Capítulo 3

COMPUTACIÓN COMPLETA

Wagner's music is better than it sounds,

Truth is the most valuable thing we have. Let us economize it,

Giving up smoking is the easiest thing in the world. I know because I've done it thousands of times,

The report of my death was an exaggeration,

Mark Twain

No one goes there nowadays, it's too crowded,

Yogui Berra

Las citas anteriores son no computables, porque contienen contradicciones internas en mayor o menor grado. Son autorreferencias que se niegan a sí mismas y enseguida hablaremos más sobre este tema. Sin embargo, nos hacen reír. Es más, el acto de entenderlas y reírse es un buen indicador de inteligencia.

Entonces ¿qué tiene la computación completa que es tan buena y tan mala a la vez, para lograr inteligencia? Lo veremos en el capítulo del mismo nombre. En este nos limitaremos a entender qué significa computación completa, a la manera de Turing. Comenzaremos por las estructuras computacionales más básicas, los grafos, y usando un tipo particular explicaremos que la computación universal es muy sencilla de lograr y que la propiedad que mejor la caracteriza es que puede simular cualquier otra máquina de cómputo universal. Terminaremos con las limitaciones que la propia computación de Turing se autoimpone. Sí. Has oído bien. Para lograr exactitud en el cómputo inevitablemente emergen tres limitaciones fundamentales respecto a lo que se puede llegar a computar.

Grafos

Los grafos son una estructura matemática simple que sirve para modelar casi cualquier cosa en sistemas discretos (en sistemas continuos es de poca utilidad). Un grafo tiene nodos y arcos que unen nodos, y puede ser orientado o no orientado, en función de cómo sean los arcos (Figura 34). Los grafos como rama de las matemáticas nacieron con el problema de los puentes de Königsberg, resuelto por Euler en 1736. En los últimos años esta rama ha recibido un nuevo impulso debido a las múltiples aplicaciones para modelar y entender redes de muy diversa índole, incluyendo los contactos en las redes sociales virtuales (Twitter, Facebook, G+, etc.), las redes de citaciones de artículos en revistas, las conexiones por avión entre aeropuertos, las redes de energía eléctrica, de agua, de gas o de teléfono, las redes ecológicas con predadores y presas, las redes de conexiones entre virus y sus proteínas objetivo, entre otras. Desde entonces ha recibido el nuevo nombre de *Network Science* (Ciencia de Redes) y si estás interesado en profundizar en este tema recomiendo leer los trabajos fundacionales de Mark Newman, Albert-László Barabási y Duncan J. Watts, que a su vez se basaron principalmente en las investigaciones de Erdös y Rényi de 1960 sobre grafos aleatorios. La aleatoriedad tiene una razón muy importante de ser a lo largo de todo este libro, y aquí la vemos de nuevo: la emergencia de propiedades nuevas en sistemas complejos debe ocurrir sin que haya una estructura *ad hoc* subyacente altamente improbable, pues de otro modo caeríamos en la mal llamada "filosofía del diseño inteligente"[30]. Dicho con otras palabras: si hay fenómenos que se dan con certeza o con alta probabilidad a partir de condiciones iniciales aleatorias, entonces estamos en el camino correcto para entender la emergencia de la complejidad.

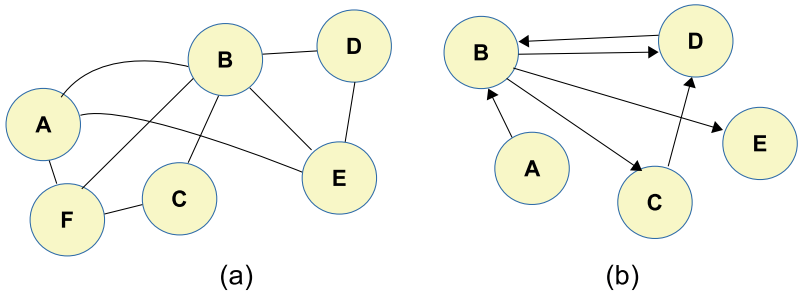
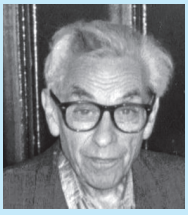


Figura 34. (a) Grafo no orientado con 6 nodos y 9 arcos; (b) Grafo orientado con 5 nodos y 6 arcos

30 Pues realmente no es una filosofía sino la base de la religión. Ver una crítica mordaz en *Mackey* (2008).

Algunas definiciones: los arcos pueden ser orientados (dirigidos) o no orientados (bidireccionales). El grado de un nodo es la cantidad de arcos que llegan o salen de él. Se puede definir el grado promedio de todo el grafo. La longitud de camino entre dos nodos es el mínimo número de arcos que hay que atravesar para llegar de uno al otro, y de ahí se puede calcular la longitud de camino promedio de todo el grafo. El diámetro de un grafo es la longitud de camino más grande que haya entre dos nodos cualesquiera. Si se considera la longitud de camino de un nodo respecto a todos los demás, su valor mayor es el radio de ese nodo; y entonces el centro del grafo es el nodo cuyo radio sea menor. Se puede asignar un grado de centralidad para cada nodo, en función de la diferencia entre su radio y el radio del centro del grafo. Hay otra forma de definir la centralidad de un nodo, que es la centralidad intermediaria y se calcula contando cuantos caminos más cortos entre cualquier par de nodos, pasan por el nodo en cuestión. El coeficiente de agrupamiento del grafo nos dice cuántos nodos forman subgrafos triangulares.



Fuente: Topsy Kretts (2003). Disponible en: https://goo.gl/3QYV8C

Personaje 3

Paul Erdös (1913-1996)

Paul Erdös fue un matemático húngaro desconocido para el común de la gente, a pesar de ser el más prolífico después de Euler. Representa muy bien al típico científico completamente absorto en su trabajo y con muy poca conexión con la vida cotidiana. Sus despistes son legendarios. Baste citar que no sabía atarse los zapatos y que no tenía casa propia, sino que viajaba por el mundo de congreso en congreso, hospedándose en casa de sus amigos matemáticos. Cuando uno de ellos (probablemente Stanislaw Ulam) fue hospitalizado a causa de un problema cerebral, Erdös se fue a acompañarlo poniéndole sin cesar problemas de ingenio, para asegurarse de la correcta recuperación del cerebro de su amigo. De él (o de alguno de sus amigos) procede el dicho de que "los matemáticos convierten el café en teoremas". También le gustaba dar pequeños premios en dinero, para quien fuera capaz de resolver algún desafío matemático. Para saber más de él puedes leer su biografía en un libro muy entretenido, *El hombre que solo amaba los números*, de Paul Hoffman.

Usando algoritmos sencillos de entender, aunque posiblemente de complejidad NP, se pueden medir las propiedades de estas redes. Podemos querer saber si hay mucha redundancia en las conexiones o si por el contrario

un fallo en un nodo puede aislar partes de la red. A lo mejor lo que queremos saber es si hay nodos con muy alta conectividad, llamados *hub*, desde los cuales es fácil acceder al resto de la red. A veces lo que nos interesa es más bien la influencia de los nodos sobre la red, por ejemplo, puede haber un nodo con pocas conexiones pero que son accesos directos a los *hub*. Y todo esto tiene interés para saber la velocidad de propagación de la información por la red: virus informáticos que saltan de un computador a otro por redes de contactos de *e-mail*, virus biológicos que se propagan por personas que están en contacto físico, rumores que se propagan por medio de canales electrónicos, incendios que se propagan de un árbol a sus vecinos, etc. El trabajo de investigación en esta área es muy entretenido y hay mucho por hacer y descubrir. Por ejemplo, en el 2014 María Andrea Cruz desarrolló con mucho éxito su trabajo de grado en EVALAB tratando de identificar agrupamientos de nodos en redes, usando técnicas de computación evolutiva y permitiendo solapamiento. En aquella época, Erdös seguramente no lo sabía, pero sus investigaciones sobre grafos iban a servir hoy día para dos cosas: analizar redes sociales virtuales, inexistentes en su época; y entender una forma de emergencia, cuando todavía no se hablaba de ese concepto. Para que te formes una idea de lo último, te propongo que intentes resolver el problema 5.

Problema 5: Teorema de la amistad
Fuente: Freepik. Disponible en: www.flaticon.com

En una fiesta hay N personas. ¿Cuál es el menor valor de N para que existan obligatoriamente al menos 3 personas que, o bien no se conozcan entre sí, o bien sí se conozcan entre sí?

El problema anterior es un ejemplo de emergencia determinista. A partir de cierto valor de *N*, aparecen propiedades nuevas que no existían antes. En otros casos, la emergencia se da de forma probabilista, es decir, no hay garantías sino solo una cierta probabilidad de que ocurra. La mayoría de las emergencias que vamos a ver en este libro son probabilistas, por lo que aquí dejo, en el problema 6, otro ejemplo determinista.

No es el objetivo de este libro entrar a detallar las redes complejas y sus aplicaciones[31], pero sí indicar que, dependiendo del algoritmo empleado en

31 Ver, por ejemplo, Watts (2010).

su fabricación (aleatoria, de mundo pequeño, con conexión preferencial u otras), la red va a tener unos parámetros macroscópicos muy característicos (longitud del camino promedio, diámetro, coeficiente de agrupamiento, centralidad e influencia de cada nodo). Y ello permite hacer razonamientos por analogía entre redes similares aunque provengan de ámbitos muy distintos (biología, ciencias sociales, física, y otras áreas).

David G. Green y David Newth (2005) nos cuentan que los patrones de dependencias en modelos de matrices, sistemas dinámicos, autómatas celulares, semigrupos y conjuntos parcialmente ordenados son todos isomórficos con los grafos dirigidos. Y que el espacio de estados de cualquier autómata o conjunto de autómatas forma un grafo dirigido. Esto quiere decir que cualquier sistema complejo hereda sus propiedades de las de los grafos dirigidos. Además, hay una fuerte relación entre conectividad en un grafo y umbrales críticos para que emerja algo nuevo. O sea, se pueden observar cambios de fase conforme se añaden arcos de forma más o menos aleatorios a un grafo.

Problema 6: Galletas para todos

Fuente: dibujo realizado y cedido amablemente por Helga R. Calvo R. (2017)

Una fábrica de galletas dispone de cajas de empaque para 6, 9 y 20 unidades. ¿A partir de qué número N de galletas es posible atender cualquier pedido?

Por ejemplo, N no es 10 porque no hay forma de empacar 10 galletas en esas cajas, ni tampoco 11. Tampoco N vale 12, porque aunque 12 se pueden empacar en 2 cajas de 6, no es posible empacar 13 galletas.

Algunos de mis estudiantes acertaron con la solución, pero sin demostrarla. La estrategia fue escribir un *software* que iterase desde 6 hasta 1 000 000 de galletas probando a empacar esas galletas usando todas las combinaciones de empaque. Es un *software* muy sencillo de escribir. Al final encontraron un valor de N a partir del cual todos los pedidos podían ser empacados, pero… hasta 1 000 000 de galletas. Sin embargo, lo que exige el enunciado es probarlo hasta infinitas galletas. Eso no se puede hacer con esta estrategia. La programación convencional no abarca la noción de infinito.

¿Tienes tú alguna estrategia mejor?

En resumen, con los grafos se puede modelar prácticamente cualquier cosa digital, y en particular capturan bien las nociones de complejidad y de emergencia de propiedades.

Computación universal

Un ejemplo de grafo muy particular son los autómatas celulares, donde las conexiones entre nodos siguen un patrón muy regular en forma de cuadrícula. Si es un autómata de dos dimensiones como el de la figura 35, con cada nodo conectado a sus vecinos de arriba, abajo, derecha e izquierda. O en forma de fila, si es de una dimensión. Todos los detalles se explican en el capítulo sobre autómatas celulares del libro anterior.

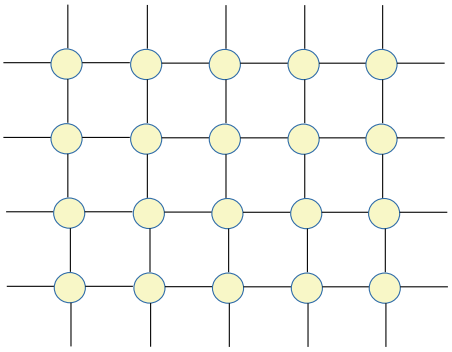


Figura 35. Un autómata celular 2D es un grafo muy simple y repetitivo

Los autómatas celulares son muy simples y eficientes computacionalmente hablando y permiten diseñar mundos artificiales con una noción de espacio básica, soportada en que para mover información de un nodo a otro hay que atravesar los nodos intermedios. No es posible dar saltos, todo el cómputo ocurre simultáneamente en cada nodo y no existe un control central.

Los autómatas celulares se han usado también para modelar interacciones entre partículas (Mitchell y Crutchfield, 1995) con éxito limitado, debido principalmente a que las métricas de distancias que surgen son de tipo Manhattan, muy alejadas de las *euclídeas* del mundo en que vivimos. Por ello, en EVALAB en el año 2013 demostramos que puede emerger una geometría *euclídea* discreta macroscópica usando grafos aleatorios a nivel microscópico, en el trabajo de grado Pablo Andrés Vélez.

Volviendo al tema que nos ocupa en este capítulo, la computación, en el 2002 Wolfram demostró que de los 256 tipos de autómatas celulares que existen en una dimensión, cuatro de ellos tienen capacidad de cómputo universal. Los detalles se pueden encontrar en el capítulo sobre autómatas celulares del libro anterior.

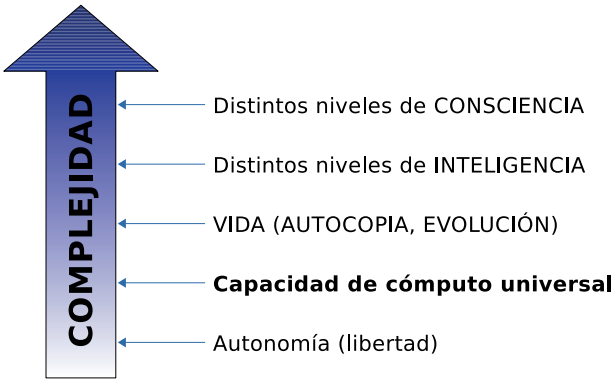


Figura 36. Con una complejidad del orden de 8 bits emerge la computación universal

Es decir, hay una alta probabilidad de lograr computación universal (un 2%) si por azar se logra una infraestructura similar a un autómata celular unidimensional, que requiere 8 bits de memoria y poco más. Concretamente, se requiere comunicación local entre objetos, es decir, cada objeto debe poder comunicar un bit de información al objeto de su derecha y al objeto de su izquierda. Esto es tan simple que ocurre en las células de un ser vivo, pero lo vemos también en las moléculas, los *spin-glasses*, los átomos, por ejemplo. Conclusión: en nuestro universo podemos contar con que aparezca espontáneamente la computación universal (Figura 36).

La capacidad de cómputo es una cosa, pero la universal es algo más potente, pues significa, entre otras cosas, que dentro de un computador así podemos simular otro, también con capacidad de cómputo universal. Y dentro de ese otro, otro más, y así sucesivamente, de manera indefinida.

Si queremos distraernos un poco de estos temas tan áridos, sin salir de ellos, hay muchas implementaciones de Máquinas de Turing extrañas, divertidas y sorprendentes. En IN6TV (2012) tenemos una construida físicamente en Lego, en Ikergarcia1996 (2016) hay otra dentro de *Minecraft* y en Bradbury (2012) podemos ver una máquina dentro de otra, implementadas en un autómata celular con las reglas de *LIFE* que vimos en el libro anterior.

