HANS HERMES

INTRODUCCIÓN A LA TEORÍA DE LA COMPUTABILIDAD

ALGORITMOS Y MÁQUINAS

tecnos

INTRODUCCIÓN
A LA TEORÍA
DE LA COMPUTABILIDAD

ALGORITMOS Y MÁQUINAS

Traducido del alemán por Manuel Garrido y Aránzazu Martín Santos.

N. de los T.: En esta traducción se ha tenido generalmente en cuenta el texto original alemán en su 2.ª edición de 1971. Pero en algunos ejemplos y en el diseño de la máquina universal de Turing se ha dado preferencia a la versión inglesa con modificaciones publicadas por el autor en 1969 con el título *Enumerability*, *Decidability*, *Computability* (Springer, Berlín). A esa misma edición aludirá, para mayor comodidad del lector castellano, la sigla *EDC*.

Diseño de cubierta:
J. M. Domínguez y J. Sánchez Cuenca

La paginación se corresponde con la edición impresa.

O'Donnell, 27 - Madrid-9
Depósito legal: M-330-1984
I.S.B.N.: 84-309-1026-3

Printed in Spain. Impreso por GAR. Villablino, 54. Fuenlabrada. (Madrid).

ÍNDICE

PRESENTACIÓN

Uno de los principales factores determinantes de la profunda revolución experimentada en el ámbito de la ciencia, la técnica y la cultura de nuestros días es el desarrollo de la informática. La palabra «informática» es un nombre colectivo que designa un vasto conjunto de teorías y técnicas científicas —desde la matemática abstracta hasta la ingeniería y la gestión administrativa— cuyo objeto es el diseño y el uso de los computadores electrónicos. Pero el núcleo teórico más sólido y fundamental de todo ese conjunto de doctrinas y prácticas es la llamada «teoría de la computabilidad» o «teoría de algoritmos», formalmente elaborada en los años 30 y 40 gracias a los descubrimientos de lógicos matemáticos como Gödel, Turing, Post, Church, Kleene.

El presente Cuaderno de Hans Hermes, titulado Introducción a la teoría de la computabilidad, *constituye la parte inicial del libro de dicho autor* Aufzahlbarkeit, Entscheidbarkeit, Berechenbarkeit *[«Enumerabilidad, decidibilidad, computabilidad»], que es considerado por la comunidad científica un tratado modelo de lógica y teoría de algoritmos.*

Editorial Tecnos proyecta publicar posteriormente en su totalidad este importante libro. Pero mientras tanto, y para un primer acercamiento de sus contenidos a un círculo amplio de lectores, se ofrecen aquí inmejorablemente expuestos los conceptos básicos de la teoría de la computabilidad concretados en la teoría de las máquinas de Turing como definición exacta del concepto de algoritmo.

José Fernández Prida, Profesor de Teoría de la Computabilidad en la Facultad de Matemáticas de la Universidad Complutense de Madrid y discípulo personal de Hans Hermes, ha confeccionado un valioso «Anexo», que permite al lector de este Cuaderno pasar sin solución de continuidad de la comprensión de los conceptos más sencillos de la teoría de algoritmos a la noción, más ardua, de máquina universal de Turing.

PREFACIO A LA PRIMERA EDICIÓN

El objetivo de desarrollar algoritmos para resolver problemas ha sido siempre considerado por los matemáticos como especialmente interesante e importante. Normalmente, un algoritmo sólo, es aplicable a un grupo muy reducido de problemas. Tal es, por ejemplo, el algoritmo de Euclides, que determina el máximo común divisor de dos números, o el bien conocido procedimiento que se usa para obtener la raíz cuadrada de un número natural en notación decimal. Cuanto más importantes sean estos algoritmos *especiales*, tanto más deseable parece ser el disponer de algoritmos de un mayor rango de aplicabilidad. A través de los siglos, los intentos de suministrar algoritmos aplicables tan ampliamente como fuera posible no tuvieron éxito. Fue en la segunda mitad del siglo pasado cuando tuvo lugar el primer avance notable. Pues entonces se expuso bajo la forma de cálculo un importante grupo de las inferencias de la lógica de predicados. (En ello jugó un papel pionero esencial el álgebra booleana.) Tal vez se podría haber conjeturado entonces que *todos* los problemas matemáticos son solubles mediante algoritmos. Sin embargo, como ya se sabe, los problemas aún insolubles (como el problema de las palabras de la teoría de grupos o el décimo problema de Hilbert, que considera la cuestión de la solubilidad de las ecuaciones diofánticas) exigían cierta cautela. No obstante, se había dado el impulso a la investigación de la esencia de los algoritmos. Ya *Leibniz* había investigado este problema, pero sin suerte.

Los matemáticos de nuestro siglo, sin embargo, experimentados en el tratamiento de problemas abstractos y especialmente en las operaciones con lenguajes formales, tuvieron éxito. Alrededor de 1936 se propusieron casi simultáneamente varias sugerencias para hacer preciso el concepto de algoritmo y otros afines (tesis de Church). Aunque estas sugerencias (cuyo número se ha incrementado después) se originaron a partir de consideraciones iniciales bastante diversas, han demostrado ser equivalentes. Los motivos que condujeron a estas sustituciones precisas, el hecho de su equivalencia, y el hecho experimental de que todos los algoritmos que han tenido lugar en matemática hasta ahora son casos de estos conceptos precisos, al menos si nos fijamos en su núcleo esencial, han convencido a casi todos los investigadores de este campo de que esas sustituciones precisas son interpretaciones adecuadas del concepto de algoritmo que primero se dio intuitivamente.

Una vez hemos aceptado una sustitución precisa de este concepto de algoritmo, se torna posible tratar la cuestión de si existen conjuntos bien definidos de problemas que no pueden ser manipulados mediante algoritmos y, si ello es así, aducir casos concretos de este tipo. Muchas de tales investigaciones se llevaron a cabo durante las últimas décadas. Se mostró la indecidibilidad de la aritmética y otras teorías matemáticas, además de la insolubilidad del problema de las palabras de la teoría de grupos. Muchos matemáticos consideran estos resultados, y la teoría sobre la que se basan, como el logro más característico de la matemática en la primera mitad del siglo XX.

Si concedemos la legitimidad de las sustituciones precisas sugeridas del concepto de algoritmo y los conceptos relacionados, entonces podemos decir que

se ha logrado mostrar, por métodos estrictamente matemáticos, que existen problemas matemáticos que no pueden ser tratados mediante los métodos del cálculo matemático. En vista del importante papel que juega hoy la matemática en nuestra concepción del mundo, este hecho es de gran interés filosófico. *Post* habla de una ley natural sobre las «limitaciones de la potencia matemática del Homo Sapiens». Aquí encontramos también un punto de partida para discutir la cuestión de en qué consiste la actividad creativa real de los matemáticos.

Este libro es una introducción a la teoría de algoritmos. Ante todo, intentaremos convencer al lector de que las sustituciones precisas dadas representan adecuadamente los conceptos intuitivos. El mejor modo de hacerlo consiste en usar una de estas sustituciones precisas —esto es, la máquina de Turing— como punto de partida. Trataremos los más importantes conceptos constructivos, como los de función computable, propiedad decidible y conjunto generado por un sistema de reglas, usando como base las máquinas de Turing.

La teoría se desarrolla desde el punto de vista de la lógica clásica. Esto se advertirá especialmente mediante la aplicación del operador existencial clásico, por ejemplo, en la definición de computabilidad: denominaremos computable a una función si *existe* un algoritmo para encontrar los valores de argumentos arbitrariamente dados. Ello se hará bien patente, sin embargo, en todos aquellos casos donde las demostraciones se lleven a cabo *constructivamente*.

En contraste con muchas publicaciones sobre el tema, pondremos cuidado en distinguir entre las fórmulas de un lenguaje simbólico y las cosas denotadas por ellas, especialmente en las definiciones básicas.

Al final de algunas secciones hay unos cuantos ejercicios, muy fáciles, que el lector debería intentar resolver.

Dado el carácter introductorio de este libro, no se discutirán en él todos los resultados pertinentes. Sin embargo, las referencias dadas al final de la mayor parte de las secciones informarán al lector de los últimos desarrollos de la teoría. Además quisiéramos, de una vez por todas, remitir al lector a una obra básica, a saber, la *Introducción a la Metamatemática* de S. C. Kleene ([1952]; traducción castellana en Madrid, Tecnos, 1974); y también a los artículos publicados en *The Journal of Symbolic Logic* (1936 y ss.). Otro libro cuya teoría se inspira en las máquinas de Turing es el de M. Davis, *Computability and Unsolvability* (Nueva York, 1958).

El presente libro se basa en las clases que el autor ha dado regularmente sobre este tema desde 1949. En 1955, un manuscrito sobre las clases del curso fue publicado por Verlag Aschendorff (Münster) bajo el título de *Entscheidungsprobleme in Mathematik und Logik*.

Me gustaría expresar mi gratitud a los doctores H. Kiesow y W. Oberschelp por la valiosa ayuda en la preparación del manuscrito. Asimismo expreso mi agradecimiento a las señoritas T. Hessling y E. Herting, y a los señores D. Titgemeyer y K. Hornung.

Münster i. W., primavera de 1960

H. HERMES

PREFACIO A LA SEGUNDA EDICIÓN

Se han corregido algunos errores. Estoy agradecido al profesor H. B. Curry por sus valiosas sugerencias.

Para una bibliografía más reciente me permito remitir al lector a la citada en el libro de H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability* (Nueva York, 1967).

Friburgo de Br., primavera de 1969

H. HERMES

REFLEXIONES PRELIMINARES SOBRE ALGORITMOS

§ 1. El concepto de algoritmo

El concepto de algoritmo o «procedimiento general» es algo más o menos sabido de todo matemático. En esta sección introductoria nos proponemos precisar mejor dicho concepto, poniendo de relieve lo que deba considerarse esencial.

1. Algoritmos como procedimientos generales. El modo específico según el cual los matemáticos conciben y elaboran sus teorías tiene múltiples aspectos. Aquí destacaremos y discutiremos más precisamente un punto de vista característico de muchos desarrollos de la ciencia matemática. Cuando los matemáticos se ocupan de un grupo de problemas, en una primera instancia son la mayoría de las veces hechos aislados los que cautivan su interés. Pronto, sin embargo, procederán a descubrir una conexión entre estos hechos, e intentarán sistematizar cada vez más la investigación con el fin de obtener una visión de conjunto y alcanzar, finalmente, pleno dominio del campo en cuestión. Con frecuencia, un tal dominio se obtiene gradualmente (o al menos se intenta obtener) delimitando grupos especiales de problemas de forma que cada uno de esos grupos pueda ser tratado con ayuda de un algoritmo. Un algoritmo es un procedimiento general con el que se obtiene la respuesta a todo problema apropiado mediante un simple cálculo de acuerdo con un método especificado.

Ejemplos de procedimientos generales pueden hallarse en toda disciplina matemática. Baste recordar al respecto el procedimiento de división de números naturales dados en notación decimal, el conocido algoritmo de computación de expresiones decimales que se aproximan a la raíz cuadrada de un número natural, o el método de descomposición en fracciones parciales para la computación de integrales con integrandos racionales.

Cuando se hable *aquí* de un *procedimiento general*, debe entenderse siempre por tal un proceso cuya ejecución está claramente especificada hasta los últimos detalles. Esto supone, en particular, que las instrucciones para la ejecución del proceso se puedan plasmar en un texto *finito* [1].

No queda lugar alguno para que intervenga la imaginación creadora del ejecutante, que ha de trabajar, a la manera de un esclavo, de acuerdo con las instrucciones que le son dadas, que todo lo determinan hasta el más pequeño detalle [2].

Los requisitos que ha de cumplir un proceso para ser considerado un procedimiento general son muy

[1] Producir un conjunto de instrucciones de longitud infinita es cosa que no está en nuestro poder. Cabría, empero, concebir la construcción de un conjunto de instrucciones cuya longitud fuese potencialmente infinita. Ello es algo que puede obtenerse especificando primero un segmento inicial finito y aduciendo después un conjunto finito de reglas que determine de qué modo habrá que prolongar en cada caso las instrucciones. Pero entonces podemos decir que el segmento inicial finito, juntamente con el texto adicional finito, representa al conjunto real (finito) de instrucciones.

[2] Obviamente, la ejecución esquemática de un procedimiento general dado no ofrece (una vez ha sido ensayado varias veces) especial interés para el matemático. Así podemos constatar el notable hecho de que, tras la tarea específicamente matemática de desarrollar un método general, el matemático creador desprecia matemáticamente, por así decirlo, el campo que justamente acaba de dominar por ese método.

estrictos. Conviene darse cuenta claramente de que el modo y manera en que un matemático acostumbra a describir un «procedimiento general» es, por lo común, demasiado vago para satisfacer realmente el requerido grado de precisión. Ello es aplicable, por ejemplo, a la descripción usual de los métodos para resolver un sistema de ecuaciones lineales. En tal descripción se deja abierto, entre otras cosas, el modo en que habría que ejecutar las adiciones y multiplicaciones pertinentes. No obstante, es claro para todo matemático que en este caso, y en otros parecidos, las instrucciones pueden ser suplementadas en su totalidad al objeto de que en ellas nada quede abierto. Las instrucciones conforme a las cuales operan los funcionarios sin especial entrenamiento matemático en un centro de cálculo se encuentran relativamente próximas al ideal que acabamos de trazar.

Hay un caso, que vale la pena mencionar, en que el matemático acostumbra a hablar de un procedimiento general aun sin pretender con ello caracterizar un modo de proceder totalmente exento de ambigüedad. Nos referimos aquí a cálculos sujetos a una variedad de reglas sin que se determine en qué orden deben ser aplicadas esas reglas. Pero estos cálculos guardan estrecha relación con los procedimientos completamente inequívocos que se acaban de describir. De ellos trataremos en el apartado 6 de la presente sección. En este libro adoptaremos la convención de *decir que un procedimiento es un procedimiento general sólo cuando el modo de proceder esté completamente exento de ambigüedad*.

Aunque hay *algoritmos conclusivos*, existen también otros con los que se puede continuar tanto tiempo como se desee. El algoritmo de Euclides para determinar el máximo común divisor de dos números

tiene un término; después de un número finito de pasos en la computación, obtenemos la respuesta y el procedimiento llega a un fin. El conocido algoritmo de la computación de la raíz cuadrada de un número natural dado en notación decimal *no* alcanza generalmente un término. Podemos continuar con la computación del algoritmo tanto tiempo como queramos, y obtendremos una y otra vez nuevas fracciones decimales que se aproximan cada vez más a la raíz.

2. Realización de algoritmos. Un procedimiento general, tal y como aquí se entiende, significa en cualquier caso, primariamente, una operación (manipulación) con cosas concretas. Estas cosas deben estar delimitadas entre sí con suficiente nitidez. Y pueden ser piedrecitas [3] (contadores, bolas de madera), como, por ejemplo, en el clásico *ábaco* o en el *sorobán* japonés, pero también pueden ser símbolos, como los utilizados en matemática (por ejemplo, 2, x, $+$, $(, \int)$, y asimismo las ruedas dentadas de una calculadora de mesa, o impulsos eléctricos, como es usual en los grandes computadores. La operación consiste en imprimir o imponer nuevas configuraciones a cosas espacial o temporalmente ordenadas.

Para la práctica de la matemática aplicada es esencial saber cuál es el *material* que se utiliza para la ejecución de un procedimiento. Pero desde el punto de vista teórico, desde el cual nos proponemos tratar los algoritmos, el material es irrelevante. Una vez se co-

[3] Los algoritmos (o en todo caso los procedimientos discutidos en el apartado 4 de la presente sección) son frecuentemente denominados *cálculos*. Este nombre tiene su origen en los *calculi* (piedrecitas calcáreas) que los romanos usaban para sus operaciones de cómputo *(calculare)*. Conviene advertir que la palabra «cálculo» es usada también a veces en un sentido que no se identifica con el de algoritmo.

noce un procedimiento realizado con un cierto material, resulta posible transferirlo (con más o menos éxito) a otro material. Así, la adición en el dominio de los números naturales puede ser efectuada añadiendo nuevos palotes a una fila de ellos, añadiendo o sustrayendo bolas de un ábaco, o mediante el giro de las ruedas en una calculadora de mesa.

Puesto que únicamente estamos interesados, dentro del ámbito de los procedimientos generales, por cuestiones que son independientes de la realización material de tales procedimientos, podemos tomar como base de nuestras consideraciones una realización que es especialmente fácil de tratar desde el punto de vista matemático. Por ello resulta preferible en la teoría matemática de algoritmos considerar aquellos algoritmos que consisten en la alteración de *filas de signos*. Una fila de signos es una secuencia lineal finita de *símbolos (signos individuales*, *letras)*. Se da por estipulado que para cada algoritmo hay un número finito de letras (al menos una) cuya colección constituye el *alfabeto* en que se basa el algoritmo. Las filas finitas de signos que pueden componerse con las letras del alfabeto se denominan *palabras*. A veces resulta conveniente disponer también de la *palabra vacía* $\square$, que carece de letras. Si $\mathfrak{A}$ es un alfabeto y W una palabra compuesta sólo de letras de $\mathfrak{A}$, decimos que W *es* una *palabra sobre* $\mathfrak{A}$.

En cierto sentido, las letras de un alfabeto $\mathfrak{A}$ en que se basa un algoritmo no son esenciales. Pues si alteramos las letras de $\mathfrak{A}$ a fin de obtener, correlativamente, un nuevo alfabeto $\mathfrak{A}'$ resulta inmediatamente posible aducir un algoritmo, basado en ese nuevo alfabeto, que es «isomorfo» con respecto al algoritmo original y que cumple, fundamentalmente, las mismas funciones.

3. *Gödelización* [4]. En principio, es posible arreglarse con un alfabeto que conste solamente de una letra, como, por ejemplo, la letra I [5]. Las palabras sobre este alfabeto son (prescindiendo de la palabra vacía): I, II, III , etc. Estas palabras pueden ser trivialmente identificadas con los números naturales 0, 1, 2... Tan extremada canonización del «material» es recomendable por determinadas razones. Por otra parte, a menudo resulta conveniente disponer de la diversidad de un alfabeto que conste de varios elementos, como, por ejemplo, cuando se utilicen letras auxiliares. Normalmente preferiremos basar nuestras consideraciones en alfabetos con más de un elemento.

El uso de un alfabeto que conste de *un elemento* no implica una limitación esencial: de hecho, las palabras de un alfabeto $\mathfrak{A}$ que conste de N elementos pueden ser asociadas biunívocamente con números naturales $G(W)$ (de tal modo que cada número natural está asociado con al menos una palabra), esto es, con palabras de un alfabeto que conste de *un* elemento. Una tal aplicación, G, se denomina una *gödelización*, y $G(W)$ el *número de Gödel* (con respecto a G) de la palabra W. Gödel fue el primero en hacer uso de tal aplicación en el artículo que se cita al final de esta sección. Los requisitos de una gödelización G son los siguientes:

1) Si $W_1 \neq W_2$, entonces $G(W_1) \neq G(W_2)$ (aplicación uno a uno).

2) Para cualquier palabra dada W, se puede com-

[4] A la gödelización (numeración gödeliana) se la suele llamar también *aritmetización*.

[5] Para mayor facilidad de escritura omitiremos frecuentemente las comillas. Escribiremos, por tanto, I en lugar de «I» (y, respectivamente, «I» en lugar de «"I"»).

putar en un número finito de pasos, mediante un algoritmo, el número natural correspondiente $G(W)$.

3) Para cualquier número natural n, puede establecerse en un número finito de pasos, si n es el número de Gödel de una palabra W sobre $\mathfrak{A}$.

4) Si n es el número de Gödel de una palabra W sobre $\mathfrak{A}$, entonces esta palabra W (unívocamente determinada según 1), debe poder ser hallada en un número finito de pasos mediante este algoritmo.

Ofreceremos aquí una sencilla gödelización para palabras sobre el alfabeto $\mathfrak{A} = \{a_1, ..., a_N\}$. Concebimos a a_j como una «cifra» que representa al número natural j. Podemos entonces considerar a cada palabra no vacía W como representación de un número en un sistema numérico de $(N + 1)$ dígitos. Además establecemos que $G(\square) = 0$. Es fácil comprobar que se cumplen los requisitos 1) a 4).

4. Observaciones sobre la palabra vacía. Para ciertas consideraciones, es conveniente disponer de la palabra vacía. Algunos análisis, sin embargo, pueden ser más sencillamente llevados a cabo si se excluye la palabra vacía. Por la exclusión de la palabra vacía no se pierde nada fundamental. De hecho, podemos aplicar biunívocamente de modo constructivo las palabras W sobre un alfabeto $\mathfrak{A} = \{a_1, ..., a_N\}$ a las palabras no vacías W' sobre $\mathfrak{A}$. Una tal aplicación φ puede realizarse, por ejemplo, como sigue: $\varphi(\square) = a_1$; $\varphi(W) = a_1W$ si la palabra W contiene solamente símbolos que coinciden con a_1; $\varphi(W) = W$ para todas las demás palabras.

Al considerar funciones y predicados en lo que respecta a sus propiedades constructivas, daremos generalmente por supuesto que los argumentos y valores que ocurran son palabras no vacías.

El procedimiento más simple para representar mediante palabras los números naturales consiste en utilizar un alfabeto de un solo elemento $\mathfrak{A} = \{|\}$, y representar el número n por la palabra que conste de n trazos. Así, el número 0 estará representado por la palabra vacía. Sin embargo, si deseamos utilizar solamente palabras no vacías, entonces se puede recurrir a la aplicación φ anteriormente indicada y representar asi el número n por $n + 1$ trazos. Esta *representación numérica* (que ya ha sido mencionada en el anterior apartado 2) *será empleada más adelante* (cf. § 6.2).

5. Idealización de algoritmos. Nos referiremos ahora a una extrapolación que generalmente se efectúa en las teorías de que trata este libro. Todo el mundo sabe que existe un procedimiento general mediante el cual para dos números naturales cualesquiera n y m (dados en notación decimal) puede calcularse la potencia n^m (asimismo en notación decimal). Para números *pequeños* (por ejemplo, para $n < 100$, $m < 10$) este cálculo puede ser realmente llevado a cabo. Con números mayores ello se torna dudoso, y con números muy grandes (situados, por ejemplo, en la región de la potencia iterada $1000^{1000^{1000}}$) es posible que la ejecución de un cálculo *real* esté en contradicción con las leyes de la naturaleza (por ejemplo, porque no hubiese en el mundo material suficiente para escribir el resultado en notación decimal, o porque el género humano no duraría el tiempo requerido para ejecutar efectivamente semejante cálculo). Sería, ciertamente, interesante investigar tales limitaciones, posiblemente impuestas por las leyes de la naturaleza, y en especial, también, por las dimensiones de la memoria humana [6]. Investigaciones de esta suerte, sin embar-

[6] A este grupo de problemas pertenece, por ejemplo, la de-

go, no son nada fáciles de realizar y, hasta el momento, apenas si se las ha emprendido. Aquí nos proponemos, de una vez por todas, *abstraer* de tales limitaciones y dar por supuesta la situación ideal de que no hubiese ningún límite de tiempo, espacio o materia, para la ejecución de un procedimiento general. Suponemos asimismo, en particular, que podemos disponer de filas (finitas) de signos arbitrariamente largas [7].

6. Derivaciones. En el apartado 1 de esta sección hemos subrayado expresamente que un algoritmo opera de un modo totalmente exento de ambigüedad. Pero ya hemos mencionado un caso en el cual el matemático habla de un «procedimiento general» sin que se dé esta falta de ambigüedad. Vamos a analizar este caso y a investigar qué conexión guarda con el concepto de algoritmo aquí introducido. A este fin partimos de un

Ejemplo 1: El mínimo conjunto de números reales que contiene los números 9 y 12 y que está clausurado

---

mostración del hecho de que es conveniente para el hombre elegir como base *b* del sistema numérico que emplee, un número cercano a 10. Es interesante considerar al respecto que *b* no debe ser mucho más pequeño que 10, pues en caso contrario los números de la vida cotidiana podrían alcanzar una longitud que resultaría inconveniente para la memoria humana. Por otra parte, *b* no puede ser mucho mayor que 10, pues en tal caso la tabla de multiplicar se convertiría en una carga demasiado pesada para nuestra memoria.

[7] Las limitaciones que se acaban de mencionar con respecto a los seres humanos son análogas en el caso de los computadores, como, por ejemplo, las impuestas por su limitada capacidad de almacenamiento. Una representación plástica de la idealización arriba propuesta se puede obtener suponiendo que, sin alterar la construcción básica de una máquina, se pudiera aumentar arbitrariamente el equipo destinado a memoria de la misma, a medida que lo fuesen exigiendo las operaciones de cálculo. Para hacerlo más simple, baste imaginar, por ejemplo, una calculadora usual de mesa que poseyera tal virtud.

con respecto a la sustracción y a la multiplicación por $\sqrt{2}$ puede obtenerse con el siguiente algoritmo:

(a) 9
(b) 12
(c) $x - y$
(d) $x\sqrt{2}$

donde (a) y (b) significan que 9 y 12, respectivamente, pueden ser anotados como números de ese conjunto; (c) significa que, para cualesquiera dos números x e y ya obtenidos como números del conjunto, la diferencia entre ambos es también miembro del mismo; y (d) significa que la multiplicación de un número ya obtenido x por $\sqrt{2}$ también pertenece al conjunto.

Es obviamente claro que lo aquí descrito es una especie de «procedimiento», con cuya ayuda pueden obtenerse los elementos del módulo numérico generado por 3 y $3\sqrt{2}$. Aqui hemos descrito este procedimiento de una manera que está de acuerdo con la praxis matemática. Sin embargo, conviene tomar conciencia de que esta descripción está plagada de imperfecciones de toda suerte. En primer lugar, no debiera hablarse de *números*, sino más bien de *representaciones de números (numerales)* $\pm a \pm b \sqrt{2}$, donde a y b son números naturales dados en su notación decimal. Por ejemplo, (a) rezaría entonces, más exactamente, $+ 9 + 0 \sqrt{2}$. En el caso de (c) y (d) se da a entender manifiestamente que hay que emplear las reglas conocidas de cálculo para obtener el nuevo número en la notación fijada. No obstante, renunciaremos aquí a dar una completa formulación de estas reglas.

