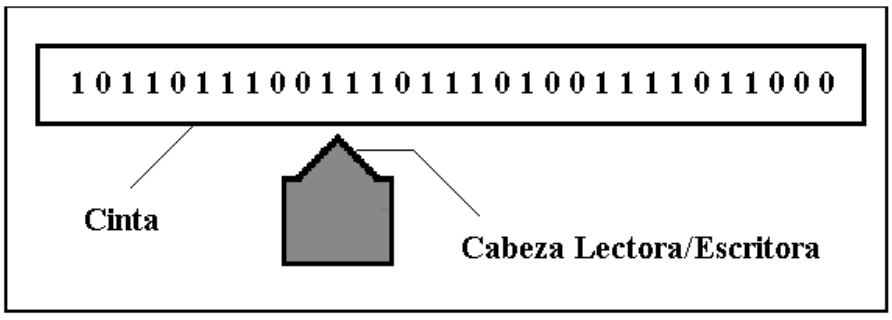
Máquinas de Turing y Funciones Incomputables

En 1936, 5 años después de que Gödel publicara su teorema de la incompletitud de los sistemas formales, Alan Turing publicó un artículo que tenía que ver con los "números incomputables". En el prólogo de ese trabajo (ahora muy famoso) Turing afirmaba haber llegado a resultados "superficialmente similares a los de Gödel". Lo que Turing hizo fue demostrar que hay algunas funciones claramente definibles pero que no se pueden computar. No hablaba de funciones "difíciles" (como encontrar el número primo más cercano a $10^{100,000}$; o determinar el dígito número $10^{10^{10}}$ de la expansión decimal de π; o calcular la probabilidad de que un átomo específico de una muestra de uranio se fisione durante los próximos 10 ns; o encontrar el valor de la función de Ackermann para *m=n=4* [ver más adelante]) que pueden calcularse si tenemos suficiente tiempo, o suficiente memoria, o suficiente información. Hablaba de funciones que, no importa qué tan sofisticado sea nuestro sistema de cómputo, o qué tan rápido sea nuestro procesador, o qué tan grande sea nuestra base de datos, son imposibles de calcular. Turing demostró que existen MUCHAS más funciones (infinitamente muchas más) incomputables que las que sí se pueden calcular.

Para lograr demostrar eso, Turing tuvo que idear un modelo de una máquina de cómputo que no tuviera que ver con asuntos asociados con la tecnología: una máquina que capturara la esencia de los procesos de cálculo. Una vez hecho eso, demostró que todas las funciones que pueden ser calculadas (a las que ahora llamamos funciones recursivas parciales o FRPs) pueden ser calculadas por alguna de sus (ahora llamadas) "máquinas de Turing" (a las que nos referiremos como MTs). Demostró, además, que cualquier MT es una instancia de alguna FRP y, por lo tanto, que podemos hablar intercambiablemente de FRPs o MTs. Una vez hecho esto, mostró cómo construir al menos una función que ninguna MT puede calcular (a pesar de que una MT no consume energía [y puede correr para siempre] y de que sus datos se encuentran en una cinta de longitud infinita [y pueden, por ello, ser ilimitados]).

Las MTs son dispositivos ideales (no pueden construirse) y conceptualmente muy sencillos. Constan de una cinta de datos infinita, una cabeza lectora/escritora y una unidad de control. Por simplicidad consideraremos que la cinta almacena datos binarios (aunque se puede usar otra base es demostrable que las capacidades de MTs binarias o de otro tipo son equivalentes). En las siguientes dos ilustraciones se muestran: una MT (cabeza y cinta) y una posible unidad de control (UC). La cinta de la MT se extiende hacia $\pm\infty$ a derecha e izquierda respectivamente.



En la siguiente figura, la UC consta de 5 estados "dinámicos" (Q1 – Q5) y un estado de "ALTO" o estado de "PARO". Una MT puede no tener estados de paro, pero son las menos interesantes porque ejecutan sus instrucciones y nunca terminan. Las MTs de interés son las que, eventualmente, llegan a algún estado de paro (puede haber más de uno).

Estado Actual	Entrada = 0			Entrada = 1		
	Nuevo Símbolo	Movimiento de Cabeza	Siguiente Estado	Nuevo Símbolo	Movimiento de Cabeza	Siguiente Estado
Q1	1	DERECHA	Q2	1	IZQUIERDA	Q3
Q2	0	IZQUIERDA	Q1	0	IZQUIERDA	Q4
Q3	1	IZQUIERDA	Q1	1	IZQUIERDA	ALTO
Q4	1	IZQUIERDA	Q2	1	DERECHA	Q5
Q5	0	DERECHA	Q4	0	DERECHA	Q2

La MT que se ilustra está diseñada para operar en una cinta inicialmente llena de 0s y, eventualmente, alcanzar el estado de paro. Comentaremos acerca de esta máquina más adelante.