¿Qué es un simulador?

El principio de Church-Turing-Deutsch (Deutsch, 1985) dice que una Máquina Cuántica Universal puede simular exactamente cualquier proceso físico. Entonces, según esta definición, el mundo es un computador cuántico. Ello no necesariamente implica que haya sido construido por alguien, ya que puede haber surgido espontáneamente a través de un proceso al azar, evolutivo u otros que desconocemos. Recordemos que es muy baja la

complejidad que se requiere para lograr una Máquina de Turing Universal Clásica. Podemos preguntarnos si hay una equivalencia entre Máquinas de Turing Universales Clásicas y Cuánticas, y la verdad es que todavía no se sabe. Durante el nacimiento de la computación cuántica se pensaba que iba a resolver problemas que no podía la clásica. Pero han pasado los años y todavía no hay un ejemplo de ello, por lo que comienza a pensarse que son equivalentes. En el momento en que ello se demuestre, también quedaría demostrado que nuestro mundo es computacional.

En cualquier caso, la tesis de este libro es que el mundo real es indistinguible de un mundo artificial que tenga similar complejidad y, por ello, es necesario explicar en detalle que un mundo artificial necesita un simulador para ejecutarse, así como las características que tiene, cómo funciona y cómo se construye.

Básicamente, un simulador es como un juego. Hay juegos en 2D, en 3D, con gafas de realidad virtual, con guantes hápticos y las tecnologías serán más y mejores en un futuro. Cuanto más complejo sea el mundo artificial, más sensación de realismo tendrá. Es más, el día que logremos conectar el juego directamente al cerebro, sin pasar por nuestros sentidos de la vista, oído y tacto podremos crear mundos con muchos más detalles, y más complejidad que el mundo real. Por ejemplo, podríamos hacer que los colores produzcan también sensaciones olfativas, y entonces el jugador no solo podrá ver el color amarillo si lo tiene delante, sino que también lo podrá oler si está cerca aunque fuera de su ángulo de visión. También podríamos crear nuevos sentidos para los ultrasonidos (como tienen los murciélagos), para los campos eléctricos (como tienen las anguilas) e incluso crear otros inexistentes e inimaginables en el mundo real. Regresar al mundo real será aburrido (y, ojo, que eso ya está ocurriendo con juegos muy adictivos).

Pero aquí hay algo que muchas personas no entienden bien. Un simulador va mucho más allá de eso. No solo permitirá a los humanos del mundo real sumergirse en el mundo artificial a la manera que han recreado en películas tan sugerentes como *Piso 13*, *Avalón*, *Matrix* y otras, sino que también el propio mundo simulado podrá tener seres artificiales propios, bien porque los hayamos diseñado ex profeso, bien porque hayan surgido allí a partir de las leyes de la física de ese mundo. Esto ya empieza a ocurrir en algunos juegos como "Spore"[32] o "No Man's Sky"[33] y solo es cuestión de tiempo que estos seres alcancen la complejidad necesaria para tener inteligencia y consciencia, y así poder reflexionar sobre el mundo en el que viven. Para estos seres,

32 Will Wright, *Electronics Arts*, 2008.

33 Hello Games, *Sony Interactive Entertainment*, 2016.

que no conocen nuestro mundo real, su mundo artificial será el real, pues no tienen ninguna experiencia que les lleve a añorar alguna característica externa. No pueden saber que hay un mundo más complejo que les abarca. E incluso si esta perspectiva parece muy fantástica, piensa que un niño real recién nacido al que se conecte su cerebro directamente a un juego (aunque sea poco sofisticado) aprenderá a vivir y a desarrollar su personalidad completamente sumergido en un mundo artificial. Si en este mundo no hay colores, sino solo grises, no los echará en falta. Si no hay sonido, tampoco. Etcétera. No puede imaginar cosas que jamás ha experimentado. Vivirá en su mundo artificial y no podrá saber que hay un mundo real más complejo ahí fuera. Esta es la razón principal que impide distinguir un mundo real de uno artificial: si estás dentro, no hay forma. Y siempre estás dentro.

Esto ya se puede hacer hoy día, si bien no se debe, por razones éticas. Hay que tener en cuenta que ya existen interfaces entre dispositivos electrónicos y las fibras nerviosas que permiten controlar directamente exoesqueletos con el pensamiento, de modo que es solo un problema técnico cablear todos los nervios que salen y entran del cerebro a un computador. Por otro lado, el ancho de banda sensorial humano es bastante más reducido de lo que se pensaba en un principio. Por ejemplo, la visión potencialmente debería requerir 10^{10} bit/s de información, que es lo que llega a la retina, pero lo cierto es que a la primera capa del córtex visual solo llega 10^4 bit/s (Raichle, 2010). Y para los otros sentidos se requiere menos ancho de banda. O sea, que los computadores actuales pueden recrear en tiempo real toda la información que recibimos.

El mundo que creemos real puede ser artificial y si uno hace las cuentas como Nick Bostrom, las probabilidades de que ello ocurra son altísimas porque hay una realimentación positiva (un proceso exponencial) involucrada.

Pero hay más cosas que tenemos que saber de un simulador. Una primera noción es que el simulador aísla dos mundos, el real y el simulado (Figura 37).

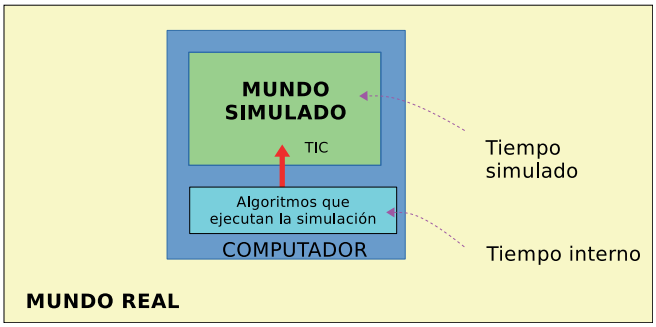


Figura 37. Dos mundos

Nosotros habitamos el mundo real y estamos sumergidos en sus "leyes de la Física". Usándolas podemos construir el simulador. Pero dentro del simulador, el nuevo mundo no tiene por qué seguir esas leyes. Podemos crear otras nuevas hasta llegar a extremos inauditos. Examinemos unos cuantos ejemplos:

Nuestro espacio es de tres dimensiones y ello ya es de por sí bastante misterioso, y da para reflexionar, pues resulta que en un universo con una o dos dimensiones espaciales no es posible generar complejidad. Mientras que en cuatro o más dimensiones la complejidad es tanta que puede ser imposible de entender o manejar, impidiendo que existan estructuras autoorganizadas y adaptativas. O sea que tres dimensiones espaciales parece que nos lleva a la complejidad justa para que exista la vida. ¿Por qué nuestro mundo tiene tres dimensiones?

Pero en el mundo simulado el número de dimensiones espaciales puede ser distinto a tres. Si pensamos en un espacio absoluto, como un contenedor, basta que lo implementemos en *software* como una matriz. La matriz puede tener una dimensión (seguramente dará lugar a mundos muy aburridos), dos o tres (como en los juegos de tablero o de primera persona). Pero nada impide construir la matriz con cuatro dimensiones o con veintisiete o con cien mil millones. Seguramente saldrán mundos terriblemente complejos, pero son fáciles de construir en *software*. También se puede construir un mundo sin la noción del espacio (o sea, con cero dimensiones). Por ejemplo, cuando simulamos la evolución de poblaciones de predadores y presas con las ecuaciones de Lotka-Volterra, el espacio no tiene relevancia. Por así decir, cualquier predador puede comerse a cualquier presa porque todos se encuentran en el mismo punto espacial.

Además, las propiedades del espacio no tienen por qué ser iguales a las del nuestro. Por ejemplo, en el nuestro se cumple el teorema de Pitágoras (o sea, $h^2=a^2+b^2$, siendo h la hipotenusa y a, b los catetos de cualquier triángulo rectángulo). Mientras que el mundo virtual podemos implementarlo con autómatas celulares, usando la distancia Manhattan, de modo que se cumplirá en su lugar $h=a+b$. O decidir cualquier otra fórmula. O implementarlo con grafos aleatorios y que las distancias tengan ruido superpuesto o que cambien con el tiempo. O que las dimensiones no sean homogéneas (dos dimensiones pueden ser Euclídeas, otras dos Manhattan, otra permite moverse hacia un lado pero no hacia el otro, etc.). Como ya dijimos, en EVALAB (Vélez, 2013) diseñamos un mundo con grafos aleatorios evolutivos, de modo que aunque en distancias cortas la conexión entre nodos era al azar (lo cual podría simular saltos cuánticos como el efecto túnel) en distancias largas se cumple el teorema de Pitágoras.

Los objetos podrían tener masa inercial pero no masa gravitatoria, o al revés. O podríamos diseñarlo de modo que la energía no se conserve. O que no exista el concepto de energía. O que haya partículas distintas al electrón, quarks, fotones, etc. Y además podríamos cambiar el valor de cualquier constante de las que llamamos fundamentales, como la velocidad de la luz, la constante de Plank, la carga del electrón… O que no sea constante. O eliminar el concepto. O añadir conceptos nuevos. No hay límite a la imaginación.

O que no haya nada estocástico. Por ejemplo, cuando se lance una moneda al aire que siempre salga cara[34]. O que siempre salga repetido tres veces seguidas cada resultado. O cualquier otra cosa. Cuando hacemos *software* tenemos libertad absoluta para cambiar el comportamiento de los objetos, y además es trivial de lograr.

El tiempo también lo podemos manipular pues en todo simulador hay dos ejes de tiempo: un tiempo del mundo simulado que podemos diseñar como queramos (puede tener incluso varias dimensiones); y un tiempo interno que coincide con el del mundo real externo, y que no es observable por hipotéticos seres que habiten dentro del simulador porque cuando corre el tiempo interno, el simulado (con el que laten los corazones de los seres virtuales) está detenido.

Para entender todo esto, vamos a contar *grosso modo* cómo funciona un simulador de circuitos digitales síncronos, usando como ejemplo el circuito de la figura 38, que tiene dos entradas conocidas (e_1, e_2), tres nodos internos (n_1, n_2, n_3) y una salida (s_1). El circuito está construido con una puerta *AND* de tres entradas, dos puertas *NOT* (que siempre son de una entrada) y un *flip-flop*.

34 El libro está dirigido a personas que sepan programar. Pero para las que no, es tan sencillo como definir:

```
def lanzar_moneda()
  return "CARA"
end
```

Mientras que en un mundo como el nuestro la función sería:

```
def lanzar_moneda()
  return rand < 0.5 ? "CARA" : "SELLO"
end
```

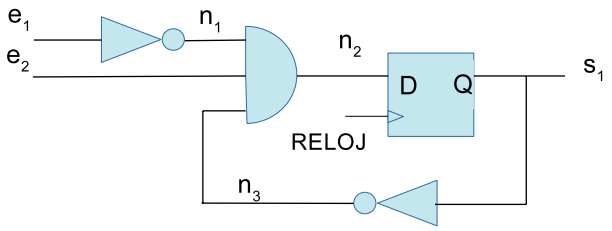


Figura 38. Un circuito digital sencillo

Para quien no recuerde cómo funcionan los circuitos digitales, cada señal tiene dos posibles niveles (0 y 1). Las tablas de verdad de las puertas están descritas en la figura 39. Mientras que el *flip-flop* es un bit de memoria que funciona así: la salida *Q* mantiene su estado permanentemente, y solo cambia cuando llega un pulso de *RELOJ*, copiando en ella lo que hubiera a la entrada *D*. Este circuito es muy sencillo, pero nos servirá para nuestros objetivos. Los circuitos reales tienen billones de puertas lógicas y *flip-flops*, y estos últimos comparten la misma señal de reloj (por eso se llaman síncronos). Con ello se logra que las salidas solo cambien en los instantes que llegan pulsos de *RELOJ*.

ENTRADAS	SALIDA
000	0
001	0
010	0
011	0
100	0
101	0
110	0
111	1

(a)

ENTRADA	SALIDA
0	1
1	0

(b)

Figura 39. Tabla de verdad de: (a) la puerta AND; (b) la puerta NOT

Este *RELOJ* es el reloj del circuito simulado (que coincide con el TIC de la figura 37). Es el que ve el usuario humano en la pantalla del computador, cada vez que avanza un pantallazo. También es el que percibirían unos hipotéticos seres que vivan dentro del simulador. Pero existe un tiempo interno que se desenvuelve entre los pulsos del reloj simulado, y que sirve para calcular los valores de los nodos internos, y que no ve el usuario humano desde su mundo externo. Tampoco lo podrían percibir unos hipotéticos seres dentro del simulador porque mientras se ejecuta el tiempo interno su vida está detenida, entre medias de dos tics de su reloj simulado. Por ejemplo, el simulador debe calcular el valor de n_1 negando la entrada e_1, debe calcular

n_2 como un *AND* de n_1, e_2 y n_3, pero a su vez n_3 también hay que calcularlo como negación de s_1. Y estos cálculos se hacen de una manera ingenua, en cualquier orden, de modo que pueden dar lugar a valores espurios que, con el paso del tiempo interno y más cálculo, finalmente se estabilizarán a los valores correctos. Lo ideal es que cuando todos los cálculos (realizados usando el tiempo interno) se hayan estabilizado, entonces llegue un pulso del *RELOJ* simulado que producirá las nuevas salidas observables al usuario humano (o un latido del corazón del hipotético habitante simulado).

Los cálculos internos en este caso son muy sencillos (ecuación 18), pero el orden en que se ejecuten estos tres pasos produce resultados distintos. Para el simulador es complicado deducir el orden correcto porque puede haber muchas dependencias cruzadas, por lo que habitualmente calcula estos pasos en cualquier orden y luego vuelve a repetir los cálculos hasta que todos los nodos internos llegan a un valor estable, que ya no cambia más. Allí termina la ejecución del tiempo interno y entonces el simulador envía un pulso por el *RELOJ* simulado, para que los cálculos efectuados produzcan cambios en las salidas.

$$\begin{aligned} n2 &= \text{AND}\ (n1, e2, n3) \\ n3 &= \text{NOT}\ (s1) \\ n1 &= \text{NOT}\ (e1) \end{aligned} \qquad \text{Ec. 18}$$

Pero no siempre es esto tan sencillo. Hay casos especiales donde los cálculos internos no convergen (ver fragmento de circuito en la figura 40, cuyos cálculos internos están en la ecuación 19). Aquí, si $n_1=0$ entonces $n_3=0$ y $n_2=1$. Y si volvemos a hacer los cálculos sale lo mismo. Perfecto. Ahora, si cambia el nodo $n_1=1$ entonces $n_3=1$ y $n_2=0$, pero si volvemos a calcular entonces $n_3=0$ y $n_2=1$. Y si otra vez lo calculamos sale $n_3=1$ y $n_2=0$, y así sucesivamente, es decir, los nodos n_3 y n_2 oscilan.

