Teoría de la Computabilidad
FCC BUAP

José de Jesús Lavalle Martínez

Primavera de 2013

Índice general

Resumen

Este documento es una traducción de partes de los capítulos 1, ... del libro *Computabiity Theory - An Introduction to Recursion Theory* de Herbert B. Enderton.

Capítulo 1

El concepto de computabilidad

1.1. El concepto informal

1.1.1. Conjuntos decidibles

La teoría de la computabilidad, también conocida como teoría de la recursión, es el área de las matemáticas que trata con el concepto de *procedimiento efectivo*, un procedimiento que se puede llevar a cabo siguiendo reglas específicas. Por ejemplo, podríamos preguntar si existe algún procedimiento efectivo -algún algoritmo- que, dada una sentencia sobre los enteros, decidirá si la sentencia es verdadera o falsa. En otras palabras, ¿es el conjunto de sentencias verdaderas sobre los enteros *decidible*? (Veremos posteriormente que la respuesta es negativa.)

O un ejemplo más simple, el conjunto de números primos es ciertamente un conjunto decidible. Esto es, existen procedimientos muy mecánicos, los cuales se enseñan en las escuelas, para decidir si cualquier número entero dado es o no un número primo. (Para un número muy grande, el procedimiento enseñado en las escuelas prodría tardar mucho tiempo.) Si queremos, podemos ejecutar un programa de computadora para ejecutar el procedimiento. Aún más simple, el conjunto de enteros pares es decidible. Podemos escribir un programa de computadora que, dado un entero, muy rápidamente decidirá si es o no par. Nuestro objetivo es estudiar qué problemas de decisión se pueden resolver (en principio) mediante un programa de computadora, y qué problemas de decisión (si hay) no se pueden resolver.

De manera más general, considere un conjunto S de números naturales. (Los números naturales son 0, 1, 2, ... En particular 0 es natural.) Decimos

que S es un conjunto *decidible* si existe un procedimiento efectivo que, dado cualquier número natural, eventualmente terminará dándonos la respuesta "Sí"si el número dado es un miembro de S y "No"si no es un miembro de S.

(Inicialmente, vamos a examinar la computabilidad en el contexto de números naturales. Posteriormente, veremos que los conceptos de computabilidad pueden ser transferidos fácilmente al contexto de cadenas de letras de un alfabeto finito. En ese contexto, podemos considerar un conjunto S de cadenas, tal como el conjunto de ecuaciones, como $x(y+z) = xy+xz$), que se cumplen en el álgebra de números reales. Pero para empezar, consideraremos conjuntos de números naturales.)

Y por un *procedimiento efectivo* entenderemos un procedimiento mediante el cual podemos dar instrucciones exactas -un programa- para llevar a cabo el procedimiento. Seguir estas instrucciones no debe demandar ideas brillantes de parte del agente (humano o máquina). Debe ser posible, al menos en principio, hacer las instrucciones tan explícitas que puedan ser ejecutadas por un empleado diligente (quien es muy bueno para seguir instrucciones pero no es muy ingenioso) o una máquina (la cual no piensa en absoluto).

Esto es, debe ser posible para nuestras instrucciones ser *implementadas mecánicamente*. (Uno puede imaginar a un matemático tan brillante que pueda ver una sentencia de la aritmética y decir si es verdadera o falsa. Pero no puede pedirle al oficinista que haga ésto. Y no exite un programa de computadora para hacer ésto. No es solamente que no hemos tenido éxito al escribir tal programa. ¡Realmente podemos probar que no es posible que exista tal programa!)

No obstante estas instrucciones deben, por supuesto, ser de longitud finita, no imponemos alguna cota superior a su posible longitud. No descartamos la posibilidad de que el número de instrucciones pueda ser absurdamente grande. (Si el número de instrucciones excede al número de electrones en el universo, solamente nos encojeremos de hombros y decimos, "Este es un programa muy grande".) Insistimos sólo en que las instrucciones -el programa- sean de longitud finita, para que podamos comunicarselas a la persona o máquina que haga los cálculos. (No hay manera de darle a alguien todo un objeto infinito.)

Similarmente, para obtener los conceptos más amplios, no imponemos cotas sobre el tiempo que el procedimiento pueda consumir antes de que nos dé la respuesta. Tampoco imponemos una cota sobre la cantidad de espacio de almacenamiento (papel de borrador) que el procedimiento pudiera necesitar. (El procedimiento puede, por ejemplo, necesitar utilizar números muy

grandes que requieren una cantidad sustancial de espacio simplemente para escribirlos.) Sólamente insistiremos en que el procedimiento nos dé la respuesta eventualmente, en cierta extensión finita de tiempo. Lo que está excluido definitivamente es hacer infinitos pasos y *luego* dar la respuesta.

En el capítulo 7, consideraremos conceptos más restrictivos, donde la cantidad de tiempo es limitada de alguna manera, así como excluiremos la posibilidad de tiempos de ejecución ridículamente grandes. Pero inicialmente, queremos evitar tales restricciones para obtener el caso límite, donde limitaciones prácticas sobre el tiempo de ejecución o espacio de memoria son eliminadas. Es bien conocido que en el mundo real la rapidez y capacidad de las computadoras ha estado creciendo sostenidamente. Queremos ignorar la rapidez y capacidad reales, en su lugar queremos preguntar cuáles son los límites puramente teóricos.

La descripción precedente de procedimiento efectivo es ciertamente vaga e imprecisa. En la siguiente sección, veremos como se puede hacer precisa esta descripción vaga, como el concepto se puede hacer un concepto *matemático*. No obstante, la idea informal de lo que se puede hacer mediante un procedimiento efectivo, esto es, lo que es *calculable*, puede ser muy útil. El rigor y la precisión pueden esperar hasta el *siguiente* capítulo. Primero necesitamos un sentido de hacia dónde vamos.

Por ejemplo, cualquier conjunto finito de números naturales debe ser decidible. El programa para el procedimiento de decisión puede incluir simplemente una lista de todos los números en el conjunto. Luego dado un número, el programa puede checarlo contra la lista. Así, el concepto de decidibilidad sólo es interesante para conjuntos infinitos.

Nuestra descripción de procedimientos efectivos, vaga como es, ya muestra que tan limitante es el concepto de decidibilidad. Uno puede, por ejemplo, utilizar los conceptos de conjuntos contables e incontables (ver el apéndice para un resumen de estos conceptos). No es difícil ver que la cantidad de posibles secuencias finitas de instrucciones, que uno puede escribir, (digamos que usando un teclado estándar) es numerable. Pero existe una cantidad incontable de conjuntos de números naturales (por el argumento diagonal de Cantor). Se sigue que casi todos los conjuntos, en un sentido, son *indecidibles*.

El hecho de que no todo conjunto es decidible es relevante a la teoría de la computación. El hecho de que existe un límite a lo que puede ser realizado por un procedimiento efectivo significa que existe un límite a lo que puede -aún en principio- hacerse mediante programas de computadora. Y esto dispara las preguntas: ¿Qué se puede hacer? ¿Qué no se puede hacer?

Históricamente, la teoría de la computabilidad surgió antes del desarrollo de las computadoras digitales. La teoría de la computabilidad es relevante a ciertas consideraciones en lógica matemática. En el corazón de la actividad matemática está la demostración de teoremas. Considere lo que se requiere para que una cadena de símbolos constituya una "demostración matemática aceptable". Antes de aceptar una demostración, y agregar el resultado que fue demostrado a nuestro almacén de conocimiento matemático, insistimos en que la demostración sea *verificable*.

Esto es, debe ser posible que otro matemático, tal como el árbitro del del artículo que contiene la demostración, cheque, paso a paso, la corrección de la demostración. Eventualmente, el árbitro o concluye que la demostración en realidad es correcta o concluye que la prueba contiene un hueco o un error y que por lo tanto aún no es aceptable. Esto es, el conjunto de demostraciones matemáticas aceptables -pensadas como cadenas de símbolos- debe ser decidible. Este hecho veremos (en un capítulo posterior) que tiene consecuencias significativas para lo que puede y no puede ser demostrado. Concluimos que la teoría de la computabilidad es relevante a los fundamentos de las matemáticas. Pero si los lógicos no hubieran inventado el concepto de computabilidad, los computólogos lo hubieran hecho más tarde.

