UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE CIENCIAS

El teorema de Gödel a partir del teorema de Rice

T E S I S

Que para obtener el título de:

M a t e m á t i c o

P R E S E N T A:

Edgar Enrique Solís de los Reyes

Tutor:
Dr. Carlos Torres Alcaraz
2008

a mis padres, Honorato e Isabel,
por ustedes estoy aquí…

a mis hermanos, Haydee, Viridiana e Ivan

Índice

Introducción

Los teoremas de Gödel son un resultado fundamental y cuya importancia no se ha hecho notar sólo en la Lógica Matemática, sino en las Matemáticas en general. En ellos Gödel nos muestra un nuevo método de demostración – la aritmetización de la teoría – mediante el cual establece ciertas limitaciones para las teorías axiomáticas; estas limitaciones han tenido serias repercusiones tanto en la filosofía como en la inteligencia artificial.

Un rasgo común a los resultados de gran importancia en las matemáticas es que tienen muchas demostraciones. Por ejemplo, con relación al teorema de Pitagoras hay incluso libros dedicados a las distintas maneras de probarlo, o a los distintos ámbitos en que aparece. En este sentido el teorema de incompletud de Gödel no es la excepción. Este teorema se ha probado de distintas maneras por autores como Gödel, Turing, Boolos, Kleene y Enderton, por mencionar algunos de ellos. Al respecto, un común denominador en todos estos casos es que el procedimiento que se sigue se ha ideado específicamente para obtener el resultado[1]. En ello hay una diferencia con la demostración que aquí se ofrece, tiene la virtud de seguir un camino original. Se trata de una demostración en la que el eje principal lo constituye la teoría matemática de la computabilidad. Al respecto, la teoría se desarrolla en sus propios términos y sin hacer alusión alguna al teorema de incompletud. Esto se hace hasta alcanzar ciertos resultados, como lo son el teorema de recursión, el del punto fijo y el teorema de Rice, que tienen una vida propia, es decir, que son relevantes para las ciencias de la computación por sí mismos y que no fueron pensados para atacar problemas de la incompletabilidad de la aritmética. Con esto esperamos tender un puente entre la teoría matemática de la computación, el estudio axiomático de la aritmética y los sistemas formales.

[1]Otra línea de acción es la utilizada por Paris y Hwuvington, quienes prueban la indecibilidad del teorema de Ramsey en la aritmética de Peano de primer orden. No obstante, en este caso la prueba no incluye la conclusión de que la aritmética de los números naturales es esencialmente incompletable.

El teorema de Rice tiene una enorme importancia en la teoría matemática de la computabilidad. Una lectura del teorema es la siguiente: *cualquier propiedad sobre Maquinas de Turing que se pueda caracterizar recursivamente es trivial*. Al compararlo con el teorema de Gödel, que dice: *en la aritmética es imposible caracterizar recursivamente la propiedad "ser teorema"*; podemos ver que hay una clara similitud entre ellos, pues la propiedad "*ser teorema"* no es trivial. Con base en esta idea se proyectó este trabajo, el cual consta de cinco capítulos

Capítulo 1: En este espacio se introduce el concepto de Maquina de Turing, tanto intuitiva como formalmente, y se repasan las diferentes formas de representarlas. El capítulo termina con la introducción de la Maquina Universal de Turing.

Capítulo 2: En este lugar se desarrolla el concepto de función parcial recursiva, una herramienta básica para lo que después se hará. El capítulo concluye con la demostración de un resultado que permite pasar de las funciones recursivas a Maquinas de Turing: *toda función parcial recursiva es Turing Calculable*.

Capítulo 3: En este apartado se trabajan los conjuntos recursivos y recursivamente enumerables, y se dan algunas caracterizaciones de ellos; *v. gr.*: *un subconjunto de* $\mathbb{N}$ *es recursivamente enumerable si y solo si es el dominio de una función parcial recursiva en* $\mathbb{N}$. Estos conceptos son la base de gran parte del trabajo subsiguiente como, por ejemplo, la no recursividad del conjunto $\mathcal{K} = \{n \in \mathbb{N} | \, n \in dom(\varphi_n)\}$. Asimismo, desarrollamos la aritmetización de las Maquinas de Turing, con lo que se concluye este capítulo.

Capítulo 4: En este lugar se demuestran dos teoremas centrales de la teoría de la computabilidad: *el teorema s-m-n de Kleene y el teorema de Rice;* el primero permite demostrar varios teoremas de gran importancia en esta teoría, como los teoremas de recursión y del punto fijo. El capítulo concluye con un resultado central de nuestro trabajo: el teorema de Rice.

Capítulo 5: Se expone la no recursividad de ciertos conjuntos, entre los que se halla el conjunto de índices de máquinas equivalentes a una máquina dada. Esto nos llevará a la meta prevista: la demostración del primer teorema de Gödel a partir del teorema de Rice; con lo que se concluye el trabajo.

Capítulo I

Máquinas de Turing

1.1 Idea intuitiva.

En la actualidad, los humanos interactuamos cotidianamente con lo que podemos llamar máquinas de Turing "materializadas"; las cuales no son otras que las modernas computadoras digitales. Estos dispositivos se pueden explicar cómo máquinas cuya función es realizar cómputos con expresiones simbólicas. Similarmente a las computadoras que trabajan con instrucciones determinadas (el programa), las máquinas de Turing cuentan con una unidad de control que contiene las instrucciones a seguir (el programa). Hay muchas presentaciones de las máquinas de Turing que guardan diferencias entre sí. No obstante, desde el punto de vista de la teoría matemática de la computación, todas ellas caracterizan una misma clase de funciones computables. La siguiente presentación la hemos elegido por conveniencia. Una máquina de Turing tiene una cinta infinita en un sentido, dividida en celdas iguales. Estas celdas son utilizadas como memoria y como espacios en los que realiza los cómputos. Para ello cuenta con una cabeza lectora/escritora que corre sobre la cinta.

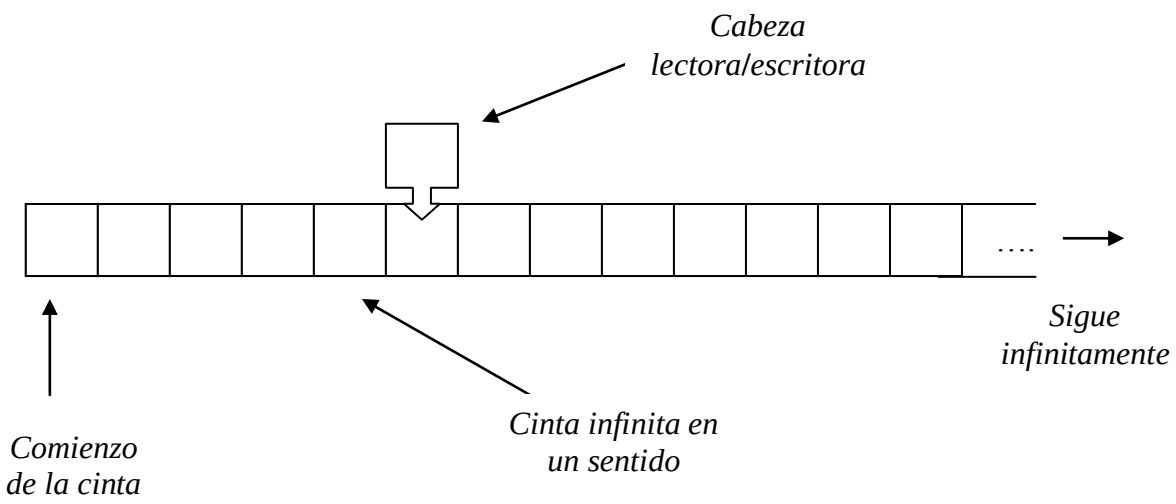


Figura 1.1. Diagrama de la cinta de una máquina y la cabeza lectora/escritora

La cabeza lectora/escritora inspecciona una celda a la vez, y está conectada a un dispositivo de control, el cual, como ya lo habíamos señalado, dirige las acciones que ésta puede realizar. Las tareas básicas son muy sencillas:

- Leer el contenido de la celda inspeccionada, la cual puede estar en blanco o contener un sólo símbolo tomado de un alfabeto finito predeterminado.
- Borrar el contenido de la celda.
- Escribir un símbolo en la celda.
- Moverse una celda a la izquierda sobre la cinta.
- Moverse una celda a la derecha sobre la cinta.

Con estas sencillas operaciones es mucho lo que en teoría la máquina puede hacer, siendo la infinitud de la cinta un indicativo de que la máquina posee una capacidad de almacenamiento ilimitado.

Internamente las computadoras actuales sólo hacen cálculos con unos y ceros, los cuales corresponden a los símbolos de su alfabeto. Con ellos realizan toda la gama de tareas que se les asigna. Hacia el exterior, tienen un lenguaje distinto para comunicarse con el usuario. De igual manera, las máquinas de Turing tienen un alfabeto de cinta con el que trabajan internamente; este alfabeto tiene un símbolo especial, "$\mathcal{B}$", para denotar que una celda está en blanco. Además, la máquina cuenta con un alfabeto de entrada/salida; el cual es un subconjunto del alfabeto de cinta.

Así como en ocasiones las computadoras se pasman por diversas circunstancias sin haber terminado ninguna tarea, las máquinas de Turing pueden llegar a una situación en la que jamás se detendrán. En tal caso decimos que la máquina no realiza un cómputo.

Básicamente, la diferencia entre una computadora moderna y una máquina de Turing es la cinta infinita. Esto hace de las máquinas de Turing objetos abstractos.

1.2 Descripción Formal.

Definición 1.2.1. Una función $f: A \longrightarrow B$ es una *función total,* cuando su dominio es todo A; en cambio, si el dominio es sólo un subconjunto propio de A, decimos que la función es *parcial con relación a* $\boldsymbol{A}$.

Definición 1.2.2. Un *alfabeto* X, es un conjunto finito no vacío de símbolos. Con X^* denotamos al conjunto de todas las cadenas finitas formadas con elementos de X y el símbolo "$\mathcal{B}$"; además, en X^* incluimos la cadena vacía ε. Un *lenguaje* sobre el alfabeto X, es un subconjunto de X^*. Por ejemplo, el español es un lenguaje sobre el alfabeto $\{a, b, c, d, e, \ldots, z\}$.

X^* se conoce como estrella de Klenne de X.

Definición 1.2.3. Una maquina de Turing es una sexteta $\mathcal{M} = (X, Y, Q, \delta, q_0, q_f)$ donde:

- X es el alfabeto de entrada/salida, Y el alfabeto de cinta y $X \subset Y$.
- Q es un conjunto finito de *estados*.
- $\delta: Q \times Y \longrightarrow Q \times Y \times \{I, D, \Lambda\}$, es una función parcial denominada *de transición.*
- $q_0 \in Q$ es un estado especial denominado *inicial.*
- $q_f \in Q$ es un estado especial denominado *final o de detención.*

La función de transición es el programa de la máquina y corresponde a lo que antes llamamos “dispositivo de control”. Los símbolos I, D, Λ los interpretamos como “mover la cabeza lectora/escritora una celda a la izquierda”, “una celda a la derecha” y “no moverla”, respectivamente.

La función de transición no está definida para el estado q_f, es decir, $\delta(q_f, x)$ no ésta definida para ninguna x. Esto se debe a que q_f corresponde a la detención de la máquina, la cual ya no realiza ninguna operación adicional. Asimismo, si la cabeza lectora/escritora se encuentra en la celda inicial de la cinta, y la

función de transición da la instrucción de moverse a la izquierda, entonces la máquina detendrá su operación.

