


Lógica Superior:

Computabilidad e Incompletitud

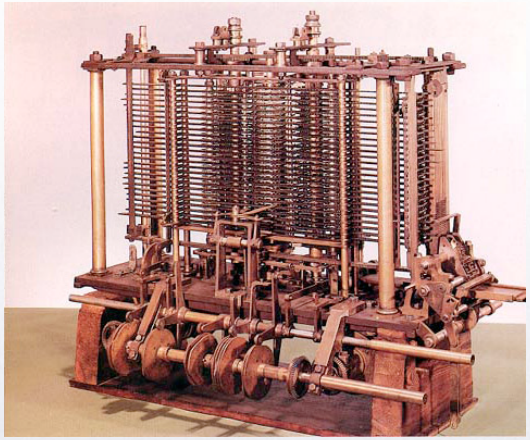
Profesor: Eduardo Barrio
UBA - Filosofía
2do cuatrimestre de 2015


Funciones y computabilidad

¿Qué funciones son computables?

Tesis de Turing:

Cualquier función efectivamente computable es Turing-Computable.




Los límites de la computabilidad

Objetivos del capítulo 4:

1. - probar que hay algunas funciones que no pueden ser computadas por una máquina de Turing.
2. - brindar un argumento general que muestra que existen funciones no computables.
3. - mostrar tres ejemplos de funciones no computables.

Máquina de Turing:

Definición de ***función computable por una máquina de Turing***

1. Los argumentos $m_1, \ldots, m_k$ de la función se representan en notación monádica de bloques de números de 1s, cada bloque separado por blancos (0). En el comienzo de la computación 3 + 2, la cinta es: 111B11.
2. Inicialmente, la cinta lee el 1 que esté más a la izquierda sobre la cinta, y estará en su estado inicial 1. Por eso, en la computación de 3 + 2, la configuración inicial será $1_1$11B11. Tenemos una *standard initial configuration.*
3. Si la función que se computa asigna un valor *n* a los argumentos que están representados inicialmente sobre la cinta, entonces la máquina parará eventualmente sobre la cinta conteniendo un bloque de 1s y el resto en blanco. Por eso, en la computación de 3 + 2, tendremos 11111.
4. En este caso, la máquina parará en el 1 que está más a la izquierda sobre la cinta. La configuración será 1_n 1111, donde *n* representa el estado en el cual no hay ninguna instrucción para continuar. Tenemos una *standard final configuration.*
5. Si la función que está siendo computada no asigna ningún valor a los argumentos que están representados inicialmente sobre la cinta, entonces la máquina nunca parará o parará en alguna configuración no estándar tal como B_n11111 o $B11_n111$ o $B11111_n$.

Máquina de Turing:

Caso más simple: funciones de un argumento

Una M define (computa) la función: f(a) = a + 1.

La función f(x) es **Turing computable** ssi hay una **máquina de Turing** (que lee sólo 0s y 1s) que estando en su configuración estándar inicial para cada entero positivo asignado a x

f(a) = b, si la máquina **PARA** siguiendo sus instrucciones quedando en su configuración final.

f(a) = indefinida, si M **NO PARA** en el estado estándar final o **NO PARA** en absoluto.

Definición f: P→P es Turing computable ssi hay una máquina de Turing (que lee sólo 0s y 1s) que define f.


Funciones no computables

¿Hay funciones no computables?

- **Si:**
- Hay “demasiadas funciones” como para que todas sean turing computables.

- El conjunto de todos los subconjuntos de enteros positivos es no numerable.
- El conjunto de funciones Turing computables es enumerable.
- Por lo tanto, existen funciones que no son computables por una máquina de Turing


Algoritmos que nos cambiaron la vida

- El algoritmo de Euclides: método antiguo (300 AC) y eficaz para calcular el máximo común divisor (MCD).
- Sin algoritmos no habría internet.
- Búsqueda y localización (GPS): Google - Facebook, etc.
- Aplicaciones militares (comunicaciones - misiles, etc)


Casos de Funciones no computables

Puede brindarse algún ejemplo de función no computable?

- Funciones diagonales: hay genuinas funciones d (de un argumento) que no son Turing computables.


Primer caso: funciones diagonales

Teorema de enumeración:

Hay una lista efectiva de todas las funciones de un argumento que son Turing computables.

Primer caso: funciones diagonales

Paso 1: Representación **canónica** de una máquina de Turing

1. comenzar con una representación cuadrática
   $q_1S_0S_1q_2$ $q_1S_1Lq_1$ $q_2S_1Lq_2$
2. Agregar un estado de detención (si fuera necesario) como el estado final. (q_3)
3. Asegurarse que un cuádruplo represente cada estado
   $q_1S_0S_1q_2$ $q_1S_1Lq_1$ $q_2S_1Lq_2$ $q_2S_0S_0q_3$
4. Abreviar eliminando los primeros dos estados.

   S_1q_2

   Lq_1

   Lq_2

   S_0q_3


Primer caso: funciones diagonales

Paso 2: Codificar cada máquina de Turing con un entero positivo

(v) Convertir la representación canónica a una secuencia de números usando 1-4

S_0: 1

S_1: 2

L: 3

R: 4

i: i

$S_1q_2\ Lq_1S_0q_3\ Lq_2$

2, 2, 3,1,1,3, 3, 2

Importante: ninguna M conduce a la misma lista

Primer caso: funciones diagonales

Paso 2: Codificar cada máquina de Turing con un entero positivo

(v) Convertir la secuencia en un único número usando un resultado de la aritmética que asegura la preservación del orden de la máquina.