1.1.2. Funciones calculables

Antes de continuar, debemos ampliar nuestras consideraciones sobre conjuntos decidibles e indecidibles y extenderlas a la situación más general de las *funciones parciales*. Sea $\mathbb{N} = \{0, 1, 2, \ldots\}$ el conjunto de números naturales. Entonces, un ejemplo de una función de aridad-dos sobre $\mathbb{N}$ es la función sustracción

$$g(m,n) = \begin{cases} m-n & \text{si } m \geq n \\ 0 & \text{en otro caso} \end{cases}$$

(donde hemos evitado números negativos). Una función sustracción diferente es la función "parcial"

$$f(m,n) = \begin{cases} m-n & \text{si } m \geq n \\ \uparrow & \text{en otro caso} \end{cases}$$

donde "$\uparrow$" indica que la función está indefinida. Así $f(5,2) = 3$, pero $f(2,5)$ está indefinida, el par $(2,5)$ no está en el dominio de f.

En general, decimos que una *función parcial* de aridad-k sobre $\mathbb{N}$ es una función cuyo dominio es algún conjunto de tuplas-k de números naturales y cuyos valores son números naturales. En otras palabras, para una función parcial f de aridad-k y una tupla-k $\langle x_1, \ldots, x_k \rangle$, posiblemente $f(x_1, \ldots, x_k)$ esté definida (es decir, $\langle x_1, \ldots, x_k \rangle$ está en el dominio de f), en cuyo caso el valor de la función está en $\mathbb{N}$, y posiblemente $f(x_1, \ldots, x_k)$ esté indefinida (es decir, $\langle x_1, \ldots, x_k \rangle$ no está en el dominio de f).

En un extremo, existen funciones parciales cuyos dominios son el conjunto $\mathbb{N}^k$ de todas las tuplas-k; de tales funciones se dice que son *totales*. (El adjetivo "parcial" cubre tanto a las funciones que son totales como a las funciones que no son totales.) En el otro extremo, existe la función vacía, esto es, la función que está definida en ninguna parte. La función vacía puede no parecer particularmente útil, pero cuenta como una de las funciones parciales de aridad-k.

Para una función parcial f de aridad-k, decimos que f es una *función parcial calculable efectivamente* si existe un procedimiento efectivo con la siguiente propiedad:

- Dada una tupla-k $\vec{x}$ en el dominio de f, el procedimiento eventualmente regresa el valor correcto de $f(\vec{x})$ y para.
- Dada una tupla-k $\vec{x}$ *no* en el dominio de f, el procedimiento no regresa un valor y no para.

(Aquí hay detalle: ¿Cómo se puede dar un número? Para comunicar un número x al procedimiento, enviamos el *numeral* para x). Los numerales son trozos de lenguage que se pueden comunicar. Los números no. La comunicación requiere lenguage. Sin embargo, continuaremos hablando de sean "m y n números dados" y así sucesivamente. Pero en unos pocos puntos, necesitaremos ser más exáctos y tomar en cuenta el hecho de que a un procedimiento se le dan numerales. Hubo un tiempo en los 1960s cuando, como parte de la "nueva matemática", a los profesores de educación básica se les animó para que distinguieran cuidadosamente entre números y numerales. Esta fue una buena idea que no funcionó.)

Por ejemplo, la función parcial para sustracción

$$f(m,n) = \begin{cases} m-n & \text{si } m \geq n \\ \uparrow & \text{en otro caso} \end{cases}$$

es calculable efectivamente, los procedimientos para calcularla, usando numerales en base-10, se enseñan en educación básica.

La función vacía es calculable efectivamente. El procedimiento efectivo para ella, dada una tupla-k, no necesita hacer algo en particular. Pero no debe parar ni regresar un valor.

El concepto de decidibilidad puede ahora ser descrito en términos de funciones. Para un subconjunto S de $\mathbb{N}^k$, podemos decir que S es *decidible* ssi su función característica

$$C_S(\vec{x}) = \begin{cases} \text{Sí} & \text{si } \vec{x} \in S \\ \text{No} & \text{si } \vec{x} \notin S \end{cases}$$

(la cual siempre es total) es calculable efectivamente. Aquí "Sí" y "No" son números fijos de $\mathbb{N}$, tales como 1 y 0.

(Esa palabra "ssi" en el párrafo anterior significa "si y sólo si". Esta es un trozo de jerga matemática que ha demostrado ser tan útil que se ha convertido en una parte estándar de la forma de hablar en matemáticas.)

Aquí, si $k = 1$, entonces S es un conjunto de números. Si $k = 2$, entonces tenemos el concepto de una relación binaria decidible sobre números, y así sucesivamente. Tome, por ejemplo, la relación de divisibilidad, esto es, el conjunto de pares $\langle m, n \rangle$ tales que m divide a n exactamente. (Para que todo esté definido, asuma que 0 divide sólo a sí mismo.) La relación de divisibilidad es decidible ya que, dados m y n, podemos efectuar el algoritmo de división que todos aprendimos en cuarto año y ver si el residuo es cero o distinto de cero.

Ejemplo 1.1.1 Cualquier función total constante sobre $\mathbb{N}$ es calculable efectivamente. Suponga, for ejemplo, $f(x) = 36$ para todo $x \in \mathbb{N}$. Hay un procedimiento obvio para calcular f; ignora la entrada y escribe "36" como salida. Esto puede parecer una trivialidad, pero compárelo con el siguiente ejemplo.

Ejemplo 1.1.2 Defina la función F como sigue

$$F(x) = \begin{cases} 1 & \text{si la conjetura de Goldbach es verdadera} \\ 0 & \text{si la conjetura de Goldbach es falsa} \end{cases}$$

La conjetura de Goldbach enuncia que todo entero par mayor que 2 es la suma de dos primos; por ejemplo, $22 = 5 + 17$. Esta conjetura sigue

siendo un problema abierto en matemáticas. ¿Esta función F es calculable efectivamente? (Elija su respuesta antes de leer el siguiente párrafo.)

Observe que F es una función total constante. (La lógica clásica entra aquí: O existe un número par que sirva como contraejemplo o no existe.) Así como se señaló en el ejemplo anterior, F es calculable efectivamente. ¿Cuál es entonces un procedimiento para calcular F? No lo sé, pero te puedo dar dos procedimientos teniendo la confianza de que uno de ellos calcula F.

El punto de este ejemplo es que la calculabilidad efectiva es una propiedad de la función en sí misma, no es una propiedad de alguna descripción lingüística usada para especificar la función. (Uno dice que la propiedad de calculabilidad efectiva es *extensional*.) Existen muchas frases en Español que servirían para definir F. Para que una función sea calculable efectivamente, debe existir (en sentido matemático) un procedimiento efectivo para calcularla. Que no es lo mismo que decir que tienes ese procedimiento en la mano. Si en el año 2083, alguna criatura en el universo prueba (o refuta) la conjetura de Goldbach, entonces eso no significa que F cambie de repente de no calculable a calculable. Siempre fue calculable.

No obstante, posteriormente habrá situaciones en las que querremos más que la mera existencia un procedimiento efectivo P; querremos alguna manera para encontrar realmente a P, dadas algunas pistas apropiadas. Esto es para después.

Es muy natural extender estos conceptos a la situación donde tenemos la mitad de decidibilidad: Decimos que S es *semidecidible* si su "función semicaracterística"

$$c_S(\vec{x}) = \begin{cases} \text{Sí} & \text{si } \vec{x} \in S \\ \uparrow & \text{si } \vec{x} \notin S \end{cases}$$

es una función parcial calculable efectivamente. Así, un conjunto S de números es semidecidible si existe un procedimiento efectivo para *reconocer* a los miembros de S. Podemos pensar en S como el conjunto que el procedimiento *acepta*. Y el procedimiento efectivo, si bien no es un procedimiento de decisión, es al menos un procedimiento de *aceptación*.

Cualquier conjunto decidible es también semidecidible. Si tenemos un procedimiento efectivo que calcula la función característica C_S, entonces podemos convertirlo en un procedimiento efectivo que calcule la función semicaracterística c_S. Simplemente reemplazamos cada instrucción "escribe No" por un ciclo interminable. O más informalmente, simplemente destornillamos el foco No.

¿Qué sobre la inversa? ¿Existen conjuntos semidecidibles que no son decidibles? Veremos que en verdad existen. El problema con la función semicaracterística es que nunca produce una respuesta No. Suponga que hemos estado calculando $c_S(\vec{x})$ por 37 años y el procedimiento aún no ha terminado. ¿Debemos rendirnos y concluir que $\vec{x}$ no está en S? O quizás trabajando sólo otros diez minutos produciría la información de que $\vec{x}$ pertenece a S. No hay, en general, manera de saberlo.