Definición 1.2.4. Sea M una máquina de Turing, y sean $u, v \in X^*$. Decimos que M realiza un cómputo con entrada o argumento u y salida v, o similarmente, M computa v con entrada o argumento u si y sólo si:

- La máquina inicia
  - Con u escrito al inicio de la cinta, estando el resto de las celdas en blanco;
  - La cabeza lectora/escritora se encuentra en la celda inicial de la cinta en el estado q_0; y
- La máquina termina
  - Con v escrito al inicio de la cinta, estando el resto de las celdas en blanco;
  - La cabeza lectora/escritora se encuentra en la celda inicial de la cinta en el estado q_f.

Cuando M realiza un cómputo en las condiciones descritas, escribimos $M(u) = v$. En caso contrario decimos que M no realizó un cómputo.

Si la máquina de Turing M computa v con la entrada u después de n aplicaciones de la función de transición, decimos que el cómputo se realiza *en* n *pasos,* o que éste consta de n *instancias de operación.*

Como ejemplo construimos una máquina de Turing M_S, que computa el sucesor de cualquier número natural.

Ejemplo 1.2.1. Consideremos un alfabeto de entrada/salida $X = \{|\}$, un alfabeto de cinta $Y = \{|, \mathcal{B}\}$ y el conjunto de estados $Q = \{q_0, q_1, q_2, q_f\}$. Definimos la máquina de Turing M_S mediante la función de transición δ como sigue:

$$\delta(q_0, |) = (q_1, \mathcal{B}, D) \qquad \delta(q_2, |) = (q_2, |, I)$$
$$\delta(q_1, |) = (q_1, |, D) \qquad \delta(q_2, \mathcal{B}) = (q_f, |, \Lambda)$$
$$\delta(q_1, \mathcal{B}) = (q_2, |, I) \qquad \text{Indefinida en otros casos}$$

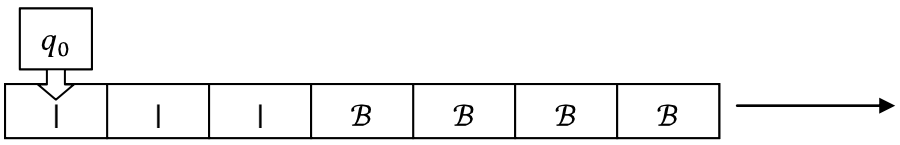
En el ejemplo nos servimos de la siguiente representación de los números naturales, la cual utilizaremos en lo sucesivo:

Número	Representación
0	|
1	||
2	|||
3	||||
4	|||||
.	.
.	.
.	.

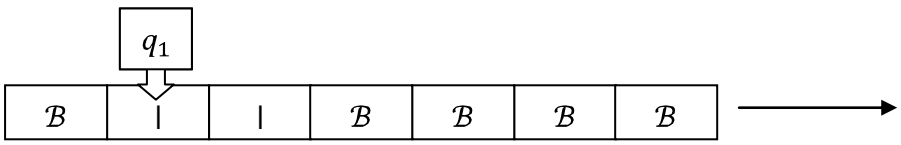
La entrada para la máquina es: escribir, con la representación anterior, un número natural a partir de la primera celda de la cinta, dejando el resto de celdas en blanco.

Probemos la máquina, con el número dos.

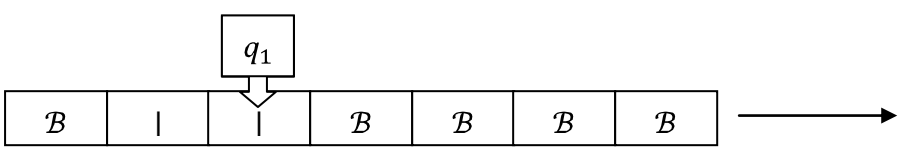
Inicia así:



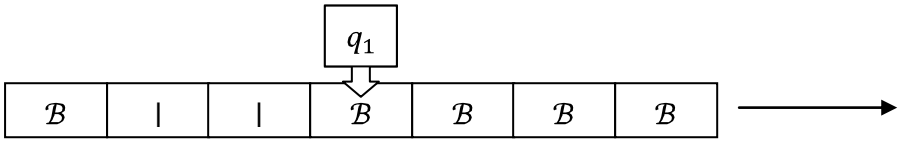
1^{er} paso:



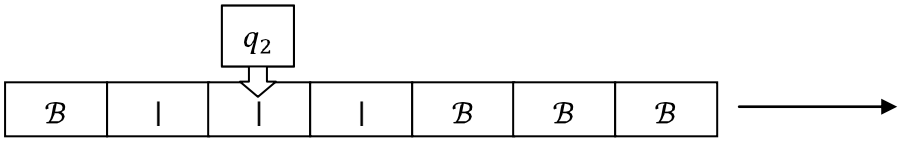
2^{do} paso:



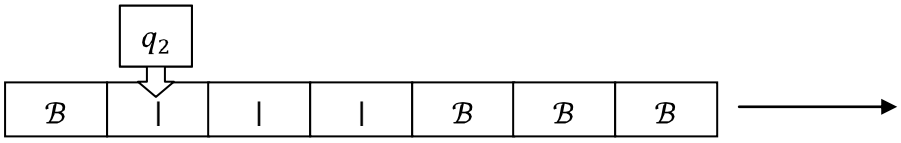
3^{er} paso:



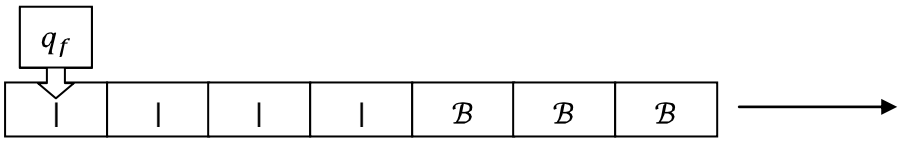
4^{to} paso:



5^{to} paso:

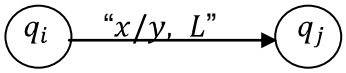


6^{to} paso:



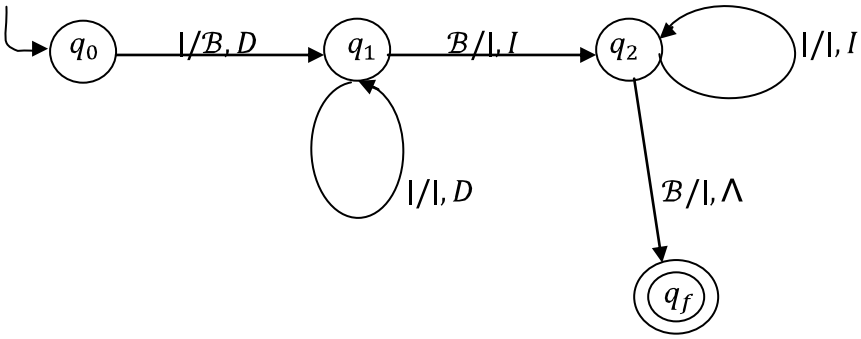
Conforme a la función de transición, M_S se detiene en el paso seis. Conforme a la definición, M_S realizó un cómputo: el del número |||| (tres) con la entrada ||| (dos). Escribimos $M(|||) =$||||.

Otra forma de representar una máquina de Turing es con un *diagrama de estados*. En esta representación encerramos los estados en círculos. Para indicar el inicio, en el estado inicial ponemos una flecha curva, y para indicar que termina encerramos en un doble círculo el estado final. Asimismo, unimos los estados con flechas sobre las cuales escribimos algo como esto: "$x/y,\ L$". Por ejemplo si el estado de donde parte la flecha es q_i, y el estado a donde llega es q_j, entonces con "$x/y,\ L$" indicamos que la cabeza lectora/escritora cambia el símbolo "x" por el símbolo "y", realiza el movimiento "L" y pasa al estado q_j:



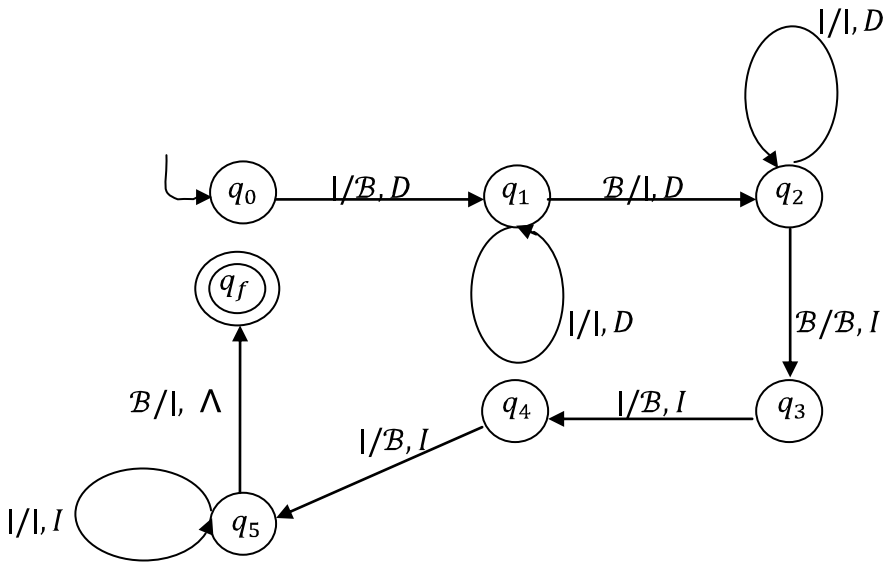
Este diagrama expresa lo mismo que la igualdad $\delta(q_i, x) = (q_j, y, L)$.

Veamos el diagrama de la máquina del ejemplo 1.2.1.

