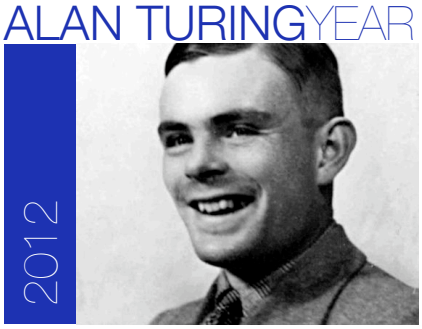
Alan Turing: Una aproximación personal a su obra

por

David de Frutos Escrig

Resumen. Con ocasión del centenario del nacimiento de Alan Turing, y la celebración a nivel internacional de «The Alan Turing Year», con el que se conmemora el mismo a nivel internacional, presentamos una visión personal de su obra, exponiendo de manera divulgativa sus principales resultados e incidiendo fundamentalmente en las ideas que los guiaron. Nos centramos en especial en los temas que hoy en día catalogamos como fronterizos entre la Matemática y la Informática, destacando sus contribuciones seminales a la Teoría de la Computabilidad y a la Inteligencia Artificial, las actividades relacionadas con la Criptografía durante la Segunda Guerra Mundial, y finalmente sus aportaciones absolutamente visionarias a temas tan actuales como la Vida Artificial, la Morfogénesis y los Algoritmos Genéticos.

Sin lugar a dudas nos encontramos ante uno de los matemáticos, y precursores de la Informática, más importantes del siglo XX. Y ello a pesar de lo muy corta que fue su existencia, que no llegó a alcanzar los 42 años, y de que buena parte de la misma se desarrolló durante un periodo tan tormentoso para todo el mundo como fue la Segunda Guerra Mundial, lo que como veremos influyó tremendamente en su quehacer científico.



Celebramos durante este año el centenario de su nacimiento, convertido en *The Alan Turing Year*, que ha venido a poblar de eventos conmemorativos toda la superficie del planeta, destacando en especial *The Turing Centenary Conference* y más de un centenar de sesiones conmemorativas, en todo tipo de congresos y reuniones relacionadas con las áreas que él cultivó.

Son también muchísimos los artículos y textos dedicados a su persona, tanto publicados desde su muerte, como en ocasión de este centenario, por lo que resultaría enormemente pretencioso por mi parte tratar de aportar aquí datos o detalles científicos novedosos. Además, la limitación de espacio es evidente, por lo que es inevitable asumir omisiones e incluso pérdida de precisión técnica en ocasiones. Por ello, mi objetivo principal será dar un punto de vista personal sobre Turing y su obra, buscando una presentación divulgativa de sus principales resultados, incidiendo fundamentalmente en las ideas que los guiaron. Es natural, por otra parte, que

me centre en especial en aquellos temas más cercanos al área que más domino, aunque trataré de citar suficientemente todos los muy diversos campos en que Turing trabajó durante su lamentablemente corta vida.

He consultado diversas obras especializadas para la realización de este trabajo, pero entre ellas quiero destacar aquí *The Essential Turing* [1], editado por B. Jack Copeland, donde se reúnen los originales de las obras más importantes de Turing en relación a la *Teoría de la Computabilidad*, la *Lógica* y diversas áreas de la Informática, que van desde la construcción y programación de los ordenadores de propósito general, hasta la tremendamente actual *Vida Artificial*. Todos los trabajos de Turing que más tarde referenciaré citando solo su título, aparecen y son brillantemente comentados por Copeland en la citada obra, por lo que me ha parecido innecesario citar las referencias originales, que serían además mucho más difíciles de localizar. Animo al lector interesado a visitar la página `http://www.mathcomp.leeds.ac.uk/turing2012/` donde se recoge todo tipo de material en relación a la vida y obra de Turing, con ocasión de la celebración de su centenario.

El artículo comienza con una Introducción en la que se se exponen de forma cronológica las fases de la vida de Turing y los temas principales en los que fue trabajando, para luego detallar un poco más las tres grandes áreas en las que hemos decidido centrarnos: Teoría de La Computabilidad, Criptografía e Inteligencia Artificial (IA), concluyendo con una sección más breve dedicada a la Vida Artificial, y un Epílogo conteniendo las principales referencias bibliográficas.

Introducción: Fases de la vida de Turing y Breve descripción de sus Logros Principales

Alan Mathison Turing nació el 23 de junio de 1912 en Londres, y tras estudiar en Doset, accede al King's College de Cambridge, donde se gradúa en Matemáticas en 1934. Solo dos años más tarde publica su obra científica más importante, *On Computable Numbers with an Application to the Entscheidungsproblem*, en la que se introducen las que luego se llamarían *Máquinas de Turing*. Por medio de ellas se formaliza el concepto de *Algoritmo*, para dar lugar a los resultados seminales de la *Teoría de la Computabilidad*, que entroncan con la *Lógica Formal*, dando respuesta negativa a la *Conjetura de Hilbert sobre la Decidibilidad de la Lógica de Primer Orden*.

Su Tesis Doctoral, *Systems of Logic Based on Ordinals*, desarrollada en Princeton bajo la dirección de *Alonzo Church*, y culminada en menos de dos años, a mediados de 1938, continúa los trabajos en esa línea, introduciendo conceptos de una mucho mayor sofisticación técnica, como la *Computabilidad Relativa*, que da pie a la *Teoría de Grados*. En palabras llanas, se trata de clasificar la dificultad de los problemas imposibles, constatando que algunos de ellos son «más imposibles que otros». En realidad, la justificación técnica de esta clasificación es bastante sencilla: supongamos por un momento que disponemos de un método mágico (*oráculo* en el argot) para resolver (todas las instancias de) un cierto problema P. Entonces, diríamos que otro problema Q es más sencillo P si contamos con una traducción (*reducción* en el argot)

de las instancias de Q a las de P, que utilizando dicho método mágico resuelve cada instancia posible de Q.

En el verano de 1938 Turing regresa a Cambridge para, tras los inicios de la Segunda Guerra Mundial, ser reclutado por el Gobierno de Su Gloriosa Majestad, para trabajar en el Grupo de Cifrado (por supuesto secretísimo) radicado en *Bletchley Park*. Allí se buscó por todos los medios posibles un mecanismo que permitiera descifrar rápidamente los mensajes en clave intercambiados por las unidades del Ejército Alemán. Este utilizaba para su transmisión ejemplares de la famosísima máquina *Enigma.* Dado que los métodos para realizar dicha decodificación exigían interminables procesos imposibles de hacer a mano, Turing participó en el diseño de diversas máquinas electromecánicas, conocidas con el nombre genérico de *Bombe*, que permitían ejecutar muchas operaciones en un tiempo reducido. Más adelante se desarrolló en Bletchley Park la máquina *Colossus*, dedicada a descifrar los mensajes codificados con medios mucho más sofisticados, con los que se comunicaba el alto mando alemán.

Solo bastante recientemente se terminó de retirar el lacre de secreto a los documentos relacionados con Bletchley Park, donde tras ganar la Guerra fueron intencionadamente destruidas todas las máquinas desarrolladas allí. Ciertamente, Colossus fue una máquina electrónica (basada en *válvulas*), pero no un computador de propósito general, pues su propósito era claramente específico, y además debía ser «reconfigurado» a nivel físico, por medio de interruptores y conectores, cuando se decidía utilizarlo para resolver problemas diferentes. Además, no queda claro hasta qué punto Turing tuvo una relación directa con su desarrollo, pero como quiera que posteriormente sí que participó en parte de los trabajos que terminaron por hacer surgir los primeros precursores de los modernos computadores, nos queda la duda de hasta qué punto ideas que surgieron en Bletchley Park fueron posteriormente cruciales en el desarrollo de esos primeros ordenadores de propósito general. Lo que sí que parece muy probable es que la durísima y siempre respetada exigencia de mantener en secreto todo lo que tuviera que ver con Bletchley Park, tuvo mucha culpa en que las primeras y grandes compañías de computadores fueran americanas y no británicas.