Aquí hay otro ejemplo de una función parcial calculable:

$$F(n) = \text{el } p > n \text{ más pequeño tal que tanto } p \text{ como } p+2 \text{ son primos}$$

Aquí se ha de entender que $F(n)$ está indefinido si no existe un número p como se describió; así F podría no ser total. Por ejemplo, $F(9) = 11$ ya que ambos 11 y 13 son primos. No se sabe si F es total o no. La "conjetura de los primos gemelos", la que dice que existen infinitos pares de primos que difieren en 2, es equivalente a la afirmación de que F es total. La conjetura de los primos gemelos es aún un problema abierto. Sin embargo, podemos estar seguros de que F es calculable efectivamente. Un procedimiento para calcular $F(n)$ es como sigue. "Dado n, primero defina $p = n + 1$. Luego verifique si p y $p + 2$ son o no primos. Si lo son, entonces pare y dé la salida p. Si no, incremente p y continue." ¿Qué si n es enorme, digamos, $n = 10^{10^{10}}$? Por un lado si existe un par de primos más grande, entonces este procedimiento encontrará al primero y para con la salida correcta. Por otro lado, si no hay un par de primos más grande, entonces el procedimiento nunca para, así nunca da una respuesta. Lo que es correcto; ya que $F(n)$ está indefinida, el procedimiento no debe darnos alguna respuesta.

Ahora suponga que modificamos el ejemplo. Considere la función total:

$$G(n) = \begin{cases} F(n) & \text{si } F(n)\downarrow \\ 0 & \text{en otro caso} \end{cases}$$

Aquí "$F(n)\downarrow$" significa que $F(n)$ está definida, así que n pertenece al dominio de F. Entonces la función G *también* es calculable efectivamente. Esto es, *existe* un programa que calcula G correctamente.

La conjetura de los primos gemelos es verdadera o falsa: O hay infinitos pares de primos, o hay un par más grande. (En este punto, la lógica clásica entra una vez más.) En el primer caso, $F = G$ y el procedimiento efectivo para F también calcula G. En el segundo caso, G es eventualmente la función

constante 0. Y cualquier función eventualmente constante es calculable (el procedimiento puede utilizar una tabla para la parte finita de la función antes de que se estabilice).

Así en cualquier caso, *existe* un procedimiento efectivo para G. Eso no es lo mismo que conocer el procedimiento. Este ejemplo indica una vez más la diferencia entre conocer que un cierto procedimiento efectivo existe y tener el procedimiento efectivo en nuestras manos (o tener razones convincentes para saber que el procedimiento en nuestras manos funcionará).

Un programa de una persona es otro dato de esa persona. Éste es el principio detrás de los sistemas operativos (y detrás de la idea de un programa de computadora almacenado). El programa favorito de uno es, para el sistema operativo, otra pieza de dato para ser recibida como entrada y procesarla. El sistema operativo está calculando los valores de una función "universal" de aridad-2. (Históricamente, ¡el flujo de ideas fue exactamente en la dirección opuesta! La siguiente digresión expande este punto.)

Digresión: El concepto de programa de computadora de propósito general almacenado es ahora muy común, pero el concepto se desarrolló lentamente sobre un periodo de tiempo. ¡La máquina ENIAC, la computadora más importante en los 1940s, era programada moviendo interruptores e insertando cables en tableros con contactos! Esto está muy lejos de tratar un programa como dato. Fue von Neumann quien, en un reporte técnico de 1945, estableció las ideas cruciales para un programa de computadora de propósito general almacenado, esto es, para una computadora universal. El artículo de Turing de 1936 sobre lo que ahora es llamado máquinas de Turing ha demostrado la existencia de una "máquina universal de Turing" para calcular la función Φ descrita posteriormente. Cuando Turing fue a Princeton en 1936-37, von Neumann estubo ahí y debió de estar consciente de su trabajo. Aparentemente, el pensamiento de von Neumann en 1945 fue influenciado por el trabajo de Turing de casi una década antes.

Suponga que adoptamos un método fijo para codificar cualquier conjunto de instrucciones mediante un solo número natural. (Primero, convertimos las instrucciones a una cadena de 0's y 1's -uno siempre hace esto con programas de computadora- y luego consideramos a la cadena como el nombre de un número natural en notación base-2.) Entonces la "función universal"

$$\Phi(w, x) = \begin{array}{l}\text{el resultado de aplicar las instrucciones} \\ \text{codificadas mediante } w \text{ a la entrada } x\end{array}$$

es una función parcial calculable efectivamente (donde se entiende que $\Phi(w, x)$

está indefinida si al aplicar las instrucciones codificadas mediante w a la entrada x no para y no regresa una salida). Aquí están las instrucciones para Φ: "Dados w y x, decodifica w para ver que hay que hacer con x y luego lo hace" Por supuesto, la función Φ no es total. Por una cosa, cuando tratamos de decodificar w, podriamos encontrar un completo sin sentido, así que la instrucción "luego lo hace" nos lleva a ningún lado. Y aún si al decodificar w obtenemos instrucciones explícitas y comprensibles, el aplicar esas instrucciones a un x en particular podría nunca producir una salida.

(Este razonamiento será repetido en el Capítulo 3, cuando tengamos material más concreto con el que tratar. Pero las ideas rectoras serán las mismas.)

La función parcial de aridad-dos Φ es "universal" en el sentido de que *cualquier* función parcial f efectivamente calculable está dada por la ecuación

$$f(x) = \Phi(e, x) \text{ para todo } x$$

donde e codifica las instrucciones de f. Será útil introducir aquí una notación especial: Sea $[\![e]\!]$ la función parcial de aridad-uno definida por la ecuación

$$[\![e]\!](x) = \Phi(e, x)$$

Esto es, $[\![e]\!]$ es la función parcial cuyas instrucciones están codificadas por e, en el entendido de que algunos valores de e podrían codificar algo no sensato, la función $[\![e]\!]$ podría ser la función vacía. En cualquier caso, $[\![e]\!]$ es la función parcial que obtenemos de Φ, cuando fijamos su primer variable en e. Así,

$$[\![0]\!], [\![1]\!], [\![2]\!], \ldots$$

es una lista completa (con repeticiones) de todas las funciones parciales calculables efectivamente. Los valores de $[\![e]\!]$ están dados por el $(e+1)-$ésimo renglón en la siguiente tabla:

$[\![0]\!]$	$\Phi(0,0)$	$\Phi(0,1)$	$\Phi(0,2)$	$\Phi(0,3)$	$\cdots$
$[\![1]\!]$	$\Phi(1,0)$	$\Phi(1,1)$	$\Phi(1,2)$	$\Phi(1,3)$	$\cdots$
$[\![2]\!]$	$\Phi(2,0)$	$\Phi(2,1)$	$\Phi(2,2)$	$\Phi(2,3)$	$\cdots$
$[\![3]\!]$	$\Phi(3,0)$	$\Phi(3,1)$	$\Phi(3,2)$	$\Phi(3,3)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Usando la función parcial universal Φ, podemos construir una relación binaria *indecidible*, la relación *de paro* H:

$$\begin{aligned} \langle w, x\rangle \in H &\Leftrightarrow \Phi(w, x)\downarrow \\ &\Leftrightarrow \text{al aplicar las instrucciones codificadas por } w \\ &\quad\ \text{a la entrada } x \text{ para} \end{aligned}$$

Desde el lado positivo, H es semidecible. Para calcular la función semicaracterística $c_H(w, x)$, dados w y x, primero calculamos $\Phi(w, x)$. Si y cuando ésta regrese un valor y pare, damos la salida "Sí" y paramos.

Desde el lado negativo, H no es decidible. Para ver esto, primero considere la siguiente función parcial:

$$f(x) = \begin{cases} \text{Sí} & \text{si } \Phi(x, x)\uparrow \\ \uparrow & \text{si } \Phi(x, x)\downarrow \end{cases}$$

(Note que estamos usando la construcción diagonal clásica. Mirando la anterior tabla de los valores de Φ ordenados en un arreglo bi-dimensional, uno ve que f ha sido construida yendo a lo largo de la diagonal de la tabla, tomando la entrada $\Phi(x, x)$ encontrada allí, y asegurando que $f(x)$ difiera de ella.)