Con ayuda de las reglas (a),...,(d) pueden formarse derivaciones; por ejemplo, la siguiente derivación de siete pasos:

(i)	12	(por (b)).	
(ii)	9	(por (a)).	
(iii)	3	(por (c),	usando las líneas (i) y (ii)).
(iv)	$3\sqrt{2}$	(por (d),	usando la línea (iii)).
(v)	$9\sqrt{2}$	(por (d),	usando la línea (ii)).
(vi)	$6\sqrt{2}$	(por (c),	usando las líneas (v) y (iv)).
(vii)	$3 - 6\sqrt{2}$	(por (c),	usando las líneas (iii) y (vi)).

Una *derivación* es, por tanto, una secuencia finita de palabras que puede ser producida de acuerdo con las reglas dadas. El número de palabras se denomina *longitud* de la derivación. La longitud del ejemplo anterior es 7.

Naturalmente, un conjunto de reglas de este tipo (llamado por lo común *sistema de reglas*) no suministra, en general, ningún algoritmo en el sentido estricto indicado en el apartado 1 de esta sección, puesto que no determina en qué orden serial deben ser aplicadas las distintas reglas. A partir de un sistema de reglas dado, se puede producir normalmente una pluralidad arbitraria de diferentes derivaciones. ¿Cuál es entonces la conexión entre un «procedimiento» dado por un sistema de reglas y un algoritmo en sentido propio? Por de pronto puede constatarse (en el Ejemplo 1 y en cualquiera de los restantes) que toda regla particular de un sistema de reglas describe un algoritmo real, y un algoritmo que es, por cierto, conclusivo [8]. Además, para cada derivación *concreta*

[8] Ocasionalmente intervienen «reglas» que no pueden ser inmediatamente concebidas como algoritmos conclusivos. Una «regla» de esa índole podría ser, por ejemplo, ésta: *Fórmese un múltiplo de un número (ya dado) n*. Obviamente, no hay aquí un algoritmo en el sentido estricto, pues no se dice por qué número *k* habría que multiplicar, de modo que el procedimiento no está exento de ambigüedad. Se puede, sin embargo, sustituir esta «regla» por reglas que son algoritmos (conclusivos), si bien recurriendo, ciertamente, a la adecuada utilización de un sistema de reglas superpuesto (véase lo

puede aducirse fácilmente un procedimiento general en el sentido especificado en el apartado 1, de acuerdo con el cual puede producirse tal derivación. Este algoritmo consta de reglas pertenecientes al sistema de reglas, junto con una instrucción complementaria que especifica en qué orden serial deben ser aplicadas dichas reglas (pudiendo una misma regla ser aplicada más de una vez). En este sentido, un sistema de reglas puede ser concebido como una fuente suministradora, en general ilimitada, de algoritmos.

Adviértase que no siempre es posible prolongar en un paso más una derivación previamente dada con ayuda de una regla previamente dada, y que no se puede comenzar una derivación con una regla arbitraria. Así, en el Ejemplo 1 no se puede aplicar en el primer paso ni la regla (c) ni la regla (d).

El Ejemplo 1 se refiere a un caso particularmente simple de sistema de reglas. En general, la situación se complica cuando se «superponen» entre sí varios sistemas de reglas $R_{11},\ldots, R_{1m_1}$; $R_{21},\ldots, R_{2m_2}$; $R_{k_1},\ldots, R_{km_k}$ (cf. Ejemplo 2). En la aplicación de una regla de un sistema posterior se da generalmente por supuesto que previamente se ha obtenido ya una o varias palabras en sistemas anteriores. Se considera que una palabra es *derivable* en el sistema total si se la puede obtener por aplicación de una regla del *último* sistema $R_{k_1},\ldots,R_{km_k}$.

En lugar de dar una definición general, que sería bastante complicada, nos limitamos a dar un ejemplo de *sistemas de reglas superpuestos*. En el sistema de reglas de este ejemplo pueden derivarse —aun-

---

que sigue en el texto principal). Pues se puede producir primero un número k con ayuda de un sistema de reglas adicional, y acogerse luego a una regla que prescriba la formación, a partir de n y k, del producto nk.

que no todas— algunas tautologías (esto es, fórmulas universalmente válidas) del cálculo de enunciados.

Ejemplo 2. El sistema total que se va a exponer a continuación consiste en la superposición de tres sistemas. El *primer sistema* sirve para la producción de *variables enunciativas*. Estas son palabras especiales sobre el alfabeto {O, I}.

R_{11}: Escríbase O.
R_{12}: Añádase la letra I.

Variables enunciativas son, por ejemplo, O, OI, OII.

El *segundo sistema* proporciona *fórmulas (expresiones)* de la lógica de enunciados. Estas son palabras especiales sobre el alfabeto {O, I, →, (,)}

R_{21}: Escríbase una palabra que haya sido obtenida mediante el primer sistema.

R_{22}. Si las palabras W_1, W_2 han sido ya obtenidas, escríbase $(W_1 \rightarrow W_2)$. Fórmulas son, por ejemplo, OII, ((OII → O II) → O II).

Con ayuda del *tercer (último) sistema* se obtienen *tautologías*. Las tautologías son fórmulas especiales.

R_{31}: Si las palabras W_1, W_2 han sido obtenidas ya en el segundo sistema, escríbase $(W_1 \rightarrow (W_2 \rightarrow W_1))$.

R_{32}: Si las palabras W_1, W_2 han sido ya obtenidas en el segundo sistema, escríbase $((W_1 \rightarrow (W_1 \rightarrow W_2)) \rightarrow (W_1 \rightarrow W_2))$.

R_{33}: Si las palabras W_1, W_2, W_3 han sido obtenidas ya en el segundo sistema, escríbase $((W_1 \rightarrow W_2) \rightarrow ((W_2 \rightarrow (W_1 \rightarrow W_2))$.

R_{34}: Si las palabras $(W_1 \rightarrow W_2)$ y W_1 han sido ya obtenidas en el tercer sistema, escríbase la palabra W_2 *(modus ponens)*.

Para el concepto de cálculo, véase:
Lorenzen, P.: *Einführung in die operative Logik und Mathematik*, Springer, Berlin-Göttingen-Heidelberg, 1955.
Curry, H. B.: *Calculuses and Formal Systems*. *Lógica*, *Studia Paul Bernays dedicata*, Editions du Griffon, Neuchátel, 1959, 45-69.
Véase también *Dialéctica*, 12 (1958), 249-273.
La gödelización fue utilizada por primera vez en:
Gödel, K.: *Sobre proposiciones formalmente indecidibles de los Principia Mathematica y sistemas afines* [1931]. Introducción de R. Braithwake. Traducción del alemán de Manuel Garrido, Alfonso Garcia Suárez y Luis Valdés, Valencia, Cuadernos Teorema, 1980. Una versión castellana del mismo artículo figura en; Kurt Gödel, *Obras completas*, introducción y traducción de Jesús Mosterín. Madrid, Alianza Editorial, 1981, pp. 45-89.
Para la realización de algoritmos, véase:
Menninger, K.: *Zahlwort und Ziffer* I, II. Vandenhoeck & Ruprecht 1957, Góttingen, 1958.

§ 2. Los conceptos fundamentales de la teoría de la constructividad

Del concepto de algoritmo podemos extraer toda una serie de importantes conceptos ulteriores. Algunos de esos conceptos son los de computabilidad, enumerabilidad, decidibilidad y generabilidad. En esta sección definiremos dichos conceptos y estableceremos entre ellos relaciones simples.

1. Funciones computables. En matemática ocurren muy frecuentemente funciones que son tales que existe un algoritmo conclusivo que proporciona, para cualquier argumento dado, el valor de la función. Tales funciones son llamadas *funciones computables*. Computables son, por ejemplo, las funciones $x + y$, xy, x^y, donde puede tomarse como valores de los argumentos cualesquiera números naturales x e y.

Al decir (aquí y en lo que sigue) que los argumentos son números naturales, conviene advertir que sería más correcto en este contexto decir que los argumentos (si, por ejemplo, preferimos la notación decimal) son palabras sobre el alfabeto {0, 1, 2,...,9}. Al dominio de nuestras consideraciones pertenecen tan sólo aquellas funciones cuyos argumentos y valores son palabras, o en cualquier caso pueden ser caracterizadas inequívocamente por palabras sobre un alfabeto. Decir, por ejemplo, que la función suma es computable significa que existe un procedimiento general con cuya ayuda podemos obtener de manera puramente esquemática, para dos números cualesquiera dados en notación decimal, la suma de ellos también en notación decimal.

El concepto de computabilidad tiene sentido no solamente en el caso en que los argumentos y valores de una función sean números *naturales*, sino también cuando son *enteros* o *racionales*. Podemos, por ejemplo, representar números enteros y racionales mediante palabras finitas (por ejemplo, —34:49). Así, por ejemplo, la función suma x + *y* es también computable si hacemos uso de argumentos racionales. Por otra parte, la situación se torna harto distinta en el caso de los números *reales*. Desde el punto de vista de la matemática clásica existe un número no-numerable de números reales. Así pues, no podemos representar todo número real por una palabra de un alfabeto finito previamente dado, puesto que hay sólo un número numerable de tales palabras (toda palabra es una línea finita de signos sobre un repertorio finito de letras).

De ahora en adelante supondremos que, para una *función computable f*, el dominio de argumentos de *f* consta de todas las palabras *W* sobre un alfabeto finito 𝔄, mientras que los valores de *f* son palabras sobre

un alfabeto finito $\mathfrak{B}$ [9]. (Funciones de este tipo son especialmente aquellas cuyo dominio de argumentos es el dominio de los números naturales, pero cuyos valores no necesitan ser necesariamente números naturales.) Que una tal función es computable significa que existe un procedimiento general que puede ser descrito en un número finito de enunciados y con cuya ayuda se puede obtener efectivamente el valor de la función *f(W)* para todo posible argumento *W*.

La generalización para funciones de varios argumentos se puede llevar a cabo sin dificultad.

Según los teoremas elementales de la teoría clásica de conjuntos (y aceptando el concepto clásico de función ideado por Dirichlet), existe un número no-numerable de funciones que son definidas para todos los números naturales y cuyos valores son números naturales. Por otra parte, sólo existe ciertamente un número numerable de dichas funciones que son computables. Esto es así, porque para toda función computable existe una instrucción de computación de longitud finita (cf. §3), y solamente puede existir un número numerable de tales instrucciones. Es, por decirlo así, una excepción que una función sea computable.

(Ejemplos de funciones reales que no son computables se ofrecen en *EDC*. §22).

Ponemos fin a este apartado con una observación fundamental. Hay un cuantificador existencial en la definición del concepto de función computable: una función *f* es computable si *existe* una instrucción tal que... Este cuantificador existencial se entenderá siempre aquí en el sentido de la *lógica clásica*. Parece

[9] Si no damos por supuesto que *toda* palabra sobre $\mathfrak{A}$ es un argumento de *f*, llegamos al concepto de *función parcialmente computable*. Pero no lo trataremos en este libro.

que nos encontraríamos con dificultades si intentásemos interpretar el cuantificador como un cuantificador existencial de una lógica *constructiva* [10]. A pesar de la interpretación clásica del cuantificador existencial, la computabilidad de funciones reales se demostrará, en general, desde luego, constructivamente, suministrando un procedimiento de computación de forma explícita.

La observación sobre la interpretación clásica del cuantificador existencial se aplica también, *mutatis mutandis*, a los conceptos de enumerabilidad, decidibilidad y generabilidad.

2. Conjuntos y relaciones enumerables. Sea f una función computable cuyo dominio de argumentos es el dominio de los números naturales. Entonces queda determinada, para el dominio M de los valores de f, una secuencia natural $f(0)$, $f(1)$, $f(2)$,..., que recorrerá M posiblemente con repeticiones. Podemos decir que la función f *enumera* los elementos de M en la secuencia $f(0)$, $f(1)$, $f(2)$,... En general diremos que un conjunto M de palabras sobre un alfabeto es un *conjunto enumerable* si existe una función computable cuyo dominio de valores coincide con M. Por lo demás, diremos también que el *conjunto vacío de palabras* es enumerable.

Por supuesto, debemos distinguir estrictamente entre el concepto de conjunto enumerable y el concepto de conjunto *numerable*. Todo conjunto enumerable es naturalmente a lo sumo numerable (es decir, finito o numerable) en el sentido de la teoría de conjuntos. Sin embargo, la imposibilidad de que todos los conjuntos numerables sean enumerables puede ha-

[10] Cf. las referencias sobre la tesis de Church al final de § 3.

cerse fácilmente plausible. Existe concretamente, al menos, un número numerable de conjuntos enumerables sobre un alfabeto finito fijado, porque (aparte del conjunto vacío) existe, para cualquier conjunto enumerable, una función computable que lo enumera y porque, como hemos visto en el apartado 1, existe solamente un número numerable de funciones computables. Por otra parte, existe un número no-numerable de conjuntos de palabras sobre todo alfabeto que contenga al menos una letra. Para ejemplos reales de conjuntos no enumerables cf. *EDC* §28.

Lo mismo que podemos hablar de conjuntos enumerables, también podemos hablar de *relaciones (predicados) enumerables*. Consideremos una relación binaria R (es decir, una relación de dos argumentos) entre palabras sobre un alfabeto $\mathfrak{A}$. Que R es enumerable significa que existen dos funciones unarias computables (es decir, funciones de una variable) f y g tales que la secuencia de pares ordenados $(f(0), g(0))$, $(f(1), g(1))$, $(f(2), g(2))$,..., recorre (quizá con repeticiones) el conjunto de todos los pares ordenados que están en la relación R. Por lo demás, diremos que la *relación vacía* es enumerable. En consecuencia, podemos introducir para cada n el concepto de relación enumerable de n argumentos.

3. Conjuntos y relaciones decidibles. Nos ocuparemos ahora de una expresión que se utiliza con frecuencia en conexión con algoritmos conclusivos. Comenzaremos aduciendo algunos ejemplos.

(1) Es *decidible* si un número natural es o no primo.
(2) Es *decidible* si un sistema de ecuaciones lineales es o no soluble.

(3) *No* es *decididle* si una fórmula del cálculo de predicados es o no válida.

Si intentamos encontrar una estructura común en estos ejemplos, vemos que en cada instancia dos conjuntos[11] M_1 y M_2 están puestos en una relación, a saber: en (1) el conjunto M_1 de los números primos con el conjunto M_2 de los números naturales, en (2) el conjunto M_1 de los sistemas de ecuaciones lineales solubles con el conjunto M_2 de todos los sistemas de ecuaciones lineales, en (3) el conjunto M_1 de las fórmulas válidas del cálculo de predicados con el conjunto M_2 de todas las fórmulas del cálculo de predicados. M_1 es siempre un subconjunto de M_2.

En todos los casos, M_1 y M_2 son conjuntos *de palabras* (o pueden, en cualquier caso, ser considerados como tales). Nos limitaremos a ilustrar esto más detalladamente empleando tan solo el Ejemplo (2). Basta con advertir que todo sistema de ecuaciones lineales puede ser escrito como una palabra. Al mostrar esto nos limitaremos a ecuaciones lineales cuyos coeficientes son dados como enteros en notación decimal[12]. Entonces, podemos representar un tal sistema de ecuaciones lineales por una palabra sobre el alfabeto $\mathfrak{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, —, =, x, ;\}$, por ejemplo, el sistema

$$3x_1 — 4x_2 = 5$$
$$2x_1 + x_2 = —6$$

[11] O propiedades (podemos identificar una propiedad con el conjunto de cosas que poseen dicha propiedad).

[12] En el álgebra lineal, la aserción (2) se establece con frecuencia sin aducir los detalles de cómo son dados los coeficientes. Sin embargo, mediante un ejemplo se explica fácilmente que es necesario postular la forma en que son dados los coeficientes. Sea a = 0 ó 1, según que la conjetura de Fermat concerniente a su «último proble-

por la palabra

$$+ 3x_1 — 4x_2 = +5 \; ; +2x_1 + 1x_2 = —6.$$

Sean M_1 y M_2 conjuntos de palabras sobre un alfabeto fijado $\mathfrak{A}$, y sea $M_1 \subset M_2$. En tal caso, diremos que M_1 *es decidible relativamente a* M_2 si existe un algoritmo conclusivo [13] con cuya ayuda podamos determinar efectivamente para toda palabra de M_2 si pertenece o no a M_1. A un tal algoritmo se le denomina un *procedimiento de decisión*.

Junto al concepto de decidibilidad *relativa* que acabamos de considerar, podemos introducir también un concepto de decidibilidad *absoluta*. Un conjunto M_1 de palabras sobre un alfabeto $\mathfrak{A}$ es denominado sencillamente *decidible*, si M_1 es decidible relativamente al conjunto M_2 de todas las palabras sobre $\mathfrak{A}$ [14].

Si, en el ejemplo (1), damos los números naturales en notación decimal con la ayuda de los símbolos 0, 1,..., 9, entonces cada palabra formada a partir de estos símbolos designa un número natural. Se puede, por tanto, decir en la terminología que acabamos de introducir que el conjunto de los números primos es decidible.

Una relación de *n*-términos puede ser considerada como un *conjunto* de *n*-tuplos. Por tanto, el concepto de decidibilidad, lo mismo que el de enumerabilidad, puede ser trasladado a las relaciones.

Todo conjunto *finito* (y, por consiguiente, toda relación *finita)* es decidible. Un procedimiento de deci-

---

ma» sea o no verdadera. Entonces, no podemos saber hoy si la ecuación $ax = 1$ es o no soluble.

[13] Este cuantificador existencial es también clásico; cf. el final del apartado 1.

[14] Se debe tener en cuenta que tanto con la decidibilidad relativa como con la absoluta nos referimos al alfabeto básico $\mathfrak{A}$.

sión consiste en escribir todos los elementos del conjunto en una lista y comprobar, para cualquier palabra dada, si aparece o no en la lista. Así, y en particular, es decidible todo conjunto que conste solamente de un único elemento [15].

Discutiremos ahora unas cuantas relaciones simples entre la decidibilidad absoluta y relativa y la enumerabilidad. Las consideraciones que conducirán a una verificación (lo mismo que algunas otras más de las que nos ocuparemos en los apartados siguientes) no pueden ser, «por la naturaleza misma de esta materia», pruebas exactas. Juntas, sin embargo, contribuyen a la clarificación de estos conceptos intuitivamen-

[15] Este es el momento de indicar que debemos tener en cuenta positivamente el hecho de que en todo lenguaje ordinario de los matemáticos podemos encontrar expresiones en las que ocurre y es utilizada la palabra «decidible» de una forma diferente a la de los ejemplos (1), (2) y (3) de arriba. Consideremos como ejemplo una posible aserción α del siguiente tipo: «El último problema de Fermat es decidible». Si se da por supuesto que por decidibilidad se entiende aquí decidibilidad absoluta, la aserción α sería trivialmente verdadera, puesto que todo conjunto de un solo elemento (aquí, el conjunto que contiene α como su único elemento) es decidible. Pero alguien que estableciese la aserción α *no* estaría de acuerdo en general con esta interpretación. Es posible que más bien significase lo siguiente: «O bien existe un contraejemplo de la aserción de Fermat (el cual se hallará después de buscarlo suficientemente), o bien existe una prueba matemática de la misma».

Si ahora suponemos que todos los métodos de pruebas matemáticas admitidos hoy y en el futuro definen juntos un algoritmo Γ (quizá muy complicado) (esta suposición es en realidad más que cuestionable), entonces, desde nuestro punto de vista podría afirmarse lo siguiente acerca de la aserción α entendida de esta forma:

(1) Podría ser que el motivo para la aserción α fuese el hilbertiano «en matemáticas no existe ningún ignorabimus». Esta motivación ha de rechazarse según nuestros conocimientos de hoy en día, pues contradice la indecidibilidad de la aritmética establecida por Gödel.

(2) La respuesta a la cuestión de si la aserción de Fermat es susceptible de prueba (en el caso de que no sea refutable por un contraejemplo) depende de Γ, y Γ no es conocido.

te dados. Más adelante, cuando reemplacemos los conceptos intuitivos discutidos aquí por otros exactos, estaremos en la posición de ofrecer una prueba estricta de las relaciones análogas.

(a) Sean M_0, M_1, M_2 conjuntos de palabras sobre un alfabeto $\mathfrak{A}$. *Si M_1 es decidible relativamente a M_2, y M_0 está contenido en M_1, entonces M_0 es decidible relativamente a M_1 si y sólo si M_0 es decidible, relativamente a M_2 (transitividad de la decidibilidad relativa)*. Pues si M_0 es decidible relativamente a M_2, entonces a fortiori es decidible relativamente a M_1. Si inversamente, M_0 es decidible relativamente a M_1, entonces es también decidible relativamente a M_2. Sea W una palabra arbitraria de M_2. Primero, se determina si W pertenece o no a M_1. Si W no pertenece a M_1, entonces definitivamente W no pertenece a M_0. Pero si W pertenece a M_1, entonces podemos decidir, de acuerdo con la hipótesis, si W es o no un elemento de M_0.

Si M_2 es el conjunto de todas las palabras sobre $\mathfrak{A}$, entonces, de acuerdo con la definición, la decidibilidad relativamente a M_2 puede ser reemplazada por la decidibilidad absoluta. De acuerdo con esto tenemos como una aplicación de (a):

(b) Sean M_0 y M_1 conjuntos de palabras sobre un alfabeto $\mathfrak{A}$. *Si M_1 es decidible, y M_0 está contenido en M_1, entonces M_0 es decidible relativamente a M_1 si y sólo si M_0* es *decidible*.

En muchos ejemplos importantes aparece claro qué alfabeto proporciona la base y si M_2 es absolutamente decidible (cf., por ejemplo, (1) y (2) más arriba). En estos casos, la relación de decidibilidad de M_1 relativamente a M_2 puede ser reemplazada por la propiedad de decidibilidad absoluta de M_1.

(c) Sea M un conjunto de palabras sobre un alfa-

beto $\mathfrak{A}$. Asignamos a este conjunto una función $\chi_M(W)$ que es definida para toda palabra W sobre $\mathfrak{A}$ y para la cual

$$\chi_M(W) = \begin{cases} 0, \text{ si } W \in M \\ 1, \text{ si } W \notin M \end{cases}$$

es válida.

χ_M es llamada *la función característica* de M. Tenemos que *M es decidible si y sólo si la función característica χ_M es computable*. A saber, si M es decidible, entonces debemos determinar, para cualquier W dada, si $W \in M$ *ó* $W \notin M$. Según lo cual debemos escribir 0 ó 1 respectivamente. Así, obtenemos el valor de la función $\chi_M(W)$. Esto es un algoritmo conclusivo de la computación de χ_M. Inversamente, si χ_M es computable, podemos decidir, por computación de $\chi_M(W)$, si $W \in M$ ó $W \notin M$.

4. Conjuntos y relaciones generables. En § 1, apartado 6, introdujimos el concepto de sistema de reglas y el de una deducción producida según dicho sistema de reglas. Decimos que una palabra *W es deducible* con la ayuda de las reglas de un sistema de reglas dado si es posible obtener una deducción con la ayuda de la regla del sistema cuya última palabra coincide con W [16]. Un conjunto M de palabras sobre un alfabeto $\mathfrak{A}$ es denominado *generable* si existe un sistema de reglas tal que una palabra W sea deducible con la ayuda de las reglas del sistema si y sólo si pertenece a M.

Podemos hablar de *relaciones generables* lo mismo

[16] En el caso de un sistema de reglas *superpuesto* (cf. § 1.6), la última palabra de la deducción debe ser formada con la ayuda de una regla del último sistema.

que en los casos de enumerabilidad y de decidibilidad.

Los ejemplos (1) y (2) de § 1.6 muestran la generabilidad del módulo generado por 3 y $3\sqrt{2}$, y la de un cierto conjunto de tautologías.

Nos ocuparemos ahora de unas cuantas relaciones entre la generabilidad y los conceptos introducidos antes. Ante todo, tenemos:

(d) *Un conjunto M de palabras sobre un alfabeto* $\mathfrak{A}$ *es generable* [17] *si y sólo si M es enumerable*. En primer lugar, sea *M* enumerable. Entonces, o *M* es vacío (y, por tanto, generado con la ayuda de una regla que no es aplicable), o *M* es el dominio de valores de una función computable *f*. Sea *R* un algoritmo con cuya ayuda podemos calcular el valor de la función *f(n)* para cualquier *n*. Ahora consideraremos un sistema de reglas superpuesto.

Primer sistema. Se utiliza para generar los números naturales *n*.

Segundo (último) sistema. Consta de *una* regla que dice: Para cualquier palabra *n* que pueda ser obtenida con la ayuda del primer sistema, genérese *f(n)* mediante el algoritmo *R*.

Es obvio que en el sistema de reglas superpuesto son exactamente generadas aquellas palabras que pertenecen a *M*.

Inversamente, sea *M* un conjunto generable. Sea generado *M* por un sistema de reglas *R* (en general, superpuesto) con un número finito de reglas. Tenemos que mostrar que *M* es enumerable. Para cualquier número *k* puede existir un número finito de deducciones de longitud *k* (quizá, incluso ninguna)[18].

[17] En la literatura, *«conjuntos generables»* se refieren con frecuencia a *«conjuntos generados»*.