Nada más terminada la guerra, Turing pasó a trabajar en el *National Physical Laboratory* (NPL) de Londres, donde una vez más se muestra como un adelantado a su tiempo, pues si bien se esperaba de él que fuera un eslabón más del equipo que buscaba construir bajo el cielo británico el primer computador de uso general, dentro del proyecto *ACE* (Automatic Computing Engine), Turing fue mucho más allá. Ello sucedió así tanto en lo relativo al diseño que realizó, tachado por sus colegas de demasiado ambicioso, como por las digresiones realizadas durante esa misma época, que le llevaron a los que hoy en día consideramos trabajos seminales sobre *Inteligencia Artificial*, abordando en un reducidísimo tiempo facetas que posteriormente solo han sido desarrolladas en profundidad en una época reciente.

De hecho su primer trabajo en este campo, *Intelligent Machinery*, dedicado a las redes neuronales, data de 1948, pero no fue nunca publicado hasta que apareció en un compendio aparecido en 1968, mucho después de su muerte. Curiosamente, el logro más divulgado en este terreno es el archiconocido *Test de Turing*, desarrollado en su

artículo *Computing Machinery and Intelligence*, que trata de establecer un criterio para reconocer una eventual inteligencia humana en un ordenador. A pesar de que siguen apareciendo múltiples trabajos que desarrollan variantes de ese test, y tratan de precisar en qué medida con él se estarían midiendo (y por tanto simulando por las máquinas que salieran victoriosas) los rasgos fundamentales de nuestra inteligencia natural, a mi entender, y al de la mayoría de los expertos que trabajan en IA, el test en sí mismo, al menos en su versión original, solo evaluaría ciertos aspectos de nuestra inteligencia, que posiblemente ni siquiera sean los más fundamentales de la misma.

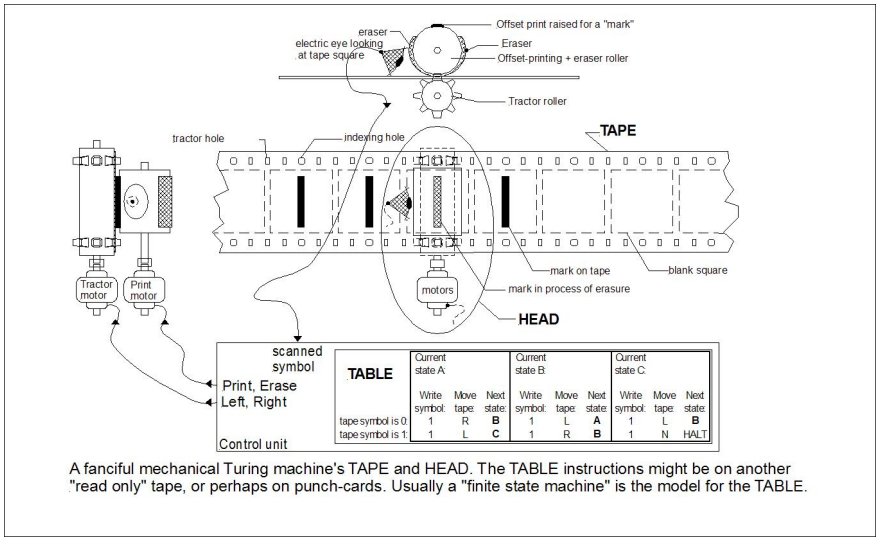
Y siguiendo con la presentación resumida de las fases de la vida de Turing y su obra, queda para el final otra de sus facetas más visionarias. Esta se desarrolla en la Universidad de Manchester, fundamentalmente durante un solo año (marzo de 1951 a marzo de 1952), pues desgraciadamente llegada esta segunda fecha se produce la acusación de homosexualidad contra él, que terminó en condena condonada por un brutal tratamiento de estrógenos, que le produjo una tremenda depresión, que le conduciría al suicidio mediado 1954. Pero centrémonos en lo realizado durante el citado año, y parte de los meses siguientes, dando lugar al trabajo *The Chemical basis of the Morphogenesis.* Intuir que gracias a los ordenadores sería posible simular con suficiente precisión un sistema tan complejo como es *La Vida*, en su componente biológica, es algo en verdad impresionante. Las bases para esa simulación se encuentran en la modelización matemática de los procesos naturales, y Turing centró su atención en dos facetas: la *morfogénesis* y los procesos de *reacción difusión.* Es en estos trabajos donde aparecen aportaciones a la matemática más clásica, de nuevo de un modo visionario, al intuir que las difíciles ecuaciones no lineales que aparecen solo serán tratables cuando se cuente con ordenadores suficientemente potentes, capaces de efectuar los interminables procesos de cómputo necesarios para aproximar sus soluciones.

Y como si se tratara de un negro presagio, los últimos trabajos de Turing tienen en cierto modo un carácter recopilatorio, al abordar problemas en los que ha de combinar ideas y resultados obtenidos previamente en diversas fases de su vida, así como una vuelta a los orígenes. En el primer caso se engloba un curioso trabajo sobre la mecanización del ajedrez, donde aparecen las primeras ideas sobre lo que hoy en día es el *Aprendizaje Autómatico*, de la mano de los *Algoritmos Genéticos.* Y en lo relativo a la continuación de sus trabajos sobre Computabilidad, tenemos un delicioso y casi póstumo trabajo *Solvable and Unsolvable Problems,* en el que se formaliza la noción de *Indecidibilidad*, facilitando una prueba más sencilla del *Segundo (o General) Teorema de Incompletitud de Gödel*, reiterando la conexión entre los trabajos de ambos, ya proclamada en su día por el propio Gödel.

Las Máquinas de Turing y los Números Computables

Las Máquinas de Turing (MT) son dispositivos ideales por medio de los cuales Turing trató de capturar la noción de algoritmo de una forma sencilla y convincente. Preferimos eludir aquí la definición precisa de MT, presente en cualquier texto de

Computabilidad, si bien las versiones son ciertamente muchas, aunque puede probarse que todas ellas son equivalentes. Trataremos entonces de hacer una descripción más informal, buscando justificar cada elemento que aparece en su definición, y explicar su cometido. El objetivo de Turing es computar, y si buscamos un marco sencillo para expresar el objetivo de dicha computación, hoy nos centraríamos sin duda en la evaluación de funciones $f : \mathbb{N}^k \to \mathbb{N}$, de manera que cada cómputo comenzaría recibiendo un dato (*input* o entrada) $\overline{d} \in \mathbb{N}^k$, para producir el correspondiente valor (*output* o salida) $f(\overline{d}) \in \mathbb{N}$. A nivel teórico, podemos reducir el marco al caso $k = 1$, utilizando cualquier biyección natural entre $\mathbb{N}^k$ y $\mathbb{N}$ para codificar las entradas por medio de un único natural. Otro marco equivalente es entonces el que genera directamente toda la *tabla* de f, por medio de la sucesión $\langle f(n)\rangle_{\{n\in\mathbb{N}\}}$. Turing, en su artículo seminal, escoge una variante aparentemente más complicada, pero a la postre equivalente a esta segunda, que es la de generar las infinitas cifras *decimales* de un número real (computable) en el intervalo $[0, 1)$. Dada esta mayor complejidad, me permitiré manejar en lo sucesivo las dos primeras variantes.



A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Turing, de acuerdo con todos quienes asumían que la computación tangible exige el uso de magnitudes discretas, se centra en la formalización de los cómputos con naturales, pero el correspondiente nivel de discretización ha de ir más allá, basándose en una representación finitaria que exige descomponer dichos naturales en cifras (base 2 o superior) o unidades (base 1 o representación con palotes), que serán visualizados y manejados por separado. Necesitamos entonces un campo de juego (memoria) dividido en casillas, donde podamos almacenar al principio la entrada, al final la salida, y en el entreacto todos aquellos datos auxiliares (papel de sucio) que se precisen para llegar a la salida deseada a partir de la entrada dada. Al efecto, Turing opta por la variante más sencilla: una cinta (potencialmente) infinita dividida en casillas, cada una de las cuales puede albergar un símbolo del alfabeto (0 o 1 en

el caso binario, y solo 1 en el unario) o un separador (representado, por ejemplo, por *nada*, la casilla vacía).

El proceso de cómputo también ha de ser discreto. Eso lleva a Turing a dotar a su máquina de un visor móvil, que en cada momento permite ver y acceder a una sola casilla, pudiéndose desplazar a una casilla vecina cuando resulte conveniente. Entonces, los pasos de cómputo se reducirán a mirar y actuar: miro por el visor y hago lo que proceda según lo que vea. Por ejemplo, podemos quedarnos con solo dos tipos de actuaciones: (borrar y) *escribir* en la casilla y *moverme* (a derecha o izquierda). Cuando nuestro objetivo es computar un valor único, añadiremos una acción final cuya ejecución detiene el proceso, facilitando la correspondiente salida computada.