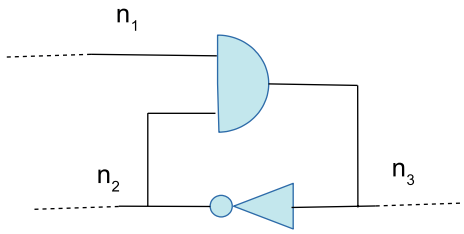


Figura 40. Circuito con problemas

$$\begin{aligned} n3 &= \text{AND}\ (n1, n2) \\ n2 &= \text{NOT}\ (n3) \end{aligned} \qquad \text{Ec. 19}$$

Dependiendo de cómo esté diseñado el simulador de circuitos lógicos, puede que aborte la simulación indicando que hay un error (llamado "condición de carrera") o puede que tenga un *timeout* que le obligue a detenerse, presentando como resultado cualquiera de los valores de la oscilación. Incluso si el simulador es sofisticado, puede aceptar ambos resultados y continuar la simulación mostrando al usuario la superposición de ambos. Al usuario le decimos "ambas posibilidades son válidas", y ello traerá consecuencias sobre los resultados mostrados en los siguientes tics del *RELOJ* simulado que serán también superposición de otros resultados[35]. Por ejemplo, si un nodo tiene una superposición de 0 y 1, y después viene una puerta *NOT*, su salida será una superposición de 1 y 0. Y si hacemos un AND entre las dos señales el resultado será siempre 0 desapareciendo la superposición. Es decir, en algunos casos la superposición puede aparecer y en otros desaparecer. La superposición puede ser molesta, pero no pasa nada si el resultado final del circuito no la tiene. La superposición de estados, que es un quebradero de cabeza para los que intentan comprender la mecánica cuántica, se entiende perfectamente suponiendo que el mundo sea computacional.

Y haciendo un pequeño paréntesis, estas oscilaciones son el equivalente a lo que los lógicos llaman "proposiciones indecidibles", de las que habla el teorema de Gödel que veremos enseguida.

Con ello espero haber aclarado que en las simulaciones tenemos total libertad de crear (o no) espacio y tiempo de diversas dimensiones. Y de la misma manera podemos crear (o no) materia, energía, partículas y cualquier cosa que se nos ocurra. También es importante entender que las simulaciones que corren sobre computadores son discretas, es decir, no existe allí el concepto de continuidad. El espacio es discreto, el tiempo corre a saltos y cualquier cantidad elemental está cuantizada.

Otro concepto importante es que en un computador todo es determinista, no hay nada estocástico. Para conseguir impredecibilidad, lo único que se puede hacer es generar secuencias de números seudoaleatorios utilizando fórmulas muy complicadas, pero debemos comprender que en el fondo son fórmulas fijas y deterministas. Otra alternativa es extraer información del mundo externo e inyectarla al mundo interno. En este caso, al mundo interno le llegan datos incorrelados con cualquier otra cosa que ya haya allí,

35 Recordemos que en nuestro universo existe esa posibilidad: si tenemos dos rendijas muy juntas y lanzamos hacia allí una partícula, ella puede pasar a la vez y simultáneamente por las dos rendijas. Este es el experimento más simple que nos muestra superposición en el mundo cuántico. Quizás haya llegado el momento de dejar de verlo como algo extraordinario, raro o mágico, y aceptar que es una consecuencia de que nuestro universo es computacional.

por lo que se pueden considerar estocásticos. Recordemos que, para ello, en Linux existe el flujo de datos */dev/random* que se puede leer por *software* y que contiene información sobre eventos externos (tiempos entre pulsaciones de teclas o entre movimientos del *mouse*, intervalos de llegada entre paquetes por el puerto de red y el ruido de la entrada del micrófono) convenientemente filtrados para eliminar autocorrelaciones.

Insisto en que los eventos externos no tienen por qué ser estocásticos en el ámbito del mundo real. Lo que importa es que sean incorrelados entre sí e incorrelados con los eventos del mundo interno simulado. Para unos hipotéticos seres que vivan dentro del simulador, un flujo de bits procedente del exterior y que esté incorrelado con cualquier otra información interna, lo interpretarán como estocástico.

Límites computacionales

Hemos visto que la computación completa es fácil de lograr, pero la pregunta que vamos a analizar y tratar de contestar ahora es: ¿todo problema se puede resolver usando computación? Obviamente, hay muchos problemas del mundo real que no tienen solución, bien sea porque yo no controlo ninguna variable que pueda modificarlo, bien sea porque las restricciones son tantas y/o tan duras que ninguna solución puede satisfacerlas a todas. Sin embargo, hay otros problemas aparentemente sencillos, donde no vemos ningún obstáculo para encontrar su solución pero que en la práctica la computación no logra solucionarlos. ¿Por qué? ¿Cuáles son los límites que no puede franquear la computación? Veremos que existen tres limitaciones.

Este tema es importante no solo para la computación en sí, sino especialmente para la inteligencia artificial (y la natural) ya que los partidarios de la inteligencia artificial débil[36], principalmente Roger Penrose y John Searle, esgrimen estas limitaciones como una prueba de que la inteligencia artificial no puede lograr cualquier cosa, al contrario que la mente humana, a la que no vemos límites. Ellos no se dan cuenta que la mente humana padece de los mismos problemas, y que si somos creativos y logramos aparentemente superar limitaciones es porque empleamos el mismo truco que los algoritmos evolutivos o las redes neuronales artificiales: usamos aleatoriedad, no somos consistentes, avanzamos por el método de prueba y error, nos equivocamos con frecuencia y nuestros logros no se pueden atribuir a ningún ser humano

36 Los que piensan que los computadores pueden imitar algunos aspectos de la inteligencia humana, pero nunca podrán superarla, y que la mera imitación no equivale al fenómeno real de la inteligencia.

sino a toda la humanidad. Por eso, los algoritmos de inteligencia artificial deben renunciar a la exactitud y las garantías de convergencia, y abrazar el error, las contradicciones y el decir de vez en cuando a las preguntas planteadas "¡no lo sé!", igual que hacemos los humanos.

Además, Penrose y Searle parecen desconocer que desde hace muchos años los algoritmos basados en la evolución, en razonamiento *bayesiano* y en redes neuronales artificiales han logrado éxitos impresionantes en campos que antes se creía que eran exclusivos de los humanos, como reconocimiento de rostros y de voz, conducción autónoma de vehículos o creación de obras de arte en música y pintura.

Por otro lado, para los partidarios de la inteligencia artificial fuerte[37] entre los que me encuentro, entender los límites de una técnica o herramienta es el primer paso para saltarlos. Dicho de otra forma, el 80% del camino que nos lleva a la solución de un problema consiste en entenderlo bien o, en otras palabras, en formular las preguntas correctas.

Entonces primero enumeraremos los límites conocidos a la computación, luego daremos un pequeño repaso a cada uno de ellos (pues realmente se conocen desde hace mucho tiempo, y existe literatura especializada en cada tema), y por último ofreceremos unas conclusiones sobre lo que ello implica para la inteligencia artificial.

Estos límites son el teorema de Gödel (que se traduce computacionalmente en el problema de la parada), la complejidad algorítmica y el teorema de *No-Free-Lunch*.

Teorema de Gödel

Para comenzar a entender a Gödel, pensemos primero en la importancia que tienen los sistemas axiomáticos (aplicación de reglas de manera mecánica). En la Grecia clásica, en vez de eso, existían unas reglas de razonamiento pero en lenguaje natural. Por ejemplo, el *Modus Ponens* de la figura 41:

Los hombres son mortales.
Sócrates es un hombre.
Luego Sócrates es mortal.

Figura 41. Modus Ponens

Sin embargo, esta forma de razonar está llena de problemas. Veamos un ejemplo en la figura 42:

37 Los que creen que una simulación que capture los detalles esenciales de un proceso es indistinguible de ese mismo proceso.

Los chinos son numerosos.
Confucio es chino.
Luego Confucio es numeroso.

Figura 42. Modus Ponens equivocado

Otro ejemplo muy bonito de razonamiento defectuoso puede encontrarse en el cuento "Lo que le dijo la Tortuga a Aquiles" en el libro *Matemática Demente*, de Lewis Carroll, donde la Tortuga ingeniosamente obliga al ingenuo Aquiles a llevar a cabo una deducción, aparentemente simple, en infinitas proposiciones lógicas. Recordemos que en la Grecia clásica, la Tortuga y el gran corredor Aquiles ya se habían enfrentado en una carrera de 100 metros, donde Aquiles dio 50 metros de ventaja a la Tortuga. Para los griegos era una paradoja que Aquiles nunca alcanzase a la Tortuga, a pesar de ser 10 veces más veloz, pues cuando él hubiera recorrido los 50 metros que le separaban de ella, la Tortuga habría avanzado 50/10=5 metros (la décima parte); y cuando Aquiles recorriera como un rayo esa distancia, la Tortuga ya no estaría allí, pues habría avanzado 5/10=0.5 metros. El resultado para los filósofos griegos de aquella época era que Aquiles jamás alcanzaría a la Tortuga. Para nosotros, que ya conocemos el cálculo diferencial de Newton y Leibnitz, los griegos simplemente estaban calculando el instante del adelantamiento como límite de una serie infinita. Que resulte una serie infinita no quiere decir que el adelantamiento no vaya a ocurrir, sino que se eligió un tortuoso camino matemático para su cálculo. Y en el cuento de Carroll de nuevo se elige un procedimiento tortuoso para hacer una demostración lógica sencilla.

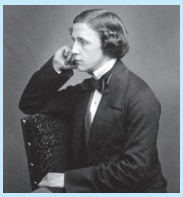
La forma que hemos encontrado de evitar estos defectos de razonamiento es formalizarlos, es decir, eliminar el lenguaje natural, el "sentido común" y la intuición, cambiándolos por un lenguaje formal matemático, esto es, un conjunto de verdades obvias de partida (axiomas) y un conjunto de reglas para encontrar nuevas verdades.

Así el primer razonamiento se traduce a lo siguiente: sea H el conjunto de los hombres, sea *M* el conjunto de los mortales y sea *s*, Sócrates. Entonces:

$$(\forall x: x \in H \rightarrow x \in M) \wedge (s \in H) \vdash (s \in M) \qquad \text{Ec. 20}$$

Mientras que el segundo razonamiento se formaliza así: sea *C* el conjunto de los chinos, sea *N* el conjunto de los conjuntos que son numerosos y sea *f*, Confucio. Entonces:

$$(C \in N) \wedge (f \in C) \vdash \text{ de aquí no se deduce nada} \qquad \text{Ec. 21}$$



Fuente: The Life and Letters of Lewis Carroll (2010). Disponible en: https://bit.ly/2ZRh2nj

Personaje 4

Lewis Carroll (1832-1898)

Es el seudónimo del matemático Charles Lutwidge Dodgson, muy conocido por sus cuentos "Alicia en el país de las maravillas" y "A través del espejo". Aprovechando que ya es de dominio público presentaremos otro cuento menos conocido, "Lo que la Tortuga le dijo a Aquiles", escrito en 1894 y recopilado en el libro Matemática Demente (1995):

Aquiles había alcanzado a la tortuga y se había sentado cómodamente sobre su caparazón.

—¿De modo que ha llegado usted al final de nuestra carrera?— dijo la Tortuga. —¿Aun cuando consistía en una serie infinita de distancias? ¿Pensó que algún sabihondo había probado que la cuestión no podía ser realizada?

—Sí puede ser realizada— dijo Aquiles. —¡Ella ha sido realizada! Solivitur ambulando. Usted ve, las distancias fueron disminuyendo constantemente y así…

—Pero, ¿si hubieran ido aumentando?— interrumpió la tortuga. —Entonces ¿qué?

—Entonces yo no debería estar aquí— replicó modestamente Aquiles, —y a estas alturas usted hubiera dado ya varias vueltas al mundo.

—Me aclama, aplana, quiero decir— dijo la Tortuga —pues usted sí que es un peso pesado, ¡sin duda! Ahora bien, ¿le gustaría oír acerca de una carrera en la que la mayoría de la gente cree poder llegar con dos o tres pasos al final y que realmente consiste en un número infinito de distancias, cada una más larga que la distancia anterior?

—¡Me encantaría, de veras!— dijo el guerrero griego mientras sacaba de su casco (pocos guerreros griegos poseían bolsillos en aquellos días) una enorme libreta de apuntes y un lápiz. —¡Empiece, y hable lentamente, por favor! ¡La taquigrafía aún no ha sido inventada!

—¡El hermoso Primer Teorema de Euclides!— murmuró como en sueños la tortuga. —¿Admira usted a Euclides?

—¡Apasionadamente! ¡Al menos, tanto como uno puede admirar un tratado que no será publicado hasta dentro de algunos siglos más!

—Bien, en ese caso tomemos solo una pequeña parte del argumento de ese Primer Teorema: solo dos pasos y la conclusión extraída de ellos. Tenga la bondad de registrarlos en su libreta. Y, a fin de referirnos a ellos convenientemente, llamémoslos A, B y Z.

(A) Dos cosas que son iguales a una tercera son iguales entre sí.

(B) Los dos lados de este triángulo son iguales a un tercero.

(Z) Los dos lados de este triángulo son iguales entre sí.

Los lectores de Euclides admitirán, supongo, que Z se sigue lógicamente de A y B, de modo que quien acepte A y B como verdaderas debe aceptar Z como verdadera, ¿no?

—¡Sin duda! Hasta el más joven de los alumnos del Liceo— tan pronto como se inventen los Liceos, cosa que no sucederá hasta dentro de dos mil años —admitirán eso.

sigue

—Y si algún lector no ha aceptado A y B como verdaderas, supongo que aún podría aceptar la secuencia como válida.
—Sin duda que podría existir un lector así. Él podría decir "Acepto como verdadera la Proposición Hipotética de que si A y B son verdaderas, Z debe ser verdadera, pero no acepto A y B como verdaderas". Un lector así procedería sabiamente abandonando a Euclides y dedicándose al fútbol.
—Y ¿no podría haber también algún lector que pudiera decir "Acepto A y B como verdaderas, pero no acepto la Hipotética"?
—Ciertamente podría haberlo. Él, también, mejor se hubiera dedicado al fútbol.
—¿Y ninguno de estos lectores— continuó la Tortuga —tiene hasta ahora alguna necesidad lógica de aceptar Z como verdadera?
—Así es— asintió Aquiles.
—Ahora bien, quiero que usted me considere a mí como un lector del segundo tipo y que me fuerce, lógicamente, a aceptar Z como verdadera.
—Una Tortuga jugando al fútbol sería… — comenzó Aquiles.
—Una anomalía, por supuesto— interrumpió airadamente la Tortuga. —¡No se desvíe del tema, Primero Z y después el fútbol!
—¿Debo forzarlo a aceptar Z, o no?— preguntó Aquiles pensativamente. —Y su posición actual es que acepta A y B pero NO acepta la Hipotética…
—Llamémosla C— dijo la tortuga, —pero no acepto que:
(C) Si A y B son verdaderas, Z debe ser verdadera.
—Esa es mi posición actual— dijo la Tortuga.
—Entonces debo pedirle que acepte C.
—Lo haré así— dijo la Tortuga —tan pronto como lo haya registrado en su libreta de Apuntes. ¿Qué más tiene anotado?
—¡Solo unos pocos apuntes!— dijo Aquiles agitando nerviosamente las hojas. —¡Unos pocos apuntes de las batallas en las que me he distinguido!
—¡Veo que hay un montón de hojas en blanco!— observó jovialmente la Tortuga. —¡Las necesitaremos todas!— Aquiles se estremeció. —Ahora escriba mientras dicto:
(A) Dos cosas que son iguales a una tercera son iguales entre sí.
(B) Los dos lados de este triángulo son iguales a un tercero.
(C) Si A y B son verdaderas, Z debe ser verdadera.
(Z) Los dos lados de este triángulo son iguales entre sí.
—Debería llamarla D, no Z— dijo Aquiles. —Viene después de las otras tres. Si acepta A y B y C, debe aceptar Z.
—¿Y por qué debo?
—Porque se desprende lógicamente de ellas. Si A y B y C son verdaderas, Z debe ser verdadera. No puede discutir eso, me imagino.
—Si A y B y C son verdaderas, Z debe ser verdadera— repitió pensativamente la Tortuga. —Esa es otra Hipótesis, ¿o no? Y, si no reconociera su veracidad, podría aceptar A y B y C, y todavía no aceptar Z, ¿o no?
—Podría— admitió el cándido héroe, —aunque tal obstinación sería ciertamente fenomenal. Sin embargo, el evento es posible. De modo que debo pedirle que admita una Hipótesis más.
—Muy bien, estoy ansioso por admitirla, tan pronto como la haya anotado. La llamaremos 'D'. Si A y B y C son verdaderas, Z debe ser verdadera. ¿Lo ha registrado en su libreta de apuntes?