[18] Esto se aplica incluso a las deducciones generalizadas para las cuales no se exige que la última palabra haya sido formada con la

Ahora, si se examinan primero todas las deducciones de longitud 1, entonces vemos que podemos producirlas y colocarlas en un sistema estándar (por ejemplo, lexicográficamente o con referencia a las reglas dadas). Sean $W_{01},...,W_{l1}$ las últimas palabras de esas pruebas en la secuencia estándar (W_{01} ,..., W_{l1} pueden coincidir parcialmente la una con la otra). Entonces definimos $f(0) = W_{01}$, $f(1) = W_{11}$,..., $f(l_1) = W_{l1}$. Ahora examinamos todas las deducciones de longitud 2. Estas pueden también ser producidas efectivamente. Sean W_{12} ..., W_{l2} las últimas palabras de estas deducciones ordenadas según el mismo método anterior. Entonces definimos $f(l_1 + 1) = W_{12},...,$ $f(l_1 + l_2) = W_{l2}$. Continuamos de esta forma. Si no existe para un k *ninguna* deducción de longitud k, vamos inmediatamente a $k + 1$. Se presentan ahora dos posibilidades: (1) No existe ninguna deducción de longitud k para cualquier k sea el que sea. Entonces M es vacío y, por tanto, enumerable por definición. (2) Hay un número k para el que existe una deducción de longitud k. Entonces existen también deducciones de longitud $2k$, $3k$, etc., que pueden obtenerse de una forma trivial mediante repeticiones de la deducción de longi-

---

ayuda de una regla del *último* sistema (cf. § 1.6). Para dichas deducciones generalizadas, la aserción se obtiene fácilmente por inducción sobre la longitud. Para ello hay que tener en cuenta los hechos de que una deducción generalizada de longitud $k + 1$ se convierte en una deducción generalizada de longitud k si quitamos la última palabra, y de que a partir de una deducción generalizada dada de longitud k se puede formar solamente un número finito de deducciones generalizadas de longitud $k + 1$. Concretamente, sólo hay un número finito de reglas a nuestra disposición y toda regla sólo es aplicable a un número finito de posibles combinaciones de líneas de la deducción generalizada de longitud k. En todo caso particular, el resultado queda determinado inequívocamente porque se ha puesto la condición de que toda regla ha de ser un algoritmo en sentido estricto.

tud k. Esto asegurará que la función anteriormente definida $f(n)$ sea definida para todos los n. Debido a la definición de f queda claro que $f(n)$ puede ser computada efectivamente para todos los n (el método dado anteriormente describe el procedimiento general). Así pues, f es una función computable. El dominio de valores de f coincide obviamente con M.

(e) Sea E el conjunto de *todas* las palabras sobre un alfabeto $\mathfrak{A} = \{a_1,\ldots,a_N\}$. E es generado con la ayuda de las siguientes reglas:

R_0: Escríbase la palabra vacía.
R_1: Añádase la letra a_1.
. .
. .
. .
R_N: Añádase la letra a_N.

(f) Sea M un conjunto de palabras sobre $\mathfrak{A}$, y sea $\overline{M}$ el complemento de M relativamente a E. Tenemos entonces que *M es decidible si y sólo si tanto M como $\overline{M}$ son generables*. En lugar de *generables* también podríamos decir (por (d)) *enumerables*. Para demostrar esta aserción suponemos primero que M es decidible. Ahora, consideramos el siguiente sistema de reglas superpuesto.

Primer sistema. El sistema de (e) para la generación de todas las palabras de E.

Segundo (último) sistema. Contiene sólo la siguiente regla: Compruébese si una palabra W generada con la ayuda del primer sistema pertenece o no a M. Si $W \in M$, escríbase W. Si $W \notin M$, entonces considérese la regla inaplicable.

Está claro que con la ayuda de este sistema se puede

generar exactamente el conjunto M. Si cambiamos el último sistema de la forma obvia, veremos que $\overline{M}$ es también generable.

Supongamos ahora que tanto M como $\overline{M}$ son generables. Si es vacío M o $\overline{M}$, entonces M es decidible de forma trivial. Por tanto, daremos por sentado que ni M ni $\overline{M}$ son vacíos. Entonces existen de acuerdo con (d) funciones computables f y g, tales que el dominio de valores de f coincide con M, y el de g con $\overline{M}$. Para decidir si una palabra arbitraria dada W de E pertenece o no a M, podemos usar ahora el siguiente algoritmo. Compútense los valores $f(0)$, $g(0)$, $f(1)$, $g(1)$, $f(2)$, $g(2)$,..., uno por uno, y compruébese después de cada computación de un $f(n)$ o un $g(n)$ si el valor de la función obtenida coincide o no con W. Si no coincide con W, prosígase la computación. Si coincide con W, entonces W es igual a un $f(n)$ o a un $g(n)$. En el primer caso es un elemento de M; en el segundo es un elemento de $\overline{M}$, y, por tanto, no es un elemento de M. En este estadio se ha logrado la decisión y termina el procedimiento. Adviértase que el procedimiento termina en *todo caso*, pues toda palabra es o un elemento de M o de $\overline{M}$ y, de este modo, se encuentra en el dominio de valores de f o g.

5. La invarianza de los conceptos constructivos bajo la gödelización. Sea G una gödelización de las palabras sobre un alfabeto $\mathfrak{A}$ de N elementos. Sea M un conjunto de palabras sobre $\mathfrak{A}$. Asignamos a M el conjunto $\tilde{M}$ de los números de Gödel de los elementos de M. Tenemos entonces el

Teorema. M es enumerable (decidible respectivamente) si y sólo si $\tilde{M}$ es enumerable (decidible respectivamente).

Demostración.

(a) Sea M enumerable. Se dice que es enumerado por la función f. Entonces, obviamente, la función f_1, definida por $f_1(n) = G(f(n))$, es una enumeración de $\tilde{M}$. Inversamente, si $\tilde{M}$ es enumerable y si es enumerado por la función h, entonces M es enumerado por la función h_1 que es definida por $h_1(n) = G^{-1}(h(n))$.

(b) Sea M decidible. Entonces M es también decidible. Para establecer si una palabra W pertenece a M debemos computar el número $G(W)$ y establecer si $G(W)$ pertenece a $\tilde{M}$. Si M es decidible, entonces M es también decidible. Para determinar si un número n pertenece a $\tilde{M}$ debemos determinar primero si n es, en suma, el número de Gödel de una palabra sobre $\mathfrak{A}$ (cf. § 1.3). Si no es éste el caso, entonces ciertamente $n \notin M$. Si, por otra parte, n es el número de Gödel de una palabra W sobre $\mathfrak{A}$, entonces debemos construir W y determinar si W pertenece a M.

Nota. Este teorema es válido también, *mutatis mutandis*, para relaciones en lugar de conjuntos.

§ 3. El concepto de máquina de Turing como sustituto matemático exacto del concepto de algoritmo

Para poder dar pruebas de indecidibilidad o de otras imposibilidades en el dominio de lo constructivo, no es suficiente tener tan sólo una representación intuitiva (por clara que pueda ser) del concepto de algoritmo. Por el contrario, es preciso contar con una definición exacta de lo que llamamos un procedimiento general como base de tales consideraciones. En esta sección trataremos cuestiones que conducen al re-

emplazo del concepto de algoritmo por el concepto matemático exacto de las llamadas máquinas de Turing.

El primero en sugerir la identificación del concepto de algoritmo, originalmente dado de modo intuitivo, con un cierto concepto exactamente definido fue Alonzo Church en 1936 (cf. § 4.3). La llamada *tesis de Church* es admitida hoy por la inmensa mayoría de los lógicos matemáticos. Una breve revisión de los argumentos en su favor se dio en el prefacio. *Críticas* de este punto de vista han sido aducidas por L. Kalmar (posiblemente no toda función computable en el sentido intuitivo es computable en el sentido de Church) y por R. Péter (posiblemente no toda función computable en el sentido de Church es computable en el sentido intuitivo). Sobre esa materia deben consultarse las referencias bibliográficas citadas al final de esta sección.

1. Observaciones preliminares. Además del concepto de máquina de Turing discutido aquí, hay otras sugerencias sobre el reemplazo del concepto de procedimiento general por un concepto matemático exacto. Discutiremos algunas de estas sugerencias más adelante. Hemos elegido las máquinas de Turing como punto de partida porque éste parece ser el enfoque más fácil y natural. Comenzamos con una observación preliminar acerca del método. Si queremos demostrar que el concepto exacto de máquina de Turing es una versión adecuada del concepto intuitivo de algoritmo, entonces, por la misma naturaleza de la materia, sólo podemos aducir argumentos de la plausibilidad de nuestra suposición. Tal consideración de plausibilidad no puede ser más que una apelación a la

mayor o menor experiencia con algoritmos adquirida por el matemático durante su vida.

Una vez definida, aparecerá inmediatamente claro que toda máquina de Turing describe un procedimiento general. La única cuestión es si todo procedimiento general puede ser realizado por una máquina de Turing adecuada. Si disponemos de un argumento con el que intentamos mostrar que, en principio, *todo* algoritmo puede ser reproducido por una máquina de Turing adecuada (o por cualquier otro reemplazo preciso sugerido), entonces probablemente encontraremos siempre ejemplos de algoritmos que no caen directamente bajo nuestra consideración. En tal caso, debemos extender la argumentación de acuerdo con esto, y así sucesivamente. Cuando hayamos llevado a cabo un número suficiente de consideraciones, y de consideraciones adicionales de este tipo, llegaremos finalmente a la convicción (y esa es la experiencia de las últimas décadas) de que no vale la pena emprender experimentos una y otra vez con nociones nuevas. La aserción de que *todos* los algoritmos pueden ser comprendidos por las máquinas de Turing se considerará finalmente verificada de la misma forma que la aserción en física sobre la imposibilidad de un *mobile perpetuum*. Esta aserción es también tal que, desde un tiempo a esta parte, la gente ha rehusado hacer nuevos experimentos sobre ella.

Resulta muy notable el hecho de que se *pueda probar de forma puramente matemática que los reemplazos sugeridos del concepto de algoritmo por varios conceptos matemáticos exactos (que tienen su origen en puntos de partida muy diferentes) sean todos equivalentes* [19]. Ello es al menos una señal del hecho

[19] Cf. *EDC* § 15, § 19, § 30 y § 31.

de que estamos tratando un concepto fundamental.

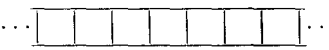
2. Algoritmos y máquinas. Hemos indicado en § 1 que para que un método sea un procedimiento general, ha de estar prescrito hasta los menores detalles, de suerte que no se requiera el más leve esfuerzo de imaginación creadora para llevarlo a cabo. Pero, si todo queda determinado así en detalle, entonces será obviamente posible abandonar la ejecución del método a una máquina, máxime si la máquina en cuestión es totalmente automática.

Las máquinas pueden tener muy complicadas estructuras. El propósito de las consideraciones que siguen es dar razón de un tipo de máquinas relativamente simples, que son fáciles de tratar desde el punto de vista matemático. Estas son las máquinas de Turing, respecto de las cuales cabe esperar que todo algoritmo pueda ser ejecutado por una máquina adecuada de este tipo. La fundamentación del hecho de que *todo* algoritmo pueda ser realizado (con las pertinentes adecuaciones) por una máquina de la citada especie, es una cuestión que abordaremos, siguiendo el método de Turing, mediante el análisis del comportamiento de un calculador o computador humano que opere de acuerdo con un método preestablecido.

3. El material de computación. Para fijar nuestras ideas, partiremos de la suposición de que la tarea de un calculador es computar el valor de una función para un argumento dado, de acuerdo con una instrucción dada que contiene toda clase de detalles. El calculador usa para su cálculo una hoja de papel (o más, según se precise). Suponemos que tal hoja está dividida en cuadrantes. Se supone asimismo que el calculador no escribe más de un símbolo en cada

cuadrante. Puede disponer de todos los símbolos de un alfabeto finito $\mathfrak{A} = \{a_1,..., a_N\}$ [20]. El argumento se escribe sobre la referida hoja en estos símbolos al comienzo del cómputo.

Para algunos algoritmos es, sin duda, conveniente tener a nuestra disposición una superficie de cálculo de dos dimensiones. Piénsese, por ejemplo, en el algoritmo usual de división para números naturales. Pero difícilmente podría poner cualquiera en duda que, en principio, no es necesario utilizar una superficie de cálculo de dos dimensiones; siempre podremos salir del paso con una *cinta de cálculo* (brevemente: *cinta)* de una sola dimensión. Dicha cinta está dividida en una secuencia lineal de *cuadrados*. Es preciso que en el curso de la computación pueda disponerse de suficiente número de cuadrados. Lo cual deberá tener lugar de modo que la cinta de cálculo se continúe en ambas direcciones hasta el infinito [21]. De este modo la cinta cobra el siguiente aspecto:



Imaginamos que la cinta va provista de una *dirección*, y hablamos en este sentido de *izquierda* y *derecha* (principio y fin, respectivamente). Los cuadra-

[20] En la práctica matemática de hoy, en que se utilizan diferentes alfabetos (en el sentido ordinario de la palabra) y otros signos, el número de símbolos realmente utilizados es del orden de 200.

[21] Podríamos calificar la frase que hace uso del infinito actual suponiendo una cinta de computación de longitud finita y exigiendo tan sólo que la cinta sea susceptible de ser extendida por cualesquiera de ambos lados *según se precise*. Podemos advertir, empero, que incluso tal suposición representa ya una *idealización* matemática de la situación existente en la realidad, y en todo caso, desde luego, si se concede que el mundo contiene sólo un número finito de partículas materiales.

En principio sería suficiente requerir la extensión de la cinta de

dos de la cinta están *vacíos*, excepto un número finito de ellos, que están *marcados*, esto es, llevan impresa una letra. Con frecuencia es conveniente incluir juntamente con los *símbolos reales* $a_1,\ldots,a_N$, el *símbolo vacío* como un *símbolo ideal*. Representamos el símbolo vacío por a_0 o por *. Así, un cuadrado es o está vacío si y sólo si en él está impreso el símbolo *. A la inscripción que hay en la cinta la denominamos *expresión de cinta*.

4. Los pasos de computación. Dirigiremos ahora nuestra atención a la computación misma. La computación discurre de acuerdo con instrucciones finitas. Intentaremos dar a estas instrucciones una forma estándar. En primer lugar es claro que toda computación puede ser concebida como un proceso susceptible de ser dividido en *pasos aislados*. Un paso de esta índole podría consistir, por ejemplo, en imprimir una letra en un cierto cuadrado vacío. La impresión en un cuadrado sería, en realidad, un proceso continuo. Sin embargo, los detalles de semejante proceso carecen de interés para nuestra presente consideración. Las únicas partes que nos importan del proceso son el comienzo (el cuadrado vacío) y el fin (el cuadrado marcado), y por ello está justificado que hablemos de un *paso* discontinuo [22]. Intentaremos dividir el proceso

cálculo al infinito solamente en una dirección. Ello traería, sin embargo, complicaciones, causadas por la existencia de un cuadrado (el primero de la cinta) que tendría en su vecindad solamente otro cuadrado.

[22] Adviértase que todo proceso puede ser concebido como continuo o discontinuo, según el modo como se lo considere. Los *computadores digitales*, por ejemplo, son considerados como máquinas que operan por pasos (discontinuamente) aun cuando el proceso mismo discurra continuamente. Por otra parte, los *integrafos* son considerados como máquinas que operan continuamente.

entero de computación en pasos de computación tan elementales como sea posible.

Un paso de computación lleva de una cierta *situación inicial* o *configuración inicial* a una nueva *situación* o *configuración*, que es a su vez la configuración inicial para el siguiente paso de computación. Pero existe también la posibilidad de que el calculador cese de operar *(pare)* al alcanzar cierta configuración. Entonces la computación del valor de la función llega a su fin.

Consideremos primero qué puede suceder en un solo paso de computación.

En primer lugar, existe la posibilidad de que se cambie la expresión de cinta. Puesto que deseamos considerar los pasos de cálculo más simples posibles, supondremos que en un paso tal se cambia únicamente el símbolo en un solo cuadrado. Todo cambio de la expresión de cinta consistirá en un cambio en un número finito de cuadrados, y puede ser dividido, por tanto, en pasos elementales aislados de esa índole. Un paso aislado puede consistir en imprimir un símbolo en un cuadrado previamente vacío, o en remover un símbolo de un cuadrado (a este procedimiento se le puede llamar *borrado)*. Mediante el borrado y la impresión subsiguiente podemos cambiar el símbolo de un cuadrado por cualquier símbolo arbitrario. Pero, por ser sumamente conveniente, deberemos permitir explícitamente el cambio inmediato de un símbolo arbitrario a_j en un cuadrado por un nuevo símbolo a_k *(j*, $k = 0,...,N$*)* en *un* solo paso. Caracterizaremos este procedimiento diciendo que se ha escrito *(impreso)* la letra a_k en el cuadrado en cuestión. Esta impresión es, en general, una impresión *sobre*, por la cual desaparece la letra previamente existente. Dicha letra queda sólo si $j = k$.

En el proceso considerado se requiere que el calculador conceda en cada instante especial atención a un cierto cuadrado de la cinta de cálculo en la que (eventualmente) se dispone a imprimir. A este cuadrado lo denominamos *cuadrado escrutado*. En general, es necesario cambiar el cuadrado escrutado en el curso de la computación. El tránsito de un cuadrado escrutado a otro situado a cierta distancia de él, no necesita ser realizado en *un* solo paso. Podemos, a nuestra mayor conveniencia, fraccionarlo en pasos simples que consisten en cambiar el cuadrado escrutado por uno nuevo situado inmediatamente a la derecha o a la izquierda del antiguo. A un tal paso simple, que consiste en cambiar el cuadrado escrutado, lo caracterizaremos brevemente diciendo que *nos movemos* (en la cinta, un cuadrado) *a la derecha* o, respectivamente, *a la izquierda*.

Con ello hemos enumerado ya los diferentes tipos de pasos de cálculo que hemos de considerar. En resumen, los posibles pasos de cálculo son:

a_k: la impresión de un símbolo a_k $(k = 0,..., N)$ en el cuadrado escrutado.
r: el movimiento a la derecha.
l: el movimiento a la izquierda.
h: la parada.

Nos referiremos a estos pasos abreviadamente mediante los respectivos símbolos introducidos a la izquierda.

5. La influencia de la cinta de cálculo en un paso de cálculo. La respuesta a la cuestión de *cuál* de los posibles pasos de cálculo es ejecutado en un cierto estadio del cómputo depende de la configuración en ese momento. Una configuración es dada por tres com-

ponentes, a saber: (1) la instrucción de cálculo, (2) qué es lo que hay ya impreso en la cinta de cálculo en el momento en cuestión, y (3) el cuadrado escrutado en ese instante.

Nos ocuparemos aquí más detenidamente de (2). Cabe plantearse la cuestión de en qué medida la expresión de cinta[23] que se encuentra impresa en la cinta en un determinado momento influye en el siguiente paso. El mínimo requisito de dependencia que podemos establecer aquí es que el símbolo de un solo cuadrado sea decisivo para el siguiente paso, mientras que los símbolos de los cuadrados restantes no jueguen papel alguno en lo que al paso próximo se refiere. Diríamos que el calculador ha de observar el cuadrado en cuestión y que el resultado de la observación, esto es, el símbolo en dicho cuadrado, es esencial para el siguiente paso. Por tanto (temporalmente) llamaremos *cuadrado observado* a este cuadrado crítico.

Aduciremos ahora un argumento de plausibilidad para mostrar que este requisito mínimo de dependencia es suficiente. En otras palabras, no es necesario admitir que un paso de cálculo esté influido por una parte mayor ni por el total de la expresión de cinta. En todo caso en que así sucediese, podríamos cambiar la instrucción de cómputo, de suerte que nos permitiese discurrir mediante pasos de cálculo tales que cada uno de ellos dependiese únicamente de la observación de

[23] Podemos concebir la cinta de cálculo con su correspondiente expresión como una extensión de la memoria humana. Pero debiéramos tener presente que, cuando un hombre calcula sobre una hoja de papel, utiliza tanto su propia memoria como la del papel. Si deseamos reproducir una computación humana mediante una máquina de Turing, habremos de transferir a la cinta la parte real de la memoria humana relevante para ese cálculo.

un *solo* cuadrado. Veamos un ejemplo característico al respecto: Si deseamos añadir dos números, cada uno de los cuales esté dado por *un* digito en notación decimal —verbigracia, 3 + 4, 3 + 2 ó 6 + 2—, entonces, y por de pronto, formularemos las respectivas instrucciones de cómputo de la siguiente manera [24]:

Si se observa un 4 en el último cuadrado y un 3 en el antepenúltimo, imprímase 7 como resultado.

Si se observa un 2 en el último cuadrado y un 3 en el antepenúltimo, imprímase 5 como resultado.

Si se observa un 2 en el último cuadrado y un 6 en el antepenúltimo, imprímase 8 como resultado. Etc.

En lugar de esta instrucción, consideremos la siguiente:

I. Observa el último cuadrado. Según que se vea en él 0, 1,..., ó 9, continúa de acuerdo con II_0, II_1,..., II_9 respectivamente [25].

.....

II_2. Observa el antepenúltimo cuadrado.

.....

Si se ve un 3, imprime el resultado 5.

.....

Si se ve un 6, imprime el resultado 8.

.....

Puede advertirse que esta instrucción conduce al mismo resultado que la instrucción anteriormente considerada y que todo paso aislado depende únicamente de la observación de *un* cuadrado. Merced al

[24] Nos limitamos en este caso a lo esencial y renunciamos a dar una versión completa de la instrucción.

[25] En el futuro expresaremos esto diciendo «salta a II_0, II_1,..., II_9, respectivamente» (es decir, escribiremos «salta a» en lugar de «continúa de acuerdo con»).

mismo método puede mostrarse que podemos dejar de lado la consideración de pasos de cálculo que dependan de la simultánea observación de *k* cuadrados en una sucesión ininterrumpida, donde *k* es un número natural fijo.

Podríamos tener una instrucción en la que el paso de cálculo estuviese determinado por las inscripciones en *k* cuadrados que no se encuentren en inmediata vecindad. Convengamos, por razones de simplicidad, en que k = 2. Si esos dos cuadrados se encuentran muy distantes entre sí, será imposible para el calculador observarlos directamente en un solo acto. El único modo de proceder a que puede ajustarse será observar primero uno de los cuadrados y dirigirse después en busca del otro. En caso de distancias muy largas, la búsqueda deberá efectuarse de acuerdo con una instrucción que sólo podrá ser ejecutada por pasos [26]. De este modo vemos que los pasos de cálculo que dependen de la observación de varios cuadrados situados entre sí a una distancia arbitraria no es necesario que se presenten como pasos elementales de cálculo.

Una consideración similar muestra que tampoco es necesario tomar en cuenta pasos de cálculo que dependan de la observación de *toda* la cinta de cálculo.

Así pues, sólo precisamos tomar en consideración pasos de cálculo tales que dependen del símbolo en un solo cuadrado, el *cuadrado observado*. Para todo pa-

[26] Aclaremos esto con un ejemplo. Pudiera ser que el segundo cuadrado efectivo sea el primer cuadrado, a la izquierda del cuadrado primeramente observado que esté inmediatamente a la izquierda de un cuadrado que contenga el símbolo l. Entonces, este segundo cuadrado efectivo puede ser encontrado en principio sólo comenzando por el cuadrado observado primeramente y yendo hacia la izquierda paso a paso hasta llegar a un cuadrado del tipo dado. Este ejercicio completo se puede solucionar mediante el acoplamiento de diversos pasos de cálculo simples.

so de computación tenemos ahora *dos* cuadrados que son esenciales (para dicho paso), a saber: el cuadrado escrutado (apartado 4) y el cuadrado observado. Una consideración análoga a la que se acaba de realizar para mostrar que es posible atenerse a *un* cuadrado observado muestra que también debe ser posible *identificar* el cuadrado observado con el cuadrado escrutado. Haremos esto a partir de ahora. Así pues, supondremos que, *en cualquier instante*, *la única parte relevante de la expresión de cinta en lo que concierne al siguiente paso de cálculo es el símbolo del cuadrado escrutado*.

De esta simplificación se sigue otra que ya habíamos hecho tácitamente en el apartado 4. Si el cuadrado observado fuese diferente del cuadrado escrutado, entonces tendríamos que incrementar en dos el número de los posibles pasos de cálculo que nos permitiría cambiar el cuadrado observado en el curso de la computación.

6. La instrucción de cálculo. El número de pasos de cálculo necesarios para la determinación de un valor *f(n)* de una función computable *f* se incrementará en general en la medida en que *n* sea más grande. Pero la instrucción para la computación de *f* es de longitud finita. Por tanto, en la computación *de f(n)*, y siendo *n* suficientemente grande, tendrá que ser aplicada una y la misma parte de la instrucción *varias veces* según sea necesario. Así, la instrucción se fraccionará en ciertas *instrucciones parciales* que tendrán que utilizarse repetidamente en el curso del cálculo si fuera necesario (la frecuencia con que tenga que ser utilizadas depende del argumento dado *n)*.

Un ejemplo simple puede ilustrar el concepto de instrucción parcial. Versa sobre el cálculo de un valor

aproximado de $\sqrt{2}$. Tomamos 1.4 como aproximación inicial X_0. Deseamos encontrar una aproximación x_n con un error absoluto menor que $2 \cdot 10^{-10}$. Ello se puede conseguir con la ayuda de una instrucción de cálculo que consta de las siguientes instrucciones parciales, comenzando con la:

Instrucción parcial número 0. Escribe 1.4 como una aproximación. Después salta a la primera instrucción parcial.

Primera instrucción parcial. Comenzando a partir de una aproximación x, computa una nueva aproximación $y = x + (2 - x^2) / 2x$ en notación decimal hasta 12 lugares. Salta después a la segunda instrucción parcial.

Segunda instrucción parcial. La computación llega a su fin si $|y-x| < 10^{-10}$. En otro caso, salta a la primera instrucción parcial, comenzando a partir de la nueva aproximación y.