Pero si siempre que vemos lo mismo hacemos lo mismo, es obvio que nuestro grado de flexibilidad es mínimo, por lo que no podríamos computar gran cosa. La flexibilidad necesaria se consigue introduciendo un mecanismo que nos indique lo que hemos de hacer en cada circunstancia, dependiendo del momento en que nos encontremos. En el argot denominamos *control* (o estado) a la formalización de dicha noción de momento, y la colección de directrices para cada momento sería el *programa*. Pero de nuevo es fundamental que el control sea discreto, dando lugar así a programas finitos. Además, debemos introducir un mecanismo que nos permita variar el control. Lo más fácil al respecto es convenir que cada paso de cómputo correspondiente a un estado nos indicará también el estado en que se realizará el siguiente.

Si ponemos ahora junto todo lo anterior, nos encontramos con que el programa será una tabla finita de instrucciones, y los cómputos asumirán que a su comienzo la entrada está ya en la cinta, por ejemplo con el visor a su izquierda, para ir ejecutando paso a paso la instrucción que proceda, hasta que la máquina se pare, o procediendo infinitamente si ello no sucediera. Solo aparece una (importante) dificultad técnica: aun en el caso de que nuestro objetivo sea computar funciones totales, podemos (se supone que equivocadamente) crear programas que ante ciertas entradas no paren nunca. En tal caso convenimos que para esas entradas el valor de la función computada no existe, por lo que se trataría de una *función parcial*. Por el momento, parece que el recurso habitual de los matemáticos de extender el campo de juego cuando así se necesita, funciona una vez más y sin complicaciones excesivas, pero más adelante veremos que es inevitable terminar pagando bastante por ello.

Una primera consecuencia inmediata de esta formalización es que la mayoría de las funciones $f : \mathbb{N}^k \to \mathbb{N}$ no son computables: hay una cantidad no numerable de ellas, pero solo puede haber una cantidad numerable de programas distintos, y por tanto solo una cantidad numerable de funciones computables, o de reales computables en el sentido de Turing. Por otra parte, la simplicidad (¿excesiva?) de las MT, nos podría llevar a pensar que ello es así porque nos hemos quedado cortos, pero las asunciones sobre discretitud del modelo parecen inevitables, y a partir de ellas la numerabilidad del conjunto de funciones computables no dependería de la definición precisa de los restantes elementos del modelo.

La Tesis de Church

La búsqueda de mecanismos sencillos que capturen la esencia de la noción de algoritmo, y caractericen por tanto el conjunto de funciones computables, no es en absoluto patrimonio de Turing. Un poco antes que él, Church había presentado su λ-*cálculo*, postulando que las funciones computables «de un modo natural» coincidirían con las λ-definibles. A Hilbert este formalismo le había parecido demasiado rebuscado, por lo que tenía serias dudas sobre su completitud. Sin embargo, Turing prueba que Turing-computable y λ-definible es lo mismo, y la simplicidad de las MT hace que Hilbert acepte que todo lo naturalmente computable podría ser definido por una MT, o equivalentemente computable por medio de una λ-expresión. Nace así la Tesis de Church, que consiste en aceptar lo anterior... hasta que no se demuestre lo contrario, lo que hasta la fecha no ha sucedido, a pesar de que se han propuesto centenares de sofisticados mecanismos que fuerzan hasta el extremo la naturalidad de la noción de computación, para demostrarse una y otra vez que todos son equivalentes a las MT, que en consecuencia han pasado a ser nuestro patrón oro: por definición, llamamos computable a lo Turing-computable.

Permítaseme comentar aquí un tercer modelo equivalente posterior debido a Kleene, que es el de las funciones μ-*recursivas*. Los matemáticos estamos totalmente acostumbrados a manejar definiciones recursivas, como por ejemplo la de la función factorial. El mecanismo de las funciones μ-recursivas es entonces una generalización razonable del tipo habitual de las definiciones recursivas que solemos utilizar, que corresponderían a lo que en el argot se denominan funciones *recursivas primitivas*. Dicha generalización permite en particular definiciones con llamadas anidadas, en las que se utilizan como parámetros los valores de la función sobre otros argumentos (más sencillos), como la que da lugar a la *Función de Ackermann*. De nuevo se probó que las funciones μ-recursivas (o recursivas generales) coinciden con las Turing-computables, lo que en particular nos indica que las aparentemente inofensivas MT son capaces de capturar, no solo el mecanismo básico de recursión primitiva, sino también la recursión general. En relación con ello mencionaremos el hecho de que las funciones recursivas no primitivas, como la citada Función de Ackermann, se caracterizan por su orden de crecimiento absolutamente bestial, pues todas las funciones computables con crecimiento «moderado» son recursivas primitivas.

La Máquina de Turing Universal

Pero siendo importante la presentación de las MT como mecanismo formal natural para capturar la noción de algoritmo, una inesperada nueva vuelta de tuerca vino a dotar al modelo de una potencia enorme, de la que se siguen una pléyade de resultados interesantes, tanto teóricos como prácticos. Turing reflexiona sobre la sencillez del proceso de ejecución de los programas de las MT para darse cuenta de que el mismo es ¡un algoritmo! Una lectura precipitada del aserto anterior nos llevaría a concluir que eso ya lo sabíamos: las MT buscaban formalizar la noción de algoritmo. Pero lo que acabamos de afirmar va mucho más allá: hemos dicho un (único) algoritmo, ¡no uno para cada máquina! Pero si las MT capturan toda la

generalidad de la noción de algoritmo... ¡tiene que haber una única máquina (la *Máquina Universal*) que represente dicho algoritmo... y a fe que la hay!

El lector escéptico podría sugerir que este modesto autor, y Turing antes, estamos haciendo trampa: nuestras MT solo manejan números naturales como datos, y la pretendida Máquina Universal debería manejar dos tipos de datos, la MT a ejecutar y sus propios datos. Pero todo en la descripción de una MT es discreto, y cada una de ellas es finita, y entonces... ¡eureka, podemos codificar con un mero número natural de un modo efectivo cada MT, para luego extraer de ese número la «forma» de la máquina, para finalmente pasar a ejecutarla sobre el resto de los datos que se nos suministren!

Vamos con las implicaciones más importantes de este hecho. A nivel práctico nos está diciendo que si pretendíamos materializar la potencia que nos dan las MT, hasta ahora necesitábamos una máquina física para cada MT, pero ahora nos basta con una... como la que estoy utilizando para componer este artículo: ¡nuestro ordenador (por ejemplo) es la materialización de la Máquina de Turing Universal!, con una única salvaguarda, cada vez las memorias son más y más grandes, pero todavía no infinitas. Pero esto en realidad no importa demasiado: para usar esa memoria infinita necesitaríamos infinito tiempo, y por suerte o desgracia no disponemos de tanto.

Si examinamos la causa principal que convierte a nuestro ordenador en la materialización de una MT universal, y no meramente en una MT más, nos encontramos con la internalización del programa, que ahora reside en la memoria, convirtiéndose así ¡en un dato más! Podría argumentarse que lo que hemos hecho es un mero juego de palabras: ¿qué más da dónde digamos que está el programa? Pero la identificación de este con un dato nos lleva mucho más allá: si rehacemos nuestra formalización centrándonos en la (única) MT universal, ¡los programas de las MT originales dejan de aparecer en la misma, o sea es como si no existieran! Más exactamente, solo hay un programa, que es el de la máquina universal. Si nos vemos como programadores que ejecutamos en nuestro ordenador exclusivamente los programas que escribimos con un determinado lenguaje de programación, vemos fácilmente que la afirmación anterior es estrictamente cierta: el programa en cuestión sería ¡el *compilador* de dicho lenguaje!, que utilizamos para compilar y ejecutar nuestros *programas*, que se «traga» como datos dicho compilador. Si preferimos movernos a un nivel menos abstracto, ese programa único también nos sería muy familiar: se trataría del *Sistema Operativo*, que en efecto controla absolutamente el funcionamiento de nuestra máquina física.

