando las décadas, prácticamente hasta la disolución del estado comunista en 1991.

En la Cuba de Fidel Castro, para citar otro ejemplo del bloque socialista, los años sesenta representaron el momento de mayor persecución. Los "culpables", considerados parte de los sujetos potencialmente peligrosos para la "nueva sociedad", eran enviados a campos de trabajo forzoso, denominados Unidades Militares de Ayuda a la Producción (véase [2]).

Aunque podrían sumarse muchas otras referencias sobre regiones, países, culturas y regímenes estas menciones nos dan un panorama general sobre el tema en las décadas siguientes a la última guerra mundial. Vale volver a señalar que en su amplia mayoría –aunque no en su exclusividad– las persecuciones y penalidades se centraban sobre los hombres.

En Inglaterra, la *Buggery act* (Ley de sodomía) fue implementada en 1533, bajo el reinado de Enrique VIII. Esta consideraba delito las prácticas antinaturales y las castigaba con la pena de ahorcamiento. Como legislación central, esta se expandió a las colonias británicas y terminó siendo una especie de ley base. Muchas décadas más tarde, en 1861, la tipificación no fue alterada, pero fue eliminada la pena de muerte (capital offence). Tal vez el caso más conocido al respecto sea el del escritor Oscar Wilde, quien en 1895 fue acusado de sodomía y grave indecencia, y condenado a dos años de trabajo forzado. Esta legislación persecutoria fue suprimida recién en 1967 (en ese mismo año también en Gales; en Escocia en 1980 y en Irlanda del Norte en 1982).

Una última y breve mención: en la Argentina la despenalización de la sodomía ocurrió en 1886. Claro que esto no impidió que durante una buena parte del siglo xx continuara el asedio policial, basado sobre todo en edictos distritales que habilitaban las redadas en baños públicos y en otros tantos lugares de encuentros.

3. Palabras finales

Con motivo de la elaboración de este conciso texto, he leído algunas notas biográficas y visto algunos documentales y películas sobre Turing y algunos de los temas vinculados a su persona. En ese trayecto me encontré con el Polari, una jerga creada y utilizada por la cultura (o subcultura) gay de Gran Bretaña. Polari viene del italiano "parlare" y al parecer sus orígenes

hay que rastrearlos hasta entrado el siglo xix. Parte del mundo gay británico fue desarrollando este lenguaje propio ante la criminalización de la homosexualidad: permitía comunicarse en público sin ser arrestado. El polari tuvo su momento de mayor difusión en las décadas de 1950 y 1960 (véase [1, Cap. 1]). Como lenguaje secreto, puede decirse que el polari fue un contralenguaje y, desde ya, un código.

Como podrá percibir el lector, no me resultó difícil pensar en cierto paralelismo (o juego de paralelas) entre ese código de comunicación y las actividades realizadas por Turing durante la guerra. No he encontrado referencias acerca del conocimiento del polari por parte del matemático londinense; y en verdad, a los fines de este sucinto cierre tampoco creo que sea imperioso saberlo. Lo cierto es que al menos como un juego imaginario podemos pensar en un "punto impropio" entre ambas paralelas. Entre las múltiples cosas que Turing legó a la posteridad, una absolutamente relevante fue la de ser un *codebreaker*, un hombre clave y fundamental en los peores momentos del siglo pasado. Bajo su función de rompedor de códigos, como sostuvo Churchill: "Turing hizo la contribución individual más grande para la victoria de los aliados".

Finalizó la guerra, pasaron los años y estas paralelas, en ese plano imaginario, continuaban tocándose. La década iniciada en 1960 se caracterizó por la expansión de la cultura popular (pop). A nivel mundial, estos representaron los "años dorados" de las economías, de la "revolución social y de la cultural". En Inglaterra, la ley represiva contra la homosexualidad estaba pronta a ser abolida, y el polari perdería un poco de su sentido original y sería readaptado por parte de la cultura iconoclasta y el consumo de drogas ilegales. Las décadas seguirían pasando y la figura de Turing emergería como un símbolo de la libertad sexual y del reconocimiento de la homosexualidad frente a la dominante cultura heterosexual. En 2016, en el Reino Unido se aprobó una ley de amnistía retroactiva a decenas de miles de varones que habían sido penados bajo la antigua legislación. De forma corriente se la denominó "Ley Alan Turing".

Breve biografía del autor

César Mónaco es doctor en Historia por la Facultad de Humanidades y Ciencias de la Educación de la Universidad Nacional de La Plata y profesor universitario de Historia (UNGS). Actualmente se desempeña como investigador-docente del Área de Historia del Instituto del Desarrollo Humano (UNGS) y es docente de la Maestría de Historia Contemporánea (UNGS). Su investigación se centra en temas de historia social de la Argentina de la segunda mitad del siglo xx, específicamente en la historia del movimiento obrero y de los trabajadores durante esa etapa.

Referencias

[1] P. Baker. *Polari: The Lost Language of Gay Men*. Londres y Nueva York: Routledge, 2002.

[2] A. Fuente. *La revolución de la comunidad gay en Cuba*. https://elpais.com/elpais/2017/05/08/planeta_futuro/1494257202_915266.html. Consultada el 27 de abril de 2020. 2017.

[3] E. Hobsbawm. *Historia del Siglo xx, 1914–1991*. Barcelona: Crítica, 1995.

[4] T. Judt. *Postguerra. Una historia de Europa desde 1945*. Barcelona: Penguin Random House, 2014.

[5] I. Kershaw. *Ascenso y crisis Europa 1950-2017: un camino incierto*. Barcelona: Crítica, 2019.

[6] ILGA World: Mendos y col. *Homofobia de Estado 2020: Actualizacion del Panorama Global de la Legislacion*. https://ilga.org/maps-sexual-orientation-laws. Consultada el 27 de abril de 2020. 2020.

[7] A. M. Turing. "Computing Machinery and Intelligence". En: *Mind* 49 (1950), págs. 433-460.

La Colección Entrecruzados presenta libros que reflexionan sobre temas científicos de orden general, con la particularidad de ser abordados desde al menos dos disciplinas (es decir, al menos dos autores), que establecen un enfoque complementario del conocimiento.

En este libro se indaga sobre las ideas que dieron lugar a nuestra vida digital moderna mediante una serie de artículos que toman la obra de Alan Turing y su contexto histórico como hilo conductor. En la compilación de estos trabajos se describe de forma amena, reduciendo tecnicismos, qué es la "máquina de Turing", el contexto matemático que le dio origen y algunas de sus consecuencias técnicas y conceptuales. Asimismo, se dan perspectivas históricas y técnicas que describen la labor de Turing como integrante del equipo de descifradores de la máquina criptográfica "Enigma", quienes trabajaban para salvar a su país de los embates alemanes durante la Segunda Guerra Mundial. El libro concluye con una introducción a la criptografía moderna de clave pública.

Universidad Nacional de General Sarmiento

L U A
Libro Universitario Argentino

ISBN 978-987-630-736-9