Hay dos cosas que decir sobre f. Primero, f no puede ser calculable efectivamente. Considere cualquier conjunto de instrucciones que pudiera calcular f. Dichas instrucciones tienen algún número de código k y así calcula la función $[\![k]\!]$. ¿Pudiera ser la misma que f? No, f y $[\![k]\!]$ difieren en la entrada k. Esto es, f ha sido construida de tal manera que $f(k)$ difiere de $[\![k]\!](k)$, ellas difieren ya que una está definida y la otra no. Así estas instrucciones no pueden calcular correctamente a f; ellas producen el resultado equivocado en la entrada k. Y como k fue arbitrario, estamos forzados a concluir que *ningún* conjunto de instrucciones puede calcular correctamente a f. (Este es nuestro primer ejemplo de una función parcial que no es calculable efectivamente. Existen muchísimas más, como veremos.)

En segundo lugar, podemos argumentar que *si* tuvieramos un procedimiento de decisión para H, entonces podriamos calcular f. Para calcular $f(x)$, usamos primero el procedimiento de decisión H para decidir si $\langle x, x\rangle \in H$ o no. Si no, entonces $f(x) =$ Sí. Pero si $\langle x, x\rangle \in H$, entonces el procedimiento para encontrar $f(x)$ caería en un ciclo infinito ya que $f(x)$ está indefinida.

Juntando estas dos observaciones sobre f, concluimos que no puede existir un procedimiento de decisión para H. El hecho de que H es indecidible

se expresa usualmente diciendo que "el problema de paro es irresoluble o también que es insoluble"; es decir, no podemos en general determinar efectivamente, dados w y x, si al aplicar las instrucciones codificadas por w a la entrada x eventualmente terminará o seguirá por siempre: **Insolubilidad del problema de paro:** La relación

$$\{\langle w, x\rangle | \text{ al aplicar las instrucciones codificadas por } w \text{ a la entrada } x \text{ para}\}$$

es semidecidible pero no decidible.

La función f del argumento anterior

$$f(x) = \begin{cases} \text{Sí} & \text{si } \Phi(x,x) \uparrow \\ \uparrow & \text{si } \Phi(x,x) \downarrow \end{cases}$$

es la función semicaracterística del conjunto $\{x|\Phi(x,x)\uparrow\}$. Ya que su función semicaraterística no es calculable efectivamente, podemos concluir que este conjunto *no* es semidecidible.

Sea K el complemento de este conjunto:

$$K = \{x|\Phi(x,x)\downarrow\} = \{x|[\![x]\!](x)\downarrow\}$$

Este conjunto es semidecidible. ¿Cómo podriamos calcular $c_K(x)$, dado x? Tratamos de calcular $\Phi(x,x)$ (lo cual es posible ya que Φ es una función parcial calculable efectivamente). Si y cuando el cálculo regrese una salida y pare, damos la salida "Sí" y paramos. Hasta ese momento seguimos intentando. (Este argumento es el mismo que vimos para la semidecidibilidad de H. Y $x \in K \Leftrightarrow \langle x, x\rangle \in H$.)

Teorema de Kleene: *Un conjunto (o una relación) es decidible si y sólo si él y su complemento son semidecidibles.*

Si estamos trabajando con conjuntos de números, entonces el complemento es con respecto a $\mathbb{N}$; si estamos trabajando con una relación de aridad-k, entonces el complemento es con respecto a $\mathbb{N}^k$.

Prueba: Por un lado, si un conjunto S es decidible, entonces su complemento $\overline{S}$ también es decidible, simplemente intercambiamos el "Sí" y el "No". Así tanto S como su complemento $\overline{S}$ son semidecidibles ya que los conjuntos decidibles son también semidecidibles.

Por otro lado, suponga que S es un conjunto para el que tanto c_S como $c_{\overline{S}}$ son calculables efectivamente. La idea es poner juntas estas dos mitades de un procedimiento de decisión para hacer uno completo. Digamos que queremos

encontrar $C_S(x)$, dado x. Necesitamos organizar nuestro tiempo. Durante los minutos impares, corremos nuestro programa para $c_S(x)$. Durante los minutos pares, corremos nuestro programa para $c_{\overline{S}}(x)$. Por supuesto, al final de cada minuto, almacenamos por ahí lo que hemos hecho, así podemos posteriormente recogerlo de donde lo hayamos dejado.

Eventualmente, debemos recibir un "Sí". Si es durante un minuto impar, encontramos que $c_S(x) =$ Sí (esto debe suceder eventualmente si $x \in S$), luego damos la salida "Sí" y paramos. Y si es durante un minuto par, encontramos que $c_{\overline{S}}(x) =$ Sí (esto debe suceder eventualmente si $x \notin S$), luego damos la salida "No" y paramos.

(Alternativamente, podemos imaginar que trabajamos de manera ambidiestra. Con la mano izquierda, trabajamos calculando $c_S(x)$; con la mano derecha, trabajamos con $c_{\overline{S}}(x)$. Eventualmente, una mano descubre la respuesta.) $\dashv$

El conjunto K es un ejemplo de un conjunto semidecidible que no es decidible. Su complemento $\overline{K}$ no es semidecidible; hemos visto que su función semicaracterística f no es calculable efectivamente.

La conexión entre funciones parciales calculables efectivamente y conjuntos semidecidibles puede describirse mejor como sigue:

Teorema:

(i) *Una relación es semidecidible si y sólo si es el dominio de alguna función parcial calculable efectivamente.*

(ii) *Una función parcial f es una función parcial calculable efectivamente si y sólo si su gráfica G (es decir, el conjunto de tuplas $\langle \vec{x}, y \rangle$ tales que $f(\vec{x}) = y$) es una relación semidecidible.*

Prueba: Para la proposición (i), en una dirección es verdadera por definición: Cualquier relación es el dominio de su función característica, y para una relación semidecidible, esa función es una función parcial calculable efectivamente.

Recíprocamente, para una función parcial calculable efectivamente, f, tenemos el procedimiento de semidecisión natural para su dominio: Dado $\vec{x}$, tratamos de calcular $f(\vec{x})$. Si y cuando tengamos éxito para encontrar $f(\vec{x})$, ignoramos el valor y simplemente decimos Yes y paramos.

Para probar (ii) en una dirección, suponga que f es una función parcial calculable efectivamente. Aquí tenemos un procedimiento de semidecisión para su gráfica G: Dado $\langle \vec{x}, y \rangle$, procedemos a calcular $f(\vec{x})$. Si y cuando

obtenemos el resultado, chacamos para ver si es y o no. Si el resultado en realidad es y, entonces decimos Sí y paramos.

Por supuesto, este procedimiento falla al tratar de dar una respuesta si $f(x) \uparrow$, lo cual es exactamente como debe ser, ya que en este caso, $\langle \vec{x}, y \rangle$ no está en la gráfica.

Para probar la otra dirección de (ii) , suponga que tenemos un procedimiento de semidecisión para la gráfica G. Buscamos calcular, dado $\vec{x}$, el valor $f(\vec{x})$, si éste está definido. Nuestro plan es checar $\langle \vec{x}, 0 \rangle, \langle \vec{x}, 1 \rangle, \langle \vec{x}, 2 \rangle, \ldots$, para ver su membresia a G. Pero para organizar nuestro tiempo sensatamente, usamos un procedimiento llamado "machihembrar". Aquí está lo que hacemos:

1. Usa un minuto probando si $\langle \vec{x}, 0 \rangle \in G$.

2. Usa dos minutos probando si $\langle \vec{x}, 0 \rangle \in G$ y dos minutos probando si $\langle \vec{x}, 1 \rangle \in G$.

3. Similarmente, usa tres minutos en cada uno de $\langle \vec{x}, 0 \rangle, \langle \vec{x}, 1 \rangle$ y $\langle \vec{x}, 2 \rangle$.

