Diagonalización y su impacto en el desarrollo de *Software*

José de Jesús Lavalle Martínez

Marzo de 2004

Resumen

La noción de computabilidad, y sus formalizaciones, es central en el estudio de la disciplina computacional; la idea informal consiste en que se pueda realizar una operación por medio de un conjunto de pasos bien definidos (no ambiguos) y finitos (usualmente llamado algoritmo), sin que se tenga que utilizar la "creatividad". De tal manera que incluso un dispositivo automático pueda efectuar dicha operación. Por supuesto, en nuestros tiempos eso significa que una computadora convencional pueda realizar la mencionada operación.

Cabe señalar que una operación puede llevarse a cabo de diversas formas o algoritmos, de tal manera que dos algoritmos efectuarán la misma operación si, ante la misma entrada simbólica, producen la misma salida simbólica.

De mayor importancia es preguntarnos si acaso existe al menos un algoritmo para producir determinada operación, en otras palabras, si hay operaciones que no puedan realizarse por medios algorítmicos.

La respuesta a este planteamiento es que, efectivamente, hay operaciones que *no* pueden realizarse algorítmicamente; la técnica para demostrar está afirmación se basa en el concepto de *diagonalización*, introducido por Cantor para demostrar que el conjunto de los números reales y el conjunto de los números naturales no son equinumerables.

A continuación se presentarán diferentes ejemplos del uso de la diagonalización para demostrar la no numerabilidad de ciertas operaciones, para establecer así los límites de lo computable. Intuitivamente, lo anterior significa que hay más operaciones que algoritmos para calcularlas.

1. No numerabilidad del conjunto de los números reales

Como es bien conocido, desde el tiempo de Cantor, los números reales son isomorfos al intervalo abierto $(0, 1)$. Grosso modo podemos listar en orden (enumerar) simbólicamente los números en el intervalo abierto $(0, 1)$, como a continuación se indica:

$$\begin{array}{ll} 0 & 0.a_{0,0}a_{0,1}\cdots a_{0,i}\cdots \\ 1 & 0.a_{1,0}a_{1,1}\cdots a_{1,i}\cdots \\ & \vdots \\ i & 0.a_{i,0}a_{i,1}\cdots a_{i,i}\cdots \\ & \vdots \end{array}$$

donde $\quad a_{k,l} \in \{0, 1, \ldots, 9\} \quad$ con $\quad k, l \geq 0.$

Ahora construyamos el siguiente número d:

$$d = 0.d_0 d_1 \cdots d_i \cdots \quad \text{donde} \quad d_j = 9 - a_{j,j}, \quad j \geq 0.$$

Como podemos observar el número d está en el intervalo $(0, 1)$, por lo tanto (y por pura curiosidad) cabe preguntarnos en qué posición de la lista está d (cuál de los números naturales le corresponde).

Supongamos que le corresponde la posición l, entonces encontramos que la igualdad $d_l = a_{l,l}$ se debe cumplir; recordando cómo se formó a d_l, se tendría que $9 - a_{l,l} = a_{l,l}$ o que $9 = 2a_{l,l}$ lo cual es imposible ya que 9 es impar. Nótese que hemos escogido un número l arbitrario, por lo que el resultado anterior no depende del número que se seleccione (aún concediendo que $d_j = a_{l,j}$, para $j \geq 0$, excepto, por supuesto, cuando $j = l$).

En suma, lo que hemos encontrado con nuestros razonamientos es que el número d que está en el intervalo $(0, 1)$ no puede estar en la supuesta lista de números reales en $(0, 1)$. En otras palabras, que existen más números reales (al menos uno, nuestro curioso d) que números naturales.

El "truco"que usamos para formar al número d fue tomar los elementos (dígitos) "diagonales"(por ello la técnica se llama diagonalización) de la matriz obtenida al listar los números reales en $(0, 1)$, e "invertir"su valor (en este caso restándolos a 9), de tal manera que se

garantiza que el número d nunca puede coincidir con ningún número en la lista, al menos en el dígito ubicado en la diagonal.