Turing discutió la llamada "MT Universal" (MTU): una MT que recibe en la cinta la descripción (llamémosle τ) de una MT en particular (M_τ), por un lado, y los datos que deberá "procesar" dicha máquina (llamémosle δ). La MTU entonces simula el comportamiento de M_τ cuando en la cinta de M_τ llega δ.

El Problema del Paro

Al problema de determinar si una MT arbitraria se detiene (es decir, no entra en un ciclo infinito) dado el contenido de la cinta de entrada se le denomina el Problema del Paro. Puede expresarse como la función $\Pi(\tau,\delta)$, de la siguiente manera:

$$\Pi(\tau,\delta)=\begin{cases}1 & \Rightarrow \quad M_\tau \\ 0 & \Rightarrow \quad \overline{M_\tau}\end{cases}$$

en donde M_τ significa "M_τ *para*" y $\overline{M_\tau}$ significa "M_τ *NO para*" (en ambos casos, cuando la cinta tiene el contenido δ). Turing demostró que $\Pi(\tau,\delta)$ es incomputable para τ y δ arbitrarios.

La prueba de Turing es por reducción al absurdo. Consta de 4 pasos que describimos a continuación.

Paso 1. Empezamos asumiendo que existe una máquina que es capaz de calcular $\Pi(\tau,\delta)$ a la cual llamamos "H" [por la inicial en inglés de "Paro" (Halting)]. La máquina H es capaz de determinar si una cierta máquina M_τ para cuando recibe como entrada los datos δ.

H opera como sigue:

a) En su cinta se alimenta la UC de M_τ codificada en τ.

b) En la misma cinta se alimentan los datos que M_τ debe procesar codificados en δ.

c) Si M_τ para, H escribe un "1" y para.

d) Si M_τ no para, H escribe un "0" y para.

De manera sintética esto lo indicamos de la siguiente manera:

$$\mathrm{H}(\tau,\delta) \to 1 \Rightarrow \tau(\delta)$$
$$\mathrm{H}(\tau,\delta) \to 0 \Rightarrow \overline{\tau(\delta)}$$

Esta última expresión dice: "Si H, al ser alimentada con la descripción de la máquina τ y los datos δ escribe un "1", ello implica que τ, al ser alimentada con δ, va a detenerse. Si, por el contrario, H escribe un "0" esto implica que τ NO va a detenerse".

Un ejemplo nos permitirá ilustrar lo anterior. Supongamos, para este efecto, que expresamos una MT como una colección de quíntuplos del tipo [1,0,1,D,2]. El primer elemento del quíntuplo indica el estado en el que se encuentra la MT; el segundo indica el valor del bit que la MT "ve" en la cinta en este momento; el tercer elemento indica el valor del bit que la MT escribe en la cinta en ese momento; el cuarto elemento indica si la cabeza se mueve a izquierda ("I") o derecha ("D"); el quinto elemento indica el estado al que pasa la MT. Para este ejemplo, el quíntuplo indica: "*Cuando la MT está en el estado '1' y lee un "0" en su cinta, reemplaza el "0" con un "1", mueve su cabeza a la derecha y pasa al estado '2' "*. Además convendremos que: a) $\mathrm{D} \to 0$ e $\mathrm{I} \to 1$, b) El último estado (el estado al que le corresponde el último número) es un estado de paro y c) Indicaremos con **negritas** el símbolo en la cinta que lee la cabeza de la MT, tendremos que la siguiente es la descripción de una MT de 4 estados y la entrada en su cinta:

τ :1,0,1,0,2;1,1,1,1,3;2,0,1,1,1;2,1,1,0,2;3,0,1,0,2;3,1,1,0,4
δ : ...**0**0000000000000000000000000000000000000000...

En donde, por claridad, hemos separada cada uno de los elementos de los quíntuplos de la cinta con comas y cada uno de los quíntuplo con puntos y comas.

Paso 2. Ahora modificamos H (y la llamamos H') de manera que:

a) En su cinta se alimenta la UC de M_τ codificada en τ.

b) En la misma cinta se alimentan los datos que M_τ debe procesar codificados en δ.

c) Si M_τ para, ahora H' entra en un ciclo infinito y no para.

d) Si M_τ no para, ahora H' para.