sigue

—¡Lo he hecho!— exclamó gozosamente Aquiles, mientras guardaba el lápiz en su estuche. —¡Y por fin hemos llegado al final de esta carrera ideal! Ahora que ha aceptado A y B y C y D, por supuesto acepta Z.
—¿La acepto?— dijo la Tortuga inocentemente. —Dejémoslo completamente claro. Acepto A y B y C y D. Suponga que todavía me niego a aceptar Z.
—¡Entonces la Lógica le agarraría del cuello y le forzaría a hacerlo!— replicó triunfalmente Aquiles.
—La Lógica le diría, "¡No se puede librar. Ahora que ha aceptado A y B y C y D, debe aceptar Z!". De modo que no tiene alternativa, usted ve.
—Cualquier cosa que la Lógica tenga a bien decirme merece ser anotada— dijo la Tortuga —de modo que regístrela en su libro, por favor. La llamaremos "E". Si A y B y C y D son verdaderas, Z debe ser verdadera. Hasta que haya admitido eso, por supuesto no necesito admitir Z. De modo que es un paso completamente necesario, ¿ve usted?
—Ya veo— dijo Aquiles. Y había un toque de tristeza en su tono de voz.
Aquí el narrador, que tenía urgentes negocios en el banco, se vio obligado a dejar a la simpática pareja y no pasó por el lugar nuevamente hasta algunos meses después. Cuando lo hizo, Aquiles estaba aún sentado sobre el caparazón de la muy tolerante Tortuga y seguía escribiendo en su libreta de apuntes que parecía estar casi llena.
La Tortuga estaba diciendo —¿Ha anotado el último paso? Si no he perdido la cuenta, ese es el mil uno. Quedan varios millones más todavía. Y le importaría, como un favor personal, considerando el rompecabezas que este coloquio nuestro proveería los Lógicos del siglo XIX. ¿Le importaría adoptar un retruécano que mi prima la Tortugacuática Artificial hará entonces y permitirse ser renombrado "Aquiles el Sutiles"?
—¡Como guste!— replicó el cansado guerrero con un triste tono de desesperanza en su voz, mientras sepultaba la cara entre sus manos. —Siempre que usted, por su parte, adopte un retruécano que la Tortugacuática Artificial nunca hizo y se permita renombrarse "Tortuga Tortura".

Podemos ver que usando lenguajes formales evitamos las ambigüedades de los lenguajes naturales. Los lenguajes formales son un gran invento. Veamos un poco cuál fue su origen.

En el siglo XVII nace la física moderna, de la mano de Galileo, así como el método científico: los sucesos que antes eran arbitrarios pueden ahora medirse de forma repetible y modelarse con ecuaciones matemáticas, las cuales se verificaban de forma tan precisa que se decía y pensaba que los objetos del mundo material estaban cumpliendo leyes, leyes de la física claro, pero leyes, como las que impone un rey a sus súbditos. Esto dio lugar a un paradigma, es decir, una forma de ver y entender el mundo, llamado mecanicismo: todos los objetos materiales siguen leyes, no hay nada arbitrario, todo se puede predecir.

El físico y matemático Laplace fue quien mejor entendió y formuló esta idea, llamada determinismo: conociendo las condiciones iniciales de un sistema se puede predecir cómo va a cambiar en el tiempo. Y si conociéramos todas las posiciones iniciales y velocidades de todas las partículas del universo, podríamos calcular sus trayectorias desde ese momento en adelante, de modo que nada de lo que ocurra en el universo nos tomaría por sorpresa,

al poder predecirlo por adelantado. Este paradigma alcanzó también la matemática: Descartes y Leibnitz pensaban que se podía construir un método mecánico universal (lo que hoy llamamos un algoritmo) para solucionar cualquier problema que pudiera ser formalizado.

El siglo XIX fue el apogeo del determinismo en la Física. Pensemos que prácticamente todos los fenómenos físicos conocidos en aquella época tenían una explicación por medio de una ley, es decir, una ecuación. Se creía que se conocía todo acerca del universo. Es en ese ambiente que Bertrand Russell y Alfred North Whitehead escriben los *Principia Mathematica* donde formalizan la lógica y la matemática de los números enteros, trabajando en el "programa de Hilbert", que intentaba construir toda la matemática a partir de un conjunto finito de axiomas y demostrar que el resultado era coherente.

En 1931 Kurt Gödel demostró que los *Principia Mathematic*a no podían ser a la vez completos y coherentes, echando por tierra el programa de Hilbert que pretendía mecanizar las matemáticas. Concretamente, el teorema de Gödel dice que, si tenemos cualquier sistema axiomático suficientemente complejo, ocurre una de estas dos cosas:

- Es incoherente: contiene contradicciones internas, es decir, contiene simultáneamente una proposición y su negación. Si un sistema es incoherente entonces no vale para nada, puesto que a partir de la contradicción es posible demostrar cualquier proposición falsa.
- Es incompleto: existen verdades que no son demostrables desde dentro del sistema. Y este sería el origen de uno de nuestros límites computacionales para la inteligencia artificial, porque viene a decir que existen cosas que son verdad, pero que no se puede demostrar que lo sean. Esto aparenta ser una grave limitación para la inteligencia artificial.

Por supuesto, una verdad indemostrable en un sistema se puede demostrar ampliando los axiomas (o las reglas de ese sistema) añadiendo a ellos esa nueva verdad. Pero eso crea nuevas verdades indemostrables en el sistema ampliado. De modo que la incompletitud es intrínseca a los sistemas formales.

Este teorema es aplicable no solo a los *Principia Mathematic*a (matemáticas de números enteros y lógica) sino a cualquier sistema formal axiomático suficientemente complejo.

Y ¿qué significa "suficientemente complejo"? Pues las matemáticas de los números enteros con las operaciones de suma y multiplicación ya son suficientemente complejas. La lógica de primer orden también es suficientemente compleja. Hay sistemas más simples —como los números enteros

solo con la operación de suma, los números reales (sin la noción de entero) con las operaciones de suma y multiplicación y la geometría— en los que no se aplica el teorema de Gödel.

Fuente: Public Domain. Disponible en: https://goo.gl/9A2pQd

Personaje 5

Lewis Carroll (1832-1898)

El matemático y filósofo Kurt Gödel nació en lo que hoy se conoce como Brno en la República Checa y se vio obligado a emigrar a los Estados Unidos, como tantos otros científicos, por el miedo al nazismo. Quizás por ello su personalidad tenía bastantes peculiaridades, como la de otras personas que se dedican intensamente a la ciencia, en este caso a las matemáticas. Dos anécdotas ilustran sus excentricidades: al ir a pedir la nacionalidad estadounidense, se ofuscó mucho porque descubrió que la constitución de USA contenía una contradicción lógica. Por suerte lo acompañaban sus amigos, Albert Einstein y Oskar Morgenstern, que no le permitieron discutir eso con el juez que le iba a dar la nacionalidad.

La otra anécdota, más dramática, se refiere a que solo recibía la comida que le preparaba su mujer porque temía ser envenenado. Murió de hambre, una vez que a ella tuvieron que hospitalizarla. Esto puede parecer sorprendente, pero con tanta información que aparece hoy en día recomendando evitar productos peligrosos (fito-hormonas, sulfitos, azúcar, aceite de palma, gluten, por citar algunos), cualquiera de nosotros podría terminar de la misma forma.

Aun cuando no es muy conocido por el público en general, Gödel fue un personaje extraordinario que cambió por completo la matemática, a la que despojó de sus ropajes de perfección.

Un libro excelente donde puede encontrarse una demostración asequible de este teorema es *Gödel, Escher, Bach: un grácil e infinito bucle* de Douglas Hofstadter, donde también se tocan aspectos filosóficos y prácticos sobre computación, evolución, creatividad e inteligencia natural y artificial. En este libro, Hofstadter propone el sistema axiomático MIU, que dejamos como ejercicio en el problema 7.

Problema 7: Acertijo MU

Hay 3 símbolos en este sistema: {M, I, U}
Hay 1 axioma: MI
Hay 4 reglas de inferencia:

Regla 1: De αI se deriva αIU
Regla 2: De Mα se deriva Mαα
Regla 3: De αIIIβ se deriva αUβ
Regla 4: De αUUβ se deriva αβ

donde las variables α y β representan cualquier cadena de símbolos, incluida la cadena vacía.
La pregunta es: ¿es MU una verdad en este sistema?
Por ejemplo, aplicando la regla 2 al único axioma sale MII, que entonces pasa así a ser una nueva verdad del sistema. Y aplicando de nuevo la regla 2 a MII sale MIIII que pasa a ser una nueva verdad. Y aplicando la regla 3 a MIIII sale MUI y también MIU que pasan a ser dos nuevas verdades. Y si comenzamos aplicando la regla 1 a MI sale MIU que pasa a ser una nueva verdad.

Vamos a ver un esbozo informal de la demostración de Gödel, que no es muy difícil pero confunde un poco. Se pueden encontrar otras demostraciones más completas en Smullyan (1987), Nagel y Newman (1994) y Hofstadter (1979). Básicamente Gödel tuvo que dar tres pasos:

- Expresar todas las proposiciones del sistema axiomático usando los símbolos de la lógica de primer orden.
- Codificar las proposiciones en un número de identificación de una manera ingeniosa.
- Plantear una proposición que hable de sí misma (a esto se le llama diagonalización o, en este caso concreto, *gödelización*), utilizando la codificación anterior y generando una contradicción.

Para establecer el procedimiento a seguir primero se deben aceptar algunos supuestos sobre las operaciones básicas permitidas que deben ser suficientemente simples, que no den lugar a ambigüedad y que sean ejecutables en un tiempo finito por medio de un algoritmo. Veremos solo algunos detalles.

El alfabeto de símbolos que requiere la lógica de primer orden es:

$$a\ x\ f\ p\ \neg\ \wedge\ \vee\ \rightarrow\ \leftrightarrow\ \forall\ \exists\ (\ )\ ,\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12 \text{ etc.}$$

Donde las constantes lógicas se expresan con *a* seguido de un número $a_0\ a_1\ a_2\ \ldots$, las variables lógicas con *x* seguido de un número $x_0\ x_1\ x_2\ \ldots$ las funciones[38] son *f* seguido de un número $f_0 f_1 f_2\ \ldots$ y las proposiciones son *p* seguido de un número $p_0\ p_1\ p_2\ \ldots$ El resto son la negación, la conjunción, la

38 Las funciones en nuestro caso son las operaciones de suma + y multiplicación * de números enteros, pero en un caso más general son cualquier tipo de función.

disyunción, la implicación, la doble implicación, el cuantificador universal, el cuantificador existencial, los paréntesis, la coma y los números naturales incluyendo el cero.

Hay muchas formas de codificar estos símbolos en números enteros[39] y una de ellas la podemos ver en la tabla 1, que usa los números impares del 3 en adelante.

Tabla 1. Numeración de Gödel

Símbolo	a	x	f	p	¬	∧	∨	→	↔	∀	∃	(	)	,	0	1	2	3	etc.
Código	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	etc.

Usando los números primos como base y estos códigos como exponente, y haciendo el producto de todo ello, podemos codificar cualquier proposición de manera biunívoca. Por ejemplo, siendo x_1 y x_2 números naturales y f_1 la relación *mayor que*, se puede escribir una proposición que diga que los números naturales son infinitos así:

$$\text{Proposición } p_1\text{: } \forall x_1 \ \exists x_2 \ f_1(x_1, x_2) \qquad \text{Ec. 22}$$

$$3^{21}*5^{5}*7^{33}*11^{23}*13^{5}*17^{35}*19^{7}*23^{33}*29^{25}*31^{5}*37^{33}*41^{5}*43^{35}*47^{27} = \text{NG1} \qquad \text{Ec. 23}$$

Se reservará el número primo 2 para indicar que a continuación sigue una secuencia de proposiciones. La coma dentro de las funciones se puede ignorar sin generar ambigüedad. Las secuencias de proposiciones se van a ligar con la conjunción (∧).

Si hacemos las operaciones matemáticas de la ecuación 6 da un número entero enorme que no soy capaz de poner aquí, y que vamos a llamar el número de Gödel de la proposición p_1 (NG1). Pero lo cierto es que ese número solo admite una única descomposición en factores primos, cuyos exponentes nos devuelven los códigos de los símbolos empleados en la fórmula. Por ello, cada número entero se puede convertir en una única fórmula, de modo que esta codificación es biyectiva. Por supuesto que, a partir de un número arbitrario, su descomposición en factores primos puede generar fórmulas mal construidas, pero eso no importa. Lo que importa es que toda fórmula bien construida tiene un número de Gödel único.

Miremos otro ejemplo. La proposición

$$\exists x \; x>2*x+1 \qquad \text{Ec. 24}$$

39 Por ejemplo, Wiki (2017a, 2017b).

Tiene una única variable libre *x*. Y no podemos decir que esa proposición sea verdadera ni falsa. Pero para el valor concreto de *x=3* se convierte en la sentencia *3>2*3+1* que es falsa. Mientras que para el valor *x=-2* se convierte en la sentencia *-2>2*(-2)+1* que es verdadera.

Ahora imaginemos que tenemos una secuencia de proposiciones *p1 p2 p3 p4* cuyos respectivos números de Gödel son NG1 NG2 NG3 NG4 con las cuales se logra demostrar otra proposición *p5*, cuyo número de Gödel es NG5. Entonces se puede calcular el número de Gödel de la secuencia *p1* ∧ *p2* ∧ *p3* ∧ *p4* así:

$$2^{*}3^{NG1*}5^{13*}7^{NG2*}11^{13*}13^{NG3*}17^{13*}19^{NG4} = NG6 \qquad \text{Ec. 25}$$

Recordemos que el primo 2 lo ponemos porque ahora esto no codifica una proposición sino una secuencia de proposiciones. Y las conjunciones, cuyo código es 13, sirven para ligar esas proposiciones. De este modo, cuando descompongamos NG6 en factores primos y uno de ellos salga 2 sabremos que los otros factores son proposiciones que se deben descomponer a su vez. Por eso es que en los códigos de los símbolos empleamos solo los números impares, para evitar ambigüedad.

En general no va a dar la casualidad de que NG5 coincida con NG6 y, de hecho, lo normal es que NG6 sea muchísimo mayor a NG5.