2. La paradoja de Russell

Otro ejemplo del uso de la diagonalización lo podemos encontrar en la famosa paradoja enunciada por Russell, quien propone (nuevamente por pura curiosidad) que consideremos al conjunto de todos los conjuntos tales que no se pertenecen a sí mismos, simbólicamente:

$$S = \{x | x \notin x\}$$

nótese que este conjunto no está vacío, piense en cualquier conjunto, por ejemplo, $\{1, 2, 3\}$ está en S, ya que él no es elemento de sí mismo, es decir, $\{1, 2, 3\} \notin \{1, 2, 3\}$. Ahora, ¿$S \in S$?, para averiguarlo basta considerar dos casos:

1. Suponer que la respuesta es afirmativa, pero si $S \in S$ entonces por la condición que caracteriza a los elementos de S tenemos que $S \notin S$, lo cual es una contradicción,
2. Suponer que la respuesta es negativa, pero si $S \notin S$ entonces cumple con la condición que cumplen los elementos de S, por lo que debemos tener que $S \in S$, lo cual es nuevamente una contradicción.

En otra palabras hemos encontrado que $S \in S$ si y sólo si $S \notin S$, lo que es muy grave, porque se supone que siempre se puede contestar a la pregunta simple de si un elemento está o no está en un conjunto, sólo hace falta verificar si el elemento se encuentra o no se encuentra presente en el conjunto de interés. Dicho de otra forma, hemos hallado un elemento S y un conjunto S (¡también!), donde no podemos constatar la pertenencia o no pertenencia del elemento al conjunto, la simple pregunta conlleva una contradicción.

3. Funciones de números naturales en el conjunto $\{0, 1\}$

Hasta este momento hemos considerado objetos matemáticos "muy grandes", el conjunto de todos los números reales, el conjunto de todos los conjuntos que no se pertenecen a sí mismos.

¿Qué pasa si consideramos algo "más pequeño", algo como el conjunto de todas las funciones que tienen como dominio los números naturales y como rango el conjunto $\{0, 1\}$? Nuevamente podemos suponer que a este conjunto lo podemos asociar con los números naturales de tal manera que f_i es la función asociada con el $i-$ésimo número natural. Podemos definir la siguiente función que tiene como dominio a los números naturales:

$$f(n) = \begin{cases} 1 & \text{si } f_n(n) = 0 \\ 0 & \text{si } f_n(n) = 1 \end{cases} \tag{1}$$

Esta nueva función que hemos definido también tiene como rango al intervalo $\{0, 1\}$, por lo tanto debe estar asociado con un número natural, digamos j, en símbolos $f(n) = f_j(n)$ pero:

1. Si $f(j) = 1$ es porque $f_j(j) = 0$,
2. Si $f(j) = 0$ es porque $f_j(j) = 1$.

lo cual es nuevamente una contradicción; así, la función f tal y como la hemos definido no tiene asociado, o no se le puede corresponder con, un número natural; también podemos decir que hay más funciones de los números naturales en $\{0, 1\}$ que números naturales.

4. Dispositivo automático universal

Nuestra experiencia cotidiana nos dice que existen dispositivos automáticos en casi todas partes. Por ejemplo, los hornos de microondas, los reproductores de: VHS, CD, DVD; el sistema de inyección de gasolina de los automóviles; los cajeros automáticos; las cajas de juegos de vídeo; etc.

Si bien, y sin lugar a dudas, tales dispositivos automáticos son muy útiles, económicamente sería desastroso que tuviéramos que pagar, por ejemplo, por un dispositivo para cada una de las tareas siguientes: procesar un documento; editar, compilar y ejecutar un programa; hacer una presentación; leer el correo electrónico; etc.

Suponiendo sin conceder, que tuviéramos el dinero suficiente para pagar por tales *dispositivos automáticos de propósito específico*, el panorama sería muy desagradable, en el sentido de que en cada lugar donde ahora hay una computadora, parecería, en el mejor de los casos, una miscelanea o, en el peor, una gran plaza comercial, dividida en secciones, al menos una por cada servicio que quisiéramos obtener.

Como se insinuó en el párrafo anterior, afortunadamente tenemos *dispositivos automáticos universales*, mejor conocidos como computadoras de propósito general, las cuales sirven para realizar una "gran cantidad"de tareas por el mismo precio.

A grandes rasgos podemos decir que un dispositivo automático universal recibe un par de datos; el primero representa la descripción (o codificación, o programa) de un dispositivo automático, y el segundo representa los datos de entrada con los que trabajaría el dispositivo automático.

Posteriormente el dispositivo automático universal (dau), basándose en la descripción (d), simula lo que haría el dispositivo automático con los datos de entrada (in), simbólicamente:

$$dau(d, in) \mapsto d(in)$$

5. Los límites

Ahora queremos hallar cuáles son los límites de lo computable, o cuáles son los límites de lo que pueden hacer los dispositivos automáticos.