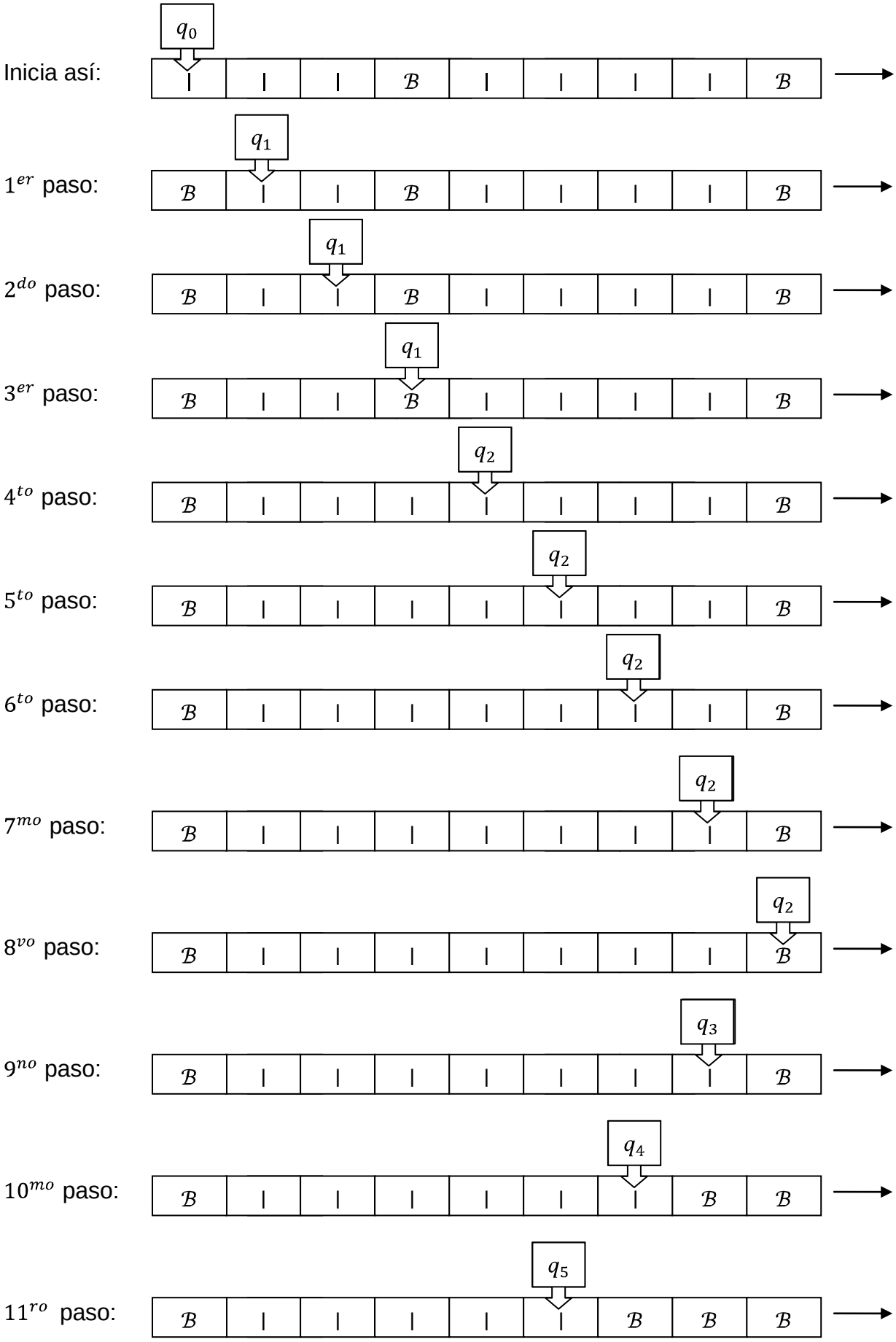


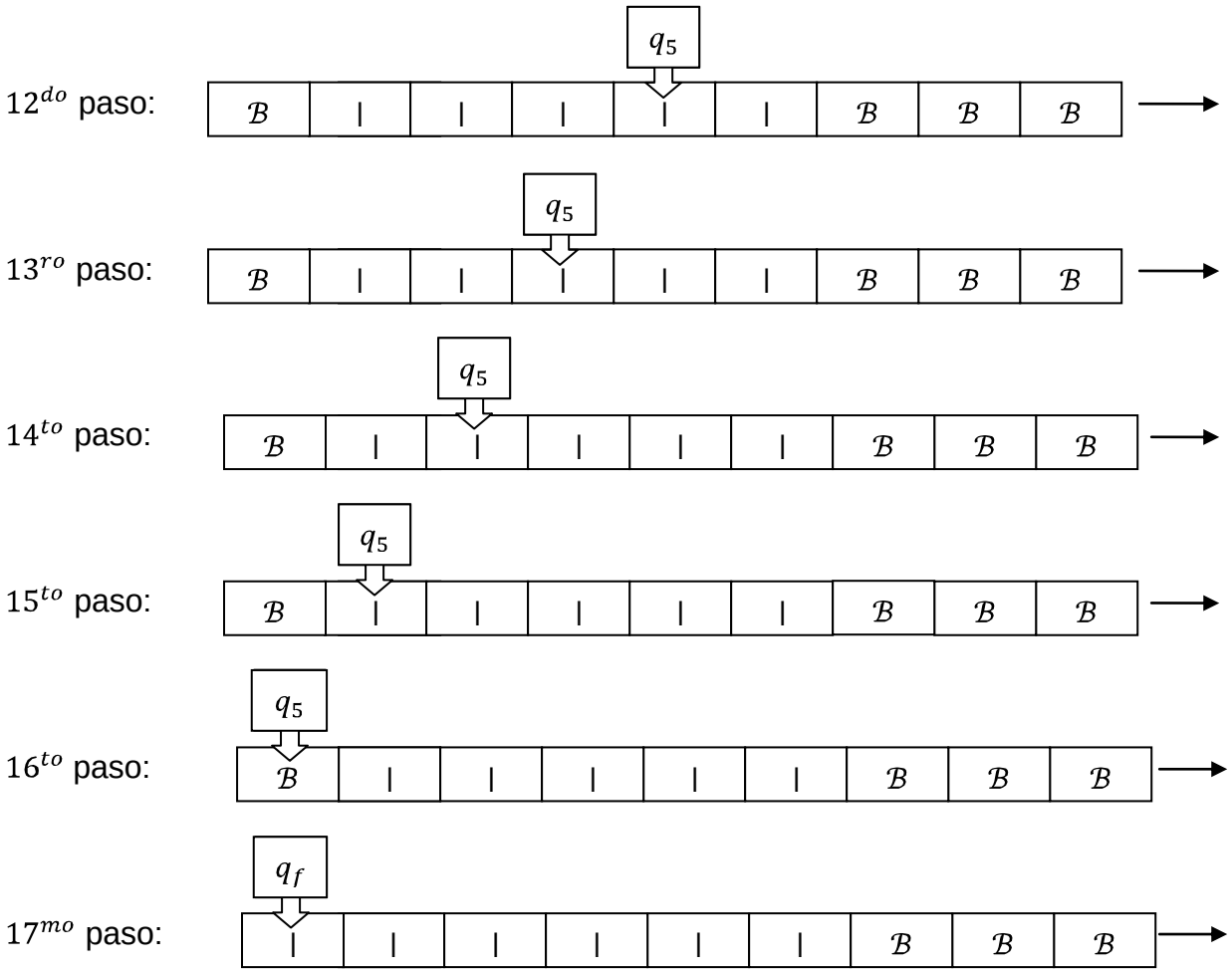
Construyamos ahora una máquina de Turing M_+, que sume parejas de números naturales. Para ello, utilizamos la misma representación del ejemplo 1.2.1. Como siempre, escribimos la entrada al inicio de la cinta, separando los sumandos con una casilla en blanco, y dejando el resto de las celdas en blanco.

Ejemplo 1.2.2. Consideremos el alfabeto de entrada/salida $X = \{|\}$ y alfabeto de cinta $Y = \{|, \mathcal{B}\}$, y el conjunto de estado $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$. En este caso especificamos la Máquina de Turing M_+ con un diagrama de estados:



Realicemos la suma de 2+3 con ésta máquina:


Inicia así:
q_0
$\mathcal{B}$
1^{er} paso:
q_1
2^{do} paso:
3^{er} paso:
4^{to} paso:
q_2
5^{to} paso:
6^{to} paso:
7^{mo} paso:
8^{vo} paso:
q_3
9^{no} paso:
q_4
10^{mo} paso:
q_5
11^{ro} paso:



Conforme al diagrama de estados, M_+ se detiene en el paso decimo séptimo. En este caso M_+ realizo un cómputo: el del número |||||| (cinco) con la entrada $|||\mathcal{B}||||$ (dos, tres).

Hasta ahora sólo hemos trabajado con máquinas de Turing cuyo alfabeto de entrada/salida tiene un solo elemento, pero se puede pensar en máquinas con alfabetos más amplios. ¿Se podrían realizar más cómputos de esa manera?, es decir, mientras más símbolos tenga un alfabeto, ¿será mayor la cantidad de cálculos que podrán realizar las máquinas con tales alfabetos? De hecho, la situación es la opuesta. Todo lo que se hace con una máquina con un alfabeto de n símbolos, se puede realizar con una máquina con un alfabeto de un solo símbolo, mediante una "*correspondencia"* o *"identificación"* entre las palabras de sus alfabetos. Por ejemplo, si M es una máquina de Turing con alfabeto

$X_2 = \{a, b\}$ de entrada/salida, y M' una máquina con alfabeto $X = \{|\}$ de entrada/salida, podemos representar cada símbolo de X_2 de la siguiente manera: $a \rightsquigarrow |$ y $b \rightsquigarrow ||$. Con esta representación de los símbolos, separando cada una de éstas representaciones con una casilla en blanco, podemos identificar cada palabra del lenguaje X_2* con palabras del lenguaje X*; por ejemplo $aab \in X_2$* se identifica con $1\mathcal{B}1\mathcal{B}11 \in X$*.

La posibilidad de hacer corresponder las palabras de dos alfabetos nos permite la siguiente definición.

Definición 1.2.5. Sean M y M' dos máquinas de Turing, y X y X' sus respectivos alfabetos de entrada/salida. Denotemos con $u', v' \in X'$* las palabras correspondientes a $u, v \in X$* mediante una identificación. Decimos que M y M' son *equivalentes* si y sólo si cada vez que $M(u) = v$, entonces $M'(u') = v'$, y viceversa.

Con la correspondencia entre alfabetos, que nos da la equivalencia de máquinas, podemos responder formalmente la pregunta hecha anteriormente: ¿se pueden realizar más cómputos con alfabetos de más símbolos? La respuesta está en el siguiente teorema, el cual enunciamos sin demostración. El lector interesado puede referirse a*: Hermes, 1969. pp 94 - 97.*

Teorema 1.2.1. Si M es una máquina de Turing con alfabeto de entrada/salida X de n símbolos y alfabeto de cinta $Y = X \cup \{\mathcal{B}\}$, entonces existe una máquina de Turing M' con alfabeto de entrada/salida $X' = \{|\}$ y alfabeto de cinta $Y' = \{|, \mathcal{B}\}$ equivalente a M. ■

A las máquinas de Turing que tienen alfabeto de entrada/salida $X = \{|\}$ y alfabeto de cinta $Y = \{|, \mathcal{B}\}$ les llamamos *simples o unitarias*. En adelante es importante recordar que para los números naturales usaremos la representación del ejemplo 1.2.1.

Como hemos visto, con $M(u) = v$ indicamos que la máquina de Turing M computa v con la entrada u. Si M no realiza un cómputo con la entrada u, entonces $M(u)$ no está definido. Una segunda notación es la siguiente: Si M computa v con la entrada u, escribimos $M(u)\downarrow$ y decimos que M *converge* en u, mientras que en el otro caso escribimos $M(u)\uparrow$ y decimos que M *diverge* en u.

De manera análoga a la anterior, podemos construir las siguientes máquinas:

1.- Una máquina de Turing que calcule la diferencia positiva dos números naturales x e y (i.e., $x \dot{-} y$)[1].
2.- Una máquina de Turing que multiplica dos números naturales.
3.- Una máquina de Turing que eleva un número natural al cuadrado.
4.- Una máquina copiadora, que al recibir de entrada varios bloques de trazos separados entre sí por una casilla en blanco, da como salida la misma sucesión de entrada escrita dos veces, separadas por una casilla en blanco. Por ejemplo; si la entrada es: $|||\mathcal{B}|||$, la salida es: $|||\mathcal{B}\,|||\,\mathcal{B}\,|||\,\mathcal{B}|||$.

1.3 Máquina Universal de Turing.

Hemos dicho que las computadoras digitales modernas son en realidad máquinas de Turing con ciertas limitaciones materiales (por ejemplo, la extensión de la memoria). Sin embargo, al hablar de máquinas de Turing no hemos hecho referencia a una cuestión cotidiana con relación a las computadoras digitales: que éstas son programables.

En términos técnicos, esto corresponde a lo que conocemos como *máquinas universales de Turing,* es decir, máquinas que pueden imitar el comportamiento de otras máquinas.