No debemos confundir tal sistema de instrucciones parciales con un proceso generador (§ 1.6). Allí quedaba abierta la secuencia en la que tenían que aplicarse las reglas, mientras que aquí la secuencia está prescrita sin ambigüedad.

Las instrucciones parciales de este ejemplo contienen en general muchos pasos de cálculo simples. Podemos imaginar ahora que una instrucción se fracciona en instrucciones parciales tan pequeñas que cada una de ellas, junto con la expresión de cinta y el cuadrado escrutado en ese momento, determina *sólo un paso de cálculo*. En ese caso, la instrucción parcial de que se trate únicamente tendrá que decir cuál sea el paso de cálculo a ejecutar y a qué instrucción parcial hay que sujetarse en lo inmediato, ambas cosas de acuerdo con el contenido del cuadrado escrutado.

Imaginemos que las instrucciones parciales están

numeradas sucesivamente, siendo la instrucción parcial número 0 la que determina el comienzo de la computación. Sea el símbolo *b* señal de comportamiento *(b* representa, por tanto, *l*, *r*, *h*, *o* a_0,..., a_N (cf. apartado 4). Entonces tenemos:

la k-ésima instrucción parcial

Si el cuadrado escrutado lleva impreso el símbolo		ejecuta		y salta después a la instrucción parcial	
	a_0		b_0		k_0
.....	a_1	.....	b_1	.....	k_1
.....					
.....	a_N	.....	b_N	.....	k_N

En lugar de la *k*-ésima instrucción parcial, hablaremos también brevemente del *k-ésimo estado*.

Todas las líneas tienen el mismo esquema. Así podemos limitarnos a reproducir la instrucción parcial mediante una tabla, a saber;

$$k \quad a_0 \quad b_0 \quad k_0$$
$$k \quad a_1 \quad b_1 \quad k_1$$

. . . . . .

$$k \quad a_N \quad b_N \quad k_N$$

La *instrucción de cálculo completa* se obtiene en forma estándar colocando las tablas de instrucción parcial unas debajo de otras. La instrucción número 0, que determina el comienzo de la computación, se situará, por tanto, en cabeza. Una instrucción estándar de esta manera recibe el nombre de *tabla de Turing* o *tabla de máquina*. De las anteriores consideraciones resulta evidente que toda instrucción de cálculo que tenga que ver con un cómputo en una cinta de

cálculo puede ser reproducida en la forma de una tabla de Turing.

Para nosotros una *máquina de Turing* no es otra cosa que una tabla de Turing. En lo que concierne a las principales consideraciones que se llevarán a cabo en este libro, es materia irrelevante el saber de qué manera se ejecuta *realmente* un cálculo de acuerdo con una tabla de Turing (cf. § 1.2). Sin embargo, es recomendable, en el curso de nuestras especulaciones, imaginar una máquina de Turing como un aparato realmente puesto en movimiento [27].

A los símbolos que aparecen en la segunda columna

[27] Con las posibilidades técnicas de hoy, toda máquina de Turing puede ser realizada de diferentes modos como una máquina real, a través de la cual corre una cinta de cálculo (por ejemplo, en forma de cinta perforada o cinta magnética). En tal realización, un cierto estadio del cómputo corresponderá (si pensamos en una máquina mecánica) a un cierto ordenamiento de las partes de la máquina. Ello explica el término *configuración interna (internal configuration)* utilizado por Turing para denotar los estados. Turing identificó las configuraciones internas con los posibles *estados de ánimo (states of mind)* del calculador. Muchos autores le siguieron en este respecto. Pero parece que semejante identificación es problemática. Porque si suponemos que cada «estado de ánimo» de un calculador tiene unos constitutivos físicos característicos (por los que puede distinguirse de los demás estados), entonces la conclusión de que para toda persona aislada existe una *cota* superior M para el número de sus estados de ánimo parece ser evidente. (Incidentalmente, el propio Turing hizo similares consideraciones; por ejemplo, en lo que concierne al número de cuadrados directamente inteligibles de la cinta de cálculo.) Pero entonces una máquina de Turing que tuviese más de M estados no caería (o en todo caso no directamente) bajo estas consideraciones.

El hecho de que un calculador pueda trabajar de acuerdo con instrucciones (programas) arbitrariamente largas no apoya la tesis de que exista un número arbitrario de «estados de ánimo» posibles. Porque un calculador no precisa aprender el programa «de memoria». Usualmente la instrucción existe fuera del calculador; por ejemplo, en una cinta especial de cálculo, la cinta de programa, que es finita, pero puede ser arbitrariamente larga. El calculador ha de hallarse en disposición de poder «leer» tal cinta de programa, esto es, de ejecutar las instrucciones en ella contenidas. Si sólo se cuenta

de una tabla de Turing se les llama *los símbolos de esa máquina*.

Naturalmente, el modo en que opera una máquina de Turing depende de la expresión de cinta al comienzo del cómputo y del cuadrado primeramente escrutado.

Al empezar el próximo capítulo volveremos sobre la definición de máquina de Turing sin las consideraciones heurísticas que aquí tomamos en cuenta. Allí se darán también ejemplos de máquinas de Turing.

REFERENCIAS

Post, E. L.: *Finite Combinatory Processes - Formulation 1:* J. Symbolic Logic 1 (1936), 103-105.

Turing, A. M.: *On Computable Numbers, with an Application to the Entscheidungsproblem:* Proc. London math. Soc. (2) 42 (1937), 230-265. *On Computable Numbers, with an Application to the Entscheidungsproblem. A correction:* Proc. London math. Soc. (2) 43 (1937), 544-546.

Wang, H.: *A Variant to Turing's Theory of Computing Machines:* J. Assoc. Computing Mach. 4 (1957), 63-92.

Para críticas sobre la tesis de *Church*, compárese:

Kalmar, L.: *An Argument against the Plausibility of Church's Thesis*, en A. Heyting (ed.), *Constructivity in Mathematics*, North-Holland Publishing Company, Amsterdam 1959.

Peter, R.: *Rekursivitat und Konstruktivität*, ibíd., 226-233.

El problema de hasta dónde un *ser humano* puede, con respecto a sus habilidades intelectuales, ser considerado como una máquina sobrepasa la estructura de este libro. Entre otros, pueden consultarse los siguientes artículos sobre el tema:

---

con *una* cinta de cálculo, entonces el programa ha de ser escrito en esta cinta y el cómputo se llevará a cabo en la parte libre de la misma. La disposición de una persona para llevar a cabo en el antedicho sentido programas *arbitrarios* muestra que hay una correspondencia en este respecto entre esa persona y una «máquina universal de Turing» (cf. §30).

Turing, A. M.: *¿Puede pensar una máquina?* [1950]. Traducción de Manuel Garrido y Amador Antón, Valencia: Cuadernos Teorema 1974.
Kemeny, J. G.: *Man Viewed as a Machine:* Sci. Amer. 192 (1955) 58-67.

§ 4. Observaciones históricas

En el curso de los siglos, los matemáticos han obtenido grandes éxitos en el descubrimiento o invención de nuevos algoritmos. Es bien natural, por tanto, que se plantease la cuestión de saber si, al cabo de cierto esfuerzo, podríamos llegar a un extremo en la investigación en que, eventualmente, todo círculo de problemas pudiera ser atacado por un procedimiento general. La idea, más o menos clara, de que la totalidad de la matemática pueda ser dominada por métodos algorítmicos ha influido poderosamente, en diferentes épocas, el desarrollo de esta ciencia. Nos limitaremos a unas pocas notas sobre estas cuestiones.

1. Ars Magna. Los árabes desarrollaron, bajo la influencia de los indios, métodos algorítmicos para el tratamiento de problemas algebraicos. El nombre de algoritmo, por el que comúnmente se denomina hoy a un procedimiento general, tiene su origen en el nombre del matemático árabe AL-JWARIZMI (hacia 800). Los métodos matemáticos introducidos por los árabes inspiraron al español RAIMUNDO LULIO (hacia 1300), cuya *Ars Magna* se supone que pretendía ser un procedimiento general de base combinatoria para hallar todas las «verdades». Considerados desde un

punto de vista sensato, los procedimientos aducidos por Lulio no tienen mucho valor. Lo importante, sin embargo, es que aquí tuvo lugar la concepción de una idea ciertamente espléndida. Moritz Cantor juzga el «arte luliana» del siguiente modo: «... Una mezcla de lógica, estupidez cabalística y estupidez personal, en la que, no se sabe cómo, hay sembrados algunos granos de sano entendimiento» [28]. Lulio tuvo una gran influencia, especialmente en la posteridad matemática. Todavía doscientos años después, Cardano (1545) concibe los algoritmos algebraicos por él publicados con el signo del arte de Lulio, como lo muestra el título de su obra: *Artis magnae seu de regulis algebraicis líber unus*. El desarrollo del álgebra en el siglo XVI parecía ser una indicación de que todas las cuestiones, en todo caso en el dominio de esta disciplina, podían ser tratadas con la ayuda de procedimientos generales. No sucedió así, al principio, en geometría, la segunda ciencia matemática conocida por aquel tiempo. Merece la pena mencionar que DESCARTES (1598-1650) era de la opinión de que lo esencial de su geometría analítica residía en el hecho de que, con su ayuda, todos los problemas geométricos podrían ser traducidos a problemas algebraicos, con lo cual se harían susceptibles de ser tratados con los, algoritmos desarrollados en el álgebra [29]. Tras la exposición de su método, Descartes era de la opinión de que no quedaba problema alguno de interés para el

[28] M. Cantor, *Vorlesungen über Geschichte der Mathematik* II, Teubner, Leipzig 1892, 104.

[29] Lo decisivo para él *no* era, por tanto, la aplicación de coordenadas como tal, de cuya presencia hay ya constancia en la antigüedad, sino la aplicación de coordenadas con el fin de hacer posible el tratamiento algorítmico de la geometría.

matemático creador. «Aussy que je n'y remarque rien de si difficile, que ceux qui seront un peu versés en la Geometrie commune et en l'Algebre, et qui prendront garde a tout ce qui est en ce traité, ne puissent trouver» [30]. Pero, ciertamente, Descartes se equivocaba (según sabemos hoy) al suponer que todo problema algebraico pudiera ser resuelto por métodos generales.

LEIBNIZ (1646-1716) realizó ímprobos esfuerzos con objeto de desarrollar los algoritmos y con vistas a obtener una visión de la esencia de los mismos. Expresamente hizo notar que estaba influido por Lulio. Advirtió que el concepto luliano de *ars magna* comprendía varios conceptos que sería mejor considerar por separado. A este fin distinguió, no siempre claramente, un *ars inveniendi* de un *ars iudicandi*. Algunas de sus observaciones indican que por un *ars inveniendi* se ha de entender un procedimiento de generación, y por un *ars iudicandi* un procedimiento de decisión. Asimismo vio con claridad que ha de ser posible confiar a una máquina la ejecución de un procedimiento general. A este respecto vale la pena recordar que Leibniz fue el primero en construir una máquina de cálculo. Sin embargo, pese a sus denodados esfuerzos, no tuvo éxito en la realización de sus proyectos (parte de los cuales no fueron generalmente conocidos hasta 1901, al ser publicados por Couturat).

[30] «De suerte que no encuentro cuestión alguna que no pueda, pese a su grado de dificultad, ser resuelta por cualquiera, con tal de que esté un poco versado en la Geometría común y en el Algebra, y tenga en cuenta cuanto se dice en este tratado.» *Oeuvres* VI, ed. Adam-Tannery, París 1902, 374.

2. La lógica moderna. Si deseamos crear algoritmos que sean aplicables con el mayor grado de universalidad posible, parece aconsejable recurrir a la formación de procedimientos generales en el dominio de la *lógica*. De hecho, la lógica es aplicable en múltiples campos, y especialmente en teorías de fundamentación axiomática. La silogística, que tiene su origen en Aristóteles, fue un primer intento en esta dirección. Pero un intento que comprende, ciertamente, sólo una fracción muy modesta de las conclusiones lógicas realmente utilizadas por un matemático. La lógica moderna advino a la existencia en la pasada centuria. Su primer pionero fue G. BOOLE (1815-1869), cuyo libro *El análisis matemático de la lógica*. *Ensayo de un cálculo de razonamiento deductivo* apareció en 1847. Los métodos de cálculo de Boole se constituyeron, en muy gran medida, de acuerdo con los métodos usuales del álgebra, de suerte que a fines del pasado siglo solía hablarse de un *álgebra de la lógica*. En el ulterior desarrollo, empero, desapareció esta estrecha analogía con el álgebra. G. PEANO (1858-1932) eligió un simbolismo cuya construcción se conecta con la estructura de los lenguajes naturales. G. FREGE (1848-1925) puso especial empeño en dar una forma exacta a las reglas lógicas. Estos esfuerzos encontraron su primera conclusión en la monumental obra *Principia Mathematica* (3 vols. 1910-1913), de A. N. WHITEHEAD y B. RUSSELL. En esta obra se estableció que una gran parte de la matemática puede ser deducida con la ayuda de un cálculo lógico. La conclusión que viene a coronar el desarrollo iniciado por Boole fue el llamado *teorema de completud* de GÖDEL (1930). Este teorema dice que las reglas lógicas que se poseen son suficientes para extraer todas las inferencias posibles a partir de un sistema arbitrario de axiomas, supuesto que nos

limitemos al lenguaje del cálculo de predicados [31]. El resultado de Gödel puede ser expresado también en la terminología introducida en § 2.4 diciendo que si nos restringimos al lenguaje del cálculo de predicados, entonces el conjunto de inferencias a partir de un sistema axiomático es generable (enumerable).

3. Demostraciones de imposibilidad. No han tenido éxito los intentos de construir para la lógica de orden superior (cf. *EDC* § 26) un sistema de reglas tal que con ayuda del mismo pueda obtenerse toda inferencia a partir de un sistema arbitrario de axiomas. De otra parte, han sido vanos los ensayos de hallar un procedimiento con cuya ayuda pueda decidirse en un número finito de pasos para un sistema arbitrario, pero finito, de axiomas y una fórmula arbitraria del lenguaje del cálculo de predicados, si esa fórmula se sigue de dicho sistema *(problema de decisión para el cálculo de predicados)*. Cabe, por tanto, hacer la conjetura de que no existe un tal algoritmo. Una aserción que establezca que un grupo de problemas bien definido *no* puede ser dominado por algoritmo alguno es un enunciado que hace referencia, en general, a *todo* algoritmo imaginable. Para demostrar semejante enunciado no es suficiente contar con el concepto intuitivo de procedimiento general con que opera un matemática, sino contar con una definición exacta de ese concepto, que comprenda a todo algoritmo (en sentido intuitivo), aunque pueda ser, en determinadas circunstancias, harto más amplia. Pues, si pudiera probarse que un grupo dado de problemas no puede ser resuelto

[31] La restricción decisiva de este lenguaje consiste en el hecho de que las operaciones *para todo* y *existe* únicamente pueden ser aplicadas a variables individuales y no a variables de predicado. Cf. también *EDC* § 24 y § 26.

por un algoritmo en el sentido de la definición, entonces quedaría mostrado a fortiori que ningún algoritmo en sentido intuitivo producirá una solución [32].

GÖDEL fue el primero en demostrar (1931) que en el dominio de la lógica de segundo orden es imposible definir un sistema de algoritmos de una cierta especie que sirviera para obtener toda inferencia (la llamada *incompletud* de la lógica de segundo orden) [33].

Hoy se cree, generalmente, que *todo* sistema de algoritmos puede ser definido mediante funciones recursivas. De la definición de las funciones recursivas se sigue que el resultado de Gödel vale para sistemas arbitrarios de algoritmos, con lo cual cobra una significación más profunda. Ello se apreciará mejor si se tiene en cuenta lo que inmediatamente se dirá con relación a Church. El concepto de función μ-recursiva y el concepto con él emparentado de función recursiva han sido investigados a fondo por KLEENE.

La línea de pensamiento de la prueba de Gödel se conforma a la clásica antinomia del mentiroso. Se puede especificar una fórmula F dependiente del sistema de reglas arbitrariamente dado en el lenguaje de la lógica de segundo orden, tal que los contenidos de F equivalgan a la afirmación de que F no es deducible en el sistema de reglas. F no contiene variables libres. Por tanto, o bien F ha de ser válida, o bien ha de ser válida su negación $\neg F$ [34]. Si F es válida, entonces F no es deducible y, por tanto, el presunto sistema

[32] Esta observación vale la pena hacerla porque, de hecho, se expresa a veces la opinión de que las precisiones que hoy día suelen utilizarse como equivalente o sustituto exacto del concepto de algoritmo son demasiado amplias.

[33] Para el teorema de incompletud, cf. *EDC* § 26; para el concepto de función μ-recursiva, cf. § 19. Puede considerarse a Finsler (1926) como un precursor de Gödel.

[34] Sobre esta terminología, cf. *EDC* § 26.

de reglas es incompleto. Pero si $\neg F$ es válida y si el sistema de reglas es completo, entonces $\neg F$ ha de ser deducible. Ello implica que F no es deducible (si se supone la consistencia del sistema de reglas), lo cual significa, por la definición de F, que F es válida. Así pues, este caso no puede darse [35].

En 1936 CHURCH expresó su opinión de que los conceptos de función λ-definible *(EDC* §30*) y* de función recursiva *(EDC* §19*)* son ambos identificables con el concepto de función computable *(tesis de Church)*. Algo más tarde, el propio Church mostró que el problema de decisión para el cálculo de predicados es insoluble. El mismo resultado fue obtenido casi al mismo tiempo por A. M. TURING, quien introdujo en su prueba el concepto de máquina de Turing [36].

Durante los últimos años se ha probado también la indecidibilidad de diversas teorías susceptibles de ser expuestas mediante sistemas axiomáticos en el lenguaje del cálculo de predicados (o en lenguajes que son extensiones no esenciales del mismo). La primera prueba de esta índole fue dada por Church en 1936 para la aritmética de Peano. Entre los múltiples resultados obtenidos en este campo merece mencionarse la indecidibilidad de la teoría elemental de grupos, probada por TARSKI en 1946.

Un problema matemático muy conocido, cuya insolubilidad ha sido ya establecida, es el problema de las palabras para grupos. La tarea abordada por este

[35] La incompletud de la lógica de segundo orden es demostrable también de un modo diferente, reduciéndola a la indecidibilidad de la lógica de primer orden.

[36] Un concepto de una máquina de esa índole fue desarrollado por Post simultáneamente a Turing, aunque independientemente de él. Turing y Post identificaron el concepto de función computable con el de función computable por tal máquina.

problema consiste en hallar un algoritmo con cuya ayuda pueda decidirse en un número finito de pasos (para cualquier grupo que esté dado por un número finito de generadores y por un número finito de relaciones definitorias entre estos generadores) para palabras arbitrarias W_1 y W_2 (compuestas de dichos generadores) si W_1 y W_2 representan o no el mismo elemento del grupo. Las pruebas existentes de la insolubilidad del problema de las palabras para grupos son todavía muy complicadas.

Un problema cuya insolubilidad ha sido mostrada recientemente (1970) es el *décimo problema de Hilbert* (1900) (cf. *EDC* § 29.3). El problema consiste en hallar un método con cuya ayuda pueda decidirse para una ecuación diofántica arbitrariamente dada si es o no soluble.

REFERENCIAS

Finsler, P.: *Formale Beweise und die Entscheidbarkeit:* Math. Z. 25 (1926), 676-682.

Gödel, K.: La suficiencia de los axiomas del cálculo lógico de primer orden [La traducción literal del título original alemán sería: La completud de los axiomas del cálculo lógico funcional] [1930]. En: Kurt Gödel, *Obras completas*. Introducción y traducción de Jesús Mosterín, Madrid, Alianza Editorial, 1981, pp. 20-34.

—*Sobre proposiciones formalmente indecidibles de los Principia Mathematica y sistemas afines*, [1931]. Introducción de R. Braithwaite. Traducción de Manuel Garrido, Alfonso García Suárez y Luis Ml. Valdés, Valencia, Cuadernos Teorema, 1980. Una versión castellana del mismo artículo por Jesús Mosterín figura en las *Obras completas* de Gödel arriba citadas, pp. 45-89.

Church, A.: *An unsolvable Problem of Elementary Number Theory:* Amer. J. Math. 58 (1936), 345-363. (La tesis de Church en página 346.)

Kleene, S. C: *General Recursive Functions of Natural Numbers:* Math. Ann. 112 (1936), 727-742.

Sobre Turing y Post, cf. las referencias bibliográficas dadas en §3. Posteriormente se discutirán con más detalle algunos problemas especiales planteados en esta sección. Oportunamente se darán, en las secciones pertinentes, referencias bibliográficas al respecto.

En este capítulo se introducirá el concepto de máquina de Turing sin las consideraciones heurísticas del capítulo anterior. Además se definirán, mediante las máquinas de Turing, los conceptos constructivos más importantes que hemos encontrado en el Capítulo 1. Conviene cerciorarse de que las definiciones sugeridas de Turing-decidibilidad, Turing-computabilidad y Turing-enumerabilidad son sustituciones exactas de los conceptos intuitivos correspondientes. Estas definiciones se pueden considerar obvias, en realidad, si acordamos que las máquinas de Turing representan un sustituto legítimo exacto del concepto de algoritmo. Al final ofreceremos unos cuantos ejemplos sencillos de máquinas de Turing. Las máquinas $\mathbf{a}_j$, $\mathbf{r}$ y $\mathfrak{l}$ introducidas en § 6.5 son de importancia fundamental.

Los conceptos definidos en este capítulo pueden considerarse como conceptos de la matemática pura. Pero es muy sugestivo elegir una terminología técnico-física sugerida por la imagen mental de una máquina.

§ 5. Definición de las máquinas de Turing

1. Definiciones. Sea dado un alfabeto $\mathfrak{A} = \{a_1, \ldots, a_N\}$, $N \geqq 1$. Denótese por a_0 al símbolo (ideal) vacío. Supóngase de una vez por todas, que ninguno de los símbolos r, l, h, pertenece a $\mathfrak{A}$.

Una *máquina de Turing* **M** *sobre* $\mathfrak{A}$ viene dada por una matriz $M(N + 1) \times 4$ (una tabla con $M(N + 1)$ filas y 4 columnas) $(M \geqq 1)$ de la forma

$$
\begin{array}{llll}
c_1 & a_0 & b_1 & c'_1 \\
\cdots & \cdots & \cdots & \cdots \\
c_1 & a_N & b_{N+1} & c'_{N+1} \\
c_2 & a_0 & b_{N+2} & c'_{N+2} \\
\cdots & \cdots & \cdots & \cdots \\
c_2 & a_N & b_{2N+1} & c'_{2N+1} \\
\cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
c_M & a_0 & b_{(M-1)N+M} & c'_{(M-1)N+M} \\
\cdots & \cdots & \cdots & \cdots \\
c_M & a_N & b_{MN+M} & c'_{MN+M}
\end{array}
$$

donde $c_1,...,c_M$ son diferentes números naturales $\geqq 0$ [1] y $c'_j \in \{c_1,...,c_M\}$ para $j = 1,...,MN+M$. Por otra parte, cada b_j es un elemento de $\{a_0,...,a_N, r, l, h\}$.

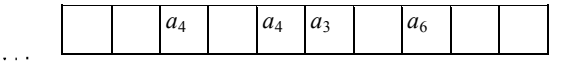
Podemos identificar a una máquina de Turing con su tabla.

Conviene advertir que para cada par c_j, a_i existe exactamente una línea **M** que comienza con $c_j a_i$. Los c_j se denominan *estados* y a c_1 se le llama el *estado inicial*. Denotaremos también al estado inicial de **M** por $c_{\mathbf{M}}$. $c_{\mathbf{M}}$ es el primer estado de que se hace mención en la tabla. Si una línea comienza por $c_j\ a_k\ h$, entonces c_j es llamado un *estado terminal*. $\mathfrak{A}$ es denominado el alfabeto de **M**.

---

[1] En § 3 hemos considerado que $c_1 = 0$, $c_2 = 1,...$, $c_M = M - 1$. Sobre la generalización aquí efectuada, cf. también el apartado 6.

2. La cinta de cálculo. En el presente libro consideramos frecuentemente *funciones B*, que están definidas para todos los *enteros x*, y cuyos valores *B(x)* son elementos de $\{a_0,...,a_N\}$. Entenderemos que los argumentos *x* son números de *cuadrados* de una *cinta de cálculo* ordenada de modo tal que el cuadrado con el número *n* (o más brevemente, el cuadrado *n)* se encuentra situado inmediatamente *a la izquierda* del cuadrado $n + 1$. Suponemos que el cuadrado *n* tiene el símbolo *B(n) escrito (impreso)* en él. Un cuadrado que tiene impreso el símbolo a_0 se denomina *vacío*. Así pues, podemos considerar que la función *B* es una *expresión de cinta* de la cinta de cálculo. Consideramos sólo *funciones para las que* $B(x) = a_0$ *para casi todos los x;* de este modo se supone que sólo un número finito de cuadrados presentan, de hecho, símbolos impresos en ellos, por ejemplo:



Cuadrado N.º —4 —3 —2 —1 0 1 2 3 4

3. Configuraciones[2] *y pasos de cálculo*. Por una configuración *K* de una máquina de Turing **M** entendemos cualquier triplo ordenado *(A, B, C)*, donde *A* es un cuadrado (dado por su número), *B* una expresión de cinta (por tanto, una función), y *C* un estado de **M**.

Una configuración *(A, B, C)* es llamada una *configuración inicial* si $C = c_{\mathbf{M}}$.

Toda configuración *K* = *(A, B, C)* corresponde

[2] Lo que aquí se denomina configuración corresponde a lo que en el artículo original de Turing se denominaba *configuración completa*.

inequívocamente a la línea de la tabla **M** que comienza con *C B(A)*, y a la que llamaremos *la línea de la configuración K*. *K* es denominada una *configuración terminal* si la línea de la configuración *K* comienza con *C B(A) h*.