A nivel teórico, el mero manejo de MT que toman MT (arbitrarias) como dato, nos conduce a un resultado tan simple y sencillo de probar, como inesperado y demoledor: no puede haber ninguna MT que sirva para decidir si una MT cualquiera va a parar o no. Obsérvese que lo que estamos haciendo ahora es mostrar una función total perfectamente bien definida que no es computable; antes solo sabíamos que había muchas de ellas, pero no conocíamos ninguna en concreto. O sea, admitida la Tesis de Church, o simplemente identificando computable con Turing-computable: el *Problema de Parada* es *indecidible*.

No detallaré aquí la prueba de este resultado, pero sí quiero destacar la idea clave del mismo, que es la de *autoaplicabilidad*. De ser decidible la parada, contaría-

mos con una máquina T capaz de decidir si una máquina cualquiera M terminará deteniéndose cuando se le suministra la entrada d. En particular, como la máquina M ha quedado codificada por un número, podríamos aplicarla a sí misma, tomando $d = M$, con lo que T nos podría informar sobre si M aplicada a sí misma para. Pues bien, a continuación «trucamos» la terminación de la ejecución de T, de manera que cuando nos dice que M pararía, le añadimos un bucle infinito ¡para que no pare! Obtenemos así una máquina *paradójica* P, que enfrentada a sí misma... ¡para si y solo si no para!

Pero demostrada la indecidibilidad del problema de parada, es posible, y Turing así lo hizo, representar el funcionamiento, y en particular la parada de una MT, por medio de una fórmula de la lógica de primer orden, lo que nos lleva a la *indecidibilidad* de esta última, y a la consiguiente frustración de Hilbert. La técnica para realizar esa codificación dio lugar a los preciosos problemas de *teselación*, que nos presentan curiosos y sorprendentes ejemplos de la sutilidad de la frontera entre lo decidible y lo indecidible. No puedo resistirme a citar aquí la referencia [2], ¡que de hecho acaba de ser reeditada con ocasión del Año Turing!, donde el lector interesado se podrá deleitar con una preciosa y sencilla exposición de estos y otros fundamentos de la algorítmica.

Como hemos indicado en nuestra introducción, tanto en Bletchley Park como posteriormente en Manchester, Turing colaboró en el diseño y realización física de máquinas que de algún modo llevaron a la práctica parte de sus ideas teóricas. Sin embargo, y por diversas razones, sería inapropiado atribuir el mérito del desarrollo de los computadores de propósito general al trabajo previo de Turing sobre la MT universal. Como indicamos antes, Turing se adelantó mucho a su tiempo, tanto que él mismo no pudo intuir las posibles consecuencias prácticas que podía tener su descubrimiento de la MT universal. Y en cuanto a la mayor parte de los pioneros que desarrollaron los primeros ordenadores de propósito general, desde luego Konrad Zuse no tuvo en absoluto ningún contacto con Turing, ni tampoco parece que lo tuviera Aiken (creador de Mark-I), y aunque von Neumann sí que conocía con detalle sus resultados teóricos, e intuía una relación entre los mismos y la posible universalidad de una eventual máquina física, sin embargo no consta que la idea de programa en memoria se haya relacionado de manera directa con los resultados del trabajo original de Turing.

Cierto es que hubiera podido suceder así, y no me cabe duda de que Turing habría previsto al menos, e incluso propuesto a nivel teórico, parte de los desarrollos futuros de la programación que hoy podemos ver como consecuencias de su MT universal, pero desgraciadamente murió demasiado pronto, cuando en particular los limitados medios físicos con los que se contaba para materializar la idea de ordenador hacían casi imposible que se pensara en desarrollos tales. En particular, los primeros ordenadores no sacaban partido del hecho de que el programa estuviera en memoria, pues para empezar no era verdad que físicamente lo estuviera en la memoria interna de acceso más o menos rápido, sino en memorias externas materializadas con lentos dispositivos mecánicos, o a través incluso de interruptores físicos con una capacidad muy limitada. Es con la introducción, bastante posterior, de los lenguajes de alto nivel y los compiladores cuando se saca verdadero partido a la idea de programa

como dato. Constatado este hecho, hoy las aplicaciones van mucho más allá, pues al ver el programa como un dato más, podemos incluso variarlo durante la ejecución, dando lugar a conceptos como la *metaprogramación* y la *reflexividad*, que han abierto la puerta a nuevas metodologías de desarrollo de sistemas.

BLETCHLEY PARK: ALGUNAS NOCIONES PRÁCTICAS SOBRE CRIPTOGRAFÍA VETUSTA

¿Cómo colaboró Turing a reducir las pérdidas de los aliados y subsecuentemente a ganar la Guerra? De nuevo tendremos que sacrificar el detalle buscando la esencia del caso, que radicó en el modo en que funcionaba la máquina Enigma, utilizada por el ejército alemán para cifrar y descifrar sus mensajes durante la guerra. La necesidad de contar con un sistema sencillo y homogéneo, pues se deseaba que desde cualquier punto se pudieran mandar y recibir mensajes, llevó a que la máquina en cuestión (de hechura similar a una máquina de escribir de la época) tuviera que ser portátil, y en consecuencia su algoritmo de cifrado no demasiado complejo. En particular, la primera (y desafortunada para los alemanes) propiedad del mecanismo era la relación uno a uno (letra a letra) entre el texto original y su cifrado: el operario tecleaba cada carácter del texto y en su lugar aparecía una letra del texto cifrado. Ese texto cifrado se retransmitía en abierto, y por tanto cualquiera podía tener acceso al mismo. En particular lo recibía su destinatario, el cual solo tenía que volver a suministrar a la máquina el texto cifrado, para que su versión recodificada no fuera otra que el texto original.

En términos matemáticos, la operación de cifrado es inversa de sí misma (*involución*). Por otra parte, a nivel local sería una función de caracteres a caracteres, si bien la gracia del asunto residía en el hecho de que dicha función variaba en el tiempo, pues dependía de la configuración interna de la máquina, que variaba cada vez que se tecleaba una letra. A grandes rasgos, dicha configuración se basaba en los movimientos sincronizados de las *ruedas dentadas* de Enigma, que a modo de contador de tres cifras (aunque en realidad se tratara de letras del alfabeto), iban rotando cada vez que se pulsaba una tecla. En consecuencia, en un momento dado al teclear una A podía producirse, por ejemplo, una F, pero poco después otra A podría producir una J, consiguiéndose de este modo un falso efecto de caos. El caos es, por supuesto, solo aparente, pues fijada la configuración inicial de la máquina la función de codificación es en efecto una función, de modo que, dado el texto a codificar, su texto codificado está totalmente predeterminado. Para recuperar el texto original basta teclear de nuevo el texto recibido en otra máquina Enigma, asegurándose de recrear la misma configuración inicial de la que se partió al emitirlo.

Obviamente, aquí radica el quid de la cuestión: aunque contáramos con una máquina Enigma y recibiéramos un mensaje cifrado, no podríamos decodificarlo en absoluto, si no dispusiéramos de la configuración inicial de la máquina. Esta venía dada por la posición en que debían ponerse las tres ruedas antes de empezar a transmitir el mensaje. Pero para complicar el tema se disponía de cinco ruedas diferentes, de entre las que se decidía cuáles, y en qué orden, iban a usarse cada día,

junto con un cuadro adicional «cableable» en el que podían insertarse cierto número de conexiones, que alteraban también el funcionamiento de la máquina. Con todo ello, el número de posibles configuraciones era inmenso, haciendo poco probable que diéramos con la configuración oportuna por mero azar. En concreto, se estima que los alemanes utilizaron la máquina con ciertas limitaciones para simplificar su uso, que dejaban en 10^{23} el número de configuraciones posibles, frente a las $3 \cdot 10^{114}$ que hubiera alcanzado el uso no limitado de la potencia del artefacto.

Cada día la combinación de ruedas que se utilizarían y el cableado del mecanismo adicional se establecía según indicaban los cuadernos de claves distribuidos mensualmente entre los operarios. La configuración inicial antes de retransmitir cada mensaje se completaba con una parte variable, que indicaba simplemente cómo debían posicionarse las tres ruedas al comienzo. El emisor decidía libremente dicha posición (indicando las tres letras de las tres ruedas, que debían verse en los visores de las mismas) y la transmitía en el preámbulo anterior a cada transmisión. Por supuesto, también utilizaba la máquina para hacerlo, por lo que la información iba codificada, utilizando en este caso una configuración inicial incluida en el cuaderno de claves, que de este modo quedaba muy poco expuesta a los probables esfuerzos del enemigo para descifrar los mensajes, pues solo se utilizaba para retransmitir estos mini-mensajes.