[1] Por definición $x \dot{-} y = \begin{cases} x - y & si\ x \geq y \\ 0 & si\ y > x \end{cases}$

Imaginemos, por ejemplo, una máquina que sólo es capaz de realizar un tipo de cómputo (como lo hace una calculadora de bolsillo o un controlador digital de la temperatura en un horno). En estos casos se trata de dispositivos que sólo son capaces de ejecutar una rutina específica, modificable si acaso con ciertos parámetros. *Grosso modo,* a esto corresponde la idea que hemos dado de las máquinas de Turing. No obstante, hay máquinas de Turing que en cierto sentido son programables. La idea es la siguiente. Mediante una técnica que explicaremos en el capítulo III, las máquinas de Turing se pueden numerar o codificar en números. Dicho número encierra la descripción de la función de transición que es, de hecho, lo que define a la máquina. Pues bien, en un trabajo publicado en 1936, Turing, véase Turing, 1936, demostró la existencia de máquinas llamadas *"universales"* que pueden imitar el comportamiento de una máquina M a través de su código, cuando ésta se aplica a cualquier sucesión de argumentos $x_1, x_2, \dots, x_n$. Esto corresponde a la idea de "programar a $\mathcal{U}$" (la máquina universal de Turing). Digamos que el *índice* i de una máquina M, actúa como un programa, es decir, como un conjunto de instrucciones para que $\mathcal{U}$ imite a M con la entrada $x_1, x_2, \dots, x_n$. Para $\mathcal{U}$, el índice i de M es un parámetro adicional, por lo que escribimos $\mathcal{U}(i, x_1, x_2, \dots, x_n) = M(x_1, x_2, \dots, x_n)$.

Por ejemplo, si M_+ es la máquina del ejemplo 1.2.2, la máquina sumadora, y si ésta tuviera índice 500, entonces tendríamos lo siguiente: $\mathcal{U}(500, 3, 4) = M_S(3, 4) = 7$.

En este sentido, las computadoras modernas son como máquinas universales de Turing, en las cuales se cargan diferentes programas para realizar diversas tareas, en lugar de construir una computadora para cada una de ellas. No obstante, y a diferencia de las computadoras digitales, la máquina universal de Turing tiene memoria infinita.

Con esto concluimos la descripción de máquinas de Turing, para seguir en el siguiente capítulo con la relación de nuestro interés entre éstas y las funciones parciales recursivas: *toda función parcial recursiva es Turing computable y viceversa*.

Capítulo II

Funciones Aritméticas Parciales Turing Computables

Ahora describiremos las funciones parciales recursivas y la relación de nuestro interés con las máquinas de Turing: toda función parcial recursiva es Turing computable. Para ello, en lo que sigue, denotamos la función proyección de n argumentos con: $P_k^n(x_1, x_2, \dots, x_n) = x_k$, donde $1 \leq k \leq n$.

Definición 2.1. Una función parcial $\varphi: S \subset \mathbb{N}^n \to \mathbb{N}$ es Turing computable, si y sólo si existe una máquina de Turing unitaria que la computa, es decir, existe una máquina de Turing M con alfabeto de cinta $Y = \{|, \mathcal{B}\}$ y alfabeto de entrada/salida $X = \{|\}$ tal que, para todo $s \in S \subset X^*$, $M(s) = \varphi(s)$, mientras que en el caso contrario, es decir, cuando $s \notin S$, $M(s)$ diverge.

Para las funciones $\varphi: \mathbb{N}^n \to \mathbb{N}^m$ tenemos lo siguiente: φ es Turing computable si y sólo si las funciones $P_k^m \circ \varphi: \mathbb{N}^n \to \mathbb{N}$ son Turing computables, para todo $k \in \{1, 2, \dots, m\}$

En virtud del teorema 1.2.1, esta definición es general, pues lo que ahí se demuestra es que, en realidad, la computabilidad no depende de la notación utilizada.

Por ejemplo, la función suma $+: \mathbb{N}^2 \to \mathbb{N}$ definida por $+(n, m) = n + m$ es Turing computable, pues la máquina de Turing unitaria M_S del ejemplo 1.2.2 la computa.

Definición 2.2. Una función f es *inicial* si y sólo si es alguna de las siguientes:

- La función cero $\mathbf{0}: \mathbb{N} \to \mathbb{N}$, definida por $\mathbf{0}(n) = 0$
- La función sucesor $s: \mathbb{N} \to \mathbb{N}$, definida por $s(n) = n + 1$
- La función proyección $P_k^n: \mathbb{N}^n \to \mathbb{N}$, definida por $P_k^n(x_1, x_2, \dots, x_n) = x_k$

Un resultado que aquí no probaremos es el siguiente. El lector interesado puede referirse a Mendelson,1997 pp 320.

Teorema 2.1. Si f es una función inicial, entonces es Turing computable. ■

Definición 2.3. Sean $\psi: \mathbb{N}^n \to \mathbb{N}$ y $\theta: \mathbb{N}^{n+2} \to \mathbb{N}$ funciones parciales, y φ la función parcial $\varphi: \mathbb{N}^{n+1} \to \mathbb{N}$ definida por las ecuaciones

$$\varphi(x_1, x_2, \dots, x_n, 0) = \psi(x_1, x_2, \dots, x_n)$$

y

$$\varphi(x_1, x_2, \dots, x_n, n+1) = \theta(x_1, x_2, \dots, x_n, n, \varphi(x_1, x_2, \dots, x_n, n))$$

En tal caso decimos que φ ésta definida por *recursión* a partir de ψ y θ.

Definición 2.4. El conjunto $\mathcal{P}$ de las funciones recursivas primitivas sobre $\mathbb{N}$ se define inductivamente como sigue:

- $\mathcal{P}$ contiene a las funciones iniciales.
- Para $m, n \geq 1$, si las funciones $g: \mathbb{N}^m \to \mathbb{N}$ y $h_k: \mathbb{N}^n \to \mathbb{N}$ con $k \in \{1, 2, \dots, m\}$, pertenecen a $\mathcal{P}$, entonces la función composición $g \circ (h_1, h_2, \dots, h_m): \mathbb{N}^n \to \mathbb{N}$ pertenece a $\mathcal{P}$.[1]
- Si las funciones $g: \mathbb{N}^{n-1} \to \mathbb{N}$ y $h: \mathbb{N}^{n+1} \to \mathbb{N}$ pertenecen a $\mathcal{P}$, entonces la función $f: \mathbb{N}^n \to \mathbb{N}$ definida a partir de g y h por recursión pertenece a $\mathcal{P}$.

[1] $g \circ (h_1, h_2, \dots, h_m)(\bar{x}) = g\big(h_1(\bar{x}), h_2(\bar{x}), \dots, h_m(\bar{x})\big)$.

- $\mathcal{P}$ es el menor conjunto con las tres propiedades anteriores.

Ejemplo 2.1. La función suma $+: \mathbb{N}^2 \to \mathbb{N}$ es recursiva primitiva; pues se obtiene por recursión de las funciones $P_1^1: \mathbb{N} \to \mathbb{N}$ y $g: \mathbb{N}^3 \to \mathbb{N}$, donde $g(n_1, n_2, n_3) = s \circ P_3^3(n_1, n_2, n_3)$. Su definición es:

$$+(m, 0) = P_1^1(m)$$
$$+(m, n+1) = g(m, n, +(m, n))$$

Definición 2.5. Un elemento $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$ es *admisible para minimalización* relativa a la función $\phi: \mathbb{N}^{n+1} \to \mathbb{N}$ si el siguiente conjunto es no vacío:

$\mathcal{D}(x_1, x_2, \ldots, x_n) = \{m \in \mathbb{N} \mid \phi(x_1, x_2, \ldots, x_n, m) = 0 \; y \; para \; todo \; 0 \leq i \leq m \; (x_1, x_2, \ldots, x_n, i) \in Dom \; \phi\}$.

La razón por la cual se pide que la función esté definida para cada $i \leq m$, es que la máquina de Turing realizará un proceso algorítmico para hallar el mínimo cero, lo cual exige buscar tal número sucesivamente a partir del cero. Al respecto, si hubiera algún elemento menor al primer cero para el cual la función no estuviera definida, al aplicar el procedimiento señalado podría no detenerse, es decir, divergir en el proceso de cómputo. Esto se verá en la demostración del teorema 2.1.

Definición 2.6. Sea $\phi(x_1, x_2, \ldots, x_n, k)$ una función aritmética. Decimos que la función parcial $\varphi: \mathbb{N}^n \to \mathbb{N}$ se obtiene *de ϕ por minimalización* si y sólo si:

i. Dom φ es el conjunto de $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$ que son admisibles para minimalización relativa a ϕ, y
ii. $\varphi(x_1, x_2, \ldots, x_n) = min \; \mathcal{D}(x_1, x_2, \ldots, x_n)$, para todo $(x_1, x_2, \ldots, x_n) \in$ Dom φ

En éste caso escribimos:

$$\varphi(x_1, x_2, \dots, x_n) = \min k\,[\phi(x_1, x_2, \dots, x_n, k) = 0]$$

Definición 2.7. El conjunto $\mathcal{R}$ de funciones parciales recursivas sobre $\mathbb{N}$ se define inductivamente como sigue:

- $\mathcal{R}$ contiene a $\mathcal{P}$.
- Si la función parcial $\psi\colon \mathbb{N}^{n+1} \to \mathbb{N}$ pertenece a $\mathcal{R}$, entonces la función parcial $\varphi\colon \mathbb{N}^n \to \mathbb{N}$ obtenida de ψ por minimalización pertenece a $\mathcal{R}$
- $\mathcal{R}$ es el menor conjunto con las dos propiedades anteriores.

Teorema 2.2 Toda función parcial recursiva es Turing computable.

Ilustraremos la demostración con los casos de las funciones obtenidas por recursión y minimalización. Para ello, representaremos $n+1$ trazos de "|" con $\bar{n}$.

- Supongamos que las funciones $\psi\colon \mathbb{N}^n \to \mathbb{N}$ y $\theta\colon \mathbb{N}^{n+2} \to \mathbb{N}$ pertenecen a $\mathcal{R}$ y que son Turing computables. Bosquejamos la prueba de que la función obtenida por recursión a partir de ψ y θ es Turing computable.

  Por hipótesis, existen M_ψ y M_θ maquinas de Turing unitarias que computan ψ y θ respectivamente. Construyamos una máquina de Turing unitaria M que computa la función $\varphi\colon \mathbb{N}^{n+1} \to \mathbb{N}$ obtenida por recursión de ψ y θ. Consideremos una entrada consistente en $n+1$ bloques de "|", separados cada uno por una casilla en blanco. Movemos la cabeza lectora/escritora a la derecha hasta el inicio del $n+1$ bloque. *V. gr.* con la entrada $\bar{2}\mathcal{B}\bar{3}\mathcal{B}\bar{m}$ el resultado sería $\bar{2}\mathcal{B}\bar{3}\mathcal{B}[\bar{m}]$, donde los corchetes indican la ubicación de la cabeza lectora/escritora en el primer trazo de m ($|||\,\mathcal{B}\,||||\,\mathcal{B}[|]\,|||\cdots\mathcal{B}$). Ahora movemos la cabeza lectora/escritora una celda a la derecha. Si encontramos un blanco, entonces dejamos el blanco, movemos la cabeza lectora/escritora una celda a la izquierda, borramos el "|" y ponemos un

blanco, regresamos la cabeza lectora/escritora al inicio de la cinta, y activamos un módulo que actúa como la máquina M_ψ, obtenemos su cómputo y detenemos la máquina. El resultado es $\left[(M_\psi)\right]$, donde (M_ψ) representa el cómputo de M_ψ. En caso contrario, si encontramos un "|", seguimos moviendo la cabeza lectora/escritora a la derecha hasta que encontremos un blanco, entonces dejamos el blanco y regresamos la cabeza lectora/escritora una celda a la izquierda, borramos el "|" y ponemos un blanco, luego regresamos la cabeza lectora/escritora al inicio de la cinta y activamos un segundo modulo que copia y detiene la cabeza lectora/escritora al inicio de la copia que realizo:

$$\bar{2}\mathcal{B}\bar{3}\mathcal{B}\overline{(m-1)}\mathcal{B}[\bar{2}]\mathcal{B}\bar{3}\mathcal{B}\overline{(m-1)}.$$

Ahora activamos un tercer modulo que actúa como la máquina M, y obtenemos su cómputo: $\bar{2}\mathcal{B}\bar{3}\mathcal{B}\overline{(m-1)}\mathcal{B}[(M)]$. Por último regresamos la cabeza lectora/escritora al inicio de la cinta y activamos un cuarto módulo que actúa como la máquina M_θ, obtenemos su cómputo y detenemos la máquina: $[(M_\theta)]$. Así, M computa la función φ obtenida por recursión.