Todos los que hemos tenido que escribir un programa de computadora (la descripción de un dispositivo automático) sabemos que una de las principales preocupaciones es saber si nuestro programa aplicado a los datos de entrada terminará (parará, lo que denotaremos por $\downarrow$) o se quedará ejecutando siempre (se ciclará, lo que denotaremos por $\uparrow$).

Por lo tanto sería buena idea diseñar un dispositivo automático especial (dae) que, dado cualquier par (d, in), nos diga **sí** cuando $d(in) \downarrow$ y luego pare. En el otro caso, que nos diga **no** cuando $d(in) \uparrow$, y luego pare, simbólicamente:

$$dae(d, in) = \begin{cases} \text{sí} \wedge \downarrow & \text{cuando } d(in) \downarrow \\ \text{no} \wedge \downarrow & \text{cuando } d(in) \uparrow \end{cases} \tag{2}$$

Nótese que nuestro dispositivo automático universal ejecuta parte de lo que queremos que realice nuestro dispositivo automático especial, "sólo"le falta decir sí o no, y parar, dependiendo de cuando $d(in)$ pare o no.

Para simplificar un poco las cosas en lugar de pedirle al dae que simule a cualquier dispositivo d con cualquier entrada in, vamos a

definir un nuevo dispositivo automático especial dae' que sólo trabaje con pares (d, d), es decir, queremos saber si un dispositivo automático para o no cuando se le da como entrada su propia descripción (no es tan extraño, piense en un compilador, por ejemplo de C, al que se le pide compilar su propio código), en símbolos:

$$dae'(d) = \begin{cases} \text{sí} \wedge \downarrow & \text{cuando } d(d) \downarrow \\ \text{no} \wedge \downarrow & \text{cuando } d(d) \uparrow \end{cases} \tag{3}$$

Ahora definamos un dae'' "mentiroso" que haga lo siguiente, que se cicle indefinidamente (lo cual no es difícil, sólo basta hacerlo entrar a un bucle cuya condición siempre sea verdadera) cuando $d(d)$ pare, y que pare cuando $d(d)$ no pare, en símbolos:

$$dae''(d) = \begin{cases} \uparrow & \text{cuando } d(d) \downarrow \\ \downarrow & \text{cuando } d(d) \uparrow \end{cases} \tag{4}$$

Por pura curiosidad preguntemos qué pasa si le aplicamos al dae'' su propia descripción, es decir, $dae''(dae'')$; tenemos que:

1. $dae''(dae'') \uparrow$ cuando $dae''(dae'') \downarrow$,
2. $dae''(dae'') \downarrow$ cuando $dae''(dae'') \uparrow$.

lo que es a toda luces una contradicción.

Así la construcción del dae'' es imposible, pero dae'' se construyó a partir de dae', este último es como el dae restringido a pares (d, d), y el dae es como el dau excepto que "puede adivinar" que $d(in)$ no parará.

Por otro lado se ha demostrado matemáticamente, incluso utilizando diferentes enfoques, que los dispositivos automáticos universales existen. Además, la experiencia nos dice que una computadora convencional es un dispositivo automático universal.

Por lo tanto lo erróneo de nuestro razonamiento fue suponer que podría existir un dispositivo automático especial, con la capacidad de adivinar que el dispositivo automático d con la entrada in no parará.

6. Conclusión

Afirmar que una computadora puede hacer todo lo que se le diga es, en el mejor de los casos, una afirmación ingenua. Como hemos

visto hay cosas que no puede hacer (adivinar si un programa terminará dados ciertos datos de entrada); aunque existiera la forma de decirle cómo adivinar, llegaríamos a contradicciones insuperables.

En [Cut80], [HMU02] y [Men97] puede encontrar el estudio de la computabilidad desde diferentes enfoques, respectivamente, desde la teoría de las funciones recursivas, desde la teoría formal de los lenguajes, y desde el punto de vista de la lógica matemática.

Independientemente del enfoque con el que se formalice la noción de computabilidad, se llega a resultados equivalentes, lo cual sugiere que, ***no importando qué teoría matemática utilicemos para modelar lo que un dispositivo automático universal puede hacer, encontramos los mismos límites***.

Referencias

[Cut80] Nigel J. Cutland. *Computability, An introduction to recursive function theory.* Cambridge University Press, 1980.

[HMU02] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introducción a la Teoría de Autómatas, Lenguajes y Computación.* Pearson Educación, segunda edition, 2002.

[Men97] Elliot Mendelson. *Introduction to Mathematical Logic.* Chapman and Hall/CRC, fourth edition, 1997.