$2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19$

$2^2 \times 3^2 \times 5^2 \times 7 \times 11 \times 13^3 \times 17^3 \times 19^2$

M: 2, 2, 3,1,1,3, 3, 2

(vii) Cada máquina M tiene un número

(viii) Ordenar la lista: $M_1, M_2, M_3, \ldots, M_n$

(ix) Producir la lista de funciones L Turing - Computables

$$f_1, f_2, f_3, \ldots, f_n$$

No hay ninguna máquina de Turing que no esté en la lista. Las funciones Turing-Computables son numerables.

Teorema fundamental de la aritmética

Número Primo: número que sólo es divisible por 1 y por sí mismo.

Todo número natural mayor que 1 se puede representar de forma única como producto de números primos.

Representación: factorización de un número.

6936: 2 x 2 x 2 x 3 x 17 x 17: $2^3 \times 3^1 \times 17^2$

Esa propiedad es algorítmica. La factorización de cualquier número se puede obtener por medio de un proceso mecánico en una serie de pasos finitos.

Los números primos codifican la posición en la cadena de símbolos.

Cada número primo elevado a una potencia correspondiente según el esquema de codificación.

Hay dos algoritmos: uno para transformar un número en una expresión y otro para transformar cualquier expresión en un número.


Primer caso: funciones diagonales

Paso 3: Diagonalización

Dada la anterior lista L:

$f_1, f_2, f_3, \ldots, f_n$

definir la función diagonal d como sigue

$d(n) = \begin{cases} 2, \text{ si } f_n \text{ está definido y } f_n: 1 \\ 1, \text{ en cualquier otro caso} \end{cases}$

Entonces d es una función total de $P \rightarrow P$, pero d es distinta de cada uno de los $f_1, f_2, f_3, \ldots, f_n$

Si d es distinta de cada uno de los $f_1, f_2, f_3, \ldots, f_n$, d no es Turing computable (ya que en la lista están todas las funciones Turing Computables).


Primer caso: funciones diagonales

A) Si la tesis de Turing es correcta, d no es efectivamente computable

B) Puzzle: ¿Por qué d no es computable? El problema de la detención (Halting Problem)

Segundo caso: el Halting Problem

Determinar si una máquina de Turing cualquiera se detendrá ante cualquier input dado.

¿existe una máquina M capaz de determinar si cualquier otra máquina se va a detener o no?

Este problema es equivalente al problema de saber si un problema se cuelga cuando corre en la computadora.

La función detención (Halting Problem)

Consideremos la lista anterior de máquinas de Turing M: M_1, M_2, M_3 , ..., M_n

Máquina de Turing Universal: esencialmente, una máquina de Turing universal **simula** el comportamiento de cualquier máquina de Turing (incluyéndose a sí misma).

Turing (1947):

“Se puede demostrar que es posible construir una máquina especial de este tipo que pueda realizar el trabajo de todas las demás. Esta máquina especial puede ser denominada máquina universal.”

Sistema operativo: un programa que puede ejecutar otros programas

Son máquinas que se comportan como otras máquinas.

Segundo caso: el Halting Problem

La función detención (Halting Problem)

Consideremos la lista anterior de máquinas de Turing M: M_1, M_2, M_3, ..., M_n

Comienza con el número (el código) de otra M_k (este número representa la tabla de la máquina) y produce como resultado el mismo que daría M_k si partiera de ese input.

Función detención

H(m,n) =

$\{1$, si M_m se detiene cuando comienza en el input n$\}$

$\{2$, si M_m no se detiene cuando comienza en el input n$\}$

Problema de la detención: encontrar un procedimiento efectivo que, dado cualquier M y cualquier input n, sea capaz de determinar si M se detiene dado el input n.


Segundo caso: el Halting Problem

Función detención

H(m,n) =

{1, si M_m se detiene cuando comienza en el input n}

{2, si M_m no se detiene cuando comienza en el input n}

Argumento informal:

Si h fuera efectivamente computable, d lo sería.
Por eso, si aceptamos la tesis de Turing, sabemos que h no es Turing computable.


Segundo caso: el Halting Problem

El problema de la **detención** es **indecidible algoritmicamente**: ninguna máquina de Turing lo puede resolver.

- No existe una manera automática computable de saber si todos los programas del mundo terminan.

- No se niega que exista la prueba para programas concretos.

- De hecho, la construcción de pruebas para programas concretos es un paso obligatorio para demostrar su corrección.

El regreso de los Super-humanos

¿**ZEUS** podría numerar el conjunto de verdades del español?

Dato: Zeus podría numerar el conjunto de oraciones del español a través de la numeración de Gödel.

Sea «s» el número de Gödel de s. (T) s ≡ True («s»). x ≡ True (y)

$True^3$ (x, y, z) =

- 1 (el valor a z), si x es verdadera de su número de Gödel y
- 0 (el valor a z), si x no es verdadera de su número de Gödel y.

Si el conjunto de verdades fuera numerable por Zeus, no podría encontrarse una oración a la cual no le correspondiera un valor.

- The Liar: Es la oración s tal que s ≡ ¬ True («s»).

◆La antidiagonal

- $True^{*3}$ (x, y, z) =

- 1, si x no es verdadera de su número de Gödel y.
- 0, si x es verdadera de su número de Gödel y.


Números Computables

Números computables:

π (3,1416...) es un número computable

1. - Pero, los números reales no pueden ser listados.
2. - Eso es cierto, pero no significa que π no sea computable.
3. - Pero, los dígitos de π son infinitos.
4. - Eso es cierto pero no quiere decir que no sea computable.
5. - Un número es computable sss existe un algoritmo (una máquina de Turing) que encuentra su n dígito.
6. - Por supuesto, no existe una M que escriba los dígitos de π.