- Supongamos que la función parcial $\psi\colon \mathbb{N}^{n+1} \to \mathbb{N}$ pertenece a $\mathcal{R}$, y que es Turing computable. Bosquejamos la prueba de que la función obtenida de ψ por minimalización es Turing computable.

  Por hipótesis, existe M_ψ máquina de Turing unitaria que computa ψ. La máquina de Turing unitaria M que computa la función $\varphi\colon \mathbb{N}^n \to \mathbb{N}$. Consideremos una entrada consistente en n bloques de "|", separados cada uno por una casilla en blanco. Movamos la cabeza lectora/escritora al final de la n-ada de entrada, y enseguida movamos la cabeza lectora/escritora una celda a la derecha, en esta celda habrá un blanco pues es el final de la n-ada, dejamos el blanco y nos movemos a la derecha, de nuevo hay un

blanco, entonces escribimos un "|" y regresamos la cabeza lectora/escritora al inicio de la cinta. *V. gr.* si la entrada es $[\overline{3}]\mathcal{B}\overline{4}$, entonces la salida es $[\overline{3}]\mathcal{B}\overline{4}\mathcal{B}$ |. A continuación activamos un módulo que copia y se detiene en la primera celda de la copia que hizo: $\overline{3}\mathcal{B}\overline{4}\mathcal{B}$ | $\mathcal{B}[\overline{3}]\mathcal{B}\overline{4}\mathcal{B}$ |; después activamos la máquina M_ψ que computa: $\overline{3}\mathcal{B}\overline{4}\mathcal{B}$ | $\mathcal{B}\big[(M_\psi)\big]$. Si esté es 0, lo borramos, vamos al inicio de la cinta y borramos la n-ada de entrada: $\mathcal{B}\mathcal{B}\mathcal{B}$ | $\mathcal{B}$. Recorremos el trazo "|" al inicio de la cinta y detenemos la máquina: [|]. Si el cómputo es distinto de 0, lo borramos, vamos al inicio de la cinta y volvemos a empezar, pero esta vez al mover la cabeza lectora/escritora al final de la n-ada, y mover la cabeza lectora/escritora una celda a la derecha, al encontrar un blanco, borra el blanco y escribe "|" y regresa al inicio de la cinta para seguir con el proceso. Repetimos esto hasta que el cómputo sea 0. Por hipótesis sabemos que esto va a pasar en algún momento, cuando la n-ada de entrada sea admisible para minimalización relativa a la función ψ. Por lo tanto, la máquina M_ψ siempre realizara un cómputo al ir aumentando de uno en uno el $n+1$ argumento de ésta función,

Por lo tanto, toda función parcial recursiva es Turing computable. ■

El resultado inverso de este teorema, es decir, *toda función Turing computable es parcial recursiva*, no lo probaremos aquí. El lector interesado puede revisar Hermes, 1969.

En lo sucesivo manejamos indistintamente "función Turing computable" y "función computable".

Con esto, hemos reunido los resultados necesarios para el fin que perseguimos en este trabajo: a saber, que toda función parcial recursiva es Turing computable y viceversa. En el siguiente capítulo introducimos la noción de conjunto recursivo, la cual es relevante en la teoría matemática de la computación.

Capítulo III

Aritmetización de las Máquinas de Turing

En este capítulo desarrollaremos formalmente el resultado mencionado en la sección 3 del capítulo 1: la aritmetización de las máquinas de Turing. Este resultado es parte fundamental para nuestro trabajo. Para llegar a éste primero necesitamos desarrollar algunos resultados sobre conjuntos recursivos.

3.1 Conjuntos Recursivos y Recursivamente Enumerables

Definición 3.1.1. Sea $A \subseteq \mathbb{N}$. La función característica de A, $\mathcal{X}_A: \mathbb{N} \to \{0,1\}$, es la función definida de la siguiente forma:

$$\mathcal{X}_A(a) = 0 \quad \text{si } a \in A$$
$$\mathcal{X}_A(a) = 1 \quad \text{si } a \notin A$$

Definición 3.1.2. Sea $S \subseteq \mathbb{N}$. S es un conjunto recursivo si y sólo si su función característica es una función total computable.

Si $f: A \to B$ es una función suprayectiva, este hecho lo denotamos de la siguiente forma: $f: A \twoheadrightarrow B$.

Definición 3.1.3. Sea $S \subseteq \mathbb{N}$. Decimos que S es *un conjunto* recursivamente e*numerable* si y sólo si $S = \emptyset$ o existe $f: \mathbb{N} \twoheadrightarrow S$ función total computable; en tal caso, decimos que f *es una enumeración recursiva de* S.

Teorema 3.1.1. Todo subconjunto recursivo de $\mathbb{N}$ es recursivamente enumerable.

Demostración. Sea $S \subseteq \mathbb{N}$ recursivo. Si $S = \emptyset$, por definición es recursivamente enumerable. Por el contrario, si $S \neq \emptyset$, entonces S tiene un elemento mínimo s_0. Por hipótesis, la función característica de S, $\chi_S: S \to \mathbb{N}$, es una función total computable. Definimos una función total recursiva $f: \mathbb{N} \twoheadrightarrow S$ como sigue:

$$f(n) = \prod(\chi_S(n), s_0) + \prod(n, sgc(\chi_S(n)))$$

Donde $\prod(n, m)$ es la función producto $n \cdot m$, y $sgc: \mathbb{N} \to \{0, 1\}$ es la función *signo contrario* dada por $sgc(0) = 1$ y $sgc(n) = 0$ si $n \neq 0$. La función f es total recursiva y enumera (con posibles repeticiones) a los elementos de S. ■

Teorema 3.1.2. Un subconjunto $S \subseteq \mathbb{N}$ es recursivo si y sólo si S y S^C son recursivamente enumerables.

Demostración. Si $S = \mathbb{N}$ o $S = \emptyset$, el resultado se cumple trivialmente. Supongamos que S es un conjunto recursivo y $S \neq \emptyset$. Debemos probar que S y S^C son recursivamente enumerables. Por el teorema 3.1.1, S es recursivamente enumerable. Por otro lado, como $S^C \neq \emptyset$, tiene un elemento mínimo s_1. Por hipótesis, la función característica de S, χ_S, es una función total. Ahora definimos una función $g: \mathbb{N} \to S^C$ como sigue:

$$g(n) = \prod(\chi_S(n), n) + \prod(sgc(\chi_S(n)), s_1)$$

Esta función es total y enumera (con posibles repeticiones) los elementos de S^C. Por lo tanto, S y S^C son recursivamente enumerables.

A la inversa, si S y S^C son recursivamente enumerables, entonces S es recursivo. En efecto, por hipótesis existen $f: \mathbb{N} \twoheadrightarrow S$ y $g: \mathbb{N} \twoheadrightarrow S^C$ funciones totales computables que enumeran recursivamente a S y S^C. Definimos la función $h: \mathbb{N} \to \{0, 1\}$ como sigue:

$$h(s) = \begin{cases} 0 & si \quad s = f(n) \quad para\ algún \quad n \in \mathbb{N} \\ 1 & si \quad s = g(n) \quad para\ algún \quad n \in \mathbb{N} \end{cases}$$

Como $Img(f) \cap Img(g) = \emptyset$, la función h está bien definida y es total, además como f y g son computables, h también lo es, y $h = \mathcal{X}_S$. Por lo tanto, S es recursivo. ■

Teorema 3.1.3. Un subconjunto de $\mathbb{N}$ es recursivamente enumerable si y sólo si es el dominio de una función parcial computable.

Demostración. Supongamos que $S \subseteq \mathbb{N}$ es recursivamente enumerable. Debemos probar que S es el dominio de una función parcial computable. Como S es recursivamente enumerable, por definición existe $f: \mathbb{N} \twoheadrightarrow S$ función total computable. Definimos la función parcial $h: \mathbb{N} \to \mathbb{N}$ como sigue:

$$h(s) = min\{n \in \mathbb{N} \mid f(n) = s\}$$

Como f es computable, h también lo es; además el dominio de h es S, por lo que S es el dominio de una función parcial computable.

A la inversa, supongamos que $g: S \subseteq \mathbb{N} \to \mathbb{N}$ es una función parcial computable. Ahora debemos probar que S es recursivamente enumerable. Si $S = \emptyset$ o si $S = \mathbb{N}$, el resultado se cumple trivialmente por definición. Supongamos que S es un subconjunto propio no vacio. Entonces S tiene un mínimo s_0. Sea M la máquina de Turing que computa a la función g. Ahora definimos la función total computable $f: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ como sigue:

$$f(i,k) = \begin{cases} i & si \quad M\ converge\ en\ a\ lo\ más\ k\ pasos\ con\ la\ entrada\ i \\ s_0 & en\ otro\ caso \end{cases}$$

Obtenemos la siguiente sucesión de los elementos de S:

$$f(0,0), f(0,1), \ldots, f(1,0), f(1,1), \ldots, f(2,0), f(2,1), \ldots, f(n,m), \ldots$$

Sea $h: \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ una función total computable biyectiva. Entonces, la función composición $f \circ h$ es una función total computable y enumera, con posibles repeticiones, a los elementos de S. Por lo tanto, S es recursivamente enumerable. ■

Un resultado más que necesitamos para lograr el objetivo de este capítulo: aritmetizar las máquinas de Turing, es la *tesis de Church-Turing,* la cual dice, en una de sus versiones: *una función aritmética es efectivamente computable si y sólo si es recursiva.* La tesis de Church-Turing equipara una noción intuitiva (efectivamente computable), con una noción formal (recursividad). En otras palabras, nos dice que si podemos calcular intuitivamente una función aritmética, entonces lo podemos hacer recursivamente y viceversa. Aunque el regreso de este enunciado es claramente cierto, el problema está en la dirección contraria con la palabra "intuitivamente".

Por el teorema 2.2 y aceptando la tesis de Church-Turing, tenemos que toda función aritmética es efectivamente computable si y sólo si es Turing computable, es decir, si podemos computar intuitivamente una función aritmética, entonces la podemos computar con una máquina de Turing unitaria, y viceversa. Este resultado lo usaremos en lo sucesivo, incluso sin hacer mención explícita de él.

3.2 Aritmetización de las Máquinas de Turing

Dada una máquina de Turing M, siempre es posible reenumerar sus estados de modo que éstos sigan la secuencia $Q = \{0, 1, 2, \ldots, n\}$, donde 0 es el índice del estado inicial y n el índice del estado final, es decir, $q_f = q_n$.