Suponiendo que los cuadernos de claves fueran inviolables (en alguna rara ocasión dejaron de serlo al incumplir el capitán de algún submarino la severísima orden de destruirlos, junto con la correspondiente máquina Enigma, antes de abandonar la nave), para intervenir con éxito las conversaciones se necesitaba contar, para empezar, con una copia de la máquina, y cuántos más mensajes del día mejor. Obviamente, mucho mejor era contar con muchas de tales máquinas, pues de este modo se podría poner en funcionamiento lo que hoy en día calificaríamos como ejemplo de *procesamiento paralelo*. El mecanismo básico de búsqueda de la configuración diaria, contando con la cual era casi inmediato decodificar todos los mensajes de ese día, era la burda *prueba y error*. Se trataba de buscar, y además rápidamente, pues aunque termináramos adivinando la clave de hoy, no serviría para nada mañana, una aguja en un gigantesco pajar. La tarea, imposible en principio, se volvió factible gracias a errores de diverso tipo cometidos por los alemanes, tanto en el diseño como en el uso de su mecanismo. A partir de los mismos, Turing y todo el equipo de Bletchley Park encontraron el imán de gran potencia capaz de encontrar la aguja, simplemente haciéndola salir a la superficie.

Comencemos por los errores de los alemanes. En relación al diseño, el limitarse a una codificación letra a letra es quizás lo más grave, aunque podría argüirse que la simplicidad de la máquina exigía cargar con esta restricción. Mucho más difícil de justificar fue una segunda propiedad del mecanismo: al teclear cada letra en un momento dado podía producirse cualquier carácter ¡menos precisamente el tecleado! La justificación técnica era que el carácter *involutivo* del proceso era más fácil de lograr agrupando las letras por pares, sin permitir nunca pares degenerados formados por una única letra repetida dos veces. Podría pensarse que no produciéndose nunca el carácter correcto estamos dificultando más dar con él. Pero si meditamos más pausadamente sobre ello, nos damos cuenta de que la situación es justamente

la contraria: por «paso al complementario», si fuéramos capaces de ver simultáneamente todos los valores con los que se codifica una determinada letra ¡obtendríamos de inmediato la misma!

Un tercer error vino precisamente de la debilidad de los sistemas de transmisión y recepción en la época. Como indicamos antes, en el preámbulo de cada mensaje el emisor enviaba una clave local inventada de tres letras, que iba a utilizar para codificar el mismo. Pongamos que escogiera XYZ. Al teclear XYZ partiendo de la configuración inicial diaria, se producía por ejemplo AJF, que decodificado produciría la deseada clave XYZ. Pero si se producía un error de transmisión, lo que era frecuente en la época, podía recibirse solo AJ, o quizás AJG, que decodificado nos daría, por ejemplo, XYD. El receptor cargaría entonces esa clave para decodificar el mensaje original, para encontrarse con un texto absurdo. La solución a este problema pasó por el uso de la *tolerancia a fallos*, generada por la *redundancia*: para enviar XYZ codificado no enviamos solo XYZ, sino XYZXYZ.

Si todo va bien, ahora al decodificar la clave la encontramos dos veces repetida, y nos fiamos de ella. En caso contrario, sabemos que ha habido un error y tratamos de informar al emisor para que repita el envío. La gravedad de este error de diseño proviene, no solo del hecho de que sepamos que al suministrar una clave codificada a la (buscada) configuración inicial diaria, esta ha de producirla dos veces repetida, sino sobre todo al hecho de que recibimos información sobre la codificación de esas tres letras en dos momentos casi consecutivos. Como quiera que se disponía de la máquina Enigma, y en consecuencia se conocía perfectamente el mecanismo periódico con el que las ruedas modificaban (lentamente) la configuración interna de la misma, contar con esas claves doblemente codificadas era todo un filón del cual tratar de extraer la deseada configuración diaria. En efecto, fue por este agujero de seguridad por donde se empezó a atacar a Enigma. De hecho, posteriormente los propios alemanes fueron conscientes de esa debilidad, y pasaron a utilizar en sus comunicaciones con la flota un doble sistema de codificación de las claves locales, que combinaba el uso de Enigma con unas complejas tablas de preproceso de dichas claves, anulando así dicha grave debilidad.

Turing y su equipo tuvieron que centrarse entonces en los textos de los mensajes originales (que obviamente solo se tenían cifrados) para tratar de buscar la configuración inicial de la máquina. Al efecto, junto con la debilidad producida a nivel local por la no autocodificación de ningún carácter, se usó la no aleatoriedad del lenguaje natural. En concreto, la búsqueda por prueba y error que mencionamos antes se basaba en el uso simultáneo de una serie de réplicas de Enigma a las que se hacía partir de cada una de sus configuraciones iniciales posibles, suministrándolas el texto de un mensaje codificado recibido. Si fuéramos capaces de leer todas y cada una de las frases decodificadas, una, y con alta probabilidad solo una de ellas, tendría sentido, habiendo dado por tanto con la configuración inicial de ese día. Una vez que se contaba con esa batería de réplicas capaz de generar todas las posibilidades en un tiempo razonable, el problema se reducía a filtrar ese único texto correcto.

A partir del estudio detallado del lenguaje natural como lenguaje formal se encontraron diversos testigos capaces de guiar esa búsqueda. Citaremos aquí solo el más sencillo de ellos: si se ponen uno bajo el otro dos textos cualesquiera escritos

en alemán (obviamente algo similar ocurriría en inglés, y por su configuración aún más en castellano), la probabilidad de que coincida la misma letra en cada casilla es 1/17, muy lejos del 1/26 correspondiente a dos textos aleatorios. Dada la forma en que trabajaba Enigma, dicha alta probabilidad permanece invariante si consideramos dos textos codificados a partir de la misma configuración inicial. Obviamente, algoritmos como este, basados en la estadística, no podían aspirar a seleccionar directamente la (única) solución posible, sino que su objeto era una drástica criba que seleccionara una cantidad manejable de candidatos, entre los cuales se buscaba después utilizando «procesadores humanos» que las iban comprobando una a una en paralelo, para tratar de dar cuanto antes con la solución buscada.

Turing empleó toda su perspicacia para diseñar algoritmos astutos que guiaran la búsqueda de las configuraciones diarias basadas en el comportamiento periódico de las ruedas dentadas de Enigma. Hoy en día resulta evidente que sus propuestas estaban plagadas de resultados relacionados con determinados *grupos* encubiertos, constituyendo por tanto un magnífico ejemplo de aplicación práctica ingenua de una sofisticada teoría matemática que tardaría aún en desarrollarse.

Inteligencia Artificial

Turing fue también absoluto pionero en lo que hoy denominamos *Inteligencia Artificial*, campo que ciertamente es de difícil determinación, pero al que trataré de referirme aquí en base a las definiciones populares más extendidas. Según estas, se han venido englobando en la IA aquellas facetas de la Informática en las que de alguna manera intervienen cuestiones relacionadas con la comprensión y la inteligencia humana más profunda, buscando emular el modo en que los humanos utilizamos nuestro intelecto y los recursos físicos a su disposición (léase los sentidos) para aprender y resolver problemas.

Se trata, por tanto, de un campo enormemente amplio, dentro del cual Turing realizó diversas contribuciones visionarias, la mayoría de las cuales quedaron en un nivel bastante embrionario. Solo fueron retomadas en parte mucho más tarde, cuando se contó con los medios necesarios para desarrollar en profundidad los correspondientes campos, pero aun siendo cierto que en la práctica su impacto directo fue muy limitado, reexaminadas ahora contribuyen a magnificar la figura de genio irrepetible que fue Turing.

A pesar de la motivación absolutamente pragmática que tuvo su trabajo en Bletchley Park, es partiendo del mismo como Turing arma sus propuestas para la mecanización del proceso de resolución de problemas (generales), apoyado en una definición del espacio de soluciones y la búsqueda heurística. Se trata de intentar avanzar cuanto antes en las direcciones más prometedoras, evitando perder tiempo en la exploración de ramas laterales, menos propensas a conducirnos pronto a una solución. Turing simplemente trató de reflejar de esta manera el modo natural de afrontar y resolver problemas no triviales por parte de un ser humano.