En español podemos decir que de la secuencia de proposiciones codificada con número de Gödel NG6 sirve para demostrar la proposición con número de Gödel NG5. Definamos entonces la función demostración f_2.

$\exists x_1\ f_2(x_1, x_2)$ significa que la proposición x_2 es demostrable. Ec. 26

Dado que cada número de Gödel identifica biunívocamente una proposición, entonces podemos decir que es verdad que $f_2(NG6, NG5)$, para esos valores concretos.

Definamos también la función de sustitución:

$$f_3(x_1, x_2, x_3) \qquad \text{Ec. 27}$$

Significa que x_3 es el número de Gödel que sale al sustituir por una constante x_2 todas las variables libres (no cuantificadas) de la fórmula cuyo número de Gödel es x_1.

Definamos otra función de autosustitución, como la anterior, pero haciendo $x_1=x_2$, es decir, sustituyendo las variables libres de una fórmula cuyo número de Gödel es x_1 por el número de Gödel x_1. El resultado será una fórmula con número de Gödel x_3. Ello se expresa así:

$$f_4(x_1, x_3) \qquad \text{Ec. 28}$$

Lo siguiente que hay que hacer es construir esta proposición:

$$\neg\exists x_1\, \exists x_2\, (f_2(x_1, x_2) \wedge f_4(x_3, x_2)) \qquad \text{Ec. 29}$$

Calculamos el número de Gödel de esta fórmula y sale NU.

Entonces si hacemos autosustitución de la única variable libre x_3 por NU queda la famosa proposición de Gödel:

$$\exists x_2\, \neg\exists x_1\, (f_2(x_1, x_2) \wedge f_4(\mathrm{NU}, x_2)) \qquad \text{Ec. 30}$$

Cuyo número de Gödel vamos a llamarlo G.

Y que significa que existe una fórmula con número de Gödel x_2 para la que no hay demostración. También dice que esa fórmula se obtiene al autosustituir NU sobre las variables libres de la ecuación 29. Dicho con otras palabras: la fórmula cuyo número de Gödel procede de la autosustitución de NU en la ecuación 29 no es demostrable.

Este formalismo se puede leer en español así:

Proposición G: "no es posible demostrar la proposición G" Ec. 31

Esta proposición quizás sea verdadera o quizás sea falsa. Examinemos cada caso:

- Si la proposición G es verdadera, entonces la proposición G no se puede demostrar, pero hemos supuesto que es verdadera. Por tanto, el sistema es incompleto.
- Si la proposición G es falsa, entonces sí se puede demostrar G. O sea, el sistema es incoherente.

Vaya lío ¿verdad? Hemos creado una proposición que es verdadera pero indemostrable; o es falsa y demostrable a la vez. Y esta contradicción la hemos logrado usando únicamente los números enteros, haciéndoles hablar de sí mismos. Las matemáticas no han sido las mismas desde entonces.

Ahora que ya hemos visto la demostración, quisiera hacer algunos comentarios adicionales.

Las contradicciones pueden parecer destructivas, pero son todo lo contrario. Son fuente de creación y de aprendizaje, y la ciencia progresa gracias a ellas. Por ejemplo, en la época en que solo existían los números naturales (positivos) y yo tenía 1000 en un banco y a continuación saqué 1500, aparentemente ello es una contradicción, pues es imposible de realizar físicamente. En la práctica, acabo de inventar los números negativos, es decir, acabo de pedir un préstamo. Lo mismo pasa con los números complejos y otras generalizaciones del concepto de número, como los tensores, los cuaterniones y los transfinitos. De la misma manera, cuando un experimento

de la física da lugar a una contradicción con las teorías existentes ello es un momento de júbilo, pues significa que se acaba de descubrir algo nuevo.

El teorema de Gödel es una versión sofisticada de la paradoja de Epiménides, el cretense, cuando dijo "Todos los cretenses son mentirosos". La frase de Epiménides solo es verdadera si es falsa, y solo es falsa si es verdadera, lo cual nos lleva a una contradicción. Podemos ya intuir que para lograr contradicciones hay que tener la capacidad de autorreferencia, o sea, sistemas que hablen de sí mismos, como decíamos en el capítulo anterior. Te propongo el problema 8.

Problema 8: Dos autorreferencias divertidas

- Esta frase contiene _____ caracteres
- Esta frase no verbo

Antes del trabajo de Gödel ya se conocían otras paradojas lógicas. Seguramente la más famosa es la de Russell cuyo enunciado está en el problema 9, para que lo pienses un poco.

Problema 9: El barbero de Sevilla

En Sevilla existía un barbero que afeitaba a todas las personas que no se afeitaban a sí mismas. ¿Se afeitaba este barbero a sí mismo?

A veces tantas paradojas y tanto razonamiento contradictorio hacen que sea difícil entendernos a nosotros mismos, como en el problema 10 que está mal resuelto en varios libros.

El razonamiento llamado "diagonalización" que usó Gödel lo inventó Cantor, y es muy elegante y útil. Con él demostró que los números reales no son numerables, es decir, no se pueden poner en correspondencia biyectiva (uno a uno) con los números naturales. De hecho, la cardinalidad del conjunto de los números reales es un infinito mucho mayor que la de los números naturales . O sea, que hay una jerarquía de números infinitos.

El argumento de Cantor es más o menos así: sin perder la generalidad, vamos a listar todos los números reales que hay en el intervalo [0,1) en cualquier orden astuto que uno desee o se haya inventado. Sin perder la generalidad, vamos a representarlos en binario y al primero le asignaremos el número natural 1, al segundo el 2, y así sucesivamente. Como están en binario tendrán la forma de 0.xxxxx donde las x son 0 o 1. Supongamos que ya tenemos la lista en correspondencia con los números naturales, como en el ejemplo de la figura 43.

Problema 10: Paradoja de Richard

Vamos a jugar con otra paradoja que habitualmente no se entiende bien. Para ello:

- Expresemos en español todas las propiedades de los números naturales de la aritmética (ej: "para todo número natural existe otro mayor que él", etc.).
- Ordenemos la lista de propiedades de forma lexicográfica (orden alfabético).
- Numeremos las propiedades (la primera es la número 1; la siguiente es la 2…).
- Como cada propiedad está asociada a un número entero, puede ocurrir que ese número cumpla con la propiedad.
- Ejemplo: si la propiedad 17 dice que "No ser divisible por ningún otro entero excepto por la unidad y por él mismo", entonces el número 17 si cumple con la propiedad 17.
- Ejemplo: si la propiedad 22 dice que "Ser producto de un entero por sí mismo", entonces el número 22 no cumple con la propiedad 22.
- Definición de nueva propiedad: Se llama Richardiano a todo número que no cumple con la propiedad que representa. Y no-Richardiano a los que si cumplen con su respectiva propiedad.
- Esta definición también estará en la lista, de modo que tendrá un número asociado R.

Número	Propiedad	¿Richardiano?
1		
2		
...		
R	No cumplir con la propiedad que representa	?
...		
17	No ser divisible por ningún otro entero excepto por la unidad y por él mismo	No
...		
22	Ser producto de un entero por sí mismo	Si
...		

¿Es R Richardiano?

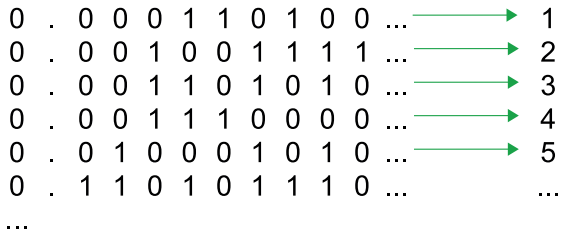


Figura 43. Una posible numeración de los reales

Si ignoramos la primera cifra y el punto de los reales, podemos tomar las cifras de la diagonal y construir un nuevo número intercambiando ceros y unos. En la diagonal (Figura 44) está 001101… que cambiamos por

110010… Ponemos el "0." delante, para que quede dentro del intervalo [0,1) y acabamos de construir un número binario que no está en la lista.

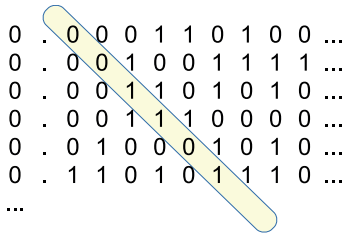


Figura 44. Tomamos la diagonal: 001101... en este ejemplo

Y no lo está porque no puede ser el primero de la lista, ya que la primera cifra decimal no coincide, por la construcción del número. Tampoco puede ser el segundo, porque la segunda cifra no coincide. Tampoco el tercero, pues no coincide la tercera cifra. Lo mismo pasa con los infinitos números de la lista; no puede ser ninguno de ellos, a pesar de que está en el intervalo [0,1). La conclusión es que la hipótesis de partida tiene que ser incorrecta: es imposible crear una lista con todos los números reales del intervalo [0,1).

Problema de la parada de Turing

Habitualmente se diseñan los sistemas axiomáticos de modo que no tengan contradicciones internas pues ello es lo peor que puede ocurrir: si aplicando una secuencia de reglas una proposición sale verdadera, mientras que aplicando otra secuencia la misma proposición sale falsa, entonces cualquier cosa se puede demostrar en ese sistema. En el recuadro 3 se muestra por qué[40].

Recuadro 3: En los sistemas axiomáticos incoherentes se puede demostrar cualquier cosa

- Supongamos que tenemos una contradicción en mi sistema, es decir, $(A \wedge \neg A)$ es verdadero.
- De allí se deduce que A es verdadero y que A es falso, simultáneamente.
- Si A es verdadero, también lo es $(A \vee B)$ siendo B cualquier proposición.
- Como $(A \vee B)$ es verdadero y A es falso, entonces B es verdadero.

Acabamos de demostrar que cualquier proposición B es verdadera. Q.E.D.

40 De todos modos, hay que recordar que existen las lógicas paraconsistentes, donde se permiten cierto tipo de contradicciones (por ejemplo, en las lógicas difusas o las multivaluadas o las temporales) y donde el sistema no se derrumba sino que tiene mucha utilidad.

Entonces, para evitar la incoherencia se opta por aceptar la incompletitud como un mal menor. Por eso, Turing se preguntó si era posible emplear un algoritmo para detectar en un sistema axiomático cuáles proposiciones son indecidibles, para filtrarlas fuera y quedarse con las demás. Empleando su definición de Máquina de Turing Universal demostró que no era posible hacerlo.

En vez de una demostración formal, vamos a verlo con un programa en Ruby, que es más sencillo (Recuadro 4).

Esto tiene consecuencias importantes a la hora de desarrollar *software*: dado que es imposible saber de antemano en un caso general y de forma automática si un programa va a llegar a su final o va a quedarse atrapado en un bucle infinito, entonces tampoco es posible saber si contiene errores. Esto se expresa mejor con una variante de la Ley de Murphy: "Todo programa contiene errores hasta que no se demuestre lo contrario, lo cual es imposible".

Recuadro 4: Demostración del Halting Problem

Hipótesis: se puede escribir una función parada(), que lea un programa y su entrada, lo analice y retorne true si detecta que el programa se va a detener, y false en caso contrario.

```
def parada(programa, entrada)
  # Aquí viene un programa maravilloso,
  # pero los detalles no nos interesan
end
```

Suponiendo que eso es posible, entonces yo puedo escribir el siguiente programa, al que llamaré diagonal.rb:

```
programa = gets   #Lee la entrada y la almacena en un string
if parada(programa, programa)
  loop do            #bucle infinito
end
else
  return true
end
```

Por último, desde el shell de Linux ejecuto:

```
./diagonal.rb < diagonal.rb
```

Y entonces puede ocurrir una de estas dos cosas:

- *Si la función parada(programa, programa) retorna true (recordemos que es una función que tú me diste y que me juraste que era capaz de detectar programas que paran, retornando true, y que no paran, retornando false), entonces mi programa diagonal.rb se queda en un bucle infinito, es decir, no para. Por tanto, tu función no funciona bien, pues se equivoca.*
- *Si la función parada(programa, programa) retorna false, entonces mi programa diagonal.rb retornará true y finalizará su ejecución. De nuevo, tu función se equivocó al predecir que no iba a terminar.*

Por tanto, la hipótesis inicial de que era posible construir la función parada() es falsa. ***Q.E.D.***

Por ello, las metodologías ágiles de programación guiadas por pruebas (como TDD) son muy buenas. Primero se escriben las pruebas, luego se escribe el programa hasta que pase todas las pruebas. Y cada vez que un usuario reporte haber encontrado un nuevo error, se escriben las pruebas correspondientes que, obviamente, el programa no puede pasar. Después se corrige el programa hasta que pase estas nuevas pruebas y también todas las pruebas anteriormente escritas. Con ello se garantiza que antiguos errores ya corregidos no vuelvan a la vida como consecuencia de los cambios realizados para corregir el nuevo error. Podemos así decir que el número de errores del programa irá decreciendo asintóticamente a cero, conforme pasa el tiempo.

Otra alternativa para garantizar programas libres de errores es evitar computación completa, básicamente impidiendo por diseño que aparezcan bucles infinitos (cualquier tipo de bucle infinito, iterativo o recursivo). Esto quita mucha expresividad a cualquier lenguaje de programación, pero de este modo permite en teoría analizar un programa antes de ejecutarlo, en busca de errores. Eso es lo que se plantea, por ejemplo, en las metodologías que usan precondiciones, postcondiciones e invariantes para cada función. O en las metodologías que parten de una especificación matemática para traducirla a un programa.

De alguna manera ello divide la computación en dos (Figura 45): si quieres un lenguaje de programación (o una metodología) con garantías de producir programas libres de errores entonces no puedes aspirar a tener computación completa. Y si quieres inteligencia artificial, que de algún modo implica creatividad, entonces no puedes limitarte a tener una computación no-completa. Se necesita computación completa o incluso más.

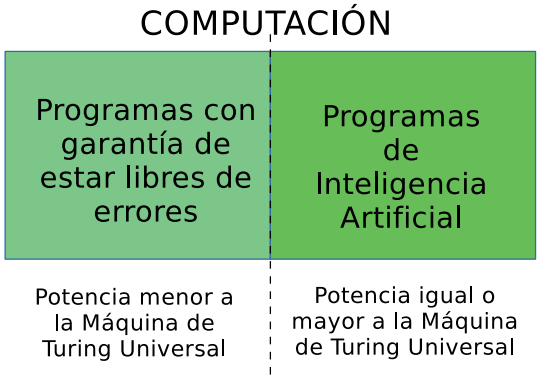


Figura 45. Las dos facetas de la computación

Entonces esto es un límite computacional: para lograr programas de inteligencia artificial hay que renunciar a que estén libres de errores. También se puede ver al revés, de una forma más optimista: los errores son una forma de creatividad. O, dicho más claramente, no se puede esperar verdadera inteligencia sin el riesgo de cometer errores, de equivocarse, de llegar a contradicciones.

Cuando pensamos que el ser humano es inteligente no nos damos cuenta de ello: somos falibles, nos dejamos engañar con facilidad, nos equivocamos con las sumas, lo que decimos un día lo desdecimos al día siguiente, razonamos mal con la lógica. Lo que realmente queremos decir es que la humanidad es inteligente, es decir, el proceso de tener muchos agentes que, colectivamente, van depurando sus errores, eliminando sus contradicciones, seleccionando las verdades (temporales) que mejor resultado les da, y transmitiéndolas por escrito a futuras generaciones. A ese proceso se le llama ciencia.