Es fácil ver que convertir H en H' es un simple problema de programación. El comportamiento de H' puede expresarse como sigue:

$$\overline{\mathrm{H}'(\tau,\delta)} \Rightarrow \tau(\delta)$$
$$\mathrm{H}'(\tau,\delta) \Rightarrow \overline{\tau(\delta)}$$

en donde $\overline{\text{H'}}$ significa “H’ no para”; mientras que H’ significa “H’ para”.

Paso 3. Hacemos una nueva modificación a H’ (y la llamamos H’’) de manera que:

a) En su cinta se alimenta solamente τ.

b) M_τ duplica el código de τ y lo toma como la entrada a la máquina M_τ. Es decir, ahora M_τ recibe en su cinta la descripción de su propia UC.

Es fácil ver que, nuevamente, este es un simple ejercicio de codificación. En el ejemplo de arriba, podemos codificar al estado *i* con el número binario correspondiente a *i-1* como sigue: (‘1’ $\rightarrow$ 00; ‘2’ $\rightarrow$ 01; ‘3’ $\rightarrow$ 10; ‘4’ $\rightarrow$ 11) y tendríamos:

$$\tau: 000100100111100101100011100110010011011011$$
$$\delta: ...\mathbf{0}000100100111100101100011100110010011011011...$$

c) H’’ opera de manera análoga a H’ en sus pasos (c) y (d).

Todo esto puede expresarse como sigue:

$$\overline{\text{H''}(\tau)} \Rightarrow \tau(\tau)$$
$$\text{H''}(\tau) \Rightarrow \overline{\tau(\tau)}$$

Paso 4. Supongamos, ahora, que tomamos la descripción de H’’ de manera que $\tau \equiv \text{H''}$. De acuerdo con las últimas ecuaciones, tenemos:

$$\overline{\text{H''}(\text{H''})} \Rightarrow \text{H''}(\text{H''})$$
$$\text{H''}(\text{H''}) \Rightarrow \overline{\text{H''}(\text{H''})}$$

Conclusión: “*Si H’’, al recibir en su cinta su propia descripción NO para, entonces H’’, al recibir en su cinta su propia descripción SI para y viceversa*”.

Es decir, si alimentamos a H’’ con sus propia descripción llegamos a una doble contradicción. Para determinar el por qué de ésta analizaremos el proceso que nos llevó a la misma:

El paso 4 no viola ninguna convención porque la naturaleza de τ no fue restringida, de manera que este paso no es el problema.

El paso 3 (la transformación de H’ en H’’) es un simple ejercicio de programación, de manera que este paso no es el problema.

El paso 2 (la transformación de H en H’) es, igualmente, un simple ejercicio de programación, de manera que este paso no es el problema.

El paso 1 (la suposición de la existencia de H) es, pues, el único eslabón en la cadena de razonamientos que nos puede haber llevado a las contradicciones anotadas y, por ello, la única conclusión posible es que “H” no existe y, por lo tanto:

"*Π(τ,δ) es incomputable*".

Es decir:

"*No existe ninguna MT que sea capaz de determinar si una MT arbitraria va a detenerse dado un contenido arbitrario en su cinta*".

Lo anterior implica, dada la equivalencia entre las FRPs y las MTs:

"*No existe una función recursiva parcial que sea capaz de determinar si una FRP arbitraria arroja resultados válidos para un argumento cualquiera*".

O sea que:

"*No existe un procedimiento capaz de determinar, en general, si un algoritmo llegará a término dado un conjunto de datos de entrada arbitrario*".

Si H existiera, podríamos resolver problemas tales como el problema de Fermat ("¿Existe una N entera mayor que 2 tal que $A^N + B^N = C^N$ para A, B, C enteros?") de la siguiente forma:

a) Escribimos un programa que calcule todos los valores de $A^N + B^N = C^N$.
b) Alimentamos este programa a H.
c) Preguntamos si el programa para.
d) Si para, respondemos a Fermat: "si"; en caso contrario respondemos "no".

La incomputabilidad de $\Pi(\tau,\delta)$ implica que existen una serie interesante de problemas irresolubles. La respuesta a todas las siguiente preguntas es negativa; si no fuera así, $\Pi(\tau,\delta)$ sería computable.

¿Es posible determinar si un programa accederá a una localidad específica de memoria?

¿Es posible determinar si un programa usará una unidad de E/S dada?

¿Es posible saber si un programa escribirá un cierto número al ejecutar?

¿Es posible encontrar el código fuente de un programa a partir del código objeto?