Y así sucesivamente. Si y cuando descubramos que, en efecto, $\langle \vec{x}, k \rangle \in G$, entonces regresamos el valor k y paramos. Observe que siempre que $f(\vec{x}) \downarrow$, tarde o temprano el procedimiento anterior determinará correctamente $f(\vec{x})$ y parará. Por supuesto, si $f(x) \uparrow$, entonces el procedimiento se ejecutará por siempre. $\dashv$

1.1.3. Tesis de Church

No obstante el concepto de calculabilidad efectiva ha sido descrito aquí en términos algo vagos, en la siguiente sección describiremos un concepto preciso (matemáticamente) de una "función parcial computable". En efecto, se describirán varias maneras equivalentes para formular el concepto en términos precisos. Se argumentará que el concepto matemático de función parcial computable es la formalización *correcta* del concepto informal de una función parcial calculable efectivamente. Esta afirmación es conocida como la *tesis de Church* o la *tesis de Church-Turing*.

La tesis de Church, relaciona una idea informal con una idea formal, no es en sí una proposición matemática capaz de ser probada. Pero uno puede buscar evidencia a favor o en contra de la tesis de Church, todo resulta ser evidencia a favor.

Una pieza de evidencia es la ausencia de contraejemplos. Esto es, cualquier función examinada hasta ahora que los matemáticos han creido que es calculable efectivamente, se ha encontrado que es computable.

Evidencia más fuerte deriva de varios intentos que diferentes personas hicieron independientemente, tratando de formalizar la idea de calculabilidad efectiva. Alonzo Church uso el cálculo$-\lambda$; Alan Turing uso un agente de cálculo idealizado (posteriormente llamado maquina de Turing); Emil Post desarrollo un enfoque similar. Extraordinariamente, todos estos intentos resultaron ser equivalentes, en que todos ellos definieron exactamente la misma clase de funciones, llamadas ¡funciones parciales computables!

El estudio de la calculabilidad efectiva se inició en 1930s con el trabajo en lógica matemática. Como se ha notado previamente, el asunto está relacionado con el concepto de una *prueba aceptable*. Más recientemente, el estudio de la calculabilidad efectiva ha formado una parte escencial de la teoría de la computación. Un científico de la computación prudente seguramente querrá saber que, además de las dificultades que el mundo real presenta, existe un límite puramente teórico para la calculabilidad.

Ejercicio 1.1.1 Realice lo que se indica:

1. Asuma que S es un conjunto de número naturales pero que es finito. (Esto es, S es un subconjunto cofinito de $\mathbb{N}$). Explique por qué S debe ser decidible.

2. Asuma que A y B son conjuntos decidibles de números naturales. Explique por qué su intersección $A \cap B$ es también decidible. (Describa un procedimiento efectivo para determinar si un número dado está o no en $A \cap B$.)

3. Asuma que A y B son conjuntos decidibles de números naturales. Explique por qué su unión $A \cup B$ es también decidible.

4. Asuma que A y B son conjuntos semidecidibles de números naturales. Explique por qué su intersección $A \cap B$ es también semidecidible.

5. Asuma que A y B son conjuntos semidecidibles de números naturales. Explique por qué su unión $A \cup B$ es también semidecidible.

6. *a*) Asuma que R es una relación binaria decidible sobre los números naturales. Esto es, es una relación decidible de aridad-2. Explique

por qué su dominio, $\{x|\langle x, y\rangle \in R$ para algún $y\}$, es un conjunto semidecidible.

b) Ahora suponga que en vez de asumir que R es decidible, asumimos sólo que es semidecidible. ¿Sigue siendo cierto que su dominio debe ser semidecidible?

7. *a*) Asuma que f es una función total calculable de aridad-uno. Explique por qué su gráfica es una relación binaria decidible.

   b) Inversamente, muestre que si la gráfica de una función total de aridad-uno f es decidible, entonces f debe ser calculable.

   c) Ahora asuma que f es una función parcial calculable de aridad-uno, no necesariamente total. Explique por qué su dominio, $\{x \in \mathbb{N}|f(x)\downarrow\}$, es semidecidible.

8. Asuma que S es un conjunto decidible de números naturales y que f es una función *total* calculable efectivamente sobre $\mathbb{N}$. Explique por qué $\{x|f(x) \in S\}$ es decidible. (Este conjunto es llamado la *imagen inversa* de S bajo f.)

9. Asuma que S es un conjunto semidecidible de números naturales y que f es una función parcial calculable efectivamente sobre $\mathbb{N}$. Explique por qué

$$\{x|f(x)\downarrow \text{ y } f(x) \in S\}$$

   es semidecidible.

10. En la expansión decimal de π, podría haber una cadena de algunos 7's consecutivos. Defina la función f tal que $f(x) = 1$ si existe una cadena de x o más 7's y $f(x) = 0$ en otro caso:

$$f(x) = \begin{cases} 1 & \text{si } \pi \text{ tiene una cadena con } x \text{ o más 7's} \\ 0 & \text{en otro caso} \end{cases}$$

    Explique, sin usar algún hecho especial sobre π o teoría de números, por qué f es calculable efectivamente.

11. Asuma que g es una función total no creciente (esto es, $g(x) \geq g(x+1)$ para todo x) sobre $\mathbb{N}$. Explique por qué g debe ser calculable efectivamente.

12. Asuma que f es una función total sobre los números naturales y que f es eventualmente periódica. Esto es, existen números m y p tales que para todo x mayor que m, tenemos que $f(x+p) = f(x)$. Explique por qué f es calculable efectivamente.

13. *a*) Asuma que f es una función total calculable efectivamente sobre los números naturales. Explique por qué el rango de f (esto es, el conjunto $\{f(x)|x \in \mathbb{N}\}$) es semidecidible.

    b) Ahora suponga que f es una función parcial calculable efectivamente (no necesariamente total). ¿Sigue siendo cierto que su rango debe ser semidecidible?

    c) Asuma que f y g son funciones parciales calculables efectivamente sobre $\mathbb{N}$. Explique por qué el conjunto

    $$\{x|f(x) = g(x) \text{ y ambos están definidos}\}$$

    es semidecidible.

1.2. Fomalizaciones - Un Panorama

En la sección anterior, el concepto de calculabilidad efectiva se describió muy informalmente. Ahora queremos hacer aquellas ideas precisas (es decir, hacerlas parte de las matemáticas). En efecto, varios enfoques para hacerlo serán descritos: dispositivos de cálculo idealizados, definiciones generativas (es decir, la clase más pequeña que contiene ciertas funciones iniciales y que es cerrada bajo ciertas construcciones), lenguajes de programación y definibilidad en lenguajes formales. Es un hecho significativo que estos enfoques, a pesar de ser muy distintos, producen conceptos exactamente equivalentes.

Esta sección da un panorama general de un número de maneras diferentes (pero equivalentes) de formalizar los conceptos de calculabilidad efectiva. En capítulos posteriores se desarrollarán con todo detalle algunas de estas maneras.

Digresión: El libro de Rogers de 1967 citado en las Referencias demostró que el concepto de computabilidad se puede desarrollar *sin* adoptar alguna de estas formalizaciones. Ese libro fue precedido por una versión preliminar mimeografiada en 1956, en la que primero vi este concepto. Todavía existen pocas copias atesoradas de la edición mimeografiada.

1.2.1. Máquinas de Turing

A principios de 1935, Alan Turing era un estudiante de 22 años de edad graduado en el King's College en Cambridge. Bajo la asesoría de Max Newman, estuvo trabajando en el problema de formalizar el concepto de calculabilidad efectiva. En 1936, se enteró del trabajo de Alonzo Church, en Princeton. Church también había estado trabajando sobre el mismo problema, y en su artículo de 1936, "Un problema insoluble de teoría elemental de números", presentó la afirmación categórica de que la clase de funciones calculables efectivamanete debe ser idéntica a la clase de funciones definibles en el *cálculo lambda*, un lenguaje formal para especificar la construcción de funciones. Aún más, Church mostró que exactamente la misma clase de funciones puede ser caracterizada en términos de derivabilidad formal de ecuaciones.

Luego Turing terminó de escribir rápidamente su artículo, en el cual presentó un enfoque muy diferente para caracterizar las funciones calculables efectivamente, pero uno que -como demostró- una vez más producía la misma clase de funciones que Church había propuesto. Con el estímulo de Newman, Turing fue a Princeton por dos años, donde escribió su tesis doctoral bajo la dirección de Alonzo Church.

El artículo de Turing sigue siendo una introducción a sus ideas muy legible. ¿Cómo podría un empleado diligente realizar un cálculo siguiendo instrucciones? Él o ella podría organizar el trabajo en una libreta. En cualquier momento dado, su atención se centra en una página en particular. Siguiendo sus instrucciones, podría alterar la página, y entonces podría cambiar a otra página. Y como la libreta es suficientemente grande (o el suministro de papel nuevo es suficientemente basto) nunca llega a la última página.