Complejidad computacional

La complejidad computacional, tanto temporal como espacial, es un tema bien conocido para cualquier científico de la computación. Hagamos un repaso rápido: se definen los **problemas de decisión** como aquellos que tienen una respuesta booleana (sí o no), mientras que los **problemas de optimización** buscan un óptimo entre un conjunto posiblemente infinito de candidatos. Un problema de optimización puede ser el del vendedor viajero: dado un conjunto de ciudades, averiguar en qué orden hay que visitarlas todas para minimizar la distancia recorrida. Y el problema equivalente de decisión es preguntar si existe un orden de recorrido tal que su distancia sea menor de 56 km.

Para resolver los problemas se usan **modelos de cómputo**. El más usado hasta ahora es la Máquina de Turing Universal. Pero existen otros modelos tan fantásticos como la hipercomputación, que se caracteriza porque sus computadores pueden trabajar con números reales en tiempo finito[41]. O tan a punto de convertirse en realidad como la computación cuántica[42], donde, aprove-

41 Por ejemplo, un computador cuya frecuencia de reloj se duplique cada cierto tiempo. Podríamos calcular así la primera cifra de π en un tiempo T, la segunda en T/2, la tercera en T/4, etc. Dado que la serie T+T/2+T/4+… converge a 2T, habremos calculado un número real trascendente en un tiempo finito. Obviamente ese computador nunca se podrá construir en nuestro universo, porque al aumentar la frecuencia del reloj tropezaríamos primero con limitaciones tecnológicas y luego físicas.

42 Ya hay varios computadores cuánticos trabajando muy bien en varios laboratorios del mundo, y falta poco para que se vendan al público, pero lo que definitivamente escasea son los algoritmos que puedan correr de forma efectiva en estas máquinas.

chando el fenómeno de superposición, se puede trabajar simultáneamente con un número enorme de candidatos a solución. Todavía no está claro si la computación cuántica es equivalente a la Máquina de Turing Universal, aunque más rápida, o si se podrá lograr hipercomputación, o si dará lugar a una clasificación intermedia de potencia de cómputo. Hay muchos artículos hablando de ello, pero son contradictorios. Lo que sí está claro es que dado que la inteligencia artificial depende fuertemente del modelo de cómputo, con la llegada de la computación cuántica todo va a cambiar drásticamente y es inminente que los computadores superen a los humanos en cualquier tarea.

A su vez cada modelo de cómputo da lugar a una clasificación de problemas: problemas tratables y problemas intratables (imposibles de resolver en un tiempo razonable). La clasificación más estudiada es la que se hace para las Máquinas de Turing Universales, cuyas clases de problemas más conocidas son (siendo N el número de bits necesarios para codificar los datos de entrada del problema en cuestión usando un código razonable):

- Problemas P (polinomiales): aquellos que se resuelven aplicando un algoritmo determinista en un tiempo máximo proporcional a un polinomio de N ($T \propto N^K$).
- Problemas NP (polinomiales no-deterministas): aquellos que se resuelven aplicando un algoritmo no-determinista en un tiempo máximo proporcional a un polinomio de N. También se pueden definir como problemas EXP pero verificables en tiempo polinomial, es decir, si tengo un candidato a solución, para verificar si realmente lo es me demoro un tiempo polinomial.
- Problemas EXP (exponenciales): aquellos que se resuelven aplicando un algoritmo determinista en un tiempo máximo proporcional a una exponencial de N ($T \propto K^N$).

Ejemplos de problemas P son buscar un elemento concreto en una lista desordenada de N elementos (el tiempo de búsqueda es proporcional a N). Ordenar los elementos de una lista también es un problema polinomial.

Un ejemplo de problema NP es el de decisión del vendedor viajero. La idea es que si tenemos N ciudades hay $N!$ formas distintas de recorrerlas, lo cual crece más rápido que una exponencial. Sin embargo, si me dan una secuencia ordenada de ciudades, me demoro muy poco (solo tengo que hacer N sumas, es decir, la verificación es polinomial) en saber si el recorrido es menor que una cierta distancia dada. Otro problema NP es el de satisfabilidad: dada una función lógica de N variables (de álgebra de Boole) averiguar que valores (0 o 1) deben tomar las variables para que la fórmula ofrezca como resultado 1. Por ejemplo:

$$f = (x_1+x_2+x_3).(x_1+x_3).\ x_3 \qquad \text{Ec. 32}$$

$$g = (x_1+x_2).(x_1+x_3).\ x_1.\ (x_2+x_3) \qquad \text{Ec. 33}$$

La función *f* se satisface con $x_1=0,\ x_2=1,\ x_3=1$ mientras que la función *g* no es satisfactible. Pero para averiguarlo hay que explorar todas las combinaciones posibles, que son 2^N. Pero dada la solución es muy rápido verificarla, en tiempo polinomial.

Un ejemplo de problema EXP es averiguar cuál es la mejor jugada que puedo hacer ante un tablero dado de ajedrez. La única forma que tengo de averiguarlo es probando todas las jugadas posibles hasta llegar al final de la partida. Y aunque me digan cuál jugada es la correcta, para verificarlo también debo de probar todas las jugadas posibles a partir de ahí, de modo que la verificación no es polinomial.

Hay muchas más clases de problemas pero solo nos fijaremos en estas tres pues son las que más frecuentemente aparecen en problemas prácticos reales actuales.

Se han encontrado muchísimos problemas NP en campos muy diversos: teoría de conjuntos, teoría de grafos, búsqueda en árboles, conectividad, trazado de rutas, particiones, compresión de datos, bases de datos, planificación de tareas por sistemas operativos, teoría de números, juegos, lógica proposicional, lenguajes formales, autómatas, etcétera. Los más usados en ámbito académico, aparte del de satisfabilidad y del viajero son el empaquetamiento de la mochila[43], y la asignación de aulas y horarios en una universidad.

Algo muy interesante: se dice que un problema es NP-completo si existe un algoritmo de tiempo polinomial que lo transforma al problema de la satisfabilidad. Los problemas NP-completos son los más complejos dentro de la clase NP. Y esto implica que todos los problemas NP-completos son equivalentes en el sentido de que si se logra solucionar uno de ellos en tiempo polinomial, entonces todos los demás también quedan resueltos. Y también lo contrario: si se demuestra que un problema NP-completo es imposible solucionarlo en tiempo polinomial, entonces es imposible para todos ellos. Hay que advertir que no es lo mismo resolver una instancia de un problema que resolver el problema general, que es lo que nos interesa aquí. Por ejemplo, hay problemas de satisfabilidad muy sencillos de resolver como $f=x_1+x_2$, pero ello no nos vale. Lo que estamos buscando es la solución al problema general. Entonces se sabe que $P \subseteq NP$, pero no se sabe si ambos conjuntos

43 Dada una mochila con una cierta capacidad L y un conjunto de N objetos de tamaños $\{c_1, c_2 \ldots c_N\}$ y con unos valores $\{v_1, v_2 \ldots v_N\}$ se trata de empaquetar los objetos que se puedan sin sobrepasar la capacidad de la mochila y maximizando la suma de sus valores.

coinciden, es decir, si existe un truco, un algoritmo, para resolver en tiempo polinomial los problemas NP. Ello es uno de los llamados "problemas del milenio", que cuentan con un premio de un millón de dólares a quien lo logre resolver.

Quien no conozca los detalles sobre las clases de complejidad computacional puede remitirse a Garey y Johnson (1979). De todos modos, lo que quiero resaltar ahora es la inutilidad de enfrentar problemas NP y EXP aumentando la potencia de cómputo. Obviamente, incluso los problemas EXP son resolubles para valores de N muy pequeños. Pero en cuanto N crece se vuelven intratables, incluso si la tecnología de cómputo avanza sustancialmente. Para entenderlo, en la figura 46 comparamos varios problemas de complejidad temporal N, N^2, N^3 2^N y 3^N, y suponemos que ya hemos logrado solucionarlos para una entrada de tamaño N_1. Entonces, si esperamos unos años hasta que aparezcan computadores 100 veces más rápidos y luego 1000 veces más rápidos, para los problemas polinomiales vemos que merece la pena, mientras que para los intratables (NP y EXP) es muy poco lo que se lograr mejorar. Eso es lo que significa que un problema sea intratable. Y ello es un límite computacional.

CLASE	COMPLEJIDAD	SOLUCIÓN ACTUAL	SOLUCIÓN CON UN COMPUTADOR 100 VECES MÁS RÁPIDO	SOLUCIÓN CON UN COMPUTADOR 1000 VECES MÁS RÁPIDO
P	N	N_1	100 N_1	1000 N_1
P	N^2	N_1	10 N_1	31,6 N_1
P	N^3	N_1	4,64 N_1	10 N_1
NP, EXP	2^N	N_1	N_1+6,64	N_1+9,97
NP, EXP	3^N	N_1	N_1+4,19	N_1+6,29

Figura 46. Diversos problemas y lo que se gana al aumentar la potencia de cómputo

Mientras tanto, hay alternativas para tratar de resolver los problemas NP:

- Modificarlo simplificándolo para convertirlo en P.
- Buscar un algoritmo que corre en tiempo polinomial la mayoría de las veces, aunque no tenga garantías de hacerlo siempre.
- Conformarse con soluciones no-óptimas. Para ello se han desarrollado herramientas que realizan búsquedas heurísticas, como un "genio mágico", un oráculo que predice cual puede ser la solución, que debe luego verificarse aprovechando esta característica de los problemas NP. Algunos de estos algoritmos son las redes neuronales y los algoritmos evolutivos. De alguna forma los algoritmos evolutivos (que vere-

mos en el capítulo con el mismo nombre) están bien adaptados a los problemas NP pues generan candidatos a solución por medio de heurísticas generales (operadores de reproducción) y verifican su bondad con una función de aptitud que es rápida de evaluar.

Teorema del No-Free-Lunch

El teorema de *No-Free-Lunch* (NFL) fue ideado en 1992 por Wolpert y Macready, como un marco donde investigar los problemas de búsqueda, enfocándose en la conexión entre funciones de aptitud y efectividad en la búsqueda. Lo que dice este teorema es: aplicando algoritmos de búsqueda a todos los tipos de problemas existentes, todos los algoritmos se comportan igual en promedio.

Concretamente, si tomamos un algoritmo de búsqueda, se comportará bien (mejor que el promedio) en la mitad de los problemas y mal (peor que el promedio) en la otra mitad. Es importante considerar el universo de todos los problemas posibles, pues de otro modo el NFL no tiene nada que decir.

También se puede formular así (Figura 47): por cada par de algoritmos de búsqueda hay tantos problemas en el que el primer algoritmo es mejor que el segundo como problemas en el que el segundo algoritmo es mejor que el primero. O sea que si comparamos algoritmos genéticos con enfriamiento simulado o incluso con búsqueda aleatoria, y el algoritmo genético se comporta mejor en cierto tipo de problemas, siempre habrá otro tipo de problemas donde se comporte peor.

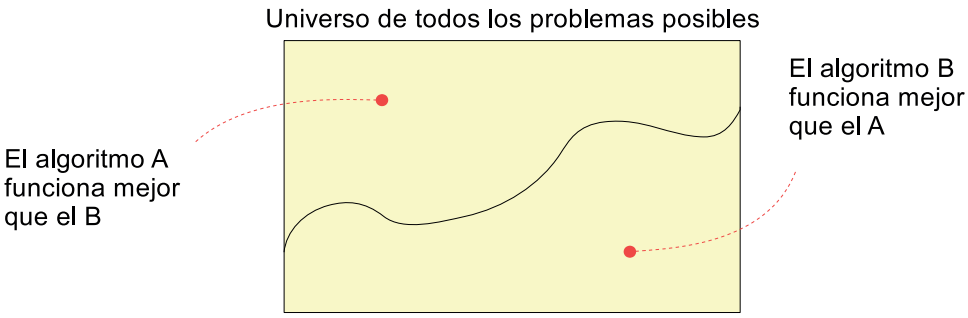


Figura 47. Explicación gráfica del NFL

Una consecuencia es que si no introducimos ningún conocimiento suplementario del dominio del problema en nuestro algoritmo, es tan probable que trabaje peor que la búsqueda estocástica, como que trabaje mejor. Esto es cierto para cualquier tipo de algoritmo de búsqueda. De modo que no se debe elegir un algoritmo basándose únicamente en lo bien que funcionan en

artículos de investigación, sino que es necesario conocer su comportamiento en el dominio del problema que vayamos a tratar (Figura 48).

Y esto debe ser un llamado de advertencia para quien diseña nuevos algoritmos: no basta con mostrar que tu algoritmo es mejor que todos los demás en un caso particular, ya que siempre es posible encontrar ese caso para cualquier algoritmo, aun cuando sea un algoritmo pésimo. Lo importante es mostrar que funciona en un amplio número de problemas, ojalá con relevancia en el mundo real. Por ejemplo, si estás diseñando un nuevo algoritmo para el vendedor viajero, no basta compararlo con otros para un único conjunto de ciudades, ni siquiera para 10 conjuntos. Es más sensato hacerlo con millones de conjuntos de ciudades cuyas posiciones se generen al azar.

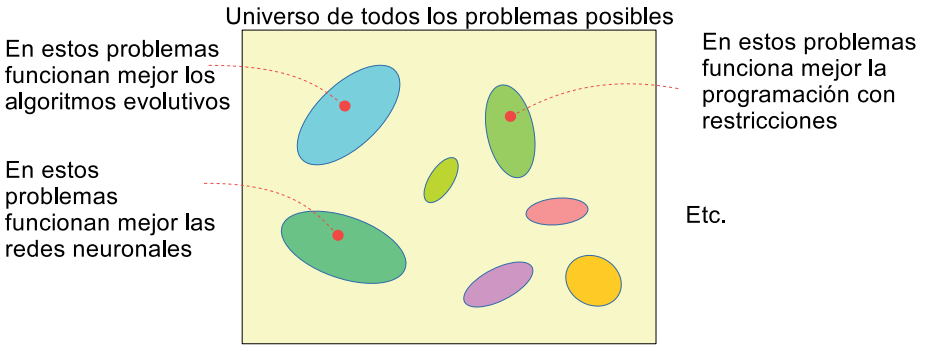


Figura 48. Dominios de problemas donde funciona mejor cada algoritmo

Volviendo al tema, resulta que el teorema NFL aparentemente echa por tierra la esperanza de conseguir el gran algoritmo inteligente que sirva para resolver todo tipo de problemas. Esta limitación computacional parece impedir que se pueda algún día alcanzar la verdadera inteligencia artificial. Porque por más que nos esforcemos en diseñar algoritmos inteligentes, fallarán para la mitad de los problemas que les planteemos. Ello es decepcionante.

En la ciencia ocurre algo similar: las teorías más generales son las que explican menos cosas, y las más particulares las que pueden dar predicciones más detalladas[44].

Sin embargo, algo hay mal interpretado en el NFL porque, por un lado, existen muchos algoritmos (evolutivos, *deep learning*) que funcionan muy bien para resolver problemas en muchas áreas. Además, el NFL conduce a una paradoja: si todo algoritmo requiere información adicional introducida por un humano para afinarlo en un dominio concreto donde funcione bien,

44 Como chiste también se dice que los ingenieros saben de todo un poco, mientras que los PhD saben muchísimo de nada.

eso sitúa a la inteligencia humana de una manera implícita por fuera de los algoritmos. Está diciendo que la inteligencia humana no se puede modelar por medio de ningún algoritmo, ni los clásicos deterministas, ni aún por los no-deterministas. Y si esto es así, se acaba no solo con la inteligencia artificial en sentido "fuerte" sino también con la "débil". Y le daría un sentido "mágico" a la inteligencia humana, que no estamos dispuestos a aceptar aquí (Figura 49).