De igual forma, dada una máquina de Turing M para la cual alguna pareja (q_i, x) no figura en el dominio de la función de transición de M, donde $q_i \in Q$ y $x \in \{\mathcal{B}, |\}$, podemos construir una máquina de Turing M' equivalente a M, tal que su función de transición δ', tiene como dominio al conjunto $(Q - \{q_f\}) \times \{\mathcal{B}, |\}$. Esto se logra definiendo $\delta'(q_i, x) = (q_f, x, \Lambda)$ en todos los casos faltantes,

y haciendo $\delta'(q_j, x) = \delta(q_j, x)$ cuando $(q_j, x) \in$ Dom δ. En otras palabras, podemos completar la tabla de la función de transición de M y con ello obtener M'. Por ejemplo, la tabla de la máquina M_S del ejemplo 1.2.1 que computa el sucesor de un número natural es:

$$\delta(q_0, |) = (q_1, \mathcal{B}, D)$$
$$\delta(q_1, |) = (q_1, |, D)$$
$$\delta(q_1, \mathcal{B}) = (q_2, |, I)$$
$$\delta(q_2, |) = (q_2, |, I)$$
$$\delta(q_2, \mathcal{B}) = (q_f, |, \Lambda)$$

Indefinida en otros casos

Esta máquina está indefinida en $(q_0, \mathcal{B})$. Completando la tabla y reenumerando los estados, obtenemos la siguiente máquina M_S' equivalente a M_S:

$$\delta(q_0, |) = (q_1, \mathcal{B}, D)$$
$$\delta(q_0, \mathcal{B}) = (q_3, \mathcal{B}, \Lambda)$$
$$\delta(q_1, |) = (q_1, |, D)$$
$$\delta(q_1, \mathcal{B}) = (q_2, |, I)$$
$$\delta(q_2, |) = (q_2, |, I)$$
$$\delta(q_2, \mathcal{B}) = (q_3, |, \Lambda)$$

Por lo anterior, y sin pérdida de generalidad, en lo sucesivo supondremos que las máquinas consideradas tienen una tabla completa y que sus estados están ordenados en la forma anteriormente descrita.

Por otra parte, podemos describir la tabla de la función de transición de una máquina de Turing sin usar paréntesis, signos de igualdad o el símbolo δ. Ejemplificamos lo anterior con la tabla de la máquina M_S' recién mencionada:

$$q_0 \mid q_1 \mathcal{B} D$$
$$q_0 \mathcal{B} q_3 \mathcal{B} \Lambda$$
$$q_1 \mid q_1 \mid D$$
$$q_1 \mathcal{B} q_2 \mid I$$
$$q_2 \mid q_2 \mid I$$
$$q_2 \mathcal{B} q_3 \mid \Lambda$$

Esta tabla nos da la misma información que la original. Una ventaja es que podemos escribir esto en forma lineal:

$$q_0 \mid q_1 \mathcal{B} D q_0 \mathcal{B} q_3 \mathcal{B} \Lambda q_1 \mid q_1 \mid D q_1 \mathcal{B} q_2 \mid I q_2 \mid q_2 \mid I q_2 \mathcal{B} q_3 \mid \Lambda$$

En esta sucesión reconocemos cada renglón de la tabla separando los símbolos en bloques de 5:

$$\underbrace{q_0 \mid q_1 \mathcal{B} D}_{1^{er}\text{ renglón}} \underbrace{q_0 \mathcal{B} q_3 \mathcal{B} \Lambda}_{2^{do}\text{ renglón}} \underbrace{q_1 \mid q_1 \mid D}_{3^{er}\text{ renglón}} \underbrace{q_1 \mathcal{B} q_2 \mid I}_{4^{to}\text{ renglón}} \underbrace{q_2 \mid q_2 \mid I}_{5^{to}\text{ renglón}} \underbrace{q_2 \mathcal{B} q_3 \mid \Lambda}_{6^{to}\text{ renglón}}$$

Ahora realizamos la siguiente asignación numérica a los símbolos:

Símbolo	Número
$\mid$	2
$\mathcal{B}$	4
D	6
I	8
Λ	0

La asignación para los estados es de la siguiente forma: tomamos el índice de cada estado, lo escribimos en base 5, y a cada dígito de esta conversión lo multiplicamos por 2 y le sumamos uno. Así, la asignación numérica a los estados es tal que el número asignado a cada estado, escrito en base 10, sólo tiene dígitos impares. Por ejemplo, si tuviéramos el estado q_{435}, escribimos el

índice en base 5: $(3220)_5$, luego cada dígito lo multiplicamos por 2 y le sumamos uno para obtener la asignación numérica para este estado: 7551.

Con esta asignación los estados de la máquina $M_S{}'$ quedan así:

Estados	Número Asignado
q_0	1
q_1	3
q_2	5
q_3	7

Con la asignación anterior podemos codificar la sucesión de símbolos de la máquina $M_S{}'$ en un número:

$$12346147403232634528525285472 0$$

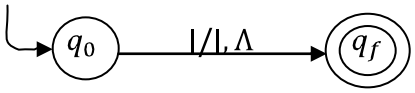
El procedimiento anterior se puede aplicar a cualquier máquina de Turing, de modo que a cada una de ellas le podemos asignar un único número. Dicho número codifica la función de transición de la máquina. Y le llamaremos el *índice* de la máquina. De este modo podemos enumerar las máquinas de Turing conforme a sus índices:

Máquina de Turing	Índice
M	$\longrightarrow i$

La enumeración se suele denotar con M_i.

De hecho, se puede establecer una función entre las máquinas de Turing (i.e., las tablas de transición) y los números naturales. La idea es más o menos la siguiente:

Para ello construyamos la máquina M_I que computa la función identidad $I_d: \mathbb{N} \to \mathbb{N}$:



El índice de la máquina M_I es 1232014340. Entonces definimos la siguiente función:

$$\varphi(n) = \begin{cases} 1232014340 & si \quad n \ no\ es\ el\ índice\ de\ una\ máquina \\ n & si \quad n \ es\ el\ índice\ de\ una\ máquina \end{cases}$$

Esta función enumera los índices de las máquinas de Turing, es una enumeración de las máquinas.

Dado un número natural cualquiera, siempre es posible saber si éste es el índice de una máquina y en tal caso, reconstruir la tabla de ésta: primero, el primer dígito del número, de derecha a izquierda, tiene que ser 1 (el estado inicial); segundo, el número se tiene que poder separar en bloques de cinco, considerando todos los dígitos impares seguidos como un sólo número del bloque de cinco; por su parte los dígitos pares se toman de uno en uno; tercero, cada bloque de cinco debe tener la siguiente estructura: *impar-par-impar-par-par* (considerando de nuevo que los impares pueden ser de más de un dígito); entre los números impares sólo pueden aparecer el 2 o el 4, lo mismo ocurre con el par que le sigue al segundo impar; finalmente el último par del bloque sólo puede ser el 6, 8 o 0. Si el número dado no satisface todo lo anterior, no es el índice de una máquina.

Para reconstruir la tabla de una máquina de Turing a partir de su índice, recordamos que cada bloque de cinco es un renglón de la tabla y que en cada uno de ellos la información se encuentra ordenada de la siguiente manera: *estado-símbolo-estado-símbolo-acción*; siguiendo este orden, decodificamos los números de acuerdo a la asignación de los símbolos y los estados, con lo cual obtenemos la correspondencia:

$$\delta(estado, símbolo) = (estado, símbolo, acción).$$

Por el teorema 2.2 sabemos que toda función parcial recursiva es Turing computable. Por lo tanto, si φ es una función parcial recursiva, existe una máquina de Turing M_i que la computa; con esta notación denotamos a la función φ con φ_i. Cuando el dominio de φ sea un subconjunto de $\mathbb{N}^n$, la notación es φ_i^n. De este modo también tenemos una enumeración de las funciones parciales recursivas.

Por todo lo anterior, en lo sucesivo podemos hablar tanto de las máquinas como de las funciones parciales recursivas a través de su índice.

Una cuestión importante es que el uso de índices permite probar algunos resultados relacionados con las máquinas de Turing y la computabilidad. Por ejemplo, ¿qué pasa cuando una máquina de Turing se aplica a su propio índice? Esto nos recuerda al método diagonal de Cantor, y sugiere que su aplicación dará como resultado algún tipo de imposibilidad. El siguiente teorema es un ejemplo de ello, y en él nos referimos al conjunto de las máquinas de Turing que convergen cuando se les aplica su propio índice.

Teorema 3.2.1. El conjunto $\mathcal{K} = \{n \in \mathbb{N} | n \in dom(\varphi_n)\}$ no es recursivo.

Demostración. Supongamos que $\mathcal{K}$ es recursivo. Por el teorema 3.1.2, $\mathcal{K}$ y $\mathcal{K}^C$ son recursivamente enumerables. Como $\mathcal{K}^C$ es recursivamente enumerable, por el teorema 3.1.3, sabemos que $\mathcal{K}^C$ es el dominio de una función parcial computable $f: \mathbb{N} \to \mathbb{N}$. Como f es computable, existe una máquina de Turing M que la computa. Sea i el índice de M, de modo que $f = \varphi_i$. Como $\mathcal{K} \cup \mathcal{K}^C = \mathbb{N}$ y $\mathcal{K} \cap \mathcal{K}^C = \emptyset$, i debe pertenecer a $\mathcal{K}$ o a $\mathcal{K}^C$ y no a ambos. Tenemos:

$$i \in \mathcal{K} \Leftrightarrow i \in dom(\varphi_i) \Leftrightarrow i \in \mathcal{K}^C$$

La contradicción anterior resulta de suponer que $\mathcal{K}$ es recursivo; por lo tanto, $\mathcal{K}$ no es recursivo. ■

Más allá de su valor intrínseco, este teorema nos ilustra el poder del método diagonal de Cantor y su utilidad en el ámbito de la teoría matemática de la computación. En lo que sigue veremos otras aplicaciones de este método, cuya posibilidad se debe al hecho de la aritmetización de las máquinas de Turing.

Capítulo IV

Teorema de Rice

En este capítulo daremos la demostración del teorema de Rice, un resultado fundamental para nuestro trabajo. Para ello, primero demostraremos otros resultados centrales en la teoría de la computabilidad: el teorema *s-m-n* de Kleene, el teorema de recursión y el teorema del punto fijo.

Teorema 4.1 (teorema *s-m-n de Kleene)*. Dados $n \in \mathbb{N}$ y $m \in \mathbb{N}$, existe una función total computable $s: \mathbb{N} \times \mathbb{N}^m \to \mathbb{N}$ tal que, para toda $i \in \mathbb{N}$ y para toda $v \in \mathbb{N}^m$ se cumple que:

$$dom\ \varphi^n_{s(i,v)} = \{u \in \mathbb{N}^n \mid (u, v) \in dom\ \varphi^{m+n}_i\}$$

y

$$\varphi^n_{s(i,v)}(u) = \varphi^{m+n}_i(u, v)$$

cuando ambos lados de la igualdad están definidos.

Dada una función de n argumentos, fijando algunos argumentos, obtenemos una función restringida. Por ejemplo, sea $f: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ una función, al fijar un argumento, obtenemos una función restringida $f(u, v_0)$ tal que $f: \mathbb{N} \to \mathbb{N}$, de hecho, podemos obtener una función restringida por cada natural. El teorema s-m-n, afirma que existe una función s tal que, dependiendo de los argumentos que fijamos y el índice de la máquina que computa a la función, entonces la función s nos da el índice de una máquina que computa la función así restringida.

Hacemos la demostración para el caso $m = n = 1$. Sean $i, v \in \mathbb{N}$, donde i es el índice de una función parcial computable. Mostraremos como obtener el valor de

la función s para estos argumentos. Para ello, construyamos una máquina de Turing M de la siguiente manera: Ante una cinta cuyo único contenido es $[\bar{u}]$, se activa un modulo que da como salida $[\bar{u}]\mathcal{B}\bar{v}$. Para cada v dada podemos construir dicho modulo. Luego, se activa un modulo que actúa como la máquina M_i y obtenemos el cómputo de M_i a partir de $[\bar{u}]\mathcal{B}\bar{v}$, es decir, $\varphi_i^2(u,v)$; no obstante, M sólo recibió de entrada a u. Para cada i y v dadas, podemos construir M de esta forma. Una vez construida, podemos obtener el índice j de M, siendo éste el valor de s para los argumentos i y v, es decir, $s(i,v) = j$. Además, por la forma en que se construyo M_j tenemos que para toda $u, v \in \mathbb{N}$

$$M_{s(i,v)}(u) = M_j(u) = M_i(u,v)$$

siempre que M_i converge. En otras palabras:

$$\varphi_{s(i,v)}(u) = \varphi_j(u) = \varphi_i(u,v)$$

siempre que $(u,v) \in dom\ \varphi_i^2$.