Sea *K = (A, B, C)* una configuración que *no es una configuración terminal*. Sea *C B(A) b c'* la línea de la configuración *K* (por tanto, *b ≠ h)*. Asociaremos (inequívocamente) con *K* una *configuración consecutiva F(K) = (A', B', C')*, en la que se tiene [3]:

$$A' = \begin{cases} A, & \text{si } b \neq r \text{ y } b \neq l \\ A + 1, & \text{si } b = r \\ A - 1, & \text{si } b = l \end{cases}$$

$$B'(x) = \begin{cases} B(x), & \text{si } x \neq A \\ B(x), & \text{si } x = A \text{ y } (b = r \text{ o } b = l) \\ b, & \text{si } x = A \text{ y } b \neq r \text{ y } b \neq l \end{cases}$$

$$C' = c'.$$

Si iniciamos una computación con una cierta configuración inicial, entonces llamamos a esta configuración *la configuración 0-ésima*. Además, si la n-ésima configuración en dicha computación tiene una configuración consecutiva, entonces llamaremos a esta última *la (n + 1)-ésima configuración*.

Introduciremos algunos términos más relativos al concepto de paso de cálculo o, brevemente, paso. Imaginamos que el *n*-ésimo paso *(n ≥ 1)* conduce de la *(n - 1)-ésima* configuración a la *n-ésima*. En este sentido llamaremos también a las configuraciones *n-ésima*

[3] Debe comprobarse que esta definición está de acuerdo con el contenido de § 3.4.

y *(n - 1)-ésima*, *las configuraciones posterior y anterior*, *respectivamente*, *al n-ésimo paso*. A la configuración 0-ésima la llamaremos también *la configuración posterior al 0-ésimo paso*. Finalmente, diremos que una máquina *se para tras el n-ésimo paso* si la *n-ésima* configuración es terminal.

4. Definiciones. Introducimos ahora unos cuantos términos, dando sus definiciones exactas. Estos términos provienen de la concepción de las máquinas de Turing como máquinas reales. Expresaremos el hecho de que en una cierta conexión elijamos una máquina **M,** un número A, y una función B, y de este modo cierta configuración inicial $K_0 = (A, B, c_{\mathbf{M}})$, diciendo que *colocamos* **M** *en la expresión de la cinta B sobre el cuadrado A*. Si K_0 no es una configuración terminal, entonces existe para K_0 una única configuración consecutiva $K_1 = F(K_0)$. Decimos en este caso que **M** *cambia en el primer paso de* K_0 *a* K_1. Si tampoco es K_1 una configuración terminal, entonces existe un único $K_2 = F(K_1)$, y decimos que **M** cambia en el segundo paso de K_1 a K_2, etc. Ahora bien, podemos imaginar dos casos. O ninguna de las configuraciones K_0, K_1, K_2,... es una configuración terminal; entonces K_n está definido para todo n y **M** no se detendrá jamás. O existe un primer n tal que K_n es una configuración terminal. Entonces $K_n + 1$ no está ya definido y diremos que **M** *cesa de operar (para)* después del *n-ésimo* paso (puede que $n=0$), y además, si $K_n = (A_n , B_n , C_n)$, decimos *que* **M** *cesa de operar en la expresión de cinta* B_n *y sobre el cuadrado* A_n. La última expresión de cinta B_n y el último cuadrado A_n determinan, en particular, la letra $B_n(A_n)$ que se encuentra impresa, al final, en A_n . Por tanto, resulta conveniente decir que situado **M** sobre B en *A se detiene en la letra*

a_j. Puede suceder que $a_j \neq a_0$. Entonces existe una parte máxima y continua de cinta que contiene A_n y que es tal que sólo letras propias se encuentran impresas en sus cuadrados, las cuales constituyen, en el orden en que se encuentran impresas en la cinta, una palabra W. Diremos en este caso que **M** *cesa de operar en la palabra* W. Si K_j y K_{j+1} están definidos y si la línea de la configuración K_j es C_jA_jbc, entonces diremos que *en el* $(j + 1)$*-ésimo paso la máquina se mueve a la derecha o a la izquierda, en caso de que* $b = r$ *ó* $b = l$ *respectivamente, o que el símbolo* a *esté impreso en* A_j*, en caso de que* $b = a$.

5. Desplazamientos. Si m es un número entero y $\tilde{B}(x) = B(x—m)$ para todo x, entonces podemos decir que $\tilde{B}$ resulta de B por desplazamiento de m lugares. Si se coloca una máquina **M** sobre B en A, entonces se obtienen las configuraciones K_0, K_1, K_2,..., donde $K_n = (A_n , B_n , C_n)$. Si se coloca a **M** sobre $\tilde{B}$ en $\tilde{A}=A + m$, entonces se obtienen las configuraciones $\tilde{K}_0 , \tilde{K}_1 , \tilde{K}_2$,..., donde $\tilde{K}_n = (\tilde{A}_n , \tilde{B}_n , \tilde{C}_n)$. Es fácil cerciorarse de que K_n y $\tilde{K}_n$ están definidos para los mismos argumentos n, y que para cada uno de tales n vale [4]:

$$\tilde{A}_n = A_n + m,\ \tilde{B}_n(x) = B_n(x—m),\ \tilde{C}_n = C_n .$$

Así, todo cuadrado escrutado $\tilde{A}_n$ y toda expresión de cinta $\tilde{B}_n$ surgen del correspondiente cuadrado escrutado A_n y de la expresión de cinta B_n respectivamente, gracias a un desplazamiento de m lugares. Si además K_n es una configuración terminal, también lo es $\tilde{K}_n$, y viceversa.

[4] Las siguientes ecuaciones también pueden ser interpretadas diciendo que los desplazamientos de la cinta no son esenciales.

Los hechos que hemos acabado de discutir permiten en ciertos casos la introducción de una terminología que es abstraída de la numeración de los cuadrados de la cinta. Tómese W como una palabra que es una secuencia no vacía de letras del alfabeto $\mathfrak{A} = \{a_1,...,a_N\}$. El enunciado *colóquese* **M** *detrás de* W significará que elegimos una expresión de cinta B que contiene en los cuadrados sucesivos las letras de la palabra W *en* el orden en que aparecen en W y cuyos restantes cuadrados son vacíos, y que tomamos como cuadrado inicial A el primer cuadrado (vacío) a la derecha de W. B y A sólo están determinados aquí hasta un desplazamiento de m lugares. De acuerdo con las anteriores observaciones, esta ambigüedad no juega ningún papel en la determinación de si **M** cesa o no de operar. Si **M** cesa de operar, lo hará en todo caso sobre la misma letra. Es conveniente, por tanto, decir que **M**, situado detrás de una palabra W (o situado detrás de la palabra vacía), cesa de operar sobre la letra a_j, o detrás de una palabra W'.

6. Equivalencia de las máquinas de Turing. Decimos que una máquina de Turing $\mathbf{M}_1$ es *equivalente* a una máquina de Turing $\mathbf{M}_2$ si existe una aplicación biunívoca φ de los estados de $\mathbf{M}_1$ a los estados de $\mathbf{M}_2$ tal que:

(1) cada línea $q_i\ a_j\ q_k$ de $\mathbf{M}_1$ es transferida a una línea $\varphi(q_i)\ a_j\ \varphi(q_k)$ de $\mathbf{M}_2$, y

(2) $\varphi(q_{\mathbf{M}_1}) = q_{\mathbf{M}_2}$.

Dada una máquina $\mathbf{M}_1$, podemos hallar siempre una máquina $\mathbf{M}_2$ equivalente donde los estados estén numerados sucesivamente 0, 1, 2..., y donde 0 sea el estado inicial. (En la última sección hemos enumerado los estados de acuerdo con este modo normal.) Es,

empero conveniente, especialmente con vistas a las combinaciones de máquinas que se discutirán en próximas secciones, tener a nuestra disposición estados que no estén dados de este modo normal o estándar. Para el uso que pretendemos hacer de las máquinas, de Turing, únicamente son esenciales las secuencias A_n y $B_n(x)$. Supóngase que la máquina de Turing $\mathbf{M}_1$ (colocada sobre B en A) da lugar a las configuraciones $(A_n, B_n(x), C_n)$. Supóngase además que $\mathbf{M}_2$ es equivalente a $\mathbf{M}_1$ en virtud de la aplicación φ de los estados de $\mathbf{M}_1$ a los estados de $\mathbf{M}_2$. Se advierte al punto que la secuencia de las configuraciones de $\mathbf{M}_2$ (si se la coloca sobre la misma expresión de cinta B y en el mismo cuadrado A) es $(A_n, B_n(x), \varphi(C_n))$. *Por tanto, máquinas de Turing equivalentes darán lugar a la misma secuencia* $(A_n, B_n(x))$ *(si se las coloca en la misma expresión de cinta B sobre el mismo cuadrado A)*. De este modo, pueden ser recíprocamente reemplazadas en todas las definiciones que utilicemos de tales secuencias únicamente. Ello se aplica en especial a las definiciones de la siguiente sección.

7. *Equivalencia, en un sentido más amplio, de las máquinas de Turing*. En § 1.2 hemos señalado el hecho de que la elección de signos particulares en los algoritmos es en principio irrelevante. Ello conduce a un concepto más amplio de la equivalencia de las máquinas de Turing. Decimos que una máquina de Turing $\mathbf{M}_1$ sobre un alfabeto $\mathfrak{A}_1$ es *equivalente en sentido amplio* a una máquina de Turing $\mathbf{M}_2$ sobre un alfabeto $\mathfrak{A}_2$ si existen aplicaciones biunívocas φ, ψ tales que:

1. ψ suministra una aplicación de $\mathfrak{A}_1$ en $\mathfrak{A}_2$ y además tenemos que $\psi(l) = l$, $\psi(r) = r$, $\psi(h) = h$.

2. φ es una aplicación de los estados de $\mathbf{M}_1$ en los estados de $\mathbf{M}_2$.
3. Toda línea
   $c\ a\ b\ c'$ de $\mathbf{M}_1$ es transferida a una línea
   $\varphi(c)\ \psi(a)\ \psi(b)\ \varphi(c')$ de $\mathbf{M}_2$.
4. $\varphi\ (c_{\mathbf{M}_1}) = c_{\mathbf{M}_2}$.

Las observaciones del apartado 6 sobre las máquinas de Turing equivalentes se aplican también, mutatis mutandis, a las máquinas de Turing equivalentes en sentido amplio.

REFERENCIAS

Cf. las referencias de §3, y el *Appendix* del artículo de Post., E. L.: *«Recursive Unsolvability of a Problem of Thue»*, *Journal of Symbolic Logic*, 12, (1947) 1-11.

§ 6. Definición precisa de los conceptos constructivos por medio de las máquinas de Turing

En los siguientes apartados ofreceremos definiciones exactas de los conceptos de computabilidad, decidibilidad y enumerabilidad. A estos conceptos exactos les daremos los nombres de Turing-computabilidad, Turing-decidibilidad y Turing-enumerabilidad. El lector deberá percatarse de que estos conceptos exactos son sustituciones naturales de los conceptos intuitivos descritos en § 2 (suponiendo que las máquinas de Turing correspondan a algoritmos).

En las consideraciones que siguen supondremos establecido de un modo fijo el alfabeto $\mathfrak{A}_0 = \{a_1, ..., a_N\}$. Consideraremos únicamente *palabras no vacías* sobre este alfabeto [5]. Sobre la terminología usada en estos apartados cf. también § 5.4.

[5] En § 1.4 se mostró que ello no es una limitación esencial.

1. Turing-computabilidad. En primer lugar damos una definición para el caso especial de funciones de un solo argumento *(unarias)*.

Sea *f* una función unaria que está definida para todas las palabras y cuyos valores son palabras. Decimos que *f* es *Turing-computable* si existe una máquina de Turing **M** sobre un alfabeto $\mathfrak{A}$ con $\mathfrak{A}_0 < \mathfrak{A}$ tal que si imprimimos una palabra arbitraria *W* (sobre $\mathfrak{A}_0$) en la cinta de cálculo, que salvo esa palabra estará vacía, y si colocamos a **M** sobre un cuadrado arbitrario de la cinta [6], entonces **M** se detendrá después de un número finito de pasos tras una palabra que represente el valor *f(W)* de la función [7].

Ahora, sea $f(W_1, ..., W_k)$ una función *k*-aria cuyos

[6] Pudiera suponerse que sería pedir demasiado si exigiéramos, para que una función sea computable, que existiese una máquina que realizase la computación *independientemente* del cuadrado de la cinta sobre el que está colocada. Esperaríamos quizá que la clase de las funciones computables definidas de este modo fuese menor de lo que sería si, por ejemplo, solamente exigiéramos que la máquina realizase la computación si estuviese colocada sobre o detrás del último símbolo del argumento. Sin embargo, se mostrará en un teorema de § 9.1 que no es éste el caso. En general, resultará naturalmente más fácil dar razón de una máquina que realiza la computación del valor de la función si (al comienzo de la computación) es colocada sobre un cuadrado *particular*. Pero la elección de un cuadrado tal deberá ser arbitraria hasta cierto punto. La definición dada en el texto nos libra de la arbitrariedad.

[7] Mientras al comienzo de la computación permitíamos un cuadrado arbitrario, no parece plausible (si elegimos representar el valor como hemos hecho) permitir eso mismo para el final de la computación. Pues, en general, habrá todavía otras palabras en la cinta junto al valor de la función (por ejemplo, argumentos o cálculos secundarios). La máquina tiene que señalar el valor de la función (mediante su posición final). La convención que hemos ofrecido aquí a efectos de este propósito resulta muy conveniente en muchos casos. Pero se podría adoptar otra convención sin dificultad y requerir, por ejemplo, que **M** cese de operar sobre un símbolo arbitrario del valor de la función, o algo similar. (La equivalencia de tales requisitos se puede establecer fácilmente: § 7, Ejercicio 2.)

argumentos son palabras arbitrarias sobre $\mathfrak{A}_0$ y cuyos valores son palabras sobre $\mathfrak{A}_0$. Entonces decimos que f es Turing-computable si existe una máquina de Turing **M** sobre un alfabeto $\mathfrak{A}$ con $\mathfrak{A}_0 < \mathfrak{A}$ tal que si se imprimen en la cinta de cálculo, que en lo demás debe estar vacía, argumentos arbitrarios W_1 ,..., W_k dejando en esta secuencia un cuadrado vacío cada vez entre dos palabras (es decir, la *palabra ideal* $W_1\, a_0\, W_2\, a_0 \ldots a_0\, W_k$), y se coloca a **M** sobre un cuadrado arbitrario de la cinta de cálculo, entonces **M** cesará de operar después de un número finito de pasos y lo hará después de la palabra $f(W_1,\ldots, W_k)$.

Es aconsejable por razones sistemáticas introducir también funciones de cero argumentos. El valor de una función de *dos* argumentos puede determinarse suministrando los dos argumentos. El valor de una función de *un* argumento puede determinarse suministrando dicho argumento. Similarmente, el valor de una función de *cero* argumentos puede determinarse no suministrando ningún argumento. Así pues, una función de cero argumentos posee *solamente un valor*. Este puede ser una palabra arbitraria W. Designaremos la función de cero argumentos que posee el valor W mediante C_0^W, o también de forma abreviada mediante W si ello no da lugar a equívocos.

Diremos, ampliando de forma natural la definición anterior de Turing-computabilidad de una función que una función de cero argumentos es Turing-computable si existe una máquina de Turing que, si está colocada en una cinta *vacía*, cesará de operar detrás del valor de la función después de un número finito de pasos. Es inmediatamente obvio aquí que *toda* función de cero argumentos es Turing-computable de forma trivial. Con objeto de computar C_0^W, solamente necesitamos tomar una máquina de

Turing que imprima la palabra W en la cinta vacía y cese de operar detrás de dicha palabra (Ejercicio 2).

2. Turing-enumerabilidad. Utilizamos aquí una representación estándar de los números naturales. Para ello convenimos en representar al número n por n + 1 trazos (cf. § 1.4); así, por ejemplo, representamos al número tres por ||||. Al convenir un tal procedimiento identificaremos al trazo | con a_1. Diremos que un conjunto M de palabras es *Turing-enumerable*, si M es vacío o si M coincide con el dominio de valores de una función Turing-computable cuyo dominio de argumentos es el conjunto de los números naturales. Dado que, de acuerdo con § 2.4, generabilidad es equivalente a enumerabilidad, podemos ahorrarnos el trabajo de dar una definición especial de Turing-generabilidad.

3. Turing-decidibilidad. Sea **M** un conjunto de palabras sobre $\mathfrak{A}_0$. Decimos que **M** es *Turing-decidible* si existe una máquina de Turing **M** sobre un alfabeto $\mathfrak{A}$ (con $\mathfrak{A}_0 < \mathfrak{A}$) y dos símbolos diferentes (reales o ideales) a_i , a_j de $\mathfrak{A}$ tales que si imprimimos una palabra arbitraria W sobre $\mathfrak{A}_0$ en la cinta de cálculo, que salvo esa palabra deberá estar vacía, y si colocamos a **M** sobre un cuadrado arbitrario de esa cinta, entonces **M** se parará después de un número finito de pasos sobre a_i o a_j, según que $W \in M$ *ó* $W \notin M$, respectivamente.

Sean M_1 *y* M_2 conjuntos de palabras sobre $\mathfrak{A}_0$. Sea que $M_2 < M_1$. Entonces decimos que M_2 es *Turing-decidible relativamente a* M_1 si existe una máquina de Turing **M** sobre un alfabeto $\mathfrak{A}$ (con $\mathfrak{A}_0 < \mathfrak{A}$) y dos símbolos diferentes (reales o ideales) a_i , a_j de $\mathfrak{A}$ tales que si imprimimos una palabra arbitraria $W \in M_1$ en

la cinta de cálculo, que salvo esa palabra debe estar vacía, y si colocamos a **M** sobre un cuadrado arbitrario de la cinta de cálculo, entonces **M** se parará después de un número finito de pasos sobre a_i o a_j según que $W \in M_2$ o $W \notin M_2$ respectivamente.

Decimos que una *propiedad* (un *predicado)* de palabras es *Turing-decidible* si el conjunto de palabras con esa propiedad es Turing-decidible.

Definimos la *Turing-decidibilidad de relaciones n-arias*, esto es, de propiedades *n*-arias ($n \geqq 2$), de un modo similar (imprimiendo el *n*-tuplo de palabras en la cinta de cálculo, como se describe en el apartado 1).

Podemos mostrar tanto en el caso de la decidibilidad absoluta como en el de la decidibilidad relativa, que la elección de las letras a_i , a_j es inesencial. Nos limitamos aquí al:

Teorema. Sea M un conjunto de palabras sobre $\mathfrak{A}_0$. Sea **M** una máquina de Turing sobre un alfabeto $\mathfrak{A} = \{a_1, \ldots, a_L\}$ (con $\mathfrak{A}_0 < \mathfrak{A}$) y a_i , a_j símbolos diferentes (reales o ideales) de $\mathfrak{A}$. Sea **M** tal que si está colocada en la cinta de cálculo que contiene solamente una palabra arbitraria W sobre $\mathfrak{A}_0$ impresa en ella, entonces cesará de operar después de un número finito de pasos sobre a_i o a_j según que $W \in M$ o $W \notin M$ respectivamente (desde ahora en adelante expresaremos esto brevemente diciendo que **M** decide M con la ayuda de a_i , a_j *)*. Además, sean a_k , a_l dos símbolos arbitrarios diferentes (reales o ideales) de $\mathfrak{A}$. Entonces podemos dar razón (usando **M**) de una máquina de Turing **M′** sobre $\mathfrak{A}$ que decide M con la ayuda de a_k , a_l.

Demostración. La tabla para una tal máquina **M'** se obtiene a partir de la tabla de **M** como sigue. Sea $\bar{c}$ un nuevo estado que no aparece en la tabla de **M**. Se

cambia toda línea de **M** que sea de la forma $c\ a_i\ h\ c'$ en $c\ a_i\ a_k\ \bar{c}$, toda línea de la forma $c\ a_j\ h\ c'$ en $c\ a_j\ a_l\ \bar{c}$, y finalmente se añade una nueva línea $L + 1$ de la forma $\bar{c}\ a_p\ h\ \bar{c}\ (p = 0\ ,...,\ L)$. Se ve inmediatamente que en cualquier caso en el que **M** cesa de operar sobre a_i o a_j la máquina **M'** ejecuta un nuevo paso y entonces se para sobre a_k o a_l respectivamente [8].

4. Observación. En todas las definiciones de esta sección hemos permitido que **M** sea una máquina de Turing sobre un alfabeto $\mathfrak{A}$ que puede contener más letras que el alfabeto $\mathfrak{A}_0$. Los símbolos de $\mathfrak{A}$ que no pertenecen a $\mathfrak{A}_0$ se utilizan como «letras auxiliares». La cuestión está en si podemos hacerlo *sin* letras auxiliares en cada caso, es decir, si podemos exigir que **M** sea una máquina sobre $\mathfrak{A}_0$. Aparece claro intuitivamente que ello sí que es posible. Sean, por ejemplo, $\mathfrak{A}_0 = \{a_1\ ,...,\ a_N\}$, $\mathfrak{A} = \{a_1\ ,...,\ a_L\}$, $L > N$. Supongamos que tenemos un problema que ha de realizarse solamente con palabras sobre $\mathfrak{A}_0$, y un algoritmo que soluciona este problema usando los símbolos de $\mathfrak{A}$. Podemos «traducir» todo *símbolo* a_j de $\mathfrak{A}$ en una secuencia de j símbolos a_1 por ejemplo, a_3 en $a_1a_1a_1$. Podemos traducir una *expresión finita de cinta*, por ejemplo, $a_4a_0a_2a_3$, usando la traducción de los símbolos individuales, en $a_1a_1a_1a_1a_0a_0a_0a_1a_1a_0a_1a_1a_1$.

Primero, traducimos el problema de esa manera, y luego le aplicamos un tipo de «algoritmo traducido» que dará finalmente la solución traducida que tendre-

[8] Empleando la notación que introduciremos en la siguiente sección puede escribirse:

$$M' = M \begin{matrix} i \\ j \end{matrix} \begin{matrix} \rightarrow a_k \\ \rightarrow a_l \end{matrix}$$

mos que volver a traducir de nuevo. Es plausible que de esta forma obtengamos un algoritmo que usa sólo los símbolos de $\mathfrak{A}_0$. No obstante, conservaremos la definición general. Pero mostraremos en un caso importante especial que el alfabeto $\mathfrak{A}_0$ es suficiente para nuestro propósito (cf. § 15).

5. *Las máquinas elementales* **r**, **l** , $\mathbf{a}_j$. Introduciremos ahora unas cuantas máquinas de Turing particularmente simples, a partir de las cuales se construirán en la próxima sección otras más complicadas. Para todo alfabeto hay máquinas de esta índole, aun cuando, a su vez, *dichas máquinas dependen del alfabeto elegido*. En el presente apartado denotaremos por $\{a_1, ..., a_N\}$ el alfabeto que sirva de base.

Definición 1. La *máquina derecha* **r** está dada por la siguiente

Tabla de la máquina derecha **r**

0	a_0	r	1
. . . . .			
0	a_N	r	1
1	a_0	h	1
. . . . .			
1	a_N	h	1

La máquina derecha, una vez se la coloque sobre una expresión arbitraria de cinta y en un cuadrado arbitrario A, cesará de operar después de haber dado un paso para colocarse en el cuadrado situado inmediatamente a la derecha de A. La expresión de cinta original no será alterada. Lo que hace la máquina es moverse un cuadrado a la derecha.

Definición 2. La *máquina izquierda* $\mathfrak{l}$ está dada por la siguiente

Tabla de la máquina izquierda $\mathfrak{l}$

0	a_0	l	1
. . . . .			
0	a_N	l	1
1	a_0	h	1
. . . . .			
1	a_N	h	1

La máquina izquierda, una vez se la coloque sobre una expresión arbitraria de cinta y en un cuadrado arbitrario A, cesará de operar después de haber dado un paso para colocarse en el cuadrado situado inmediatamente a la izquierda de A. La expresión de cinta original no será alterada. Lo que hace la máquina es moverse un cuadrado a la izquierda.

Definición 3. La *máquina* $\mathfrak{a}_j$ *(j = 0 ,..., N)* está dada por la siguiente

Tabla de la máquina $\mathfrak{a}_j$

0	a_0	a_j	0
0	a_1	a_j	0
. . . . .			
0	a_j	h	0
. . . . .			
0	a_N	a_j	0

En todas las líneas aparece el símbolo a_j en tercer lugar, excepto en la línea $0a_jh0$. Sea que la máquina $\mathfrak{a}_j$ es colocada sobre la expresión de cinta B en el cuadrado A. Si $B(A) = a_j$, entonces la máquina $\mathfrak{a}_j$ cesará de operar después de haber dado cero pasos en el cuadra-

do A y sin alterar la expresión de cinta. Si $B(A) \neq a_j$, la máquina a_j, cesará de operar, después de haber ejecutado un paso, en el cuadrado A. La expresión de cinta B_1 diferirá de B, por cuanto que la letra a_j se habrá impreso en el cuadrado A. Así tenemos que, *en cualquiera de ambos casos*, $\mathbf{a}_j$ cesa de operar en A. La expresión final de cinta coincide en todo con la original, con la posible excepción del símbolo impreso en A, que es a_j al término de la computación. Así pues, la máquina $\mathbf{a}_j$, imprime el símbolo a_j en el cuadrado escrutado A.

Muchas veces representaremos a_0 por * y a_1 por |. Por razones tipográficas usaremos esos mismos símbolos * y |, respectivamente, para las correspondientes máquinas de Turing.

6. Ejemplos. Todas las máquinas que se van a aducir en este apartado son máquinas sobre un alfabeto | de un solo elemento. Este alfabeto es suficiente para la representación de los *números naturales* (cf. § 1.4).