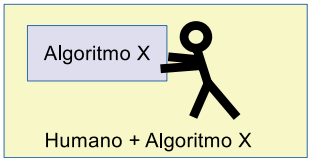


Figura 49. Este nuevo algoritmo (Humano + Algoritmo X) ¿está libre del NFL?

Para salir de esta paradoja tenemos que pensar si hay algo mal interpretado en los NFL. ¿Quizás los problemas que habitualmente tratamos de resolver están sesgados y no abarcan toda la gama de problemas posibles? ¿Es el mundo en que vivimos el que está sesgado? ¿Es nuestra percepción la sesgada? Posiblemente, la respuesta sea "sí" a todas estas preguntas, ya que el mundo es un producto de la evolución natural, por lo que dentro de él no se dan todos los problemas posibles.

En este sentido, el filósofo Malcolm R. Forster (2002) plantea el siguiente experimento mental: supongamos que hay un universo distinto al nuestro, que dura exactamente 2 días y en cada día existe exactamente 1 objeto, que puede ser una esfera o un cubo. El objeto podría ser el mismo o distinto, en cada uno de los 2 días. Entonces, ese universo tiene 4 posibles historias (Figura 50):

	Historia 1	**Historia 2**	**Historia 3**	**Historia 4**
Primer día	esfera	esfera	cubo	cubo
Segundo día	esfera	cubo	esfera	cubo

Figura 50. Posibles historias del universo de Forster

Hay exactamente 4 posibles algoritmos que intentan predecir lo que ocurrirá en el segundo día, sabiendo lo que ocurrió en el primer día:

- Igual: en el segundo día habrá el mismo objeto que en el primer día.
- Diferente: habrá un objeto distinto.
- Esfera: habrá una esfera, independientemente del primer día.
- Cubo: habrá un cubo, independientemente del primer día.

La probabilidad de las cuatro historias la suponemos igual (1/4). Por tanto, la probabilidad de acierto de cada algoritmo predictor es:

- Igual: 1/2
- Diferente: 1/2
- Esfera: 1/2
- Cubo: 1/2

Tal como indica el NFL, no hay ningún algoritmo privilegiado. En el conjunto global de todos los problemas posibles, todos los algoritmos se comportan en promedio igual.

Sin embargo, en nuestro universo real, sabemos que hay "uniformidad en la naturaleza", que se traduce a que las cosas no aparecen y desaparecen por si solas, sin ninguna causa. Ello implica que las únicas historias posibles son la 1 y la 4. Nuestro universo está "sesgado". Y entonces, la probabilidad de acierto de cada algoritmo es:

- Igual: 1
- Diferente: 0
- Esfera: 1/2
- Cubo: 1/2

En este universo sesgado, el algoritmo "igual" es el mejor de todos. Y si el algoritmo de predicción hubiera sido el resultado de un proceso de búsqueda evolutiva, no habría sido necesario proporcionarle ninguna información adicional para que encontrase la solución a cómo funciona nuestro universo.

Es decir, el NFL es inviolable, pero probablemente nunca tengamos que enfrentarnos a resolver todos los problemas posibles. Gracias a ello es que funciona nuestra inteligencia humana, y nada impide que lleguemos a lo mismo con inteligencia artificial. En ambos casos siempre habrá problemas que no se puedan resolver (por ejemplo, descifrar algoritmos de criptografía fuerte).

¿Qué importancia tienen estos límites?

Durante mucho tiempo los principales filósofos pensaban que la mente humana estaba exenta de estas limitaciones debido a que había algún elemento misterioso no sometido a las leyes físicas (Searle, 1997) o a las computacionales (Penrose, 1991, 1994). La ciencia avanza y estas ideas han sido descartadas. Lo más bonito es que ahora podemos investigar y entender cómo el cerebro lidia con estas limitaciones. Vamos a discutir varios ejemplos.

En cualquier empresa hay que asignar recursos a personas para que realicen tareas. Por ejemplo, en las universidades hay que asignar aulas a profesores en determinados horarios, para que puedan dar allí sus asignaturas. Incluso sin tener en cuenta que también hay que asignar estudiantes a cada asignatura, el problema es NP y tiene varias restricciones. Las principales son: cada asignatura tiene un conjunto limitado de profesores que la pueden dar; un profesor no puede estar simultáneamente en dos aulas; y un aula no puede ser asignada a varias asignaturas en horarios que se solapen. Para resolverlo, básicamente hay que probar todas las combinaciones posibles de aulas, asignaturas, profesores y horarios, e ir descartando las que no cumplen con alguna de las restricciones. El objetivo principal a satisfacer son las asignaturas. Conforme el número de asignaturas crece, el tiempo para calcular una solución aumenta exponencialmente. Y no hay garantías de que existan soluciones. Pero, además, ¿de qué sirve saber que hay una solución si el computador va a demorarse 5 años en encontrarla? Y si la tarea se le encomienda a un humano, va a ser mucho más.

La verdad es que sí hay una forma de solucionar problemas cuyo tiempo de cálculo es exponencial, gracias a que las exponenciales al principio tienen un comportamiento muy suave, casi lineal, de aumento casi imperceptible, como se puede ver en la figura 51. Es la función e^n, y allí nos da la impresión de que hasta n=96 pareciera que no está pasando nada.

Por ejemplo, la globalización es un fenómeno exponencial que lleva ocurriendo prácticamente desde que se formó la Tierra hace 4.500 millones de años, siendo los hitos humanos recientes más significativos la salida del *Homo Sapiens* de África hacia Europa y Asia, así como el salto hacia América. Y luego en sentido inverso: la colonización de África, la comunicación con Asia, el descubrimiento de América y la invención de Internet. Pero solo después de que ocurriera este último acontecimiento es que hemos sido conscientes de lo que significa el fenómeno que permite comunicaciones instantáneas con cualquier persona del globo, interacciones económicas y políticas desde y hacia cualquier rincón del planeta, tener noticias de lo que ocurre en cualquier parte así como poder influir también en ello[45], que hasta entonces pasó desapercibido y ni siquiera tenía nombre. Eso es una exponencial. Otro ejemplo más dramático es el calentamiento global, que también es exponencial, pero que nadie lo advierte porque estamos en una fase de subida del mar de unos milímetros cada año, lo cual es imperceptible. La ciencia nos lo cuenta porque tiene equipos sofisticados para medirlo, pero

45 Usando plataformas como Avaaz.org o Change.org, que se están configurando como los nuevos actores de la política global.

no parece que la gente le dé la importancia debida. Y para cuando seamos conscientes de la gravedad del asunto, será demasiado tarde.

Volviendo al problema de asignación de aulas, horarios y profesores a asignaturas, se puede solucionar fácil y rápidamente si tenemos mucha holgura, de modo que los tiempos de cómputo sean rápidos porque hay muchísimas soluciones posibles. Holgura en este caso significa tener muchas aulas y profesores, dado que los horarios no se pueden estirar (en una semana se puede disponer de 6 días laborables x 14 horas hábiles/día = 84 horas), siendo imposible físicamente superar las 168 horas/semana. De modo que la única solución es contratar más profesores y construir más aulas. Pero entendamos bien qué significa este concepto de holgura.

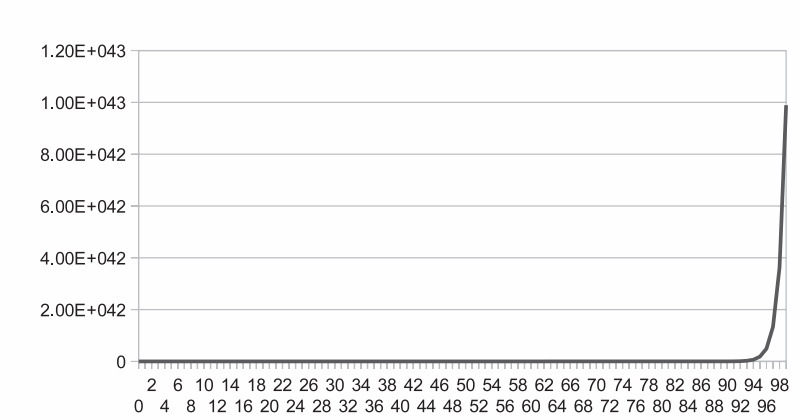


Figura 51. Cien primeros valores de e^n

Si no hay holgura, el problema no tiene solución. Por ejemplo, si hay muchos profesores pero solo hay 2 aulas, y el número de asignaturas es 140 (suponiendo que cada asignatura dure 3 horas/semana), entonces se requieren 3*140/2 horas/semana = 210 horas/semana. Como ya dijimos, la semana no tiene tantas horas. Es imposible asignar todas las asignaturas.

Si construimos muchas aulas, por ejemplo 10, habrá mucha holgura: Los cálculos ahora nos dan 3*140/10 horas/semana = 42 horas/semana. Dado que la semana dispone de 84 horas, tenemos un 50% de aulas desocupadas a lo largo de la semana. Hay una holgura del 50%. Y eso, que puede parecer ineficiente, es muy bueno desde el punto de vista de la efectividad de asignar las aulas.

También podemos verlo de esta manera: si tenemos un tiempo fijo limitado para solucionar un problema, conforme las variables restrictivas

del problema aumentan (número de estudiantes, número de asignaturas), la cantidad de posibilidades que se deben analizar crece exponencialmente (Figura 52).

Figura 52. Crecimiento exponencial de las combinaciones a analizar

Los científicos de la computación saben que si el tiempo de cómputo es limitado, como siempre ocurre, el número de combinaciones que se alcancen a explorar será una fracción muy pequeña del total. De modo que la probabilidad de encontrar una solución decrece bruscamente a cero al superarse un cierto umbral en el tamaño del problema (Figura 53).

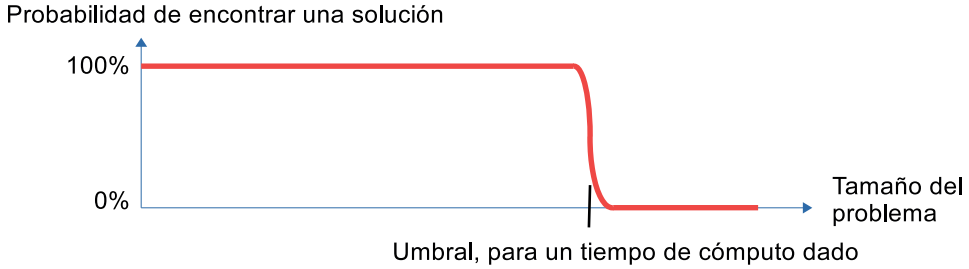


Figura 53. Probabilidad de encontrar una solución con tiempo de cómputo limitado

Un gerente observa lo mismo pero desde otro punto de vista: la probabilidad de encontrar una solución aumentará bruscamente cuando la holgura supere un cierto umbral (Figura 54), que es el mismo de la exponencial. En problemas NP esta transición es muy brusca. Tener o no tener holgura (por encima o por debajo de un cierto umbral) produce un cambio de fase entre lograr o no lograr solucionar el problema. Este cambio de fase es típico de sistemas complejos, de modo que los problemas NP generan una complejidad sistémica y no solo algorítmica.

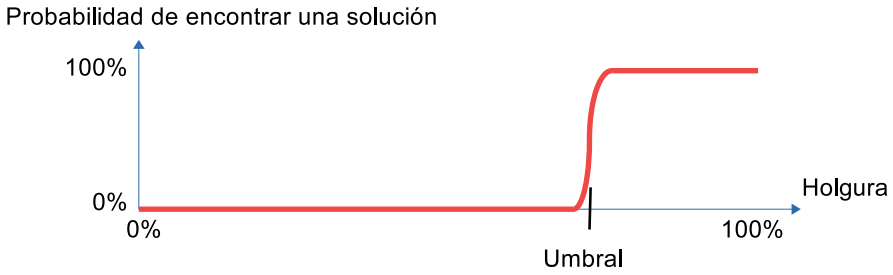


Figura 54. Probabilidad de encontrar solución, en función de la holgura que haya

Podemos calcular la holgura en este caso como resta de dos rectángulos (Figura 55).

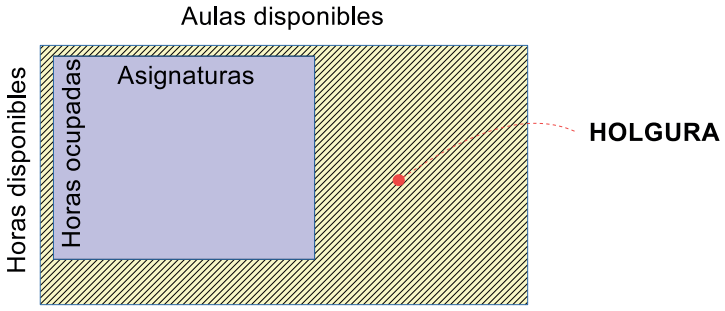


Figura 55. Holgura = (Rectángulo disponible - Rectángulo ocupado) / Rectángulo ocupado

Pero eso no es suficiente para entender la holgura. En problemas combinatorios como este, un 50% de holgura (como la que hay en la figura 56) puede ser poco para solucionarlo y ello depende de la complejidad de los detalles. Se requerirá más holgura si hay muchas restricciones, si las aulas no son todas del mismo tamaño, sino que hay mucha diversidad, si las asignaturas no ocupan todas la misma cantidad de horas, si hay asignaturas compartidas por más de un profesor, si algunas sesiones no se dan en el aula sino en otro espacio (laboratorio), etc.

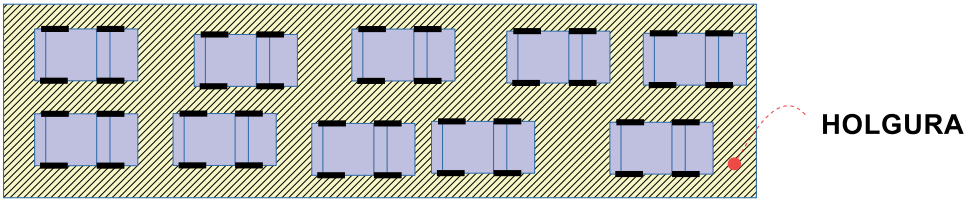


Figura 56. Una calle con automóviles. Hay congestión a pesar de que la holgura parece alta

Para entender por qué un modelo de eficiencia basado en rectángulos no es suficiente, consideremos un problema más intuitivo: el tráfico en una ciudad. Un alcalde que no quiera gastar en construir más calles podría preguntar el área asfaltada de la ciudad, el número de automóviles que circula por allí y el área que ocupa cada automóvil. Supongamos que son respectivamente 2 000 000 m^2, 100 000 automóviles y 10 m^2. Este alcalde podría decir que en su ciudad caben 2 000 000 m^2 / 10 m^2 = 200 000 automóviles. Y que nadie debe quejarse, pues en este caso hay solo la mitad, o sea, que caben todavía 100 000 más. ¡Pobre ciudad si tiene un alcalde así! No entiende la importancia de la holgura y, obviamente, nadie desearía vivir en esa ciudad tan congestionada.