Incomputabilidad vía Diagonalización

Otra función incomputable (de hecho una colección infinita de ellas) se deriva del argumento de diagonalización usado por Cantor para probar que el conjunto de puntos de una recta es un infinito mayor que el de los números naturales. Para detalles de este tipo de funciones incomputables, consultar la documentación del programa TURIMACH.EXE.

Incomputabilidad vía el Problema del Castor Ocupado

Otra familia interesante de funciones incomputables nacen del llamado "Problema del Castor Ocupado". Este problema se plantea de manera muy simple:

"*Dado un número de estados n ¿Cuál es el máximo número de 1s que una MT puede escribir en una cinta inicialmente llena de 0s Y parar*".

Esta función se denota como $\sigma(n)$ y se conoce como la función de Rado. Por convención, se asume un alfabeto binario y el estado de paro no se considera; de manera que n = 1 consta de un estado (q1) más el estado de paro; n = 2 consta de dos estados (q1, q2) más el estado de paro, etc. $\sigma(n)$ es incomputable. $\sigma(n)$ se conoce para n = 1, 2, 3, 4 y 5. Para este efecto, la más productiva MT (conocida) de 5 estados se acepta como el Castor Ocupado de 5 estados.

Claramente, la siguiente MT (que llamaremos C_0) definida por

q0,0,1,D,q0

escribe un número infinito de 1s en una cinta llena de 0s. Pero esta MT no califica como castor ocupado porque no se detiene nunca: no incluye un estado de paro. En la siguiente figura se ilustra la MT que corresponde a n = 3. El lector puede verificar que $\sigma(3) = 6$.

Estado Actual	Entrada = 0			Entrada = 1		
	Nuevo Símbolo	Movimiento de Cabeza	Siguiente Estado	Nuevo Símbolo	Movimiento de Cabeza	Siguiente Estado
Q1	1	DERECHA	Q2	1	IZQUIERDA	Q3
Q2	1	IZQUIERDA	Q1	1	DERECHA	Q2
Q3	1	IZQUIERDA	Q2	1	DERECHA	ALTO

El Castor Ocupado de 3 Estados

Para ver por qué la función de Rado es incomputable consideremos cuántas MTs existen para n estados y un alfabeto binario. En la descripción de la UC de una MT tenemos dos posibles valores de entrada. Para cada uno de ellos puede haber 2 X 2 X (n+1) posibles valores. Es decir, el número de posibles valores por fila es 4(n+1) X 4(n+1) = $16(n+1)^2$. Para n estados, por tanto, el número de combinaciones es $[16(n+1)^2]^n$. Si n=3 existen, por ejemplo, 68,719,476,736 MTs distintas; para n = 10 hay más de $7.4 \cdot 10^{32}$ posibles MTs. Para determinar la MT más "productiva" hay que simular cada una de las MTs y hacer eso es extremadamente tardado. Pero el verdadero problema radica en que: a) Hay que fijar un límite a la longitud máxima que permitiremos que la MT recorra antes de considerarla un "corredor" (un ejemplo de un "corredor" es C_0 definida arriba) y b) Hay que fijar el tamaño de iteraciones máximas antes de considerar que la MT entró en un ciclo infinito. Determinar la máxima extensión que recorre una MT desde su punto de inicio es, en si mismo, un problema incomputable. Determinar la entrada de una MT en un ciclo infinito es otro problema incomputable.

Ejercicios.

1. La función de Ackerman se define de la siguiente manera:
2.

$$A(m,n) = \begin{cases} f(0,n) = n+1 \\ f(m,0) = f(m-1,1) \\ f(m,n) = f(m-1,\ f(m,n-1)) \end{cases}$$

Esta es una función recursiva parcial. Es fácilmente programable en cualquier lenguaje que soporte la recursión y, por lo tanto, computable. Determine el valor de *A(m,n)* para *m=4*, *n=4*.

2) Escriba un programa que encuentre el número primo que ocuparía el lugar 10^{1000} si cada número primo encontrado fuera numerado consecutivamente.

3. a) Suponga una MT de 4 estados, codificada como:

τ : 0001001001111001011000111001100100110110 11

b) Suponga que el contenido de su cinta es el siguiente:

δ : **0**0010010011110010110001110011001001101101 1

(con 0s en el resto de la cinta a izquierda y derecha).

c) Suponga que la cabeza lectora se encuentra apuntando al '0' de la extrema izquierda de δ.

Determine el contenido de la cinta al terminar la ejecución de la MT.

4. Encuentre el Castor Ocupado para n=1, 2, 3, 4 y 5.