Tabla 1

0	*	|	0
0	|	*r*	1
1	*	*h*	1
1	|	*h*	1

Tabla 2

0	*	|	2
0	|	*l*	1
1	*	*h*	1
1	|	*l*	0
2	*	*h*	2
2	|	*h*	2

Tabla 3

0	*	|	0
0	|	*r*	1
1	*	*r*	0
1	|	*h*	1

Examinaremos cómo operan estas máquinas en casos especiales.

(1) Si la máquina **M** dada en la *Tabla 1* es colocada inmediatamente detrás del último trazo de una secuencia de trazos impresa en una cinta que está en lo

que resta vacía, entonces **M** añadirá un nuevo trazo a la secuencia y se parará a la derecha de ese trazo. (Las líneas 1, 2, 3 son aquí decisivas; la línea 4 es inesencial y podría haber sido establecida de modo diferente.) Así pues, **M** computa la función sucesor, *supuesto* que se la coloque de la manera descrita en la cinta que lleva el argumento. En caso contrario no efectuaría tal computación. La exposición de la máquina **M** no nos permitiría probar que la función sucesor es Turing-computable en el sentido indicado en el apartado 1. Para ello sería preciso aducir la descripción de una máquina que ejecute dicha computación independientemente del cuadrado en que se la sitúa al comienzo de la misma (véase § 9).

(2) Si se coloca la máquina **M** dada en la *Tabla 2* sobre el último trazo de una palabra *W*, entonces **M** se parará después de un número finito de pasos sobre el símbolo * o |, según que *W* represente, respectivamente, un número natural par o impar. (Similarmente al ejemplo (1), tampoco aquí queda mostrado aún que el conjunto de números pares es Turing-decidible en el sentido del apartado 3.) Ello puede comprobarse siguiendo el proceso de computación. Al principio es decisiva la línea 2. A continuación habrá que distinguir dos casos, según que *W* conste o no de un solo trazo. En el primer caso, la línea 3 es decisiva para el paso subsiguiente. La máquina se parará sobre el símbolo *, mostrando así que 0 (representado por *un* trazo) es un número par. En el segundo caso, la línea 4 es decisiva para el subsiguiente paso, a continuación de lo cual serán decisivas las líneas 1 ó 2, según que $W = \text{||}$ ó $W \neq \text{||}$. En el primer caso, la línea 6 decidirá el subsiguiente paso. La máquina se parará sobre |, mostrando que 1 es un número impar. En el segundo

caso se habrá alcanzado una situación que corresponde a la inicial, y se volverá a repetir el proceso módulo 2. La máquina se parará, cuando se agote la palabra W, o bien sobre * o bien sobre | según que W represente, respectivamente, un número par o un número impar. (La línea 5 es inesencial.)

(3) Se requiere una máquina de Turing que, colocada en un cuadrado arbitrario de la cinta, originalmente vacía, imprima en ella (empezando por el cuadrado escrutado inicialmente) la secuencia infinita |*|*|*... Esta tarea es llevada a cabo por la máquina dada en la *Tabla 3*.

(4) Ejemplo de una máquina de Turing que computa la *diferencia* $x \dot{-} y$, siendo $x \dot{-} y = x - y$, si $x \geqq y$, y $x \dot{-} y = 0$, si $x < y$. Sea que W_1 * W_2 figuren impresas en una cinta de cálculo en lo demás vacía, donde W_1 y W_2 representan números en el sentido de la Sección 2. Se pide una máquina **M** que ejecute lo siguiente: Si se coloca a **M** en el cuadrado vacío situado inmediatamente después de W_2, entonces **M** se parará después de un número finito de pasos sobre un cuadrado vacío situado justamente detrás de una secuencia W de trazos, que a su vez tiene delante un cuadrado vacío y que representa (de nuevo en el sentido de la Sección 2) el valor de la función $x \dot{-} y$.

La producción de la tabla de **M** se efectuará a través de tres estadios:

(a) Describiremos primero en términos muy generales el procedimiento mediante el cual se lleva a cabo el cálculo.

(b) Fraccionaremos el procedimiento en partes distintas (de una cierta extensión) que deberán ser ejecutadas en un orden fijo.

(c) Finalmente, volviendo a fraccionar las partes del procedimiento obtendremos la construcción de una tabla para **M**.

Aclaraciones a (a). Borrando alternativamente un trazo al extremo derecho de W_2 y al extremo izquierdo de W_1 , resultará que o bien W_2 o bien W_1 queda totalmente borrada. Si W_2 es borrada primero, entonces $y \leqq x$, y por tanto $x \dot{-} y = x—y$. El resto de W_1 (sin borrar ya otro trazo en ella) representará el valor requerido de la función. Por otra parte, si W_1 es borrada primero, entonces $x<y$ y, por tanto, $x \dot{-} y=0$. Dicho valor 0 estará impreso ahora en la forma de un solo trazo.

En lo sucesivo hablaremos de la palabra derecha y la palabra izquierda significando por tales W_2 y W_1 respectivamente, o las nuevas palabras que resulten de acortar las anteriores.

Aclaraciones a (b). Consideraremos las siguientes partes del procedimiento (así como también el orden secuencial de ellas, que se inicia con (a)).

(α) Borrar un trazo al extremo derecho de la palabra derecha. Saltar a *(β)*.

(*β*) Comprobar si la palabra derecha ha sido totalmente borrada. Si no, saltar a (γ); si sí, saltar a *(η)*.

(γ) Ir a la palabra izquierda. Saltar a (δ).

(δ) Borrar un trazo al extremo izquierdo de la palabra izquierda. Saltar a (*ϵ*).

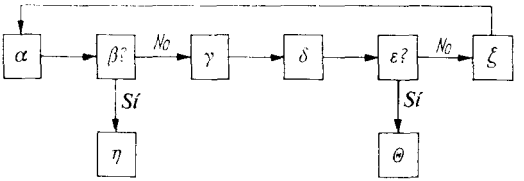
(*ϵ*) Comprobar si la palabra izquierda ha sido totalmente borrada. Si no, saltar a (*ζ*); si sí, saltar a (*θ*).

(*ζ*) Ir a la palabra derecha. Saltar de nuevo a *(α)*.

(*η*) Ir al cuadrado situado inmediatamente detrás de la palabra izquierda.

(*θ*) Ir a un cuadrado vacío situado inmediatamente detrás de un trazo aislado.

El conjunto de instrucciones que se acaban de exponer puede ser fácilmente llevado a cabo si se lo representa mediante lo que se llama un *diagrama de flujo:*



Aclaraciones a (c). Fácilmente puede verse que la siguiente tabla de máquina realiza el programa (b). Para cada una de sus líneas se indicará a qué parte del procedimiento corresponde. Se han dejado fuera de consideración algunas líneas que serían necesarias para una descripción completa de la tabla de esta máquina; p. ej., la línea que empezaría con 1 *. Son líneas que no juegan papel alguno en la computación y que pueden ser, por tanto, arbitrariamente dadas. Para dejar bien patente que la tabla corresponde por entero al diagrama de flujo, no se ha procurado obtener la más corta posible. La tabla aducida podría acortarse , p. ej., reemplazando las dos líneas 3 * * 10 y 10 * *h* 10, por una nueva línea 3 * *h* 3.

7. *Procedimientos de computación no periódicos*. En primer lugar, pudiera pensarse que una máquina de Turing que, tras ser colocada en la cinta vacía, nunca cesa de operar, lo único que hace es reiterar procesos periódicos. Aduciremos, al respecto las siguientes consideraciones. Si seguimos el proceso de computación de la máquina, advertiremos que ciertas líneas de la tabla de esa máquina serán decisivas en él. Una tabla de máquina tiene solamente un número fi-

$$
\begin{array}{ll}
(\alpha) & \left\{\begin{array}{llll} 0 & * & l & 1 \\ 0 & | & r & 0 \\ 1 & | & * & 2 \end{array}\right. \\
(\beta) & \left\{\begin{array}{llll} 2 & * & l & 3 \\ 3 & * & * & 10 \\ 3 & | & l & 4 \end{array}\right. \\
(\gamma) & \left\{\begin{array}{llll} 4 & * & l & 5 \\ 4 & | & l & 4 \end{array}\right. \\
(\delta) & \left\{\begin{array}{llll} 5 & * & r & 6 \\ 5 & | & l & 5 \\ 6 & | & * & 7 \end{array}\right. \\
(\epsilon) & \left\{\begin{array}{llll} 7 & * & r & 8 \\ 8 & * & * & 11 \\ 8 & | & r & 9 \end{array}\right. \\
(\zeta) & \left\{\begin{array}{llll} 9 & * & r & 0 \\ 9 & | & r & 9 \end{array}\right. \\
(\eta) & 10 \quad * \quad h \quad 10 \\
(\theta) & \left\{\begin{array}{llll} 11 & * & l & 12 \\ 12 & * & | & 12 \\ 12 & | & r & 13 \\ 13 & * & h & 13 \end{array}\right.
\end{array}
$$

nito de líneas. Así pues, debe haber una línea que sea la primera que por segunda vez resulte decisiva en el proceso considerado. Puesto que el comportamiento está determinado por las líneas, el procedimiento debe ser periódico a partir de ese momento. Pero esta reflexión no es del todo acertada, porque no tiene en cuenta que una computación está gobernada no sólo por la tabla de máquina, sino también

por el contenido de la cinta de cálculo. Más exactamente, podríamos decir lo siguiente. Supóngase, por ejemplo, que en el paso de cálculo k_1-ésimo debemos proceder de acuerdo con la línea n | r m de la tabla de máquina, y que en el paso k_2-ésimo $(k_2 > k_1)$ debemos hacer lo mismo otra vez. Supóngase además que delante del paso k_1-ésimo hay un cuadrado vacío en la cinta de cálculo situado inmediatamente a la derecha del cuadrado escrutado en ese momento. En este caso, tenemos que proceder en el paso de cálculo $(k_1 + 1)$-ésimo de acuerdo con la línea que empieza con m*. Pero solamente estamos obligados a proceder de acuerdo con la misma línea en el paso $(k_2 + 1)$-ésimo, si antes del paso k_2-ésimo hay también un cuadrado subsiguiente vacío a la derecha del cuadrado escrutado. De ningún modo es éste necesariamente el caso.

Que ningún argumento modificado sería suficiente, se sigue del hecho de que se puede aducir un ejemplo de una máquina de Turing que imprime en la cinta vacía la secuencia no periódica | * || * ||| * |||| * ... (cf. § 8.10).

Ejercicio 1. Construir una máquina de Turing que compute la función $f(n)$ = al resto de n dividido por 3, si se la coloca detrás del último cuadrado marcado por n en una cinta que no tenga escrita ninguna otra palabra.

Ejercicio 2. Sea W una palabra arbitraria sobre un alfabeto $\mathfrak{A}$. Aducir una máquina de Turing que imprima W en la cinta vacía y se pare sobre el cuadrado situado detrás de W.

§ 7. Combinación de máquinas de Turing

Resulta con frecuencia difícil de interpretar el comportamiento de la máquina a partir de una tabla cuando ésta es compleja. Parece recomendable introducir operaciones con cuya ayuda podamos combinar tablas simples para formar otras más complicadas. Nuestro propósito es construir todas las máquinas dadas en este libro a partir de las máquinas introducidas en § 6.5. El tipo de combinación que discutiremos aquí es análogo al «diagrama de flujo» (u «organigrama») que se emplea en la programación de computadores electrónicos (cf. ejemplo (4) en § 6.6).

Consideremos aquí una observación importante. Cuando combinamos las máquinas $\mathbf{M}_1$,..., $\mathbf{M}_r$ para formar una máquina $\mathbf{M}$, entonces $\mathbf{M}$ no queda determinada inequívocamente, sino sólo «dentro de los límites de la equivalencia», es decir, dentro de una reenumeración de los estados. Ello no nos perturba en absoluto, puesto que hemos dejado resuelto en § 5.6 que para los fines perseguidos aquí podemos reemplazar unas máquinas por otras equivalentes. Mediante instrucciones suplementarias podríamos determinar la máquina $\mathbf{M}$ totalmente y no sólo dentro de los límites de la equivalencia. Pero eso solamente sería posible con normalizaciones innecesarias, molestas y arbitrarias.

1. Diagramas. Sean $\mathbf{M}_1$,..., $\mathbf{M}_r$ símbolos de máquinas de Turing dadas, todas ellas sobre un alfabeto fijado $\{a_1, ..., a_N\}$. Con ayuda de estos símbolos $\mathbf{M}_i$ podemos producir diagramas D que satisfacen los siguientes requisitos.

(1) Al menos un símbolo $\mathbf{M}_j$ ocurre en *D*. Un símbolo puede ocurrir más de una vez. En total, sólo ocurrirá un número finito de tales símbolos.

(2) Exactamente una ocurrencia de uno de estos símbolos está marcada como *símbolo inicial* (p. ej., mediante un círculo que lo rodea).

(3) Los símbolos que ocurren en *D* están conectados mediante líneas con una dirección (flechas). Una flecha comienza en uno de estos símbolos y termina en otro, que puede coincidir con el primero (flecha que retorna). Toda flecha lleva un número *j (j* = 0 ,..., *N)*.

(4) Desde cualquier símbolo puede partir a lo sumo una flecha llevando el número *j (j* = 0 ,..., *N)*.

Se recomienda hacer uso de algunas *abreviaturas* (cf. fig. 7.2 que es una abreviatura de la fig. 7.1, donde *N=2)*. Si un símbolo de máquina está conectado con otro por *todas* las flechas $\xrightarrow{0}$,..., $\xrightarrow{N}$ (en la misma dirección), entonces utilizamos *una* sola flecha sin número. Si únicamente falta $\xrightarrow{j}$, entonces escribimos $\xrightarrow{\neq j}$. Si solamente ocurre un símbolo de máquina en el que no termine ninguna flecha, entonces ese símbolo ha de ser el símbolo inicial, y en tal caso no será preciso utilizar el círculo mencionado en (2). Si en cualquier caso no se da símbolo inicial, entonces ha de serlo el que esté situado *más a la izquierda*. En lugar de $\mathbf{M} \rightarrow \mathbf{M}$, escribiremos $\mathbf{M}^2$; en lugar de $\mathbf{M}^2 \rightarrow \mathbf{M}$, $\mathbf{M}^3$, etc. En lugar de $\mathbf{M}_1 \rightarrow \mathbf{M}_2$ escribimos $\mathbf{M}_1\mathbf{M}_2$; en lugar de $\mathbf{M}_1 \rightarrow \mathbf{M}_2 \rightarrow \mathbf{M}_3$, $\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3$, etc. Por $\mathbf{M}^0$ entendemos una máquina que es dada por la siguiente tabla:

$$
\begin{array}{cccc}
0 & a_0 & s & 0 \\
0 & a_1 & s & 0 \\
 & \cdots\cdots\cdots & & \\
0 & a_N & s & 0
\end{array}
$$

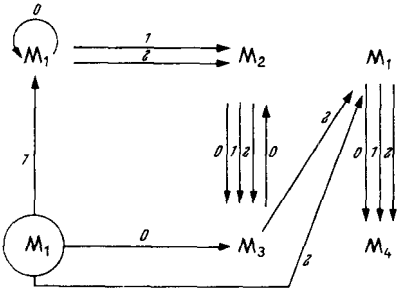


Fig. 7.1

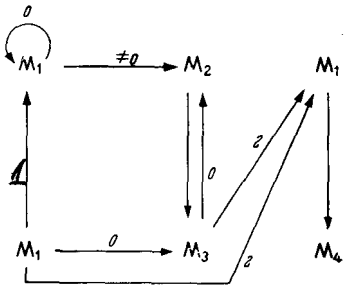


Fig. 7.2

Ejemplos de diagramas

2. Definición de la máquina **M** *representada por un diagrama D.* Esta máquina, como se acaba de indicar, queda determinada tan sólo dentro de los límites de la equivalencia. La *tabla de* **M** se obtiene como sigue:

(1) Primero se producen las tablas de las máquinas $\mathbf{M}_j$, representadas por símbolos en el diagrama. Si una máquina aparece en el diagrama más de una vez, en-

tonces se produce un número correspondiente de tablas de dichas máquinas y se asocian estas tablas con los símbolos correspondientes del diagrama.

Las tablas se elegirán de forma que dos tablas separadas no contengan el mismo *estado*.

(2) A continuación se produce una tabla «grande» escribiendo las tablas aisladas una debajo de otra. La secuencia es arbitraria, con la sola excepción de que la tabla asociada con el símbolo inicial esté situada en cabeza [9].

(3) Al objeto de obtener, a partir de esta tabla preliminar, la tabla definitiva de **M**, se llevarán a cabo las siguientes alteraciones. Si el símbolo $\mathbf{M}_i$ que ocurre en un cierto lugar del diagrama, está conectado por $\xrightarrow{k}$ con el símbolo $\mathbf{M}_j$ que ocurre en otro lugar, entonces se cambia cada línea de la forma

(*) $$c\ a_k\ h\ c'$$

que ocurre en la tabla correspondiente al primer símbolo (y en el caso de que se dé una línea tal) por

(**) $$c\ a_k\ a_k\ c_{Mj},$$

donde c_{Mj} es el estado inicial en la tabla correspondiente al símbolo $\mathbf{M}_j$. (Esta instrucción puede ser llevada a cabo de forma inequívoca, ya que se ha dado

[9] Este proceso puede efectuarse para el ejemplo dado en la fig. 7.1 ó 7.2 del siguiente modo:
Tabla de la máquina $\mathbf{M}_1$ que se indica abajo a la izquierda.
Tabla de $\mathbf{M}_4$.
Tabla de la máquina $\mathbf{M}_1$ que se indica arriba a la izquierda.
Tabla de $\mathbf{M}_2$.
Tabla de $\mathbf{M}_3$.
Tabla de la máquina $\mathbf{M}_1$ que se indica arriba a la derecha.

por supuesto que para cada símbolo hay, a lo sumo, una flecha $\xrightarrow{k}$ que a él conduce. En casos especiales es posible que no se requiera efectuar alteración alguna de acuerdo con (3).)

De la definición de la tabla de **M** se sigue que dicha tabla queda inequívocamente determinada dentro de los límites de la equivalencia.

3. El método de operación de la máquina **M** *obtenida a partir de un diagrama D* puede ser fácilmente seguido. Si colocamos a **M** sobre una expresión de cinta *B(x)* en un cuadrado *A*, entonces **M** ejecutará primero los mismos pasos que la máquina **M'** que esté denotada por el *símbolo inicial*, ya que la tabla de dicha máquina se encuentra situada en cabeza y proporciona, por tanto, el estado inicial. Tal será el caso hasta que la máquina **M'** eventualmente se pare tras haber dado un paso en la configuración (A_n, B_n, C_n). En dicho caso será decisiva para **M'** una línea de la forma (*), Es posible que **M'** no esté conectada por una flecha $\xrightarrow{k}$ con otro símbolo de máquina. Entonces la línea (*) no es alterada, y **M** se para también. Pero si **M'** está conectada por una flecha $\xrightarrow{k}$ con un símbolo de máquina **M''**, entonces **M** contiene, en lugar de (*), la línea (**) que es ahora decisiva. El efecto de ello es que la expresión de cinta y el cuadrado escrutado en ese instante no sufren alteración, pero se pone en marcha el estado inicial de **M''**. Lo cual significa que $A_{n+1} = A_n$, $B_{n+1} = B_n$, $C_{n+1} = c_{\mathbf{M''}}$. La siguiente configuración corresponderá ahora a la configuración de la máquina **M''** colocada sobre B_{n+1} en A_{n+1}, etc.

Por consiguiente, las configuraciones de **M**, si se la coloca sobre *B* en *A*, son primero las de **M'** si se la coloca sobre *B* en *A;* y luego (si **M'** se para sobre la

expresión de cinta B' en A') las de **M"** colocada sobre B' *en* A', etc.

Ello puede expresarse intuitivamente diciendo que **M** ejecuta sucesivamente la operación de las máquinas **M'**, **M"**, ... en una secuencia que está determinada por el diagrama (juntamente con los B y A originales).

4. Ejemplo. En § 6.1 se introdujeron las funciones C_0^k de cero argumentos que tienen el valor constante k. En general, podemos considerar para todo n una función de n argumentos que tiene el valor constante k. (Para estas funciones usaremos también la abreviatura k si no es posible ningún malentendido.) Todas estas funciones C_n^k son Turing-computables. Una computación se llevará a cabo, por ejemplo, mediante la máquina

$$* (\mathbf{r}\ \mathbf{l})^{k+1}\ \mathbf{r}\ *$$

Usando * se genera un cuadrado vacío que señala el comienzo del valor de la función que se va a computar. Entonces, usando $(\mathbf{r}\ \mathbf{l})^{k+1}$ imprimimos el valor de la función k (durante lo cual, los signos ya impresos en los cuadrados en cuestión serán quizás «sobreimpresos»). Usando **r*** nos cercioramos de que existe un cuadrado vacío situado a la derecha del valor de la función.

Ejercicio 1. Puede decirse que dos máquinas **M** y **M'** son *intercambiables* si para cualquier expresión de cinta B y para cualquier cuadrado inicial A se cumple lo siguiente. Si **M** (si está colocada en B sobre A) alcanza una configuración terminal (A_n, B_n, C_n), entonces **M'** (si está colocada en B sobre A) alcanza también una configuración terminal (A_m, B_m, C_m) con

$A_n = A_m$, $B_n = B_m$, y viceversa. Muéstrese que para cualquier máquina de Turing **M** sobre $\{a_1, ..., a_N\}$ es posible dar efectivamente una combinación de máquinas elementales **r**, $\mathfrak{l}$, $\mathbf{a}_0, ..., \mathbf{a}_N$ (§ 6.5) que es intercambiable con **M**.

Ejercicio 2. Probar que si una función *f* es computable según la definición de § 6.1, entonces es computable también si cambiamos esa definición en tanto que requerimos que la máquina de cálculo **M** se pare sobre un símbolo arbitrario del valor de la función, y viceversa (cf. nota 6**,** en la pág. 80).

§ 8. Máquinas de Turing especiales

Introducción. Construiremos aquí, utilizando los procedimientos discutidos en § 7, unas cuantas máquinas de Turing (que más tarde necesitaremos), a partir de las máquinas elementales **r**, $\mathfrak{l}$, $\mathbf{a}_0, ..., \mathbf{a}_N$ introducidas en § 6.5. Las definiciones juntamente con indicaciones sobre el método de operación de las máquinas más importantes se resumen en la tabla-sumario de la pág. 104. Para caracterizar el método de operación utilizaremos el siguiente y sencillo simbolismo:

m	un cuadrado con un símbolo real impreso en él; lo llamaremos, más brevemente, un *cuadrado marcado*,
~	un cuadrado marcado o vacío,
*	un cuadrado vacío,
...	una secuencia finita de cuadrados vacíos (al menos uno),
*...	una parte vacía de la cinta que va al infinito por la derecha,

W una parte de la cinta en la que está impresa la palabra no vacía *W* y que no contiene cuadrados vacíos. A una tal parte de la cinta la llamaremos frecuentemente, para abreviar, *la palabra W*,

X designará a $W_1 * W_2 * W_3 \ldots W_{n-1} * W_n$, donde $W_1, \ldots, W_n$ $(n \geqq 1)$ son palabras no vacías. En esta sección hablaremos también, para abreviar, de *la proposición X*.

Los cuadrados de los que no se da razón pueden llevar impresos símbolos arbitrarios [10]. Al cuadrado escrutado en cada instante lo caracterizaremos mediante un subrayado. A la izquierda de la flecha $\Rightarrow$ se describe la expresión inicial de cinta y el cuadrado escrutado original, y a la derecha la expresión de cinta y el cuadrado escrutado que resultan después de que la máquina en cuestión ha cesado de operar. Todas las máquinas que se exhiben en nuestra tabla (excepto posiblemente ϱ λ y **S**) se detienen después de un número finito de pasos.

Las máquinas que vamos a definir no son las únicas que resolverían las tareas en cuestión. Se las ha elegido porque su *método de operación* es fácil de seguir. No son, por tanto, todo lo simples que pudieran ser. Permítasenos indicar en este punto que no es claro cuándo hay que decir que una máquina es más simple que otra. Si de hecho realizamos la construcción de las máquinas a partir de las elementales (a las que consideramos ahora como los elementos constitutivos), entonces podríamos convenir en que el número de

[10] Aquí debemos tener presente la suposición, a la que nos sujetamos forzosamente, de que sólo esté marcado un número finito de cuadrados.

máquinas elementales necesario para la construcción fuese decisivo para la simplicidad. Por otra parte, puede interesarnos tomar en consideración el tiempo (i.e. el número de pasos) de solución de la tarea. Ello nos llevaría a cuestiones de mayor dificultad que no abordaremos en este libro.

A continuación aduciremos unos cuantos comentarios sobre las máquinas definidas en el Sumario.

1. *La máquina grande derecha* **R** *(la máquina grande izquierda* **L**). Pasa del cuadrado en que se la coloca al cuadrado vecino de la derecha (izquierda). Si este cuadrado está vacío, se detiene. Si, por el contrario, el cuadrado está marcado, entonces **R**(**L**) recorre todos los cuadrados marcados a la derecha (izquierda) hasta alcanzar el primer cuadro vacío, donde se para. La expresión de la cinta no se altera en ningún caso (cf. el sumario de la página 104, en el que figuran también las máquinas que a continuación se comentan.)

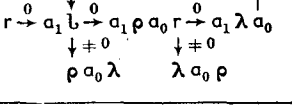
2. *La máquina de buscar derecha* ρ (*máquina de buscar izquierda* λ). Es, en cierto sentido, dual con respecto a **R**(**L**). ρ(λ) se desplaza desde el cuadrado en que está colocada a un cuadrado a la derecha (izquierda). Si ese cuadrado está marcado, se para. Pero, si ese cuadrado no está marcado, entonces ρ(λ) se desplaza a la derecha (izquierda) hasta alcanzar el primer cuadrado marcado, sobre el cual se detiene. La expresión de cinta no es alterada. Así pues, ρ(λ) «busca» el primer cuadrado marcado a la derecha (izquierda) del cuadrado escrutado original (y se para sobre dicho cuadrado marcado, en el supuesto de que lo haya).