Hoy en día podríamos ver en los algoritmos de *branch and bound* una primera concreción de sus ideas, mientras, a un nivel más general, los *sistemas basados en*

reglas son un mecanismo para plasmar los procesos de decisión, por medio de los cuales se hace efectiva la aplicación de nuestra inteligencia (o la de una máquina que nos emule).

Ciertamente, resulta sorprendente que Turing concibiera estas técnicas generales de resolución automática de problemas sin contar, ni por asomo, con máquinas con las que poder aplicarlas a problemas que justificaran su uso. Muy probablemente, esta fue la causa por la que sus ideas seminales sobre el tema no fueran valoradas ni desarrolladas por otros científicos, hasta mucho tiempo después.

Turing también fue consciente de que en ese modo de resolver problemas de manera efectiva, el aprendizaje es fundamental. A través del aprendizaje pulimos nuestro proceso de búsqueda haciéndolo más eficiente. Por otra parte, en la automatización inconsciente de nuestra forma de proceder vemos reflejado el hecho de que lo hacemos así porque «sabemos» que esa es la forma correcta de hacerlo, aunque a partir del momento en que lo aprendemos ya no retomamos la justificación de nuestro conocimiento, simplemente lo aplicamos a sabiendas de que funcionará.

En lo anterior hemos visto ejemplos de lo que podríamos llamar abstracción de la inteligencia humana, en los que buscamos mecanismos que, utilizando cierta forma de razonamientos lógicos (automatizados de una manera consciente), consigan simular la mente humana. Pero Turing también exploró el punto de vista complementario, en el que el fin deja paso a los medios, no buscando solo emular la capacidad abstracta (inteligencia), sino el propio recurso físico que la genera (el cerebro). Tan pronto como en 1948, Turing sugiere el uso de *Redes Neuronales* que simulen las presentes en nuestro cerebro, buscando que las mismas realicen de un modo inconsciente los procesos inteligentes que habitualmente realiza nuestro cerebro.

Una vez más, Turing relaciona sus propios trabajos aparentemente faltos de relación, sugiriendo que nuestro cerebro es (al menos) una MT universal. En consecuencia, estaría a la espera de que le suministremos (como datos, o sea ¡aprendiendo!) los programas que podrá ejecutar. A través de la experiencia dichos programas se irán modificando con la esperanza de hacerlos mejores. Todo el proceso estaría controlado por algún mecanismo de organización (¿los *genes* en el cuerpo humano?, ¿el *Sistema Operativo* en las máquinas?). ¡Es increíble cómo, casi sin querer, Turing fue soltando ideas que el mismo consideraba todavía demasiado vagas, pero que intuía que pronto serían extraordinariamente útiles!

El Test de Turing

Esta es sin duda una de las contribuciones por las que más se conoce a Turing, quizás más que nada por su originalidad a la hora de lanzar una propuesta sobre un posible modo de juzgar cuándo una máquina ha conseguido simular la inteligencia humana. Una vez más, lo más sorprendente de su propuesta fue el momento en que fue lanzada, que nos hace sospechar que Turing intuía que, una vez concebida la Máquina Universal y realizadas físicamente estas a través de los primeros computadores, no pasaría mucho tiempo antes de que estos evolucionaran de manera que sus limitaciones físicas, en lo relativo a capacidad de memoria y velocidad de pro-

cesamiento, permitieran plantearse la cuestión de hasta qué punto un ordenador ha devenido en un ser inteligente.

Además del carácter innovador y visionario de su propuesta, las otras cualidades por las que permanece siendo famosa y respetada (en parte) en los ambientes científicos, son su tremenda sencillez y el modo aparentemente informal con que se captura la potencial inteligencia humana de una máquina. Para más inri, la propuesta nace de un juego de imitación relativamente popular en la época, que trataba de delimitar las diferencias innatas entre hombres y mujeres. Se trataba de discernir entre dos interlocutores, a los que no se podía ver, quién de ellos era un hombre y quién una mujer, con uno de ellos tratando de alargar el juego a base de confundir al jugador con sus respuestas. El Test de Turing cambia al interlocutor humano que trata de engañar al jugador por una máquina, y el objetivo del jugador es descubrir a esta, que se vale de su inteligencia para tratar de confundirlo.

Una máquina pasaría el Test de Turing si tras ser interpelados ambos interlocutores por un espacio máximo de cinco minutos, el jugador solo acierta la identidad de la máquina con una frecuencia similar en términos estadísticos de la que se daría si lanzara su respuesta al azar. Se entiende entonces que la máquina ha mostrado una habilidad análoga a la de un humano para confundir al jugador, y con ello ha puesto de manifiesto su inteligencia.

Son por supuesto múltiples las pegas que podemos plantear al Test de Turing para medir la inteligencia humana de una máquina. Hoy en día parece existir un claro consenso en que, en el mejor de los casos, se estaría evaluando una determinada faceta de esa inteligencia natural. Pero incluso admitido esto, podríamos dar argumentos para descalificar por completo la asignación de inteligencia humana a las máquinas que lleguen a pasar el test. La objeción no la hace solo este autor, ni siquiera algún investigador reciente, sino gente tan prestigiosa y veterana en el área como John McCarthy, padre oficial de la IA, y Claude Shannon, también pionero, en este caso de la *Teoría de la Información*.

Los dos de manera conjunta, ya en 1956, argumentaban que tras estudiar la manera en que los jugadores que aplican el Test hacen sus preguntas, un paciente programador podría recopilar todas las preguntas esperables junto a la respuesta más astuta, de cara a confundir al jugador. Entonces, la máquina programada por él se limitaría a buscar en su base de datos las preguntas que se le hicieran para dar la correspondiente respuesta, sin pensar ni razonar en absoluto, por lo que resultaría francamente discutible calificar de inteligente a una máquina que no ha utilizado (ni simulado) proceso alguno de razonamiento. Obviamente, junto con el fondo que constituye la respuesta «adecuada», también se debería contar con la forma en que esta debe darse, tras «pensar» el tiempo necesario, y adornándola cuando proceda con dudas, suposiciones o titubeos, como haría un ser humano.

Como dije en mi introducción, personalmente no estoy demasiado interesado en esta aportación de Turing, pero resulta imprescindible incluirla con la debida extensión en toda reseña de su obra. Eso sí, como desarrollo personal de la crítica anterior aprovecharé para hacer una digresión que me parece interesante, que entronca también con las características de las Máquinas de Turing y la noción de problema decidible o función computable que de ellas se infiere, así como con los

mecanismos de búsqueda de soluciones de los que hablamos al principio de esta sección. De alguna manera entendemos que la inteligencia natural normal (pues Turing se contentaba con emular a un humano normal, sin pensar en las capacidades de los humanos más inteligentes) se manifiesta al resolver problemas de cierta dificultad, que exigen la capacidad de razonar. Por tanto, no estamos pensando, en absoluto, en realizar largos pero simples cálculos, que ya las máquinas precursoras con las que trabajó Turing eran capaces de hacer en un tiempo impensable para un ser humano. La cuestión es entonces medir la dificultad de esos problemas.

Volvamos a considerar como problemas la evaluación de funciones $f : \mathbb{N} \to \mathbb{N}$. Hoy en día contamos con una definición oficial de *complejidad* de tales funciones, que podemos dar utilizando MT. A grandes rasgos, dicha complejidad viene dada por el menor tiempo que tardará una MT que compute dicha función, en calcular el valor de la misma en un punto cualquiera. Naturalmente, se espera que ese tiempo crezca en función de lo grande que sea el punto en cuestión, por lo que hablamos de costes logarítmicos, lineales, exponenciales, etc. Una cuestión interesante es que la noción de computable es independiente de la base con la que representemos los naturales, que puede ser incluso 1. Sin embargo, a la hora de definir la complejidad no importa la base con la que se trabaje, siempre que esta no sea la unaria, que nos llevaría a complejidades exponencialmente mayores. En efecto, es gracias a que unas cifras tienen «más peso» que otras, que podemos conseguir una mayor rapidez de cálculo, esencialmente porque de ese modo el propio tamaño de los números se hace logarítmico. Ciertamente, no podemos acusar a Turing de no haber sido también pionero en el campo de la *Complejidad*, pero sí resulta curioso que habiéndose embarcado en cuestiones de mucha mayor enjundia, y buscando sus trabajos una incontestable aplicación práctica, no entrara en la cuestión de la impracticabilidad de los algoritmos que potencialmente terminarían por calcular una determinada función, pero solo tras un tiempo verdaderamente astronómico.