Como el lector podrá comprobar, el caso general, aunque más complicado, no es distinto de éste. La prueba correspondiente la dejamos como ejercicio al lector[1]. ■

Teorema 4.2 (teorema de recursión). Dado $n > 1$ y $f: \mathbb{N}^n \to \mathbb{N}$ función parcial computable, existe $k \in \mathbb{N}$ tal que

$$f(x_1, x_2, \ldots, x_{n-1}, k) = \varphi_k^{n-1}(x_1, x_2, \ldots, x_{n-1})$$

Demostración. Para $n \in \mathbb{N}$ y $m = 1$ por el teorema *s-m-n,* existe $s: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tal que $\varphi_i^{n+1}(u,v) = \varphi_{s(i,v)}^n(u)$. Con base en s, por composición definimos la función

[1] Esta demostración para el caso $n = m = 1$ se puede considerar completa con base en la tesis de Church-Turing, pues hemos indicado un procedimiento efectivo para calcular el índice j de la máquina M. No obstante, también es posible exhibir la correspondiente función recursiva $s(i,v)$, una engorrosa tarea que aquí no haremos.

$f(x_1, x_2, \ldots, x_{n-1}, s(x_n, x_n))$ que es parcial computable; sea r el índice de la máquina de Turing que computa ésta función. Por definición, r es tal que $\varphi_r^n(x_1, x_2, \ldots, x_{n-1}, x_n) = f(x_1, x_2, \ldots, x_{n-1}, s(x_n, x_n))$. Por el teorema *s-m-n*, $\varphi_r^n(x_1, x_2, \ldots, x_{n-1}, x_n) = \varphi_{s(r, x_n)}^{n-1}(x_1, x_2, \ldots, x_{n-1})$. Sea $k = s(r, r)$. Entonces:

$$f(x_1, x_2, \ldots, x_{n-1}, k) = f(x_1, x_2, \ldots, x_{n-1}, s(r, r)) = \varphi_r(x_1, x_2, \ldots, x_{n-1}, r) = \varphi_{s(r,r)}(x_1, x_2, \ldots, x_{n-1}) = \varphi_k(x_1, x_2, \ldots, x_{n-1}).$$ ■

En la demostración anterior, en la función $s(x_n, x_n)$, el valor x_n hace las veces de índice y de argumento, permitiéndonos una especie de autoreferencia, similar a la que juega en la demostración tradicional del teorema de Gödel; donde un enunciado habla de sí mismo. En nuestro caso, con la autorrefencia hablamos de cómo la función compuesta actúa sobre su propio índice, es decir, "sobre ella misma".

Teorema 4.3 (teorema del punto fijo). Si $f: \mathbb{N} \to \mathbb{N}$ es una función total computable, entonces hay un $i \in \mathbb{N}$ tal que $\varphi_i = \varphi_{f(i)}$.

El teorema anterior nos dice que toda función recursiva computa en alguno de sus argumentos, el índice de una máquina de Turing que computa a la función que tiene como índice el argumento dado. Es decir, hay al menos un índice (función parcial) que en la imagen de f es el índice de otra máquina que la computa a ella.

Demostración. Definimos una función $h: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ como sigue:

$$h(x, u) = \varphi_{f(u)}(x)$$

Por el teorema de recursión, existe $i \in \mathbb{N}$ tal que $h(x, i) = \varphi_i(x)$. Por otro lado, por definición $h(x, i) = \varphi_{f(i)}(x)$. Por lo tanto, $\varphi_i = \varphi_{f(i)}$. ■

Este último resultado nos permite demostrar el teorema de Rice, teorema central de este capítulo.

Teorema 4.4 (teorema de Rice). Sea I un subconjunto recursivo propio no vacío de $\mathbb{N}$. Entonces, existen $i \in I$ y $j \in I^C$ tales que $\varphi_i = \varphi_j$.

Demostración. Sea I un subconjunto recursivo propio no vacío de $\mathbb{N}$. Por hipótesis, hay un $k \in I$ y un $j \in I^C$. Definimos la función total $f: \mathbb{N} \to \mathbb{N}$ como sigue:

$$f(n) = \begin{cases} j & si \quad n \in I \\ k & si \quad n \in I^C \end{cases}$$

Como I es recursivo, f es computable. Por el teorema del punto fijo, existe un $i \in \mathbb{N}$ tal que $\varphi_i = \varphi_{f(i)}$. Como $I \cup I^C = \mathbb{N}$, i debe pertenecer o bien a I o bien a I^C. Si $i \in I$, entonces:

$$\varphi_i = \varphi_{f(i)} = \varphi_j$$

y si $i \in I^C$, entonces:

$$\varphi_i = \varphi_{f(i)} = \varphi_k$$

En ambos casos, renombrando, hay un $i \in I$ y un $j \in I^C$ tales que $\varphi_i = \varphi_j$. ■

El teorema de Rice afirma que sólo son decidibles las propiedades que son triviales, es decir, aquellas propiedades que o bien las cumplen todas las máquinas de Turing, o bien no las cumple ninguna máquina de Turing. Digamos que P es una propiedad recursiva acerca de las máquinas de Turing, la cual la cumple alguna máquina pero no todas, entonces por el teorema de Rice, hay un índice j tal que $P(j)$, pero $j \notin \{i \mid P(i)\}$. El teorema de Rice también nos da la indecibilidad de los siguientes conjuntos:

- $\{i \mid \varphi_i \, es \, total\}$
- $\{i \mid a \in dom(\varphi_i)\}$ donde a es un natural dado

- $\{ i \mid \varphi_i \, es \, constante \}$
- $\{ i \mid dom(\varphi_i) \, es \, finito \}$

Capítulo V

Teorema de Incompletud de Gödel

Las cosas están casi dispuestas para demostrar el resultado central de este trabajo, el teorema de incompletud de Gödel para la aritmética recursiva. Lo único que nos falta son algunos conceptos tomados de la lógica matemática (como los de *sistema formal, aritmética recursiva, enunciado recursivo, deducción formal, expresabilidad en un sistema formal y completud de un sistema formal),* y algunos resultados sobre estos conceptos, como por ejemplo, la noción de *expresabilidad de relaciones aritméticas* en un sistema formal. Entre estos resultados, el más importante para nosotros es el siguiente: Para una gran variedad de sistemas formales $\mathcal{S}$ para la aritmética recursiva, una relación aritmética R es expresable en $\mathcal{S}$ si y sólo si R es recursiva. El lector interesado hallará estos conceptos en Mendelson, 1964. Capítulo III, o bien véase el apéndice para un ejemplo de un sistema que formaliza la aritmética recursiva. Por lo que a nosotros respecta, a continuación presentamos el concepto recién aludido.

En general decimos que un sistema formal $\mathcal{S}$ definido en un lenguaje de primer orden $\mathcal{L}_{\mathcal{S}}$ contiene una representación de la aritmética recursiva cuando satisface lo siguiente:

Para toda relación aritmética recursiva $R(x_1, x_2, \dots, x_n)$ existe en $\mathcal{L}_{\mathcal{S}}$ una fórmula $\mathfrak{r}(x_1, x_2, \dots, x_n)$ con n variables libres tal que, para todo $(m_1, m_2, \dots, m_n) \in \mathbb{N}^n$, si $R(m_1, m_2, \dots, m_n)$, entonces la fórmula $\mathfrak{r}(\overrightarrow{m}_1, \overrightarrow{m}_2, \dots, \overrightarrow{m}_n)$ se deduce en el sistema, (*i. e.*, $\vdash_{\mathcal{S}} \mathfrak{r}(\overrightarrow{m}_1, \overrightarrow{m}_2, \dots, \overrightarrow{m}_n)$); y si no $R(m_1, m_2, \dots, m_n)$ entonces, la negación de la fórmula se deduce en el sistema (*i. e.*, $\vdash_{\mathcal{S}} \neg \mathfrak{r}(\overrightarrow{m}_1, \overrightarrow{m}_2, \dots, \overrightarrow{m}_n)$). En cuanto a la notación, se supone que en $\mathcal{L}_{\mathcal{S}}$ hay ciertos términos cerrados (que no tienen variables libres) que sirven como nombres o representaciones de los números

naturales. A estos les llamamos *numerales* y los denotamos en el metalenguaje con la notación $\overrightarrow{m}$ para cada $m \in \mathbb{N}$. Por lo general, se trata de la forma $\underbrace{ss\ldots s}_{m\ veces}0$, donde s es el símbolo formal para la función sucesor. Por ejemplo, $\overrightarrow{5}$ sería la expresión $sssss\,0$.

La propiedad anterior la podemos resumir diciendo que el sistema $\mathcal{S}$ satisface lo siguiente: Para toda relación $R(x_1, x_2, \ldots, x_n)$,

Si $R(m_1, m_2, \ldots, m_n)$, entonces $\vdash_{\mathcal{S}} \mathfrak{r}(\overrightarrow{m}_1, \overrightarrow{m}_2, \ldots, \overrightarrow{m}_n)$
Si no $R(m_1, m_2, \ldots, m_n)$, entonces $\vdash_{\mathcal{S}} \neg\mathfrak{r}(\overrightarrow{m}_1, \overrightarrow{m}_2, \ldots, \overrightarrow{m}_n)$

Asimismo, en lo que sigue haremos uso del siguiente resultado, el cual es una consecuencia del teorema de Rice.

Teorema 5.1. Sea $A_i = \{j \mid \varphi_i = \varphi_j\}$, donde i es un natural dado. El conjunto A_i no es recursivo.

Demostración. Si A_i fuera recursivo, entonces, por el teorema de Rice, habría un j tal que $\varphi_i = \varphi_j$ y j no pertenecería a A_i (Obviamente, i sí pertenece a A_i). No obstante, en A_i figuran precisamente todos los índices de máquinas que computan la función φ_i, por lo que j tendría que pertenecer a A_i. Por tanto, lo anterior no es posible, a partir de lo cual concluimos que A_i no es recursivo. ■

Otro elemento necesario para demostrar el teorema de Gödel es el predicado $T(i, a, x)$ de Kleene. Este predicado se define así: $T(i, a, x)$ es verdadero si y sólo si la máquina M_i computa un valor para el argumento a en exactamente x pasos de operación. Este predicado es recursivo, y como tal, es expresable en $\mathcal{S}$, habiendo una fórmula $t(x_1, x_2, x_3)$ que lo expresa. Al respecto véase Kleene,1967. Capitulo V.

De la misma manera, la relación $R(i,a,y,x)$ que es verdadera si y sólo si $M_i(a) = y$ y el computo se realiza en exactamente x pasos, es recursiva, por lo tanto, hay en el lenguaje de $\mathcal{S}$ una fórmula $r(x_1,x_2,x_3,x_4)$ que corresponde a este predicado.

Con estos elementos, estamos en posibilidad de demostrar el resultado de Gödel.

Primer teorema de incompletud de Gödel. Si un sistema $\mathcal{S}$ para la aritmética recursiva es consistente y correcto, entonces es incompleto, es decir, existen enunciados $\mathcal{A} \in \mathcal{L}_S^0$ tales que $\mathcal{S} \nvdash \mathcal{A}$ y $\mathcal{S} \nvdash \neg\mathcal{A}$.