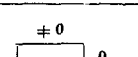
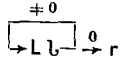
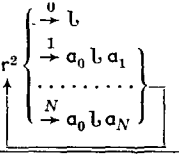
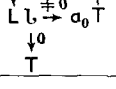
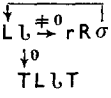
3. La máquina de buscar **S**. Realiza la siguiente operación: Si colocamos a **S** sobre un *cuadrado arbitrario* de la cinta, en la que haya al menos un cuadrado marcado, entonces **S** se parará después de un número finito de pasos sobre un cuadrado marcado. La expresión de cinta original coincidirá con la expresión de cinta terminal (en el curso de la computación, sin embargo, la expresión de cinta puede cambiar).

El método de construcción de **S** se basa en las siguientes consideraciones: Si pudiéramos suponer que existe un cuadrado marcado a la derecha (izquierda) del cuadrado escrutado original, entonces alcanzaríamos nuestro objetivo utilizando sencillamente $\rho(\lambda)$. Pero, dado que no podemos suponerlo, hemos de buscar sistemáticamente y de modo alternativo a la derecha y a la izquierda del cuadrado escrutado original hasta que, finalmente, alcancemos un cuadrado que esté marcado. Los límites (tanto a la derecha como a la izquierda) hasta donde se vaya extendiendo la búsqueda deben ser recordados en todo instante. A este fin utilizamos un «marcador» a ambos lados, que consiste en la letra a_1. Este marcador será llevado, paso a paso, cada vez más lejos, hasta que finalmente accedamos al cuadrado marcado. Sin duda, los marcadores de límite deberán ser borrados al término de la computación.

4. La máquina de extremo derecho (izquierdo) $\mathfrak{R}(\mathfrak{L})$. Se desplaza desde el cuadrado en que esté colocada hacia la derecha (izquierda) hasta alcanzar, por vez primera, el segundo de dos cuadrados vecinos que estén vacíos («doble intervalo») (sin contar el cuadrado escrutado original). Entonces retrocede un cuadra-

Sumario de máquinas de Turing importantes

N.º de Serie	Denominación simbólica	Nombre	Ilustración del método de operación[11]	Estructura
1a	R	**Máquina grande derecha**	$\simeq W * \Rightarrow \sim W \underline{*}$ $\simeq * \Rightarrow \sim \underline{*}$	
1b	L	**Máquina grande izquierda**	$* W \simeq \Rightarrow \underline{*} W \sim$ $* \simeq \Rightarrow \underline{*} \sim$	
2a	ρ	**Máquina de busca derecha**	$\simeq * \cdots * m \Rightarrow \sim * \cdots * \underline{m}$ $\simeq m \Rightarrow \sim \underline{m}$	
2b	λ	**Máquina de busca izquierda**	$m * \cdots * \simeq \Rightarrow \underline{m} * \cdots * \sim$ $m \simeq \Rightarrow \underline{m} \sim$	
3	S	**Máquina de buscar**	**Busca un cuadrado marcado**	

4a	$\mathfrak{R}$	**Máquina extremo derecho**	$\simeq X** \Rightarrow \sim X\underline{*}*$ $\simeq *X** \Rightarrow \sim *X\underline{*}*$ $\simeq ** \Rightarrow \sim\underline{*}*$	
4b	$\mathfrak{L}$	**Máquina extremo izquierdo**	$**X\simeq \Rightarrow **\underline{*}X\sim$ $**X*\simeq \Rightarrow **\underline{*}X*\sim$ $**\simeq \Rightarrow **\underline{*}\sim$	
5	T	**Máquina transporte izquierda**	$\simeq *W* \Rightarrow \sim W\underline{*}**$	
6	σ	**Máquina de desplazamiento**	$*W_1*W_2\underline{*} \Rightarrow *W_2\underline{*}\cdots*$	
7	C	**Máquina de borrar**	$\sim **X*W\underline{*} \Rightarrow \sim W\underline{*}\cdots*$	

Sumario de máquinas de Turing importantes (continuación)

N.° de Serie	Denominación simbólica	Nombre	Ilustración del método de operación [11]	Estructura
	K	**Máquina copiadora**	$*W\underline{*}\cdots \Rightarrow *W*W\underline{*}\cdots$ $*\underline{W}*\cdots \Rightarrow *W*W\underline{*}\cdots$	$\mathsf{L\,r}\left\{\begin{array}{l}\overset{0}{\rightarrow}\ \mathsf{R}\\ \overset{1}{\rightarrow}\ a_0\,\mathsf{R}^2 a_1\,\mathsf{L}^2 a_1\\ \cdots\cdots\cdots\cdots\\ \overset{N}{\rightarrow}\ a_0\,\mathsf{R}^2 a_N\,\mathsf{L}^2 a_N\end{array}\right\}$
9	K_n	**Máquina n-copiadora** *($n \geq 1$)*	$\dfrac{*W_n*W_{n-1}\cdots*W_1\underline{*}\cdots \Rightarrow}{*W_n*W_{n-1}\cdots*W_1*W_n\underline{*}\cdots}$	$\mathsf{L}^n\mathsf{r}\left\{\begin{array}{l}\overset{0}{\rightarrow}\ \mathsf{R}^n\\ \overset{1}{\rightarrow}\ a_0\,\mathsf{R}^{n+1} a_1\,\mathsf{L}^{n+1} a_1\\ \cdots\cdots\cdots\cdots\\ \overset{N}{\rightarrow}\ a_0\,\mathsf{R}^{n+1} a_N\,\mathsf{L}^{n+1} a_N\end{array}\right\}$

[11] Aquí no se enumeran todos los casos posibles, sino sólo los que son interesantes en relación con las aplicaciones consideradas. *Los cuadrados no especificados nunca son objeto de escrutinio y así quedan inalterados.* Para la notación, cf. la introducción a § 8, pág. 100.

do a la izquierda (derecha). La expresión de cinta no es alterada.

5. La máquina de transporte a la izquierda **T**. Desplaza una palabra un cuadrado a la izquierda. (Conforme al mismo principio podría construirse una máquina de transporte a la derecha, pero su uso no se requerirá en este libro). El desplazamiento de la palabra se lleva a cabo letra por letra partiendo de la izquierda.

6. La máquina de desplazamiento σ. Lleva a cabo lo siguiente. Dadas dos palabras consecutivas $*W_1*$ W_2*, sea el cuadrado escrutado el que se encuentre detrás de W_2 . Entonces W_1 es borrada y W_2 es desplazada a la izquierda hasta que el comienzo de ella coincida con el comienzo original de W_1. Después de lo cual σ se para en el cuadrado que hay detrás de la palabra desplazada W_2.

Seguiremos el proceso con más exactitud.

(1) Por virtud de **L** ι la máquina se colocará, después de algunos pasos, sobre el segundo cuadrado a la izquierda de W_2 . Este cuadrado podría estar marcado (véase (2)) o no marcado (véase (3)). (Cuando se alcance por primera vez dicho cuadrado se hallará, de acuerdo con nuestra hipótesis, que está marcado. Así pues, en este caso la máquina pasará a proceder de conformidad con (2).)

(2) Si el cuadrado está marcado, la máquina borra mediante $\overset{\neq 0}{\longrightarrow}$ $\mathbf{a}_0$ el símbolo en él impreso y desplaza a W_2 un cuadrado a la izquierda con ayuda de **T**. Ahora, en principio, estamos de nuevo en la situación inicial, aunque con la palabra W_1 acortada. No debemos dejar de tener presente, sin embargo, que la palabra

W_1 puede haber desaparecido ya por completo. La computación ulterior se lleva a cabo de acuerdo con su retroacoplamiento a (1).

(3) Ahora la palabra W_1 ha desaparecido por completo, pero W_2 ha de ser desplazada todavía un cuadrado más a la izquierda. Ello sucede de acuerdo con $\xrightarrow{0}$ **T**.

7. *La máquina de borrar* **C**. Necesitaremos ulteriormente una máquina **C** para *borrar computaciones*. La tarea de esta máquina consiste en borrar cálculos secundarios y trasladar el resultado final a una cierta posición. Supondremos que los cálculos secundarios están impresos en la cinta en forma de una sentencia X (es decir, en una secuencia de palabras que están separadas unas de otras por intervalos). El resultado queda situado en forma de una palabra W detrás de esos cálculos secundarios con un cuadrado en medio. Todos los cuadrados situados a la derecha del resultado están vacíos. A la izquierda de los cálculos secundarios hay al menos dos cuadrados vacíos (de otro modo no podríamos reconocer el «comienzo» de los cálculos secundarios) (cf. el sumario). **C** borra los cálculos secundarios comenzando por la derecha y lleva el resultado hasta que el comienzo de la palabra desplazada W esté en el primero de los dos cuadrados vacíos que se acaban de mencionar.

La forma de operar **C** se completa detalladamente como sigue.

(1) En virtud de **L** ı , la máquina se colocará, después de algunos pasos, sobre el segundo cuadrado a la izquierda de W. Este cuadrado podría estar marcado (véase (2)) o no marcado (véase (3)). (Cuando se alcance por primera vez dicho cuadrado se hallará, de

acuerdo con nuestra hipótesis, que está marcado. Así pues, en este caso la máquina pasará a proceder de conformidad con (2).)

(2) La máquina desplaza a W, mediante $\overset{\neq 0}{\rightarrow}$ **rR**σ, al comienzo de la última parte de la proposición X. Entonces se vuelve a acoplar con **L**.

(3) Los cálculos secundarios han desaparecido por completo. Queda la tarea de mover a W dos cuadrados más a la izquierda. Ello sucede de acuerdo con $\overset{0}{\rightarrow}$ **TL** ι **T**.

8. La máquina copiadora **K**. La máquina de traslación **T** desplaza una palabra, y así, en cierto modo, la copia. Pero si aplicamos esa computación, la expresión original de la cinta se perderá. Sin embargo, a menudo se nos plantea la tarea de copiar una palabra de suerte que se preserve la expresión original. A este propósito construiremos una *máquina copiadora* **K**. **K** ejecuta la siguiente computación. Damos por supuesto que se coloca a **K** en el cuadrado detrás de la palabra W. A la derecha de esta palabra, todos los cuadrados de la cinta están vacíos. Entonces **K** se parará después de un número finito de pasos. Una vez se haya detenido, todos los cuadrados de la cinta de cálculo que estuviesen marcados al comienzo de la computación seguirán marcados del mismo modo. En adición a ello se habrá impreso una copia de W a la derecha de la palabra original W, dejando en medio un cuadrado. **K** se para detrás del último de los cuadrados que estén marcados al final de la computación. Seguiremos el método de operación de **K** considerando separadamente las distintas partes del procedimiento en el ejemplo $W = babb$. (Suponemos aquí que el alfabeto $\{a_1, a_2\} = \{a, b\}$ suministra la base.)

(1) La cinta de cálculo al comienzo de la computación es:

$$* \; b \; a \; b \; b \; \underline{*} \; \ldots .$$

(2) Con ayuda de **L** la máquina retrocede al primer cuadrado que haya en frente del primer símbolo *b:*

$$\underline{*} \; b \; a \; b \; b \; * \; \ldots .$$

(3) Por medio de **r**, la máquina pasa a situarse ahora en el cuadrado vecino a la derecha:

$$* \; \underline{b} \; a \; b \; b \; * \; \ldots .$$

(4) El símbolo que haya en el cuadrado escrutado en ese momento es borrado por medio de $\mathbf{a}_0$; por medio de **R**, la máquina se mueve hacia el cuadrado a la derecha de *W*, y por medio de la segunda **R** vuelve a moverse un cuadrado más a la derecha, donde, por medio de **b**, imprime *b:*

$$* \; * \; a \; b \; b \; * \; \underline{b} \; * \; \ldots .$$

(5) Con ayuda de $\mathbf{L}^2$, la máquina retrocede al lugar donde se borró la letra *b* y la reproduce mediante **b**:

$$* \; \underline{b} \; a \; b \; b \; * \; b \; * \; \ldots .$$

(6) La máquina pasa, por medio de **r**, al cuadrado inmediatamente a la derecha:

$$* \; b \; \underline{a} \; b \; b \; * \; b \; * \; \ldots \; .$$

(7) Por medio de $\mathbf{a}_0$ se borra el símbolo en este cuadrado. Por medio de $\mathbf{R}^2$ la máquina se mueve hacia el cuadrado a la derecha de la letra *b* ya copiada y, por medio de **a**, imprime *a:*

$$* \; b \; * \; b \; b \; * \; b \; \underline{a} \; * \; \ldots \; .$$

(8) Por medio de $\mathbf{L}^2$, la máquina vuelve al lugar donde se borró *a*. Aquí puede advertirse que la importancia de esta operación de borrar las letras de la palabra original reside en el hecho de que con ayuda de tal marcador puede hallarse la letra que hay que copiar a continuación. La máquina reproduce *a* por medio de **a**:

$$* \; b \; a \; b \; b \; * \; b \; \underline{a} \; * \; \ldots .$$

(9) Aducimos a continuación las expresiones de cinta y los cuadrados escrutados tras ulteriores pasos que son característicos:

$$* \; b \; a \; \underline{b} \; b \; * \; b \; \; a \; * \ldots$$
$$* \; b \; a \; \underline{*} \; b \; * \; b \; \; a \; * \ldots$$
$$* \; b \; a \; * \; b \; * \; b \; \; a \; \; \underline{b} \; * \ldots$$
$$* \; b \; a \; \; \underline{b} \; b \; * \; b \; a \; \; b \; * \ldots$$
$$* \; b \; a \; b \; \underline{b} \; * \; b \; a \; \; b \; * \ldots$$

$$
\begin{aligned}
&* \; b \; a \; b \; \underline{*} \; * \; b \; a \; b \; * \; \ldots \\
&* \; b \; a \; b \; * \; * \; b \; a \; b \; \underline{b} \; * \; \ldots \\
&* \; b \; a \; b \; \underline{b} \; * \; b \; a \; b \; b \; * \; \ldots \\
&* \; b \; a \; b \; b \; \underline{*} \; b \; a \; b \; b \; * \; \ldots \; .
\end{aligned}
$$

Podemos comprobar que el procedimiento funciona asimismo en el último estadio, cuando la última letra de *W* ha sido ya borrada y reproducida. Ahora el cuadrado escrutado está vacío. No procede, por tanto, copiar nada más. La máquina se mueve entonces, con ayuda de **R**, hasta colocarse detrás de la última letra de la palabra copiada, con lo que se obtiene finalmente:

$$* \; b \; a \; b \; b \; * \; b \; a \; b \; b \; \underline{*} \; \ldots \; .$$

9. La máquina n-copiadora $\mathbf{K}_n$. Con frecuencia se plantea la tarea (especialmente al computar funciones de varios argumentos) de copiar una palabra que no está completamente a la derecha, de forma que el procedimiento de copia se ha de llevar a cabo pasando por encima de unas cuantas palabras impresas en medio. La máquina de copiar $\mathbf{K}_n$ realiza esta computación. *Suponemos a este respecto que las palabras en cuestión están separadas entre sí por intervalos de sólo un cuadrado*. $\mathbf{K}_1$ es idéntica a **K**. $\mathbf{K}_n$ se construye conforme al patrón de **K**, con la diferencia de que hay que saltar cada vez por encima de las palabras intermedias sin interés para el cómputo. Téngase en cuenta que *n* es un número *fijado;* para todo *n*, existe una máquina $\mathbf{K}_n$.

10. Máquinas de Turing y periodicidad. Afirmábamos en § 6.7 que podemos dar razón de una má-

quina de Turing que imprime en la cinta de cálculo, inicialmente vacía, la secuencia aperiódica

$$| * || * ||| * |||| * \ldots$$

Ello se lleva a cabo mediante la máquina $|\mathbf{rK}$. El proceso es fácil de seguir. Primero se marca un cuadrado con |, a continuación de lo cual la máquina se mueve un cuadrado a la derecha. Entonces **K** copia este trazo. Merced a |**r** se añade un nuevo trazo y se vuelve a mover la máquina otro cuadrado a la derecha. Ahora la máquina **K** copia la última palabra, que consta de dos trazos, y luego se añade un nuevo trazo, etc.

§ 9. Ejemplos de Turing-computabilidad y Turing-decidibilidad

En la definición de Turing-computabilidad de una función (§ 6.1) y de Turing-decidibilidad de un predicado (§ 6.3) hemos exigido que la máquina que ejecuta la tarea pueda colocarse sobre un cuadrado *arbitrario* de la cinta de cálculo. Como ya hemos subrayado, estas definiciones gozan de la ventaja de verse libres de la arbitrariedad que nace de prescribir la elección del cuadrado inicial. Por otra parte, no está inmediatamente claro cómo puede satisfacerse esa grave condición. Esta es una de las razones por las que hasta aquí sólo hemos aducido un ejemplo trivial de Turing-computabilidad (cf. § 7.4). Con ayuda de las máquinas desarrolladas en la sección anterior, especialmente con la máquina de buscar, nos encontramos ahora en situación de solucionar este problema.

1. Cuadrados iniciales especiales y arbitrarios de la computación de funciones y de la decisión de predicados. Discutiremos aquí el caso de la computabilidad de funciones. Lo mismo vale también, mutatis mutandis, para la decidibilidad de predicados. Partimos del supuesto de que se conoce una máquina **M'** que lleva a cabo la computación del valor de una función *n*-aria *(n* $\geqq$ 1), con tal que **M'** se coloque sobre un *cierto* cuadrado inicial $a_{W_1 \ldots W_n}$. (Podemos, por ejemplo, tomar el último cuadrado que tiene un símbolo de los argumentos impresos en él para que sea el cuadrado inicial, o también podemos tomar el cuadrado vacío que le sigue inmediatamente; cf. los ejemplos de § 6.6.) Suponemos también que se puede encontrar el cuadrado $a_{W_1 \ldots W_n}$ con la ayuda de una máquina dada **N** de tal modo que si se coloca **N** detrás del último cuadrado marcado con los argumentos W_1 ,..., W_n , entonces se parará después de un número finito de pasos sobre $a_{W_1 \ldots W_n}$. Entonces podemos describir efectivamente una máquina **M** con la ayuda de **M'** y **N** que computa *f* en el sentido de § 6.1; así pues, se puede colocar inicialmente sobre un cuadrado arbitrario. Ello se muestra en:

Teorema 1. Sea *f* una función *n*-aria ($n \geqq 1$) que está definida para todas las palabras sobre un alfabeto $\mathfrak{A}_0 = \{a_1, \ldots, a_{N0}\}$ *y* que toma palabras sobre dicho alfabeto como valores [12]. Sea **N** una máquina sobre $\mathfrak{A}_0$ que realiza lo siguiente: Si se coloca **N** detrás del último símbolo de un *n*-tuplo arbitrario *(*W_1 ,..., W_n*)* de palabras, entonces **N** se parará sobre un cuadrado que denominaremos $a_{W_1 \ldots W_n}$. La expresión de cinta original no deberá ser alterada al final de la computación realizada por **N**. Sea **M'** una máquina tal que si imprimimos un *n*-tuplo W_1 ,..., W_n de argumentos en la

cinta de cálculo, que en lo demás debe estar vacía, y si colocamos a **M'** sobre el cuadrado $a_{W_1 \ldots W_n}$ de la cinta marcada de esa forma, entonces **M'** se parará después de un número finito de pasos detrás del valor $f(W_1, \ldots, W_n)$ de la función. *Entonces f es Turing-computable. Una máquina* **M** *que Turing-computa f será dada por:*

$$\mathbf{M} = \mathbf{S}\,\mathfrak{R}\,\mathbf{N}\,\mathbf{M}'.$$

La aserción es evidente. **S** busca un cuadrado que esté marcado por los argumentos; entonces $\mathfrak{R}$ conduce al cuadrado situado detrás del último cuadrado marcado por el argumento y **N** al cuadrado $a_{W_1 \ldots W_n}$ sobre el cual debe ser colocada **M'** para computar $f(W_1, \ldots, W_n)$ [13].

Similarmente se puede probar el siguiente:

Teorema 2. Sea R una relación n-aria $(n \geqq 1)$ en el dominio de palabras sobre un alfabeto $\mathfrak{A}_0 = \{a_1, \ldots, a_{N_0}\}$. Sea **N** el mismo tipo de máquina que **N** del teorema anterior. Sea **M'** una máquina tal que si imprimimos un n-tuplo $W_1, \ldots, W_n$ de palabras en la cinta, que en lo demás debe estar vacía, y si colocamos a **M'** en la cinta marcada de esta forma sobre $a_{W_1 \ldots W_n}$, enton-

[12] Recuérdese que en § 1.4 acordamos permitir solamente palabras no vacías como argumentos y valores de una función.

[13] Puede suceder que **M'** utilice letras auxiliares, es decir, que esté definida sobre un alfabeto $\mathfrak{A} = \{a_1, \ldots, a_N\}$ con $N > N_0$. En este caso, las máquinas **S**, $\mathfrak{R}$ y **N** han de ser reemplazadas por las máquinas $\overline{\mathbf{S}}$, $\overline{\mathfrak{R}}$ y $\overline{\mathbf{N}}$ respectivamente, que son definidas también sobre $\mathfrak{A}$. En tal caso, las tablas de $\overline{\mathbf{S}}$, $\overline{\mathfrak{R}}$ y $\overline{\mathbf{N}}$ son extensiones de las tablas de **S**, $\mathfrak{R}$ y **N** respectivamente. En las líneas adicionales, las dos últimas columnas pueden elegirse arbitrariamente puesto que $\overline{\mathbf{S}}$, $\overline{\mathfrak{R}}$ y $\overline{\mathbf{N}}$ serán colocadas en este caso sólo en una expresión de cinta sobre un alfabeto $\mathfrak{A}_0$.

ces **M'** se parará después de un número finito de pasos sobre el símbolo a_i ó a_j $(i \neq j,\ 1 \leqq i\ ,\ j \leqq N_0)$, según que $RW_1 \ldots W_n$ sea o no válida. *Entonces R es Turing-decidible. Una máquina* **M** *que Turing-decide R será dada por:*

$$\mathbf{M} = \mathbf{S}\ \mathfrak{R}\ \mathbf{N}\ \mathbf{M'}.$$

2. Ejemplos de funciones Turing-computables. Consideraremos aquí funciones cuyos argumentos y valores son números naturales. Los números naturales son representados de la manera descrita en § 1.4. Todas las máquinas de Turing siguientes son máquinas sobre el alfabeto {|}. *Supondremos además que estas máquinas están colocadas sobre el primer cuadrado situado detrás de los argumentos dados.* Dicho cuadrado es ya el cuadrado inicial de **M'**. Así, podemos tener, por ejemplo, $\mathbf{N} = \mathbf{r}^0$. Por consiguiente, podemos aplicar el teorema 1 del apartado anterior y con su ayuda podemos encontrar efectivamente máquinas que computen las funciones consideradas en el sentido de la definición de § 6.1:

(1) La *función sucesor S(x) es computada por* **|r** (cf. § 6.6).

(2) La *función suma* $x + y$ es computada por:

$$\mathbf{S}_0 = \mathbf{L}\ |\ \mathbf{R}\ 1\ *\ 1\ *.$$

Primeramente, la máquina une los argumentos, separados por un intervalo de un cuadrado, llenando dicho intervalo. Luego, debemos quitar dos trazos puesto que el número n está representado por $n + 1$ trazos. La interpretación de la suma como el número

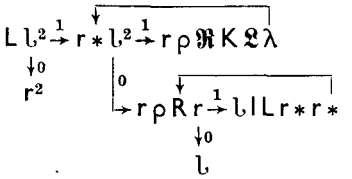
cardinal de una unión de conjuntos está reflejada en el siguiente proceso de computación.

(3) $f(x) = 2x$ es computada por:

$$\mathbf{K}\mathbf{S}_0.$$

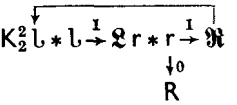
Primero, **K** copia el argumento, y luego $\mathbf{S}_0$ computa la suma (cf. (2)).

(4) La *función producto* $x \cdot y$ es computada por:



Ilustremos el método de operación sólo brevemente con la observación de que (aparte del caso especial cuando el primer argumento es igual a cero) la palabra representada por el segundo argumento es copiada por **K** tantas veces como esté determinado por el primer argumento. Entonces, estas copias (junto con el segundo argumento original) son unidas conjuntamente llenando los intervalos. Al hacer esto tenemos que quitar dos trazos cada vez (cf. (2)).

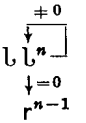
(5) La función max(x, y) puede ser Turing-computada por:



La computación se lleva a cabo esencialmente como sigue. Reproducimos los dos argumentos por $\mathfrak{K}_2^2$. Luego, reducimos alternativamente las palabras situadas más a la derecha y más a la izquierda por medio de un trazo comprobando cada vez si se han agotado dichas palabras después de la reducción. Si la palabra situada *más a la derecha* ha sido borrada primero, entonces el *primer* argumento es el máximo requerido. Éste se conserva todavía intacto como una copia. Estamos justamente detrás de él y sólo necesitamos parar. Si, por otra parte, la palabra situada *más a la izquierda* ha sido agotada primero, entonces el *segundo* argumento es igual al máximo. Este está todavía en su posición original. Hay que colocarse detrás de él.

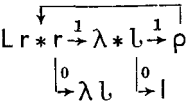
3. Ejemplos de relaciones Turing-decidibles. Nos limitaremos, lo mismo que en el apartado 2, a dar razón de máquinas que realizan la decisión (con $a_i = a_0$ y $a_j = a_1$) si están colocadas detrás del último cuadrado marcado por un argumento.