El alfabeto de símbolos disponibles para el empleado debe ser finito; si hubiera infinitos símbolos, entonces habrían dos que serían arbitrariamente similares y se podría confundir. Luego podemos sin perdida de generalidad pensar que podemos escribir sobre una página de la libreta un solo símbolo. Y podemos imaginar que las páginas de la libreta están puestas una al lado de la otra, formando una cinta de papel, que consiste de cuadrados, cada cuadrado está en blanco o tiene impreso un símbolo. (Por uniformidad, podemos pensar que un cuadrado blanco contiene el símbolo "blanco" B.) En cada etapa de su trabajo, el empleado -o la máquina mecánica- puede alterar el cuadrado que está examinando, puede poner atención al siguiente cuadrado o al previo, y puede ver las instrucciones para saber cual de ellas debe seguir a continuación. Turing describió la última parte como un "cambio de estado mental".

Turing escribió, "Ahora podemos construir una máquina para hacer el trabajo". Tal máquina es, por supuesto, ahora llamada *máquina de Turing*, una frase que usó primero Church en su revisión al artículo de Turing que apareció en *The Journal of Simbolic Logic*. La máquina tiene una cinta potencialmente infinita. Inicialmente el numeral o palabra dada de entrada se escribe en la cinta, la cual es blanca en los restantes cuadrados. La máquina es capaz de estar en cualquiera de los "estados" finitos (la palabra "mental" es inapropiada para un máquina).

En cada paso del cálculo, dependiendo del estado en que se encuentre en el momento, la máquina puede cambiar el símbolo en el cuadrado que está examinando en ese momento, y puede dirigir su atención al cuadrado a la izquierda o la derecha, y puede entonces cambiar su estado a otro estado. (La cinta se extiende sin fin en ambas direcciones.)

El programa para esta máquina de Turing se puede dar mediante una tabla. Donde los posibles estados de la máquina son $q_1, \ldots, q_r$, cada línea de la tabla es una quíntupla $\langle q_i, S_j, S_k, D, q_m \rangle$ la que será interpretada como que siempre que la máquina esté en el estado q_i y el cuadrado examinado contiene el símbolo S_j, entonces el símbolo debe ser cambiado a S_k y la máquina debe cambiar su atención al cuadrado a la izquierda (si $D = L$) o a la derecha (si $D = R$), y debe cambiar su estado a q_m. Posiblemente el símbolo S_j es el símbolo "blanco" B, lo que significa que el cuadrado examinado es blanco; posiblemente S_k es B, lo que significa que lo que sea que esté en el cuadrado tiene que ser borrado. Para que el programa no sea ambiguo, no deben haber dos quintuplas diferentes con los primeros dos componentes iguales. (Al relajar el requerimiento con respecto a la ausencia de ambigüedad, obtenemos el concepto de máquina de Turing *no determinista*, la que será útil posteriormente, en la discusión de computabilidad factible.) Uno de los estados, digamos, q_1, es designado como el estado inicial, el estado en el que la máquina empieza a calcular. Si empezamos a ejecutar la máquina en este estado, y examinamos el primer cuadrado de su entrada, podría (o tal vez no), después de algún número de pasos, alcanzar un estado y un símbolo para los cuales su tabla carece de una quintupla que tenga ese estado y ese símbolo como sus primeros dos componentes. En ese punto, la máquina *para*, y podemos mirar en la cinta (iniciando con el cuadrado que estaba bajo consideración) para ver que numeral o palabra de salida tiene.

Ahora suponga que Σ es un alfabeto finito (el símbolo B no cuenta como un miembro de Σ). Sea Σ^* el conjunto de todas las palabras sobre este alfabeto (esto es, Σ^* es el conjunto de todas las cadenas, incluyendo a la ca-

dena vacía, que consisten de miembros de Σ). Suponga que f es una función parcial de aridad-k de Σ^* en Σ^*. Diremos que f es *computable a la Turing* si existe una máquina de Turing $\mathcal{M}$ que, cuando empieza en su estado inicial escaneando el primer símbolo de una tupla$-k$ $\vec{w}$ de palabras (escritas sobre la cinta, con un cuadrado blanco entre las palabras, y con el resto de la cinta en blanco), se comporta como sigue:

- Si $f(\vec{w}) \downarrow$ (es decir, si $\vec{w} \in \text{dom}f$) entonces $\mathcal{M}$ eventualmente para, y en ese momento, está escaneando el símbolo más izquierdo de la palabra $f(\vec{w})$ (la cual es seguida por un cuadrado blanco).
- Si $f(\vec{w}) \uparrow$ (es decir, si $\vec{w} \notin \text{dom}f$) entonces $\mathcal{M}$ nunca para.

Ejemplo 1.2.1 Tome un alfabeto de dos letras $\Sigma = \{a, b\}$. Sea $\mathcal{M}$ la máquina de Turing dada por el siguiente conjunto con seis quintuplas[1], donde q_1 es designado como el símbolo inicial:

$$\langle q_1, a, a, R, q_1\rangle$$
$$\langle q_1, b, b, R, q_1\rangle$$
$$\langle q_1, B, a, L, q_2\rangle$$
$$\langle q_2, a, a, L, q_2\rangle$$
$$\langle q_2, b, b, L, q_2\rangle$$
$$\langle q_2, B, B, R, q_3\rangle$$

Suponga que empezamos esta máquina en el estado q_1, escaneando la primer letra de una palabra w. La máquina se mueve (en el estado q_1) hacia el final derecho de w, donde añade la letra a. Entonces se mueve (en el estado q_2) de regreso hacia el final izquierdo de la palabra, donde para (en el estado q_3). Así, $\mathcal{M}$ calcula la función total $f(w) = wa$.

Necesitamos adoptar convenciones especiales para manejar la palabra vacía λ, la cual ocupa cero cuadrados. Esto se puede hacer de diferentes maneras; la siguiente es la menera escogida. si la máquina para escaneando un cuadrado blanco, entonces la palabra de salida es λ. Para una función de aridad-uno f , para calcular $f(\lambda)$, simplemente iniciamos con una cinta en blanco. Para una función de aridad-dos g, para calcular $g(w, \lambda)$, iniciamos

[1]En el original hay un error, la quinta quintupla aparece como $\langle q_2, b, b, R, q_2\rangle$. Diga cuál es el comportamiento de esa máquina y qué calcula. N. del T.

sólo con la palabra w, escaneando el primer símbolo de w. Y para calcular $g(\lambda, w)$, también empezamos con solo la palabra w sobre la cinta, pero escaneando el cuadrado blanco justo a la izquierda de w. Y en general, para darle a una función de aridad-k la entrada $\vec{w} = \langle u_1, \ldots, u_k \rangle$ consistiendo de k palabras de longitudes $n_1, \ldots, n_k$,iniciamos la máquina escaneando el primer cuadrado de la configuración de entrada de longitud $n_1 + \cdots + n_k + k + 1$

$$(n_1 \text{ símbolos de } u_1)B(n_2 \text{ símbolos de } u_2)B \cdots B(n_k \text{ símbolos de } u_k)$$

con el resto de la cinta en blanco. Aquí cualquier n_i puede ser cero; en el caso extremo, todos pueden ser cero.

Un inconveniente obvio de estas convenciones es que no existe diferencia entre el par $\langle u, v \rangle$ y el triple $\langle u, v, \lambda \rangle$. Otras convenciones evitan este inconveniente, a costa de introducir su propia idiosincracia.

La definición de computabilidad a la Turing puede ser adaptada fácilmente a funciones parciales de aridad-k sobre $\mathbb{N}$. La manera más simple de hacerlo es usando numerales en base-1. Tomemos el alfabeto de una letra $\Sigma = \{|\}$ cuya única letra es el símbolo $|$. O para ser más convencionales, sea $\Sigma = \{1\}$, usando el símbolo 1 en lugar de $|$. Entonces la configuración inicial para el triple $\langle 3, 0, 4 \rangle$ es

$$111BB1111$$

Entonces la *tesis de Church*, también llamada -particularmente en el contexto de máquinas de Turing- *tesis de Church-Turing*, es la afirmación de que este concepto de computabilidad a la Turing es la formalización correcta del concepto informal de calculabilidad efectiva. Ciertamente la definición refleja las idea de seguir instrucciones predeterminadas, sin limitar la cantidad de tiempo que puede requerir. (El nombre "tesis de Church-Turing" obscurece el hecho de que Church y Turing siguieron caminos muy distintos para llegar a conclusiones equivalentes.)

