Autómatas finitos y otras máquinas

El estudio de la computabilidad que hemos llevado a cabo hasta este momento parece preocupado de manera casi exclusiva por establecer una frontera precisa entre tareas ejecutables de manera efectiva y aquellas otras que no lo son. Tomado como criterio clasificatorio, este análisis parece demasiado grueso. ¿Qué tipos de conducta pueden distinguirse dentro del dominio de la computabilidad? ¿Son todas las tareas efectivas computables del mismo modo? ¿Es posible establecer criterios de clasificación y medida que arrojen alguna luz sobre estos aspectos?

Pretender analizar el tipo de complejidad o la categoría que le corresponde a un determinado algoritmo puede parece un objetivo determinado por necesidades prácticas más que teóricas. Aunque es innegable que muchas de las cuestiones que se van a plantear a continuación tienen un interés práctico manifiesto, veremos que todas ellas poseen consecuencias teóricas de considerable alcance. Lo que vamos a discutir en las próximas sesiones constituye tan sólo una aproximación a una serie de áreas de investigación sumamente especializadas y complejas. No se puede pretender que un curso introductorio como este vaya más allá. Me conformaré con indicar de qué modo pueden llegara a afectar los resultados que se persiguen en estos dominios al tipo de problemas fundamentales que básicamente han constituido nuestro objeto de estudio.

Empezaremos por analizar modelos de la computabilidad más limitados que el que representan las máquinas de Turing. La pregunta a la que se intenta dar respuesta es ¿qué sucede si limitamos las capacidades de una máquina de Turing de este modo o de aquel otro? ¿Se obtiene una clase menor de tareas ejecutables de manera efectiva? ¿Qué clases de procedimientos pueden caracterizarse de este modo? El capítulo siguiente sacará partido de este tipo de consideraciones para

proponer una genuina teoría de el medida reconocida en la actualidad bajo el rótulo de *Teoría de la Complejidad*. Se trata de una subdisciplina con profundas consecuencias prácticas, pero que se ve afectada de forma un tanto paradójica por uno de los problemas teóricos de mayor alcance en Teoría de la Computación, el denominado problema de la identidad entre la clase P de rutinas y la clase NP. Finalmente me ocuparé de trasladar algunos de los resultados más característicos de la Teoría de la Computación al dominio de la Lógica. Una de las consecuencias de este proceso es la definición de una interesante jerarquía asociada al lenguaje de la Lógica la cual puede ser vista sin especiales problemas como un peculiar sistema de medida de la complejidad de enunciados formales.

Ya hemos visto que el modelo de cálculo efectivo de una rutina que suministran las máquinas de Turing resulta ser extraordinariamente universal, viéndose limitado tan sólo por la Tesis de Church. Pero, ¿cuán *estable* se muestra ante modificaciones de sus rasgos definitorios? Este es un asunto de índole muy distinta.

No es difícil imaginar modificaciones apreciables en el alfabeto y arquitectura que definen cada una de estas máquinas con el fin de obtener nuevos ingenios más flexibles o sencillos de manejar. Podemos pensar, de hecho, que las máquinas de registros son modificaciones del patrón original de una máquina de Turing introducido con unos fines prácticos. ¿Cuánto podemos deformar el modelo original sin perder potencia expresiva (ya que ganarla es algo que parece imposible, al menos para el estado actual de la cuestión)?

Parece claro que no perdemos ni ganamos nada si en lugar de un alfabeto en el que el único símbolo que cabe encontrar en una celda es "*" se admite otro con un mayor número de elementos. En el límite, podemos admitir todo el conjunto de los naturales sin problema. También es posible operar sobre las acciones que la cabeza lectora puede realizar alterando su disposición inicial. No es infrecuente separar los momentos en que la cabeza lectora imprime o borra contenidos de una celda de aquellos en que esa misma cabeza se desplaza a la izquierda o la derecha. En lugar

de admitir instrucciones del tipo *nXYm* nos veríamos obligados a trabajar con otras del tipo *nXYZm*, donde *Z*={i,d}, pero eso sería todo. Las modificaciones sugeridas aquí no producen especial problema en la medida en que no suponen, ni siquiera desde un punto de vista intuitivo, restricciones de las condiciones bajo las que opera una máquina de Turing, sino más bien al contrario. Consideremos un ejemplo algo menos obvio. Supongamos que impedimos que la cabeza lectora pueda borrar ningún contenido de la celda sobre la que se encuentra estacionada: sólo se le permite imprimir símbolos. En este caso podríamos estar ante una restricción siempre que el contexto no determine otra cosa. ¿Qué tiene que ver aquí el contexto? Si en lugar de celdas en blanco tenemos celdas ocupadas por un símbolo convencional, sea éste "0" y en lugar de una marca –"*"- disponemos de un símbolo como "1", permitir que la cabeza lectora imprima cualquiera de los dos símbolos, "1" o "0", no establece diferencia alguna con el modelo original: imprimir "0" sobre una celda ocupada con "1" reemplazando este símbolo por el anterior equivale en todos los sentidos a borrar el contenido de la celda. Hay que ser prudente con los cambios ya que no cabe confundir ciertas acciones con sus posibles descripciones físicas. Borrar es, desde nuestro punto de vista, una función lógica dentro de un sistema simbólico que puede ser representada por muchas acciones reales distintas: borrar realmente un símbolo, sobreimprimir otro, etc.