(1) La propiedad de que un número x que es divisible por un número fijado n $(n \geqq 1)$ es Turing-decidible. Aducimos una máquina que, si está colocada detrás del último trazo de x en una cinta en lo demás vacía, se para después de un número finito de pasos y, más precisamente, sobre l si n no divide a x y sobre * si n divide a x.



(2) La *relación de igualdad* entre números es Turing-decidible (el enunciado correspondiente es na-

turalmente válido también sobre un alfabeto arbitrario (cf. ejercicio 2)). Para ello, damos razón de una máquina que, si se coloca detrás del último trazo de la expresión de cinta... * z_1 * z_2 * ..., se para después de un número finito de pasos, y efectúa el mismo proceso sobre * si $z_1 = z_2$, y sobre | si $z_1 \neq z_2$. La máquina determinada aquí quita alternativamente (comenzando por la mitad) un trazo de z_1 y z_2 y comprueba cada vez si con la eliminación del último trazo ha desaparecido el argumento en cuestión. El resultado * aparece al final si y sólo si los dos argumentos han sido borrados al mismo tiempo (es decir, si y sólo si $z_1 = z_2$).



Ejercicio 1. Dar explícitamente una tabla de $\mathfrak{R}$.

Ejercicio 2. Aducir una máquina que decida la relación de igualdad entre palabras arbitrarias sobre un alfabeto $\{a_1,..., a_N\}$.

ANEXO

INTRODUCCIÓN
Por José Fernández-Prida

El objetivo de este anexo es la definición de una máquina de Turing capaz de simular, en un determinado sentido, la computación a que da lugar una máquina de Turing cualquiera cuando parte de una configuración inicial arbitraria. Puesto que de la Tesis de Church se sigue que todo algoritmo puede a su vez ser simulado por una máquina de Turing, el carácter universal de la máquina que se va a definir queda aún más patente si se tiene en cuenta su idoneidad para simular la computación a que da lugar cualquier algoritmo imaginable, cuando recibe cualquier posible «input».

Para precisar la forma en que se realiza la referida simulación y para definir de forma efectiva la máquina universal es necesario introducir previamente algunos nuevos conceptos y hacer referencia a algunos resultados relativos a los mismos.

a) *Turing-computabilidad normada*

«Este concepto difiere del de Turing-computabilidad introducido en § 6.1. En primer lugar, se refiere a funciones cuyos valores y argumentos son números naturales (i.e., secuencias de trazos), mientras que para la Turing-computabilidad habíamos acordado que

los valores y argumentos fuesen palabras arbitrarias no vacías. En las máquinas de Turing tratadas en esta sección, el símbolo | debe necesariamente pertenecer a su alfabeto $\mathfrak{A}$. Además, estableceremos incluso que $\mathfrak{A}^c\{|\}$. Por otra parte, se nos ofrecen dos diferencias dignas de consideración: 1) El cuadrado escrutado original está definido (lo cual es una simplificación). 2) Se aducirá una lista de condiciones agravantes que enseguida trataremos. Tales condiciones agravantes presentan la ventaja, como veremos más adelante, de permitirnos construir con facilidad máquinas que computen funciones más complicadas a partir de máquinas que satisfacen estas condiciones. Constituye una importante limitación que no sigamos suponiendo vacía la cinta de cálculo al comienzo, excepto el argumento dado. Además, concederemos que puedan imprimirse inscripciones arbitrarias sobre la cinta *a la izquierda* de los argumentos dados (con cierto intervalo entre ellos), mientras que supondremos (como antes) que *a la derecha* de los argumentos la cinta está vacía. Para una formulación más conveniente de la definición vale la pena introducir dos conceptos:

a) *Media cinta* está determinada por un cuadrado. Consta de este cuadrado (a título de primer cuadrado) y de todos los cuadrados *a la derecha* del mismo.

b) Por un *segmento argumental* de una función *n*-aria entendemos una parte finita de la cinta sobre la cual se ha impreso $* W_1 * W_2 *...* W_n$. Esto es, el primer cuadrado del segmento argumental está vacío, y el último tiene impreso sobre él el símbolo W_n. Esto se cumple si $n \geqq 1$. Un segmento argumental para una función *n*-aria no contendrá ningún cuadrado.

Supondremos que, al comienzo de la computación de una función, los argumentos dados están sobre un

segmento argumental, y que la media cinta que comienza inmediatamente a la derecha de este segmento argumental está vacía. Por otra parte, la cinta puede estar marcada de forma arbitraria a la izquierda del segmento argumental. Naturalmente, nos aseguraremos de que la inscripción de la izquierda del segmento argumental no perturbe la computación de la función. Por consiguiente, supondremos que durante la computación todos los cuadrados que se escruten estén situados en el segmento argumental o en la media cinta mencionada arriba. Más precisamente tenemos la siguiente:

Definición. Llamamos Turing-computable estándar a una función n-aria $(n \geqq 1)$ si existe una máquina de Turing M sobre un alfabeto {|} de un elemento que posea la siguiente propiedad: Si imprimimos el n-tuplo τ de argumentos sobre la cinta de cálculo en la forma habitual (cf. § 6.1), y si la mitad de la cinta H cuyo primer cuadrado es el cuadrado que se encuentra inmediatamente a la derecha del segmento argumental está vacía, entonces la máquina situada sobre el primer cuadrado de H se detiene tras un número finito de pasos, y al término de la computación tenemos que:

0) Los argumentos dados ocupan el mismo lugar que al comienzo.

1) El valor de la función comienza en el segundo cuadrado de H quedando, por tanto, un espacio de un intervalo entre el valor de la función y los argumentos.

2) M está sobre el cuadrado inmediatamente posterior al último trazo del valor de la función.

3) *H* está vacía excepto para el valor de la función.

Además tenemos que:

4) Durante la computación, únicamente son escrutados los cuadrados correspondientes al segmento argumental, determinado por los argumentos, y los cuadrados de *H.»*[*]

Aunque, obviamente, toda función Turing-computable en forma normada es también Turing-computable, la proposición contraria, igualmente cierta, es difícil de probar. Una demostración detallada de este resultado, en la que se hace uso esencial de la noción de recursividad, se encuentra en Hermes (1974).

b) *Tesis de Church*

En § 3 se hizo referencia a la propuesta, realizada por Alonzo Church en 1936, de identificar la noción intuitiva de función computable con la de función definible en el λ-cálculo, formalismo introducido por el propio Church en 1932 e investigado en colaboración con S. C. Kleene a lo largo de los años siguientes. Puesto que las funciones definibles en el λ-cálculo son precisamente las Turing-computables (Turing, 1936), teniendo en cuenta el resultado referido en el último párrafo del apartado anterior, la tesis de Church puede formularse en los siguientes términos: una función es computable si y sólo si es Turing-computable en forma normada. Y, en consecuencia, una relación

* Traducción del texto correspondiente a Hermes, *Aufzählbarkeit. Entscheidbarkeit. Berechenbarkeit*, Springer-Verlag, Berlin-Heidelberg-New York ^{2}1971, 96-97.

es decidible si y sólo si su función característica es Turing-computable en forma normada.

Aunque no tiene sentido pretender demostrar la tesis de Church, *puesto que propone la identificación de dos objetos de naturaleza distinta*, *uno perfectamente definido y otro un tanto impreciso*, *en la actualidad es admitida por la práctica totalidad de los matemáticos (cf. § 3) y son muchos los argumentos aducidos a su favor*, *que pueden distribuirse en tres grupos:*

1) Argumentos de tipo empírico, *basados en el fracaso de todos los intentos realizados desde 1936 de encontrar una función computable que no sea Turing-computable*.

2) Argumentos basados en el análisis de una computación realizada por una mente humana y en la posibilidad de ser llevada a cabo por una máquina de Turing.

3) Argumentos apoyados en la coincidencia de la clase de funciones a que dan lugar numerosos formalismos de tipo muy diverso, *tales como los sistemas de Thue*, *el λ-cálculo de Church*, *el cálculo de las funciones recursivas generales de Herbrand y Gödel*, *las máquinas de Turing*, *los sistemas de producción de Post*, *los algoritmos de Markov*, *las máquinas de registros de Sturgis y Sheperdson*, *los operadores de registros de Ródding*, *las funciones programables en ciertos lenguajes tales como el LISP de McCarthy*, *el PL de Brainerd y Landweber*, *etc*. *Coincidencia muy difícil de explicar si la tesis de Church no fuese cierta*.

c) *Gödelización de las máquinas de Turing*

El término «gödelización» se utiliza para denotar una asignación efectiva de un número natural a los

elementos de una cierta clase de objetos matemáticos, susceptibles de ser identificados con ciertas palabras o expresiones de un determinado lenguaje. Kurt Gödel, de cuyo apellido deriva el término, introdujo esta técnica en su célebre prueba del teorema de incompletitud de la aritmética (1931).

Sea $B = \{b_1, b_2, ...\}$ *un cierto alfabeto, finito o infinito, y sea* B* *el conjunto de palabras formadas con símbolos de* B. *Una gödelización de* B* *puede ser definida haciendo corresponder a la palabra* $W = b_{i_0} b_{i_1} ... b_{i_k}$ *el número natural* $g(W) = 2^{i_0} \cdot 3^{i_1} ... p_k^{i_k}$, *donde* p_k *denota el número primo de lugar k, asignándose el número cero a la palabra vacía.*

Puesto que una máquina de Turing puede ser identificada con una palabra sobre el alfabeto { # , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, r, l, s, a_0, a_1... }, *correspondiendo, por ejemplo, la máquina*

$$M = \begin{array}{cccc} 0 & a_0 & r & 1 \\ 0 & a_1 & r & 1 \\ 1 & a_0 & s & 1 \\ 1 & a_1 & s & 1 \end{array}$$

a la palabra

$$\# 0a_0r1 \# 0a_1r1 \# 1a_0s1 \# 1a_1s1 \#,$$

de acuerdo con las anteriores consideraciones a cada máquina puede asignarse un número natural. Concretamente, a la máquina M *se asignará el número*

$$g(M) = 2^1 \cdot 3^2 \cdot 5^{15} \cdot 7^{12} \cdot 11^3 \cdot 13^1 ... p_{20}^{\ 1}.$$

Abusando del lenguaje, si g(M) = x *llamaremos a* M *la máquina* x.

Por supuesto, *no todo número natural es el número de una máquina*. *Sin embargo*, *el predicado* T* *tal que* T*x *se verifica si y sólo si* x e*s el número de una máquina es claramente decidible*. *Además*, *si existe una máquina* M *tal que* g(M) = x, *es claro que* M *(la máquina* x) *puede determinarse a partir de* x.

Mediante consideraciones completamente similares es posible asignar un número natural a los elementos A, B, C *(campo de trabajo*, *inscripción y estado)*, *que integran una configuración*. *Por ejemplo*, *numerando los campos de la cinta de una máquina de Turing de la forma*

… 13 11 9 7 5 3 1 0 2 4 6 8 10 12 14 16 …

a la inscripción dada por el diagrama

…	*	*	a_3	a_4	*	*	a_5	a_3	a_1	*	*	…
…	9	7	5	3	1	0	2	4	6	8	10	…

identificable con la palabra

$$a_0 a_0 a_5 a_4 a_3 a_3 a_1$$

puede hacerse corresponder el número $2^0 \cdot 3^0 \cdot 5^5 \cdot 7^4 \cdot \cdot 11^3 \cdot 13^3 \cdot 17^1$.

Finalmente, *si* K *es la configuración* (A, B, C), *asignaremos a* K *el número natural* $2^a \cdot 3^b \cdot 5^c$ *donde*

a, b, c, *son respectivamente los números asignados a* A, B, C.

Obviamente puede decidirse si k *es el número de una configuración y, en caso afirmativo, ésta puede ser reconstruida a partir de* k.

d) *El predicado* E *y la función* F

Sea E *el predicado binario tal que* Exk *se verifica si y sólo si* x *es el número de una máquina de Turing y* k *es el número de una configuración final de la máquina* x. *Puesto que claramente* E *es decidible, existe una máquina de Turing* E *que computa en forma normada su función característica.*

Sea F *la función tal que* F(x,k) *es igual a* k' *o a* 0, *según que se verifiquen o no los cuatro siguientes supuestos:*

1) x *es el número de una máquina de Turing.*
2) k *es una posible configuración de la máquina* x.
3) *No se verifica* Exk (k *no es una configuración final de la máquina* x).
4) *Colocada la máquina* x *en la configuración* k, *al cabo de un paso adquiere una configuración de número* k'.

Puesto que F *es computable, de acuerdo con la tesis de Church existe una máquina de Turing* F *que la computa en forma normada.*

e) *Reducibilidad*

Sean R *y* S *predicados numéricos tales que* $R \subset N^p$ *y* $S \subset N^q$. *Se dice que* R *es reducible a* S *si existen fun-*

ciones computables de p *argumentos* f_1, f_2,..., f_q *tales que para todo* x_1, x_2, ..., x_p *se verifica:*

$Rx_1x_2...x_p$ *si y sólo si* $Sf_1(x_1,..., x_p)... f_q(x_1,..., x_p)$.

Obviamente se tiene:

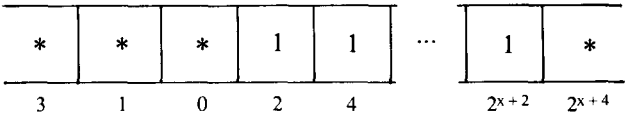
Si S *es decidible y* R *es reducible a* S, *entonces* R *es decidible. Así pues, de este resultado se sigue inmediatamente:*

Si R *es reducible a* S *y* R *es indecidible*, S *es también indecidible.*

f) *El «halting problem» de una máquina de Turing*

El problema de parada («halting problem») de una máquina de Turing consiste, en su caso más general, en decidir el predicado binario E_1 *tal que* E_1xk *se verifica si y sólo si* x *es el número de una máquina*, k *es el número de una posible configuración de la máquina* x *y, además, la máquina* x *termina parándose cuando parte de la configuración* k.

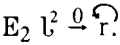
Un caso particular del «halting problem» consiste en decidir el predicado unitario E_2 tal que E_2x *se verifica si y sólo si* x *es el número de una máquina de Turing que termina parándose cuando se la coloca tras* x, *i.e., cuando parte de la configuración* (A, B, 0), *donde* A *y* B *vienen dados por el diagrama*



Probaremos que E_2 *es indecidible y que, al ser* E_2 *reducible a* E_1 *también lo es* E_1.

En efecto, supongamos que E_2 *es decidible, con lo que existirá una máquina de Turing* E_2 *que computa*

en forma normada su función característica. *Sea* k *el número de la máquina*



Claramente se tiene:

Si se verifica E_2k *entonces* E_2 *realiza la computación*

$* k * \Rightarrow * k * l *$ *(donde* $k = l\, l\, l_{k+1} \ldots l$)
con lo que M *no se para cuando se la coloca tras* k *y*, *en consecuencia*, *no se verifica* E_2k.

Por otra parte, *si no se verifica* $E_2\chi$ *entonces* E_2 *realiza la computación* $* k \underline{*} \Rightarrow * k * l\, l \underline{*}$, *con lo que* M *se para cuando se la coloca tras* k *y*, *en consecuencia*, *se verifica* E_2k.

Así pues, *la hipótesis de la decidibilidad de* E_2 *implica la contradicción*

$$E_2k \Leftrightarrow E_2k.$$

Para probar que E_2 *es reducible a* E_1 *basta tener en cuenta que el número de la configuración* (A, B, 0), *donde* A *y* B *vienen definidos a partir del diagrama* (*), *puede ser efectivamente calculada a partir de* x, *o*, *en otras palabras*, *es función computable de* x. *Denotando por* f *esa función*, *se tiene*

$$E_2x \leftrightarrow E_1xf(x),$$

con lo que, *de acuerdo con* e), E_1
es indecidible,

Mediante consideraciones completamente semejantes puede probarse sin dificultad la indecidibilidad del predicado E_3 *tal que* E_3x *se verifica si y sólo si* x *es el número de una máquina que termina parándose cuando comienza a actuar sobre una cinta vacía*.

§ 30. Máquinas universales de Turing

1. Definición. A una máquina de Turing U la llamamos *máquina universal de Turing* si U opera como sigue:

Sea B_0 una expresión de cinta arbitraria (naturalmente finita). Sea A_0 un cuadrado arbitrario de la cinta de cálculo. Sea **M** una máquina de Turing arbitraria. Colocamos a **M** en A_0 sobre B_0. De esta forma obtenemos la configuración inicial (A_0, B_0, C_0), donde $C_0 = c_{\mathbf{M}}$. Ahora bien, **M** recorrerá una secuencia (A_1, B_1, C_1), (A_2, B_2, C_2), ... (A_n, B_n, C_n), ... de configuraciones que, supuesto que **M** se detenga después de un número finito de pasos, concluirá eventualmente en una configuración terminal (An_0, Bn_0, Cn_0) (cf. la columna izquierda del esquema que se da al final de este apartado).

Colocamos la máquina U, que ha de simular **M**, sobre la expresión de cinta t^*k_0, donde $t = G(\mathbf{M})$ y $k_0 = g(A_0, B_0, C_0)$. Esto significa, más precisamente, que primero imprimimos en la cinta de cálculo, que en lo demás está vacía, $t + 1$ trazos para representar el número t (cf. 1.3), después de lo cual imprimimos, dejando un cuadrado vacío (representado por *), $k_0 + 1$ trazos para representar el número k_0, y luego seleccionamos el cuadrado subsiguiente al último cuadrado marcado como cuadrado originalmente escrutado de U. Sea C_0^{U} el estado inicial de U. Con lo cual queda determinada la configuración inicial de U, $(A_0^{\mathsf{U}}, B_0^{\mathsf{U}}, C_0^{\mathsf{U}})$.

Ahora suponemos que **M** realiza un paso que conduce desde la 0-ésima configuración (A_0, B_0, B_0) a la primera configuración (A_1, B_1, C_1). Entonces requeri-

mos que U realice al menos un paso y que U, después de un número finito r_1 de pasos, alcance una configuración (Ar_1^U, Br_1^U, Cr_1^U), en la cual el cuadrado escrutado es el cuadrado subsiguiente al último cuadrado marcado y la expresión de cinta está dada por $t*k_1$, donde $k_1 = g(A_1, B_1, C_1)$. Caracterizaremos este hecho brevemente diciendo que la r_1-ésima configuración de U *corresponde* a la primera configuración de **M**. (Obsérvese que, en esta terminología, la 0-ésima configuración de U corresponde a la 0-ésima configuración de **M**.)

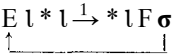
Si **M** efectúa ahora un paso ulterior que conduzca a la configuración (A_2, B_2, C_2), entonces deberá existir una configuración correspondiente (Ar_2^U, Br_2^U, Cr_2^U), de U con $r_2 > r_1$, etc. Finalmente, si **M** alcanza una configuración terminal (An_0, Bn_0, Cn_0) después de un número finito de pasos, entonces se requiere que U se detenga en una configuración terminal correspondiente $(Arn_0^U, Brn_0^U, Crn_0^U)$.

Si (A_0, B_0, C_0) es ya una configuración terminal, es decir, $n_0 = 0$, entonces deberá admitirse que $rn_0 = r_0 > 0$.

Máquina M	Máquina U (simulando M)
(A_0, B_0, C_0)	(A_0^U, B_0^U, C_0^U), con $B_0^U = t * k_0$ y $A_0^U =$ = cuadrado subsiguiente al último cuadrado marcado
(A_1, B_1, C_1)	(Ar_1^U, Br_1^U, Cr_1^U), con $Br_1^U = t * k_1$ y $Ar_1^U =$ = cuadrado subsiguiente al último cuadrado marcado
(A_2, B_2, C_2)	(Ar_2^U, Br_2^U, Cr_2^U), con $Br_2^U = t * k_2$ y $Ar_2^U =$ = cuadrado subsiguiente al último cuadrado marcado

El método de operación de una máquina universal U

2. Construcción de una máquina universal de Turing $\mathfrak{U}_0$. Es fácil presentar una máquina $\mathfrak{U}_0$, que realice lo requerido en el apartado 2. Si colocamos a $\mathfrak{U}_0$ tras $t * k_0$, lo primero que hemos de comprobar es si **M** no da ningún paso. Tal es el caso si y sólo si $E'tk_0$, esto es, si $e(t,k_0) = 0$. La construcción de $\mathfrak{U}_0$ se efectúa a partir de máquinas más simples, comenzando por la máquina E, que, de acuerdo con el apartado 1, computa normadamente *e*.



La máquina universal de Turing $\mathfrak{U}_0$

Después de la computación realizada por **E** tenemos sobre la cinta de cálculo la inscripción $t*k_0*e(t,k_0)$. Por medio de $\mathfrak{l} * \mathfrak{l}$ borramos el cuadrado inmediato a la izquierda y nos movemos un cuadrado más a la izquierda. Ahora, $e(t,k_0) = 0$ ó 1, según que el cuadrado escrutado después de este paso esté vacío o marcado, respectivamente.

Supongamos primero que **M** no efectúa ningún paso, es decir, $e(t,k_0) = 0$. En este caso la computación ha terminado.

Si, por otra parte, **M** efectúa un paso, entonces $e(t,k) = 1$. Ahora borramos el valor $e(t,k_0)$ y mediante $\xrightarrow{1}$ r * $\mathfrak{l}$ * $\mathfrak{l}$ volvemos al cuadrado subsiguiente al extremo derecho de la inscripción original. Nuestra siguiente misión es computar k_1 e imprimirlo en lugar de k_0. Tenemos que $k_1=F(t,k_0)$. De acuerdo con el apartado 1, *F* s computada normadamente por **F**. Después de la computación realizada por **F** tenemos sobre la cinta de cálculo la inscripción $t*k_0*k_1$.

Ahora debemos borrar k_0 y hacer retroceder k_1 hacia t. Ello se efectúa mediante la máquina de desplazamiento σ, que introdujimos en § 8.6.

De esta forma, después de un número finito r_1 de pasos hemos alcanzado una configuración de U_0 que corresponde a (A_1, B_1, C_1).

Ahora hemos de determinar si k_1 es el número de Gödel de una configuración terminal, etc. Ello puede hacerse sin ninguna dificultad ulterior por medio de una realimentación de **E**. Así, la construcción de una máquina universal de Turing está completa.

3. Consecuencias. Primero, para compendiar los resultados del apartado anterior, podemos caracterizar una propiedad esencial de la máquina universal de Turing U por el siguiente:

Teorema 1. Si U *es una máquina universal de Turing, entonces tenemos: una máquina de Turing arbitraria* **M** *con número de Gödel t, colocada en una expresión de cinta* B_0 *sobre el cuadrado* A_0 (que, junto con $C_0 = c_M$, determina una configuración con el número Gödel k_0), *se detiene después de un número finito de pasos si y sólo si* U, *colocada detrás de* $t * k_0$, *se detiene después de un número finito de pasos*.

Volvamos ahora nuestra atención al apartado f) de la introducción a este Anexo. Allí considerábamos la propiedad E_2, que se satisface mediante una máquina de Turing **M** si y sólo si **M**, colocada detrás de su número de Gödel $G(\mathbf{M})$, *se detiene después de un número finito de pasos. Y mostramos que* E_2 no es decidible.

Mostraremos ahora que para una máquina universal de Turing U no existe ningún algoritmo con cuya ayuda podamos decidir si U, colocada sobre una expresión arbitraria de cinta en un cuadrado arbitra-

rio, se detiene o no después de un número finito de pasos. Demostración por *reductio ad absurdum*. Supongamos que existe un algoritmo tal. Entonces podemos decidir, mediante la ayuda de este algoritmo, la propiedad indecidible E_2 como sigue. Sea **M** una máquina de Turing arbitraria. Sea $t = G(\mathbf{M})$. Imprimimos t sobre la cinta de cálculo, que en lo demás está vacía, y obtenemos de esta forma una expresión de cinta B_0. Sea A_0 el cuadrado que sigue a t. Entonces $k_0 = G(A_0, B_0, C_0)$ es el número de Gödel de la configuración inicial de **M**. Ahora colocamos U detrás de $t*k_0$. Entonces, de acuerdo con el teorema 1, la máquina U se detiene después de un número finito de pasos si y sólo si **M**, colocada detrás de t, se detiene después de un número finito de pasos, es decir, si y sólo si **M** tiene la propiedad E_2. De acuerdo con nuestra suposición podemos decidir si U se detiene después de un número finito de pasos y así podemos decidir si **M** tiene la propiedad E_2.

Sintetizamos el resultado en el siguiente:

Teorema 2. No es decidible si una máquina universal de Turing U, *colocada sobre una expresión de cinta arbitraria en un cuadrado arbitrario, se detiene o no después de un número finito de pasos.*

Para completar estas consideraciones terminaremos formulando el resultado principal del apartado 3.

Teorema 3. Puede aducirse explícitamente una máquina universal de Turing U_0.

REFERENCIAS

Turing, A. M.: *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proc. London math. Soc. (2) 42 (1937), 230-265.

CUADERNOS DE FILOSOFÍA Y ENSAYO

Director: MANUEL GARRIDO

José Luis L. Aranguren: *Propuestas morales*.
Y. Bar-Hillel y otros: *El pensamiento científico*.
Mario Bunge: *Controversias en física*.
Mario Bunge: *Economía y filosofía*.
J. N. Crossley y otros: *¿Qué es la lógica matemática?*
Charles Darwin: *Ensayo sobre el instinto*.
Javier Esquivel y otros: *La polémica del materialismo*
Andrew Feenberg: *Más allá de la supervivencia: el debate ecológico*.
Jurgen Habermas: *Sobre Nietzsche y otros ensayos*.
Jurgen Habermas: *Ciencia y técnica como «ideología»*.
Hans Hermes: *Introducción a la teoría de la computabilidad*
José Jiménez: *La estética como utopía antropológica*.
Ramiro Ledesma Ramos: *La filosofía, disciplina imperial*.
H. O. Mounce: *Introducción al «Tractatus» de Wittgenstein*.