Demostración. Supongamos que $\mathcal{S}$ es completo. Con las fórmulas t y r podemos construir la siguiente fórmula Φ:

$$\Phi(i,j) = \forall a \exists b \exists c\big(\big(t(i,a,b) \leftrightarrow t(j,a,c)\big) \wedge \forall y\big(r(i,a,y,b) \leftrightarrow r(j,a,y,c)\big)\big)$$

Grosso modo, $\Phi(i,j)$ "dice" lo siguiente: M_i calcula algo para a si y sólo si M_j calcula algo para a, y los valores calculados son los mismos". En otras palabras, $\Phi(i,j)$ "dice" que $\varphi_i = \varphi_j$. Ahora, como $\mathcal{S}$ es completo, para cada pareja (i,j) se tiene que alguna de las siguientes fórmulas es teorema en $\mathcal{S}$:

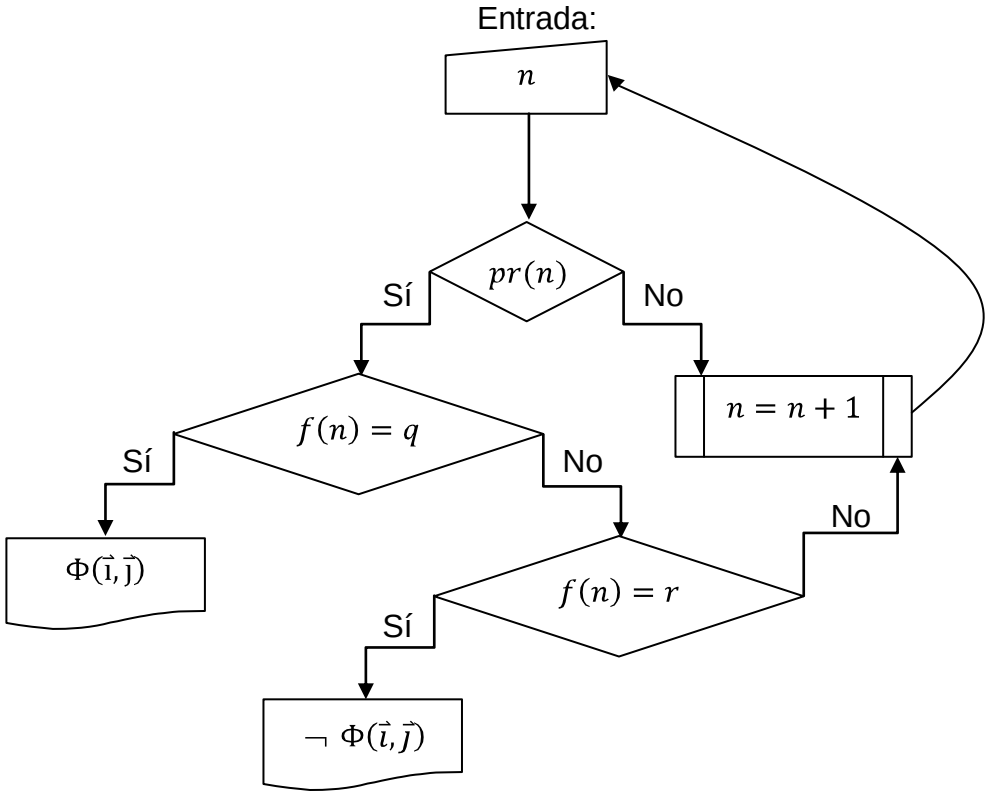
$$\Phi(\vec{i},\vec{j}) \quad \text{ó} \quad \neg\,\Phi(\vec{i},\vec{j})$$

Así, para decidir si j pertenece o no a A_i basta con enumerar los teoremas de $\mathcal{S}$ hasta que en la lista aparezca $\Phi(i,j)$ o su negación. En vista de la completud, esto sucederá en algún momento, por lo que, conforme a la tesis de Church, el conjunto A_i sería recursivo, contradiciendo el teorema 5.1. Por lo tanto $\mathcal{S}$ es incompleto. ■

Cabe aclarar que la hipótesis acerca de la validez de la tesis de Church-Turing, aplicada en la prueba anterior, se puede eliminar, mostrando que el procedimiento referido (enumerar las pruebas, etc.) es recursivo.

Veamos. Dada una numeración de Gödel para $\mathcal{L}_{\mathcal{S}}$, tanto el predicado de prueba $pr(x)$ (x es el número de Gödel de una prueba en $\mathcal{S}$) como la función $f(x) \stackrel{\text{def}}{=}$ último exponente de x en su descomposición como producto de factores primos, son recursivos. Por lo tanto, la relación $p(x,y) \stackrel{\text{def}}{=} x$ es el número de Gödel de una prueba y y es el número de Gödel de la fórmula demostrada, también es recursiva.

Sean q el número de Gödel de la fórmula $\Phi(\mathrm{i},\mathrm{j})$ (es decir, $q = \gamma\left(\Phi(\vec{\imath},\vec{\jmath})\right)$) y $r = \gamma(\neg\, \Phi(\vec{\imath},\vec{\jmath}))$. Ahora, con base en estos números podemos esquematizar el procedimiento recién referido mediante un diagrama de flujo como sigue.



Este algoritmo se puede definir recursivamente, pues cada uno de sus elementos sólo involucra predicados y funciones recursivas. Además, dada la supuesta completud del sistema $\mathcal{S}$, el proceso siempre terminaría.

Muchas de las cosas que hemos hecho se pueden presentar con un alto nivel de rigor. No obstante, ese no ha sido el propósito de este trabajo, cuya finalidad es mostrar cómo el teorema de Gödel es una consecuencia del teorema de Rice. Llenar todas las lagunas sería algo encomiable, aunque los elementos básicos ya se encuentran en la literatura (Hermes, Rogers, Kleene, Mendelson, Enderton, etc.)

Con esto tenemos concluido el trabajo; probamos que cualquier sistema formal consistente y correcto para la aritmética recursiva, es incompleto; y lo hicimos mostrándolo como consecuencia del teorema de Rice, un resultado importante dentro de la teoría de la computabilidad

Apéndice

Un sistema formal para la aritmética de Peano

El sistema se denota con *AP*. Su lenguaje L_{AP} es muy reducido: sólo tiene un símbolo de relación para la igualdad «=», una constante «*0*» y tres símbolos de operación: «*s*» (sucesor), «+» (suma) y «·» (producto). Por comodidad, en vez de escribir la suma en la forma +(*x*, *y*) como es usual en lógica, escribiremos *x* + *y*, y lo mismo haremos para el producto (notación infija). El sistema de axiomas *AP* es el siguiente (donde *A*(*x*) es una fórmula cualquiera):

Axiomas para la igualdad

$x = x$	(Identidad)
$x = y \to (A(x) \to A(y))$	(Axioma de Leibniz)

Axiomas aritméticos

$\neg(sx = 0)$	(axiomas para el sucesor)
$sx = sy \to x = y$	
$x + 0 = x$	(axiomas para la suma)
$x + sy = s(x + y)$	
$x \cdot 0 = 0$	(axiomas para el producto)
$x \cdot sy = x \cdot y + x$	
$(A(0) \wedge \forall x(A(x) \to A(sx)) \to A(x)$	(axiomas de inducción)

A partir de estos axiomas es posible derivar en el cálculo de predicados los teoremas de la aritmética de primer orden. Dada la inexistencia de un símbolo especial para cada número natural, dichos números se representan mediante

las expresiones *s0*, *ss0*, *sss0*, etc. que por comodidad escribimos $\overline{1}$, $\overline{2}$, $\overline{3}$, etc.; asimismo, aunque el lenguaje no tiene signos especiales para relaciones como la desigualdad o la divisibilidad, éstas se pueden definir mediante predicados aritméticos «$x < y$» y «$x \mid y$» como sigue:

$x \leq y \equiv_{\text{def}} \exists z(x + z = y)$	x es menor o igual que y.
$x < y \equiv_{\text{def}} \exists z(\neg(z = 0) \wedge x + z = y)$	x es estrictamente menor que y.
$x \mid y \equiv_{\text{def}} \exists z(x \cdot z = y)$	x es un divisor de y.

Asimismo, la propiedad de ser un número primo se puede definir de la siguiente manera:

$$prim(x) \equiv_{\text{def}} x > \overline{1} \wedge \forall z(z \mid x \rightarrow (z = \overline{1} \vee z = x))$$ x es un número primo.

Pese a lo reducido de su lenguaje, el poder expresivo de *AP* es enorme, pudiéndose traducir y probar en él una multitud de proposiciones relativas a los números naturales, como, por ejemplo, el teorema de Euclides sobre la existencia de una infinidad de números primos o el algoritmo de la división.

Representabilidad

Si bien el lenguaje de *AP* tiene una notoria carencia de signos de función, esta deficiencia no es una seria amenaza para su poder expresivo. En este sentido, en 1931 Gödel demostró un resultado (conocido como *lema de la correspondencia*) que pone en evidencia la relación existente entre el sistema *AP* y la aritmética recursiva: *todos los predicados y todas las funciones recursivas son representables en AP*.

Entre otras cosas, lo anterior significa que para toda relación recursiva $R(x_1, ..., x_n)$ existe en L_{AP} una fórmula $r(x_1, ..., x_n)$ con n variables libres tal que

si $R(k_1, ..., k_n)$, entonces $AP \vdash r(\bar{k}_1, ..., \bar{k}_n)$, y

si no_$R(k_1, ..., k_n)$, entonces $AP \vdash \neg r(\bar{k}_1, ..., \bar{k}_n)$

de modo que en *AP* tenemos una representación exacta de las relaciones numéricas que se dan en el ámbito de la aritmética recursiva; en particular, la relación $y = f(x_1, ..., x_n)$ es representable cuando la función f es recursiva. Este resultado es fundamental para la demostración de los teoremas de incompletud de Gödel y se puede parafrasear así: *el sistema* AP *contiene una formalización de la aritmética recursiva*.

Conclusiones

Si bien en este trabajo no hemos probado un nuevo resultado, lo que sí ha sucedido es que el teorema de Gödel se ha demostrado siguiendo un camino original y bajo un enfoque diferente. Aunado a ello, hemos mostrado el alcance y poder de un resultado como el teorema de Rice de serena apariencia: *cualquier propiedad sobre Maquinas de Turing que se pueda caracterizar recursivamente es trivial*. Este teorema es limitativo dentro de la teoría de la computación y muestra cómo problemas aparentemente sencillos no se pueden resolver; por ejemplo, dado el índice de una máquina de Turing, saber si ésta computa una función constante, o a la identidad, o bien, dado el índice de una máquina de Turing, saber si otra máquina computa la misma función. Si bien todo esto encierra interés en sí mismo, en estas páginas hemos puesto en claro que el alcance de este teorema va más allá de la teoría de la computabilidad; de hecho, su fuerza es tal que implica la incompletabilidad de la aritmética. Así, detrás de tan apacible apariencia se oculta un resultado de consecuencias imprevistas.

Con nuestra labor, esperamos haber tendido un puente entre la teoría matemática de la computación, el estudio axiomático de la aritmética y los sistemas formales. Se trata de una línea interesante de investigación en estas áreas, la cual da muestras de encerrar un enorme interés para filosofía de las matemáticas.

Bibliografía

Bridges, Douglas, (1994). *Computability*. New York, Springer-Verlag

Davis, Martin, (1993). The undecidable. New York, Dover publications Inc

Hermes, Hans, (1969). *Enumerability Decidability Computability An Introduccion to the Theory of Recursive Functions.* Traslated by G. T. Herman and O. Plassmann. New York, Springer-Verlag

Kleene, Stephen, Cole, (1967). *Mathematical Logic,* New York, John Wiley & Sons, INC

Mendelson, Elliott, (1997). *Introduction to Mathematical Logic*, Cuarta Edición. New York, Champan & Hall

Turing, Alan, M, (1936). “On computable numbers with an application to the Entscheidgungsproblem”, reimpreso en (Davis, 1993)