Una vez hecha esta observación, ¿qué restricciones podemos considerar?, ¿cuáles de ellas suponen auténticas limitaciones? Considerar restricciones del modelo original de Turing exige algo más que modificaciones con un cierto significado físico. Hace falta ser muy consciente de la función lógica de la que es responsable cada uno de sus elementos característicos. Hay muchos tipos de modificaciones que es posible considerar, pese a lo cual el fenómeno de la relativa estabilidad del concepto de calculabilidad efectiva vuelve a manifestarse dando lugar a una interesante jerarquía. Esta jerarquía de procedimientos incluye una serie de modelos notables por sus propiedades formales: cada uno de ellos es demostrablemente más capaz que el anterior. Esto supone que es posible encontrar tareas que uno de estos modelos no es capaz de representar aunque sí lo es el siguiente modelo en la jerarquía: en la cúspide

se hallan de nuevo las máquinas de Turing. Este sistema suele recibir el nombre de *jerarquía de Chomsky* y presenta el siguiente aspecto:

[1] *Jerarquía de Chomsky*:

i. Autómatas finitos.
ii. Autómatas con memoria activa (*Pushdown Automata*)
iii. Autómatas acotados linealmente
iv. Máquinas de Turing.

La jerarquía debe interpretarse en el siguiente sentido: todo autómata perteneciente a una clase x puede ser reproducido igualmente por un autómata perteneciente a la clase x+1. Es evidente que esto se hace extensivo a las tareas que cada uno es capaz de ejecutar. En el límite vemos que cualquiera de estos autómatas restringidos no es sino un tipo especial de máquina de Turing. La conversa no es cierta, sin embargo. De hecho, la razón por la cual cada uno de estos modelos posee una cierta entidad es la existencia de una limitación relevante en potencia expresiva. Por desgracia el estudio de cuales sean éstas supera los objetivos de un curso elemental como éste.

Autómatas finitos. Es frecuente emplear este término para referirse no sólo al tipo de máquina a la que aquí hace referencia, sino a todo ingenio que restringe de forma demostrable la potencia de las máquinas de Turing. A medida que se han ido conociendo cada vez mejor las diferencias existentes entre distintos tipos de restricciones este es un uso en progresivo retroceso. Un autómata finito opera sobre una cinta de cálculo del tipo de las de las máquinas de Turing, pero sus instrucciones tienen el aspecto siguiente:

[2] *Instrucciones para autómatas finitos*:

- nXIm, donde

i. n,m son enteros positivos
ii. X puede ser “*” o “B”
iii. I indica que la cabeza lectora ha de desplazarse una celda a la izquierda.

En realidad, se puede prescindir por completo de la indicación que corresponde al desplazamiento de la cabeza lectora, ya que no se opone a nada, no hay más opciones. Si renunciamos a la deliberada semejanza con las instrucciones propias de las máquinas de Turing, el tipo de instrucción característica de estas máquinas, nXm, da idea de lo poco ambiciosas que pretenden ser. Sólo son capaces de analizar el contenido de la celda dando paso a otro estado al tiempo que desplaza la cabeza lectora hacia una única dirección previamente convenida, la izquierda en nuestro caso.

Su única habilidad parece ser la de reconocer secuencias de celdas ocupadas y vacías sobre la cinta de cálculo. Cada una de estas secuencias recibe el nombre de *palabra*. Es fácil emplear este tipo de máquinas para conseguir que reaccionen exactamente a una palabra o a un tipo de palabras y no a otras. Téngase en cuenta que la reacción de una de estas máquinas nunca puede interpretarse en términos de un output que pueda ser leído en su cinta, ya que carecen de esa habilidad. Lo único que podemos hacer es ver si consigue sortear todos los obstáculos que su programa impone para alcanzar un estado de parada estándar. Si una palabra es computada por una de estas máquinas haciendo que ésta progrese desde su estado inicial hasta su estado final se dice que esa máquina *acepta* esa palabra. El estudio de los autómatas finitos demuestra que es posible analizar bastante bien el tipo de palabras que una máquina acepta a partir del número de estados de que consta y sus conexiones, ya que en este caso el aspecto interno de las instrucciones es muy poco informativo. Pero

más relevante resulta aún el hecho de comprobar que el tipo de palabras que estas máquinas son capaces de aceptar pueden caracterizarse de manera uniforme:

[3] *Expresiones regulares*

i. * y B son expresiones regulares.
ii. Si X e Y son expresiones regulares también lo son su *concatenación* XY y su *suma* X+Y
iii. Si X es una expresión, también lo es su *iteración* X'.