Volviendo al argumento de Shannon y McCarthy, lo que haría el programador a la hora de facilitar su programa a la máquina perspicaz, en términos de cálculo de una función, sería facilitar la tabla de la misma. Buscando en dicha tabla obtenemos cada valor de la función en un tiempo mínimo que no depende en absoluto de la «dificultad» de la función tabulada. ¿Nos acabamos de cargar la teoría de la complejidad? Pues no, ya que la definición de la complejidad necesita que el dominio de las funciones que se consideren sea infinito, como sucede con $\mathbb{N}$. Por contra, el programa ha de ser finito, y por tanto la hipotética tabla sería también finita, por lo que jamás podría incluir todos los valores posibles. En consecuencia, podríamos contraargumentar a Shannon y McCarthy diciéndoles que el jugador que aplica el Test siempre puede preguntar algo nuevo. Pero es más que probable que ellos también se habrían defendido considerando poco realista tamaña creatividad del encuestador.

Volviendo al caso de la evaluación de funciones, si necesitáramos estas para algo útil, es bastante probable que solo tengamos que aplicarlas a datos pequeños, pero entonces... ¿podríamos tirar de tabla y desaparecería por completo la noción de complejidad? Pues no, la complejidad inherente de las funciones se mantiene, y por tanto hay que lidiar con ella, tanto a nivel formal como en la práctica. A nivel formal, si vamos a tener que manejar una cantidad finita, pero muy grande, de valores, para

dar respuesta a nuestros accesos razonables, necesitaríamos facilitar la tabla, y el único lugar donde podemos hacerlo es el propio programa, o la cinta si permitimos que la configuración inicial de la máquina sea una cualquiera que aporte información finita. En consecuencia, si limitamos el tamaño del programa y/o la ocupación inicial de la cinta, recuperaríamos la noción natural de complejidad, al no poder contar con ninguna tabla suficientemente grande. Vemos así capturada la típica contraposición *tiempo/memoria*, a tener en cuenta cuando se realizan valoraciones «realistas» de la complejidad.

Por otra parte, si vamos a permitir el uso de una tabla, esta tendrá que haber sido calculada en algún momento, y si la función es compleja, habrá sido necesario invertir muchos recursos en ello. Surge así la técnica de programación con *precondicionamiento*, en la que en primer lugar se distingue una fase de preparación probablemente costosa, que sin embargo se ejecuta solo una vez, para luego dejar sus resultados en memoria, o en una base de datos si el uso es a más largo plazo. En lo sucesivo tendríamos la fase de uso, en la que se realizan múltiples accesos baratos a la tabla elaborada. Es evidente que la noción de complejidad práctica en tal caso es bastante compleja, teniendo un cariz *amortizado*: la inversión muy costosa de la fase de preparación se verá amortizada en la medida en que buena parte de la información sea utilizada repetidamente en el futuro. Resulta inmediato trasladar este proceso a nuestra vida: durante nuestra niñez y juventud nos educamos aprendiendo muchas cosas a base de esfuerzo, que luego veremos recompensado en la medida en que lo aprendido haya merecido la pena, quedando formados para afrontar los retos y dificultades durante nuestra vida laboral, y por extensión durante todo el resto de nuestra existencia.

Espero se me disculpe esta digresión, en la que he expuesto una técnica de programación relativamente sofisticada, pero de evidente aplicación práctica, y he comentando la noción de complejidad, todo ello motivado por una propuesta de Turing, que aparentemente no tenía nada que ver, y la respuesta que dos ilustres coetáneos le dieron en su día. Y para terminar esta sección, regreso a la obra de Turing, recomendando la lectura de la charla *Can Digital Computers Think?* emitida por la BBC en 1951, y la de la preciosa discusión con sus colegas Richard Braithwaite, Geoffrey Jefferson y Max Newman, *Can Automatic Calculating Machines Be Said To Think?*, también radiada un año después. Una vez más, el preclaro Turing nos deja impresionados con lo acertado de sus predicciones en relación al desarrollo de la Informática, a pesar de los logros relativamente modestos que habían conseguido las costosísimas, enormes y poco fiables muestras de las que se disponía por entonces.

Vida Artificial, Morfogénesis y Algoritmos Genéticos

En la última fase de su vida, Turing vuelve a encontrar otra faceta de las Matemáticas y la Computación, completamente diferente en apariencia a las anteriores que había cultivado, pues el motor conductor de la misma es tan exótico como la *Biología*. Turing analiza la *Vida* como Naturaleza en *Evolución*, y aunque sabe que la *Química* será la responsable natural última de la misma, a nivel abstracto sospecha y propone que es la Matemática la que mueve los hilos. Centrándose en el

crecimiento, encuentra a la *Sucesión de Fibonacci* en todas partes, regulando a nivel discreto múltiples procesos de crecimiento, con la espiral derivada del *número áureo* como manifestación continua del fenómeno.

Pero si las manifestaciones de ese crecimiento son continuas, la causa última de esos patrones discretos ha de encontrarse en las *ecuaciones* que regulen las reacciones químicas a través de las cuales se lleva a cabo. Nos encontramos con las *reacciones de reacción-difusión*, que de alguna manera se autorregulan a través de la presencia de un *catalizador*, cuya concentración produce notables cambios en el proceso de evolución a nivel local. En principio, la simetría en el proceso de crecimiento debería producir cuerpos esféricos completamente regulares, pero la presencia de singularidades causa dramáticos efectos que dan lugar efectos aparentemente caprichosos: rayas y lunares en la piel, pero también a un nivel mucho más operativo, los distintos tejidos y órganos que conforman un ser vivo complejo, que se desarrollan cuando los genes responsables aparecen como catalizadores.

Las ecuaciones diferenciales que hay detrás de estos procesos son ciertamente complejas, algo que en cierta forma cabría esperar, pues la excesiva simplicidad tiende a no producir singularidades. Turing ya pronosticó que el uso de los ordenadores sería imprescindible para examinar y simular los modelos resultantes. A lo que Turing ya no pudo llegar fue a desenterrar la sencillez que, pese a todo, estaba oculta tras estos aparentemente complejos procesos, llenos de singularidades. Pero de nuevo podemos relacionar esa sencillez con la universalidad de las MT. Como indicamos antes, las MT tienen exactamente la misma potencia que las funciones recursivas, pero la prueba de este resultado dista de ser inmediata, pues exige el despliegue de los medios mediante los cuales la recursión puede ser simulada por un algoritmo iterativo que usa *pilas* para almacenar los argumentos de las llamadas (anidadas) que estén pendientes en cada momento. Es bien sabido, por quien esté bien ducho en programación, y maneje con suficiente soltura tanto la recursión como la iteración, que hay algoritmos describibles de forma muy sencilla utilizando recursión, pero cuya traducción iterativa resulta muy difícil de seguir.

Pues esta es justo la situación que se da en el *Conjunto de Mandelbrot*, no descubierto por este investigador hasta 1979, dando paso al estudio de los *fractales* íntimamente ligados con las susodichas ecuaciones no lineales y sus soluciones aparentemente caprichosas y muy complicadas, que sin embargo pueden ser generadas utilizando algoritmos recursivos cuya sencillez nos deja impresionados. Obviamente, Turing no llegó ni a intuir esa simetría a escala repetitiva presente en estos fenómenos a todos los niveles, si bien los efectos a gran escala, con la omnipresencia de la Sucesión de Fibonacci, sí que le eran conocidos. Si se me permite un exceso de simplificación, es introduciendo pequeñas perturbaciones en esos sistemas plenos de simetría como se producen los modelos de la morfogénesis. Turing utilizó en su artículo todo el arsenal sobre ecuaciones diferenciales que conocía y consideró procedente, y usó los ordenadores a su disposición para obtener resultados numéricos que aproximaban las soluciones de las mismas.