Incluso en un estacionamiento público donde la mayoría de los vehículos están detenidos y no necesitan holgura delante para poder frenar, se requiere que alrededor de un 50% del área disponible esté libre para que los automóviles y peatones puedan movilizarse y abrir las puertas. Otro ejemplo similar lo vemos en la figura 57, que muestra un atasco habitual en nuestras ciudades donde, a simple vista, podemos estimar que hay todavía una holgura de más o menos un 50% entre los automóviles. Y, a pesar de ello, hay congestión. Si disminuyéramos la holgura al 0% nadie podría abrir las puertas de su vehículo ni nadie podría avanzar. Los automóviles quedarían en una especie de fase sólida para siempre.

Figura 57. Congestión vehicular en la Puerta de Alcalá de Madrid

Por tanto, que haya holgura es importante para que los sistemas funcionen y sean computables. Y nunca debería verse la holgura como un desperdicio de recursos, sino más bien como la flexibilidad que requiere un sistema para operar normalmente e incluso para poder reaccionar rápidamente ante imprevistos.

¿Cuánta holgura se requiere? Depende de cada sistema, y suele ser difícil de calcular. Es más rápido hallarlo experimentalmente. Sin embargo, en algunos casos sí se conoce teóricamente cuál es el umbral donde una holgura menor hace el sistema inviable. Por ejemplo, otro recurso muy importante que se suele asignar en cualquier empresa es el tiempo. Típicamente cada empleado debe hacer un conjunto de tareas, la mayoría de las cuales son periódicas y se conocen de antemano. Así, un profesor universitario debe dar clases, atender reuniones administrativas, participar en grupos de investigación, asistir a conferencias y otras responsabilidades. Muchas de estas tareas tienen periodicidad semanal, semestral o anual. Los algoritmos de planeación de tareas se conocen desde los albores de la computación (pues el sistema operativo sufre del mismo problema de asignar tiempos de *CPU* a tareas), y el primer límite teórico de lo que se puede lograr fue calculado por Liu y Layland en 1973: si todas las tareas son periódicas y conocidas de antemano, y si no hay restricciones entre ellas (recursos compartidos, prioridades, etc.) entonces lo máximo que se puede ocupar una *CPU* es un 69%. Es decir, es imposible dar garantías si la holgura es menor al 31%. Remarquémoslo: debemos conformarnos con que haya un 31% del tiempo disponible que no se use para nada (o como mucho para tareas de baja prioridad que no importe si se quedan sin hacer), si queremos que las tareas asignadas se ejecuten sin problemas. Y este es el mejor caso. Conforme aumentan las restricciones, la holgura que debe dejarse puede ser mucho mayor. Lo que Liu y Layland obtuvieron para *CPUs* de computador es igualmente aplicable a personas, de modo que tratar de planificar el 100% del tiempo de una persona en tareas de obligatorio cumplimiento es un disparate. Lo malo es que los administradores de empresas no suelen conocer estos resultados de ciencias de la computación.

La habitación china

El filósofo y lingüista John Searle y el matemático y filósofo Roger Penrose son los máximos exponentes de una corriente que usa el teorema de Gödel como prueba de que las máquinas jamás podrán ser inteligentes.

Como ambos atacan el funcionalismo y cualquier teoría computacional de la mente, vamos a analizar aquí sus ideas y a desmontarlas.

Su argumento es que el teorema de Gödel limita lo que un formalismo (que ellos identifican correctamente con un programa de computador determinista) puede llegar a hacer. Mientras que, para ellos, la mente humana no sufre de esas limitaciones. Por tanto, los programas de computador nunca podrán llegar a ser tan inteligentes como los humanos. Lo más que pueden hacer es simular algunos procesos mentales, lo que se llama "inteligencia

artificial débil" como jugar muy bien ajedrez. Pero tener pensamientos y sensaciones humanas (lo que se llama "inteligencia artificial fuerte") está por fuera de sus posibilidades.

A Searle y a Penrose se les puede criticar tres cosas:

- Identifican los formalismos de Gödel con programas de computador, lo cual es correcto siempre y cuando esos programas no incluyan aleatoriedad. Es decir, es válido para formalismos computacionales como los programas en LISP. Pero cuando se incluye aleatoriedad, por ejemplo, permitiendo que una proposición sea a veces verdadera y a veces falsa, o que coexistan proposiciones afirmando algo y su contrario simultáneamente (como ocurre con los cromosomas de un algoritmo evolutivo) el teorema de Gödel no limita las verdades alcanzables. Dicho de otra manera, un formalismo que acepte contradicciones internas de una manera creativa y no destructiva, evita el límite de las verdades inalcanzables.
- Ambos pensadores se sienten muy inteligentes. Se miran en su interior y no creen que tengan ninguna limitación. Eso es un vano orgullo. Todas las personas tenemos limitaciones y los grandes desarrollos en matemáticas, ciencias y tecnología se los debemos al conjunto de la humanidad, a los vivos y a los ya fallecidos. No hay una única persona que pueda presumir de entender y potencialmente haber podido llevar a cabo cualquier avance científico. Y la ciencia funciona de manera similar a los algoritmos evolutivos, si hacemos el símil de una persona igual a un cromosoma. Entre los científicos hay contradicciones y hay cambios de opinión constantemente. No se puede decir que sean modelables con un formalismo matemático en el cual, una vez establecida la verdad, es verdad para siempre, como cuando decimos que 2+2=4. Lejos de eso, una vez establecida una verdad científica (llamada teoría), no suele pasar mucho tiempo hasta que alguien encuentra un experimento que no casa, y hay que sustituirla por otra teoría más sofisticada. El teorema de Gödel es completamente aplicable a cada humano y también a la humanidad, y para alcanzar nuevas verdades tenemos que aceptar la falibilidad y las contradicciones.
- Como repetiremos varias veces a lo largo del libro, no hay forma de distinguir lo real de lo simulado, siempre que tengan el mismo grado de complejidad. De modo que la diferencia entre inteligencia artificial fuerte y débil es una falsa dicotomía.

Además, Searle creó un experimento mental para asentar más aún esa idea, la habitación china, que pasaremos a explicar:

Searle no sabe nada de chino, pero está en una habitación cerrada que se comunica con el exterior por una rendija. A través de ella recibe papeles con mensajes en chino, que no entiende, pero tiene un libro enorme que le explica en su idioma como manipular esos símbolos (con reglas de tipo *if-then-else*) para generar como respuesta otro mensaje adecuado en chino, y entregarlo hacia afuera por la rendija.

Searle simula ser un computador y su enorme libro es el programa con las instrucciones para manipular símbolos, resultado de años de investigaciones en inteligencia artificial fuerte.

La persona china que esté afuera, puede conversar con la habitación intercambiando papelitos pero —y aquí viene la paradoja— nadie en esa habitación comprende chino.

Como resumen de la paradoja, Searle afirma que:

- Los programas de computador son solo sintaxis (lenguajes formales).
- Las mentes humanas tienen semántica.
- La sintaxis no es suficiente para construir la semántica.

Y con ello llega a una contradicción, lo que indica que la habitación china es irrealizable. Es decir, el programa de computador con las instrucciones para manipular símbolos es una entelequia.

Se ha escrito demasiado sobre la habitación china, para apoyarla y para refutarla. Simplemente quiero proporcionar mi punto de vista. El error está en que Searle separa la sintaxis y la semántica como mundos completamente aislados e incomunicados. Pero lo que sabemos actualmente es que la semántica se construye acumulando muchas reglas sintácticas. A veces son reglas deterministas, que siempre se cumplen. A veces son reglas probabilistas con probabilidades condicionadas al disparo de otras reglas. A veces son reglas que pueden coexistir en contradicción. Eso es la semántica. No hay nada más.

En el manejo del lenguaje cada vez se alcanzan nuevos éxitos, como los *chatBot*, los traductores automáticos, los programas que hacen resúmenes de textos, los generadores de poesía y de novelas. Tienen fallos pero cada vez que acumulan nuevas reglas van mejorando. Van construyendo su semántica.

Los humanos somos robots biológicos muy complejos y la semántica también procede de la acumulación de mucha sintaxis. Además, somos propensos a errores, como los computadores. A veces incluso es preferible un error simple de una máquina, fácilmente identificable y corregible, que un error de un humano que entiende, pero mal[46]. Pero con el tiempo las

46 Como ejemplo que acaba de ocurrir, un revisor de inglés, experto nativo, me cambió la frase "[…] similar processes can take place on human beings" ("[…] procesos similares pueden ocurrir en se-

máquinas alcanzarán la misma semántica y las mismas posibilidades de error que los humanos.

Para terminar, quiero mencionar que Penrose supone que dentro de la tubulina, una molécula que conforma el citoesqueleto de las neuronas, ocurren procesos mecánicocuánticos que otorgan al cerebro una capacidad de cómputo que viola el teorema de Gödel. Desde que lo formuló sonaba mal, pues no está claro cómo se puede violar un teorema matemático a través de un proceso físico. Y conforme pasa el tiempo ni se encuentran fenómenos especiales dentro de la tubulina[47], ni tampoco se ha encontrado un solo caso de una máquina cuántica violando el problema de la parada. Para aclarar bien las cosas recordemos que el teorema de Gödel dice que cualquier sistema formal suficientemente complejo, o tiene contradicciones internas o hay verdades que no puede demostrar. Los algoritmos evolutivos y, en general, los algoritmos que usan probabilidades o variables estocásticas, aceptan tener contradicciones internas, de modo que se vuelve posible alcanzar todas las verdades sin violar este teorema.

Resumen

Usando autómatas celulares la computación universal es muy fácil de lograr de forma espontánea: se requiere una estructura espacial, localmente conectada, y unos 8 bits de información. Y una vez que se tiene computación completa, se abre el camino para la inteligencia artificial, a la vez que se pueden fabricar simuladores. Hemos visto cómo funciona uno (que puede servir simular ambientes en juegos, circuitos electrónicos, procesos industriales, o nuevos mundos) y hemos resaltado que requiere dos dimensiones temporales.

También hemos visto que la computación universal tiene tres límites fundamentales. Primero vimos el teorema de Gödel junto con el problema de la parada, que desde un cierto punto de vista, más que una limitación lo que indican realmente es que las matemáticas y la computación son creativas.

Después vimos la complejidad computacional, que indica que hay muchos problemas que requieren una cantidad de tiempo enorme para resolverse, por lo que son intratables en la práctica. La forma de saltar esa limitación es emplear aproximaciones o algoritmos probabilistas (como los

res humanos" que tiene un pequeño error sintáctico) por "[…] similar processes can take the place of human beings" ("[…] procesos similares pueden sustituir a los seres humanos" que contiene una grave alteración de significado).

47 Durante la edición de este libro, parece que ya se han encontrado. Pero eso no cambia el resto de la argumentación.

evolutivos), que no tienen garantías de funcionar siempre, pero que pueden dar buenos resultados en muchas ocasiones.

Por último vimos el teorema de *No-Free-Lunch* que dice que para el conjunto de todos los problemas de búsqueda posibles, no hay ningún algoritmo que funcione mejor que otro. Y vimos que este límite se puede superar siempre que no tratemos de abarcar todos los problemas posibles.

Para saber más

- Douglas Hofstadter (1979). *Gödel, Escher, Bach: un eterno y grácil bucle*. Barcelona: Tusquets Editores.

A pesar de su antigüedad, sigue siendo un gran libro, de los que se leen una y otra vez, ganando más en conocimiento en cada oportunidad. Y se sigue editando, pues la última versión en inglés es de 1999. Además, se puede leer en español, pues la traducción está muy bien lograda. Hofstadter nos presenta una perspectiva inusual de la inteligencia artificial, mezclando de forma muy amena las preguntas filosóficas con teoremas matemáticos, discusiones de casos, temas de computación, fractales y caos, así como cuentos divertidos que ejemplifican algunos de los tópicos. El hilo conductor del libro es la autorreferencia como generadora de complejidad, que se observa tanto en matemáticas (Gödel) como en dibujo (Escher) y música (Bach). Y en tantos otros campos. Reconozco que este es el principal libro que me inspira a escribir el mío.

Referencias

Libros, artículos y enlaces web

Carroll, L. (1995). *Matemática demente*. Barcelona: Tusquets Editores.

Chaitin, G. J. (1999). *The Unknowable*. Singapore: Springer-Verlag.

Deutsch, D. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society, 400*, pp. 97–117. London. DOI: https://doi.org/10.1098/rspa.1985.0070

Erdös, P. y Renyi, A. (1960). *On the Evolution of Random Graphs*. Mat. Kutato. Int. Kozl. 5, pp. 17-61.

Forster, M. R. (1999). Notice: No-Free-Lunches for Anyone, Bayesians Included. *11th International Congress of Logic, Methodology and Philosophy of Science*. Cracow, Poland. Recuperado el 27 de agosto de 2017. Disponible en: https://goo.gl/GQYCo3

Garey, M. R. y Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*. New York: W. H. Freemand and Company.

Green, D. G. y Newth, D. (2005). Towards a Theory of Everything? - Grand Challenges in Complexity and Informatics. *Complexity International, 8*.

Liu, C. L. y Layland, J. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM, 20*(1), pp. 46-61. DOI: https://doi.org/10.1145/321738.321743

Mackey, P. (2008). *Adventures of a Wetware Hacker*. Recuperado el 12 de junio de 2017. Disponible en: https://goo.gl/Nqg3eT

Mitchel, M. y Crutchfield, J. P. (1995). The Evolution of Emergent Computation. *Proceedings of the National Academy of Science of the United States of America*, *92*(23), pp. 10742-10746.

Nagel, E. y Newman, J. R. (1994). *El teorema de Gödel*. Madrid: Editorial Tecnos.

Penrose, R. (1991). *La nueva mente del emperador*. Barcelona: Grijalbo-Mondadori.

___. (1994). *Las sombras de la mente*. Barcelona: Grijalbo-Mondadori.

Raichle, M. E. (2010). Two Views of Brain Function. *Trends in Cognitive Sciences, 14*(4), pp. 180-190.

Searle, J. R. (1997). *El misterio de la conciencia*. Barcelona: Paidós.

Smullyan, R. (1987). *Forever Undecided. A Puzzle Guide to Gödel*. New York: Knopf.

Wiki (2017a). *Numeración de Gödel*. Wikipedia. Recuperado el 16 de mayo de 2017. Disponible en: https://goo.gl/5WdCPd

___. (2017b). *Gödel's incompleteness theorems*. Wikipedia. Recuperado el 22 de noviembre de 2017. Disponible en: https://goo.gl/C2YDac

Wolfram, S. (2002). *A New Kind of Science*. Canadá: Wolfram Media Inc.

Wolpert, D. H. y Macready, W. G. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), pp. 67-82.

Películas y videos

Bradbury, P. (2012). *Life in life*. Recuperado el 27 de agosto de 2017. Disponible en: https://goo.gl/jJhSuo

Ikergarcia1996 (2016). *Minecraft - Maquina de Turing | Redstone*. Recuperado el 27 de agosto de 2017. Disponible en: https://goo.gl/MZKMrP

IN6TV (2012). *Máquina de Turing - Lego*. Recuperado el 27 de agosto de 2017. Disponible en: https://goo.gl/Ac3wd8

Tesis y trabajos de grado en EVALAB

Cruz, M. A. (2014). *Búsqueda de solapamiento en clusters, usando técnicas de computación evolutiva*. Cali: Universidad del Valle.

Vélez, P. A. (2013). Software *para la búsqueda de espacios discretos que se aproximen al espacio continuo euclidiano*. Cali: Universidad del Valle.