La suma de dos expresiones regulares alude a varias palabras a la vez, mientras que la iteración de una expresión no es sino la concatenación consigo misma. Esta descripción es suficientemente general como para pensar que estas máquinas son capaces de aceptar, en realidad, cualquier palabra o cualquier clase de ellas que pueda ser descrita como una expresión regular. Sin embargo esto mismo no es cierto para una colección de palabras conocidas desde antiguo. Me refiero a los *palíndromos*: expresiones que es posible leer igual de izquierda a derecha que de derecha a izquierda - en castellano la palabra *anilina* es uno de los ejemplos más conocidos-. Dentro de éstos es posible definir unos en los que el punto de inversión se describa mediante una marca. Si por Z^{-1} se representa el resultado de invertir los símbolos en Z, y *k* representa la marca de inicio de la inversión los palíndromos a que me refiero se representaría como ZkZ^{-1}. No hay ningún autómata finito que los compute pese a que esa representación gráfica es ya ella misma una forma de computar tales palabras.

La clase formada por todas las expresiones que acepta uno de estos autómatas recibe el nombre de *lenguaje regular*.

Autómatas con memoria activa (*Pushdown Automata*). Este nuevo formalismo parte de considerar un autómata finito al cual se el añade una memoria activa en forma de cinta de cálculo que actúa según el principio de almacenamiento apilado de la información –esto a lo que hace referencia la expresión “pushdown”-. La cinta de memoria es leída por una segunda cabeza lectora que actúa de manera simultánea a la primera y que añade una marca a continuación de la última que hubiera en la memoria o elimina la última que ha sido introducida en esa memoria. Esto explica la razón por la cual nos referimos a esa memoria como una cinta en la que la información se apila y recupera por orden inverso al de entrada. Esto explica que aunque la cabeza auxiliar pueda desplazarse en ambas direcciones no pueda hacerlo más allá de una marca que indica el fondo o inicio de la memoria. En realidad, cuesta trabajo representar adecuadamente las acciones admitidas sobre esta cinta con un formalismo tan permisivo como el de las máquinas de Turing. Resulta mucho más conveniente introducir dos instrucciones específicas que podrían denominarse “adición de X” y “eliminación”. La primera añade un dato -“*”- o una celda en blanco -“B”- a la cinta de cálculo en la primera celda no ocupada hacia la izquierda, mientras que la restante elimina la marca “*” si está en la celda escrutada desplazándose a continuación una celda hacia la derecha. Esto da lugar a lo siguiente:

[4] *Instrucciones sobre la cinta de memoria activa.*

- nX(AdY)m
- nX(El)m,

X es como antes, “*” o “B”.

Cada estado de una de estas máquinas debe indicar las acciones a realizar, tanto sobre la cinta normal, como sobre la cinta correspondiente a la memoria activa.

La memoria activa permite ampliar de forma notable las habilidades de estas máquinas: entre otras cosas las capacita para aceptar palabras con patrones mucho más complejos que en el caso de los autómatas finitos. Además, se admite que su arquitectura sea de tipo no-determinista. ¿Qué quiere decir esto?

Hasta ahora siempre hemos considerado que cada una de las acciones que una máquina ejecuta siguiendo su programa está rígidamente determinada por las condiciones antecedentes. Esto es, nunca hemos considerado la existencia de más de un curso de acción posible una vez satisfechas las condiciones relativas al contenido de la cinta de cálculo. Esto no significa que no pueda hacerse. En el caso de este tipo de máquinas la opción surge de manera bastante natural: existen varias transiciones posibles coherentes con los contenidos de las cintas de cálculo sobre las que actúa en cada estado cada estado. Ante una situación en la que caben varios cursos de acción le está permitido elegir cualquiera de ellos, pero se considera que en caso de que la máquina disponga de una combinación de estados dispuesta a aceptar una palabra, sus elecciones serán favorables a la aceptación de la palabra. También podemos imaginar que la máquina ha seguido todos los posibles cursos de acción que se derivan de todas las distintas decisiones que puede adoptar en cada momento eligiendo aquella que le permite aceptar la palabra. Seguramente esta es una forma a afrontar el problema del carácter no-determinista de un algoritmo mucho más conveniente para lo que sigue.