La impenitente manía de Turing de relacionarlo todo, fuera de un modo consciente o inconsciente, le llevó a proponer lo que bien pudo ser el primer ejemplo de *algoritmo genético*. La idea es que las singularidades caprichosas que están in-

volucradas en los procesos de crecimiento producen mutaciones de las cuales solo sobreviven los especímenes mejor preparados, dando así lugar a la *evolución*. Como vimos en la sección dedicada a la IA, la resolución general de problemas exige de reglas de búsqueda que tratan de encontrar la solución adecuada en el mínimo tiempo posible. En muchas ocasiones, la dificultad de un problema hace complicado intuir cuál debería ser el proceso de búsqueda. Sin embargo, sí que podemos facilitar un amplio abanico de alternativas, entre las cuales nos gustaría poder seleccionar la más adecuada. El marco de los *juegos* resulta ideal para explicar de forma sencilla estas ideas. Fue precisamente al desarrollar un prototipo de jugador de *Ajedrez* en su ensayo *Chess*, cuando Turing realizó sus aportaciones pioneras en este campo.

Supongamos que, a la hora de jugar, disponemos de varias estrategias que intuimos razonablemente interesantes, siendo difícil escoger a priori entre ellas la más efectiva. La solución que aporta el símil de la evolución no puede ser más sencilla: organicemos un campeonato virtual masivo en el que nuestros jugadores, cada uno provisto de su propia estrategia, compiten tantas veces como sea posible. Examinamos los jugadores que van quedando mejor clasificados y eliminamos a los perdedores, repitiendo después el proceso, introduciendo quizás pequeñas variantes de los jugadores seleccionados, para así seguir contando con suficientes contendientes y además ir perfeccionando a estos, que serán cada vez más complejos, y por ello esperamos que más efectivos. Después... solo puede quedar uno, el campeón, el único superviviente, al que múltiples *mutaciones* habrán hecho potencialmente invencible.

Por supuesto que, en cierta medida, el buen funcionamiento de los algoritmos genéticos a la hora de resolver un determinado problema puede llegar a resultar hasta desconcertante al constatarlo en efecto, pero no ser capaces de saber explicar por qué funcionan bien. El símil deportivo, o si se prefiere el aprendizaje académico, nos ofrecen sendos ejemplos de aplicación de estos principios, que Turing sin duda consideraba casi imprescindibles a la hora de programar (ciertos) sistemas complejos. Ello no contradecía, sino completaba adecuadamente, el hecho de que, de cara a resolver problemas más sencillos, Turing utilizó y hubiera propugnado algoritmos más convencionales, cuya corrección hubiera podido ser demostrada formalmente, y no meramente comprobada en la práctica.

El campeón de tenis se entrena diariamente perfeccionando golpes y técnicas, con la esperanza de que, al llegar al campeonato, los sabrá ir utilizando a lo largo de un partido, muchas veces de forma intuitiva, no planificada, confiando en que su preparación (aprendizaje) y la pericia innata (algoritmo de selección utilizado de forma inconsciente) le lleven a la victoria. De la misma manera, nuestros alumnos de matemáticas van enfrentándose a problemas, adquiriendo técnicas de demostración generales, que después utilizan de forma inconsciente, para eventualmente llegar a la demostración de teoremas complejos, cuando el adecuado aprendizaje se ha unido con la imprescindible predisposición, como sin duda sucedió un día con Turing...

Epílogo con citas bibliográficas

Como indiqué, he basado mi exposición en la selección realizada y comentada por B. Jack Copeland [1]. Como se detalla en la revisión de esta obra, publicada en un

número especial de las *Notices of the AMS* [4] dedicado a Turing, durante su carrera Turing publicó bastantes más trabajos, y por otra parte alguno de los presentados en [1] no llegó a ver la luz durante su vida, e incluso es probable que su autor no lo considerara listo para su publicación. Elsevier publicó unas Obras Completas en cuatro caros volúmenes [5, 6, 7, 8], que probablemente debido a ello tuvieron muy poca difusión. El tercer volumen dedicado a las *Matemáticas Puras* contiene trabajos sobre distintas ramas clásicas de la matemática. Se trata de trabajos mucho menos conocidos y de menor impacto, pero que ponen de manifiesto la amplitud de los conocimientos matemáticos de Turing, que de alguna manera fueron utilizados como base en sus trabajos de mayor impacto, destacando la facilidad con la que aplicó resultados de distintas ramas, en campos aparentemente totalmente desconectados con ellos, dando así prueba de su genialidad.

Andrew Hodges publicó *la* biografía por antonomasia de Turing, *Alan Turing: the enigma* [3], en la que, junto a una descripción detallada de las distintas fases de su vida, se enumeran y comentan sus trabajos más importantes. La página web `http://www.turing.org.uk/turing/index.html` que le dedica Hodges a Turing es también digna de ser visitada, y contiene en particular una magnífica página sobre la bibliografía de Turing. Y como quiera que me parece conveniente que el lector interesado acuda tanto a la página anterior como a la dedicada a la celebración de su centenario, permítaseme que agote ya mi propia lista de referencias, seguro de que quien lo desee encontrará allí con facilidad caminos para proseguir su exploración sobre la figura de uno de los más importantes matemáticos del siglo XX.

Sin embargo, su obra ha sido muy poco conocida por los matemáticos «clásicos» de nuestra era, al menos hasta la fecha, muy probablemente porque los campos en los que centró sus trabajos quedan en buena medida dentro de lo que hoy podríamos catalogar como *Teoría de la Programación*, en la frontera por tanto entre la Matemática y la Informática, y todos los terrenos fronterizos son, en muchos casos, difíciles de cultivar, fundamentalmente debido a que quienes nacen y viven en la frontera a menudo son incomprendidos, e incluso tratados como ajenos, a ambos lados de la misma.

Confío en que esta modesta aportación ayude a que en el país de la Matemática, a cuyos pobladores va dirigida la publicación en la que aparece, se conozca mejor y se acepte como lugareño distinguido a Turing, y sobre todo crezca el interés hacia los temas a los que dedicó su obra, encontrando en los mismos la justificación para enaltecer la figura de quien sin duda fue un genio de la ciencia.

Y como quiera que de bien nacidos es ser agradecidos, desde aquí agradezco las dos invitaciones que recibí en su momento, a dar una conferencia sobre Turing y su obra, en el Seminario de Historia de las Matemáticas de mi Facultad, y en el de

Historia de la Ciencia de mi Universidad. El trabajo invertido en la presentación de ambas facilitó enormemente la redacción del presente artículo. Con Luca Aceto, de la Universidad de Reykjavik, intercambié material con motivo de esas presentaciones y otras que hizo él en su institución, tras nuestras respectivas visitas de intercambio en 2010. Finalmente, Ignacio Fábregas y David Romero me ayudaron a preparar las transparencias utilizadas en esas charlas, y el archivo LaTeX de este artículo. Por último, vaya mi más sincero agradecimiento al revisor anónimo, y a la menos anónima Yolanda Ortega, que tras sendas lecturas obviamente muy concienzudas, aportaron diversas sugerencias muy acertadas, que espero yo haya sabido seguir, mejorando así el resultado final que aquí presento.

Referencias

[1] B. J. Copeland (editor), *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life, plus The Secrets of Enigma*, Oxford Univ. Press, 2004.

[2] D. Harel y Y. Feldman, *Algorithmics: The Spirit of Computing*, Addison-Wesley, 3.ª edición, 2004.

[3] A. Hodges, *Alan Turing: the Enigma*, Burnett Books, London, UK, 1983.

[4] A. Hodges, *The Essential Turing*–A book review, *Notices of the Amer. Math. Soc.* **53** (2006), 1190–1199.

[5] A. M. Turing, *Collected Works of A. M. Turing. Mechanical intelligence*, Elsevier, 1992.

[6] A. M. Turing, *Collected Works of A. M. Turing. Morphogenesis*, Elsevier, 1992.

[7] A. M. Turing, *Collected Works of A. M. Turing. Pure Mathematics*, Elsevier, 1992.

[8] A. M. Turing, *Collected Works of A. M. Turing. Mathematical Logic*, Elsevier, 2001.

David de Frutos Escrig, Dpto. de Sistemas Informáticos y Computación, Fac. CC. Matemáticas, Universidad Complutense de Madrid
Correo electrónico: defrutos@sip.ucm.es