La tesis de Church ha alcanzado aceptación universal. Kurt Gödel, escribiendo en 1964 sobre el concepto de "sistema formal" en lógica, implicando la idea de que el conjunto de deducciones correctas debe ser un conjunto decidible, dijo que "debido al trabajo de A. M. Turing, ahora se puede dar una definición precisa e incuestionablemente adecuada del concepto general de sistema formal". Y otros concuerdan.

La robustez del concepto de computabilidad a la Turing se evidencia por el hecho de que es insensible a ciertas modificaciones a la definición de una máquina de Turing. Por ejemplo, podemos imponer limitaciones al

tamaño del alfabeto, o podemos insistir en que la máquina nunca se mueva a la izquierda de su punto de inicio. Ninguna de éstas afectará a la clase de funciones parciales computables a la Turing.

Turing desarrollo estas ideas antes de la introducción de las computadoras modernas digitales. Después de la segunda guerra mundial, Turing jugó un papel activo en el desarrollo de las primeras computadoras, y en el campo emergente de la inteligencia artificial. (Durante la guerra, trabajó descifrando el código Enigma usado por los Alemanes en el campo de batalla, trabajo militarmente importante, el cual permaneció clasificado hasta después de la muerte de Turing.) Uno puede especular si Turing habría formulado sus ideas de manera diferente, si su trabajo lo hubiera hecho después de la introducción de las computadoras digitales.

Digresión: Hay un ejemplo interesante, que lleva el nombre[2] de "el problema del castor ocupado".

Suponga que queremos una máquina de Turing, iniciando con su cinta en blanco, que escriba tantos unos como pueda, y luego pare. Con un número limitado de estados, ¿cuántos 1's podemos obtener?

Para hacer las cosas más precisas, tome máquinas de Turing con el alfabeto $\{1\}$ (así, los únicos símbolos son B y 1). Permitiremos que tales máquinas tengan n estados, más un estado de paro (puede ocurrir como el último miembro de una quintupla, pero no como el primer miembro). Para cada n, sólo existen finitas máquinas de Turing esencialmente distintas. Algunas de ellas, iniciando con la cinta en blanco, podrían no parar. Por ejemplo, la máquina de un-estado

$$\langle q_1, B, 1, R, q_1 \rangle$$

sigue escribiendo por siempre sin parar. Pero entre aquellas que paran, buscamos las que escriben muchos 1's.

Defina $\sigma(n)$ como el número más grande de 1's que pueden escribirse por una máquina de Turing de n estados como se describió aquí antes de que pare. Por ejemplo, $\sigma(1) = 1$, ya que la máquina de un-estado

$$\langle q_1, B, 1, R, q_H \rangle$$

(el estado de paro q_H no cuenta) escribe un 1, y ninguna de las otras máquinas de Turing de un-estado lo hacen mejor. (No hay muchas otras máquinas de un-estado, y uno puede examinarlas todas en un plazo de tiempo razonable.)

[2]Este nombre le ha dado muchas dificultades a los traductores.

Acordemos que $\sigma(0) = 0$. Entonces σ es una función total. También es no decreciente ya que no es un impedimento tener un estado extra para trabajar. A pesar del hecho de que $\sigma(n)$ es meramente el miembro más grande de cierto conjunto finito, no existe un algoritmo que nos permita, en general, evaluarlo.

Ejemplo 1.2.2 Aquí hay un candidato de dos-estados:

$$\langle q_1, B, 1, R, q_2\rangle$$
$$\langle q_1, 1, 1, L, q_2\rangle$$
$$\langle q_2, B, 1, L, q_1\rangle$$
$$\langle q_2, 1, 1, R, q_H\rangle$$

Empezando con una cinta en blanco, esta máquina escribe cuatro 1's consecutivos, y luego para (después de seis pasos), leyendo el tercer 1. Está invitado a verificarlo corriendo la máquina. Concluimos que $\sigma(2) \geq 4$.

Teorema 1.2.1 Teorema de Rado (1962): La función σ no es computable a la Turing. Aún más, para cualquier función f computable a la Turing, tenemos que $f(x) < \sigma(x)$ para todo x suficientemente grande. Esto es, σ domina eventualmente a cualquier función total computable a la Turing.

Prueba: Asuma que hemos dado alguna función f computable a la Turing. Debemos mostrar que σ eventualmente la domina. Defina (por razones que pueden parecer inicialmente misteriosas) la función g:

$$g(x) = \text{máx}(f(2x), f(2x+1)) + 1.$$

Entonces g es total y uno puede mostrar que es computable a la Turing. Así existe alguna máquina de Turing $\mathcal{M}$ con, digamos, k estados que la computa, usando el alfabeto $\{1\}$ y notación base-1. Para cada x, sea $\mathcal{N}_x$ la máquina de Turing de $(x+k)$ estados que primero escribe x 1's sobre la cinta, y luego imita a $\mathcal{M}$. (Los x estados nos permiten escribir x 1's en la cinta de manera directa, y luego están los k estados de $\mathcal{M}$.)

Entonces, $\mathcal{N}_x$, cuando inicia con una cinta en blanco, escribe $g(x)$ 1's en la cinta y para. Así, $g(x) \leq \sigma(x+k)$, por la definición de σ. Así, tenemos

$$f(2x), f(2x+1) < g(x) \leq \sigma(x+k),$$

y si $x \geq k$, entonces

$$\sigma(x+k) \leq \sigma(2x) \leq \sigma(2x+1).$$

Poniendo las dos líneas juntas, vemos que $f < \sigma$ de $2k$ en adelante. $\dashv$

Así σ crece más, eventualmente, que cualquier función total computable a la Turing. ¿Qué tan rápido crece? Entre los números más pequeños, $\sigma(2) = 4$. (El ejemplo precedente mostró que $\sigma(2) \geq 4$. La otra desigualdad no es enteramente trivial ya que existen miles de máquinas de dos-estados.) También se ha mostrado que $\sigma(3) = 6$ y $\sigma(4) = 13$. De aquí en adelante, sólo son conocidas cotas inferiores. En 1984, se encontró que $\sigma(5)$ es al menos 1915. En 1990, se elevó a 4098. Y $\sigma(6) > 3{,}1 \times 10^{10566}$. Y $\sigma(7)$ debe ser astronómico. Estas cotas inferiores se establecen usando codificaciones ingeniosamente complejas para crear pequeñas máquinas de Turing que escriben muchos 1's y luego paran.

Probar además cotas superiores sería difícil. En efecto, uno puede mostrar, bajo algunas suposiciones razonables, que cotas superiores para $\sigma(n)$ son probables sólo para finitos n's.

Si pudieramos solucionar el problema de paro, tendríamos el siguiente método para calcular $\sigma(n)$:

- Lista todas las máquinas de n-estados.
- Descarta aquellas que nunca paran.
- Ejecuta aquellas que paran.
- Selecciona la puntuación más alta.

El segundo paso es el que nos da problemas. (Nueva información sobre la función σ de Rado sigue descubriéndose. Noticias recientes se pueden obtener de la página Web mantenida por Heiner Marxen, `http://www.drb.insel.de/~heiner/BB`.)

1.2.2. Recursividad primitiva y búsqueda

Para una segunda formalización del concepto de calculabilidad, definiremos cierta clase de funciones parciales sobre $\mathbb{N}$ como la clase más pequeña que contiene ciertas funciones iniciales y es cerrada bajo ciertas construcciones.

Para las funciones iniciales, tomamos las siguientes funciones totales muy simples:

- Las funciones *cero*, esto es, las funciones constantes f definidas por la ecuación:
$$f(x_1, \ldots, x_k) = 0.$$
existe una de tales funciones para cada k.

- La función *sucesor* S, definida por la ecuación:
$$S(x) = x + 1.$$

- Las funciones *proyección* I_n^k de k-dimensiones sobre la coordenada n-ésima,
$$I_n^k(x_1, \ldots, x_k) = x_n,$$
donde $1 \leq n \leq k$.

Queremos formar la cerradura de la clase de funciones iniciales bajo tres construcciones: composición, recursión primitiva y búsqueda.