Siempre que hay una variante no-determinista de una clase de procedimientos existe también una opción plenamente determinista que se concibe como caso límite del primero. Así, los autómatas con memoria activa deterministas forman una subclase propia de los no-deterministas. Aunque sólo demos el dato, es conveniente saber que los lenguajes que aceptan los autómatas con memoria activa no-deterministas reciben el nombre de *lenguajes libres de contexto,* mientras que aquellos que son aceptados por la alternativa determinista se denominan *lenguajes libres de contexto deterministas.*

Que todo autómata finito puede ser simulado por uno de estos ingenios es tarea trivial: basta tomar un autómata con memoria activa e ignorar en su programa cualquier contenido procedente de la cinta auxiliar.

Autómatas acotados linealmente. Un autómata acotado linealmente es en todo similar a una máquina de Turing salvo por el hecho de tener limitada su cinta de cálculo a una cantidad de celdas que viene dada por la longitud del input que computa. De hecho, puede considerarse sin restricción especial que esa cantidad viene dada por una constante *k* que se aplica como coeficiente a la longitud del input, medido todo ello en términos del número de celdas consecutivas necesarias para expresar ese input.

Esta restricción se va a mostrar mucho más importante de lo que en principio pudiera parecer. De hecho, si lo pensamos por un momento, es la que marca la diferencia entre el tipo de ingenios que podemos llegar a ver sustanciados en muchos de los equipos informáticos con los que nos solemos relacionar y las propias máquinas de Turing. En general, los modernos ordenadores siempre realizan sus cálculos sobre unidades de memoria dotadas de una capacidad limitada. No se supone que sus cálculos puedan disponer de cantidades no acotadas de memoria. Esto lleva a situaciones de saturación literal de su capacidad que lleva a tratar sus resultados de forma análoga a como lo hace un autómata acotado linealmente. Pero hay algo aún más importante que esto. Desde un punto de vista practico es muy conveniente contar con algoritmos cuya conducta se muestre acotada, es más, es importante saber reconocer si un algoritmo está acotado de forma razonable o si, por el contrario, su actividad crece a velocidad considerable en cuanto se le exigen tareas de cierta entidad –traducidas ahora en inputs de cierta longitud-. Es razonable pensar que un algoritmo acotado linealmente constituye un modelo de cálculo en el que el tamaño del input se muestra bastante próximo al que conllevan todas las operaciones que conducen al resultado. En ese mismo sentido se suele considerar que aquellos algoritmos acotados exponencialmente son un ejemplo de lo contrario, esto es, de procedimientos en los que el tamaño del input hace crecer la complejidad de la rutina

de forma desmesurada. Pocos ingenieros se mostrarían dispuestos a aceptar una rutina con crecimiento exponencial si pueden evitarlo.

Es fácil ver de qué modo cada una de estas máquinas no es sino un tipo especial de Máquinas de Turing. Tal vez es más complejo ver el modo en que pueden reproducir la actuación de máquinas pertenecientes a la clase precedente en la jerarquía, pero el detalle ciertamente no nos interesa ahora. Los lenguajes que aceptan los autómatas acotados linealmente reciben el nombre de *lenguajes sensibles al contexto*, mientras que aquel que reciben los que pueden ser aceptados por genuinas máquinas de Turing son, como no podía ser de otro modo, *recursivamente enumerables.*

Los autómatas linealmente acotados y las propias máquinas de Turing pueden ser deterministas o no-deterministas. En el caso de las segundas el concepto considerado básico es determinista. La razón es que cualquier evolución no determinista de una máquina puede ser reproducida –mediante una serie de desdoblamientos de estados- mediante máquinas puramente deterministas. Esto permite prescindir de la versión no-determinista, al menos desde un punto de vista general. Sin embargo esto mismo no es posible en el caso de los autómatas acotados linealmente. En la actualidad se ignora si es posible hallar un algoritmo determinista para cada máquina no-determinista. Bajo esta pregunta se esconde un problema del máximo interés al que dedicaremos el siguiente capítulo.

Orientación bibliográfica.

La peculiar obra de Dewdney, **[Dewdney, 1993]**, contiene varias secciones en las que se presentan y discuten distintos tipos de autómatas finitos y variantes de máquinas de Turing: cap. 2, 14, 17, 31. La jerarquía de Chomsky se presenta en el cap. 7. Para la ocasión este tratamiento puede ser razonablemente suficiente. Para profundizar algo más disponemos del manual de Salomaa, **[Salomaa, 1985]**, cuyo cap.3 está íntegramente dedicado al tema.