Una función h de aridad-k se dice que se obtiene por composición de la función f de aridad-n y las funciones $g_1, \ldots, g_n$ de aridad-k si la ecuación

$$h(\vec{x}) = f(g_1(\vec{x}), \ldots, g_n(\vec{x}))$$

se cumple para todo $\vec{x}$. En el caso de funciones parciales, se entiende aquí que $h(\vec{x})$ está indefinida al menos que $g_1(\vec{x}), \ldots, g_n(\vec{x})$ estén todas definidas y que $\langle g_1(\vec{x}), \ldots, g_n(\vec{x})\rangle$ pertenezca al dominio de f.

Una función h de aridad-$k+1$ se dice que se obtine por *recursión primitiva* de la función f de aridad-k y la función g de aridad-$(k+2)$ (donde $k > 0$) si el par de ecuaciones

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, y+1) &= g(h(\vec{x}, y), \vec{x}, y) \end{aligned}$$

se cumple para todo $\vec{x}$ y y.

Nuevamente, en el caso de funciones parciales, se entiende que $h(\vec{x}, y+1)$ está indefinida al menos que $h(\vec{x}, y)$ esté definida y que $\langle h(\vec{x}, y), \vec{x}, y\rangle$ esté en el dominio de g.

Observe que en esta situación, conocer las dos funciones f y g determina completamente la función h. Más formalmente, si tanto h_1 como h_2 se obtinen por recursión primitiva de f y g, entonces para cada $\vec{x}$, podemos mostrar por inducción sobre y que $h_1(\vec{x}, y) = h_2(\vec{x}, y)$.

Para el caso $k = 0$, la función h de aridad-uno se obtiene por recursión primitiva de la función g de aridad-dos usando el número m si el par de ecuaciones

$$h(0) = m$$
$$h(y + 1) = g(h(y), y)$$

se cumplen para todo y.

Posponiendo el asunto de la búsqueda, definimos a una función como *recursiva primitiva* si se puede construir de las funciones cero, sucesor y proyección mediante el uso de composición y recursión primitiva. (Ver el principio del Capítulo 2 para algunos ejemplos.) En otras palabras, la clase de funciones recursivas primitivas es la clase más pequeña que incluye nuestras funciones iniciales y es cerrada bajo composición y recursión primitiva. (Aquí decir que una clase $\mathcal{C}$ es "cerrada" bajo composición y recursión primitiva significa que siempre que una función f se obtiene por composición de funciones en $\mathcal{C}$ o se obtiene por recursión primitiva de funciones en $\mathcal{C}$, entonces f misma pertenece a $\mathcal{C}$.)

Claramente todas la s funciones recursivas primitivas son totales. Esto es porque todas las funciones inciales son totales, la composición de funciones totales es total, y una función obtenida por recursión primitiva a partir de funciones totales será total.

Decimos que una relación R sobre $\mathbb{N}$ de aridad-k es recursiva primitiva si su función característica es recursiva primitiva.

Luego uno puede mostrar que una gran cantidad de las funciones comunes sobre $\mathbb{N}$ son recursivas primitivas: adición, multiplicación, ..., la función cuyo valor en m es el $(m + 1)$-ésimo primo, ... En el capítulo 2 emprenderemos el proyecto de mostrar que muchas funciones son recursivas primitivas.

Por un lado, parece claro que toda función recursiva primitiva debe considerarse como calculable efectivamente. (Las funciones iniciales son muy fáciles. La composición no presenta grandes obstáculos. Siempre que h se obtiene por recursión primitiva de f y g calculables efectivamente, entonces vemos que podemos encontrar efectivamente $h(\vec{x}, 99)$, encontrando primero $h(\vec{x}, 0), h(\vec{x}, 1), \ldots, h(\vec{x}, 98)$.) Por otro lado, es posible que la clase de funciones recursivas primitivas no pueda incluir a todas las funciones calculables totales ya que podemos "diagonalizarlas" la clase. Esto es, mediante un indexado apropiado del "árbol familiar" de las funciones recursivas primitivas, podemos hacer una lista $f_0, f_1, f_2, \ldots$ de todas las funciones recursivas primitivas de aridad-uno. Luego considere la función diagonal $d(x) = f_x(x) + 1$.

Entonces d no puede ser recursiva primitiva; difiere de cada f_x en x. Sin embargo, si hicimos nuestra lista muy pulcramente, la función d será calculable efectivamente. La conclusión es que la clase de funciones recursivas primitivas es una clase extensa pero es una subclase de las funciones calculables totales.

Luego, decimos que una función h de aridad-k se obtiene de una función g de aridad-$(k+1)$ mediante *búsqueda*, y escribimos

$$h(\vec{x}) = \mu\, y[g(\vec{x}, y) = 0]$$

si para cada $\vec{x}$, el valor $h(\vec{x})$ o es el número y tal que $g(\vec{x}, y) = 0$ y $g(\vec{x}, y)$ está definida y no es cero para todo $s < y$, si tal número y existe, sino está indefinida, si tal número y no existe. La idea detrás de este "operador-μ" es la de buscar por el número más pequeño y tal que es la solución a una ecuación, probando sucesivamente con $y = 0, 1, \ldots$.

Obtenemos las funciones *recursivas generales* agregando búsqueda a nuestros métodos de cerradura. Esto es, una función parcial es recursiva general si se puede construir a partir de las funciones cero, sucesor y proyección, usando composición, recursión primitiva y búsqueda (es decir, el operador-μ).

La clase de funciones parciales recursivas generales sobre $\mathbb{N}$ es (como lo demostró Turing) exactamente la misma como la clase de funciones parciales computables a la Turing. Este es un resultado muy llamativo, a la luz de las maneras muy distintas en las que las dos definiciones fueron formuladas. Las máquinas de Turing parecerían, a primera vista, tener poco que ver con recursión primitiva y búsqueda. Y aún así, obtenemos exactamente las mismas funciones parciales a partir de los dos enfoques. Y la tesis de Church, por tanto, tiene la formulación equivalente a que el concepto de función general recursiva es la formalización correcta del concepto informal de calculabilidad efectiva.

¿Qué si tratamos de "diagonalizar" la clase de funciones recursivas generales, como lo hicimos con las funciones recursivas primitivas? Como argumentaremos más tarde, podemos hacer nuevamente una lista ordenada $\phi_0, \phi_1, \phi_2, \ldots$ de todas las funciones parciales reursivas generales. Podemos definir la función diagonal $d(x) = \phi_x(x) + 1$. Pero en esta ecuación, $d(x)$ está indefinida al menos que $\phi_x(x)$ esté definida. La función diagonal d está en realidad entre las funciones parciales recursivas generales, y así es ϕ_k para algún k, pero $d(k)$ debe estar indefinida. Ninguna contradicción resulta.

El uso de la palabra "recursiva" en el contexto de las funciones recursivas primitivas es enteramente razonable. Gödel, escribiendo en Alemán, había

usado simplemente "rekursiv" para las funciones recursivas primitivas. (Fue Rózsa Péter quien introdujo el término "recursiva primitiva".) Pero la clase de funciones recursivas generales tiene -como esta sección muestra- otras caracterizaciones en las que la *recursión* juega un papel nada obvio.

Esto nos guía a la pregunta: ¿Cómo llamar a esta clase de funciones? Tener dos nombres ("computable a la Turing" y "recursiva general") es una vergüenza de ricos, y la situación sólo empeorará. Históricamente, el nombre "funciones recursivas parciales" ganó. Y las relaciones sobre $\mathbb{N}$ se dicen *recursivas* si sus funciones características pertenecen a esa clase. El estudio de esas funciones por años fue llamada "teoría de funciones recursivas", y luego "teoría de recursión". Pero esto fue más un accidente histórico que una elección razonada. No obstante, la terminología se ha vuelto estándar.

Pero ahora se está haciendo un esfuerzo para cambiar lo que ha sido la terminología estándar. En consecuencia, este libro, *Teoría de la Computabilidad*, habla de funciones parciales *computables*. Y le llamaremos a una relación *computable* si su función característica es una función computable. Así, el concepto de relación computable corresponde a la noción informal de relación decidible. (El manuscrito de este libro ha, no obstante, sido preparado con macros TEX que facilitarían un cambio rápido de terminología.)

En cualquier caso, categóricamente existe la necesidad de tener adjetivos separados para el concepto informal (aquí "calculable" se usa para funciones, y "decidible" para relaciones) y el concepto definido formalmente (aquí "computable").

1.2.3. Programas Loop y While