La máquina de Turing: un modelo para la computación

Mayte Maeso Bayo

Trabajo de fin de grado de Matemáticas

Director del trabajo: José Carlos Ciria Cosculluela

5 de julio de 2023

"It is one of the great ironies of scientific history that Turing invented computers in order to show that there are things that computers are fundamentally incapable of doing —that some problems are inherently undecidable". Wooldridge, M. (2021). A brief history of artificial intelligence: what it is, where we are, and where we are going. Flatiron Books.

Summary

In 1936 Alan Turing wrote an article called “On computable numbers, with an application to the Entscheidungsproblem” where he gave an answer to the decision problem proposed by David Hilbert. This problem was about the foundation of mathematics and, specifically, about the notion of decidability. This means, if we are able to decide, given a mathematical sentence, whether it is provable or not. To give an answer to this problem, Turing devised the celebrated Turing machines.

In this final degree project we will analyze in depth Turing’s article. First we will give an historical context about the decision problem and we will explain it through an example. We will talk a little about Gödel results which are closely related to this subject and then we will focus on the resolution of the Entscheidungsproblem.

We will introduce the notion of Turing machines. What they are, what elements compose them and how they work. Simultaneously we will give examples of Turing machines, both its operation and its python implementation. Once understood these concept we will explain the Universal Turing machine, a machine able to simulate any other machine. After that we will get in to some theoretical results necessary to approach to the decision problem and finally, in the last chapter, we will give a negative response to the Entscheidungsproblem.

Índice general

Summary III

1. Introducción 1

2. Máquinas de Turing 3
2.1. Introducción a las máquinas de Turing . . . 3
2.2. Definición formal de las máquinas de Turing . . . 4
2.3. Ejemplos . . . 5
2.4. Tablas esqueleto . . . 6

3. Máquina universal de Turing 9
3.1. Máquina Universal de Turing . . . 9
3.1.1. Codificación de las instrucciones . . . 9
3.1.2. Funcionamiento de la máquina universal . . . 10
3.2. Resultados teóricos . . . 13

4. El problema de la decisión 19
4.1. El problema de la decisión . . . 19

Bibliografía 25

Anexo 27
.1. Estados máquina que escribe un número irracional . . . 28
.1.1. Explicación . . . 28
.1.2. Diagrama . . . 28
.1.3. Código . . . 29
.2. Estados máquina que invierte un número . . . 30
.2.1. Explicación . . . 30
.2.2. Diagrama . . . 32
.2.3. Código . . . 32
.3. Estados máquina que suma dos números en bianrio . . . 34
.3.1. Explicación . . . 34
.3.2. Código . . . 35
.4. Estados Máquina Universal de Turing . . . 40

Capítulo 1

Introducción

En 1936 Alan Turing publicó "On computable numbers, with an application to the Entscheidungsproblem", un artículo de carácter científico en el que daba respuesta al Entscheidungsproblem o problema de la decisión. Para ello ideó un nuevo sistema conceptual dotado de reglas propias que sirvió no solo para resolver el problema, sino que también sentó las bases de lo que hoy conocemos como teoría de la computación. Para poner en contexto el artículo recordaremos brevemente en qué consiste el problema de la decisión.

En 1928 David Hilbert propone, en su obra Principios de la lógica Matemática, escrita junto con Wilhelm Ackermann, [1] su visión sobre la fundamentación de las matemáticas. Todas las ramas de las matemáticas se basan en un conjunto de axiomas que, junto con una serie de reglas de inferencia, permiten construir teoremas. Según Hilbert dicho sistema debe cumplir ciertas propiedades que enumeraremos a continuación. Pero antes, las ilustraremos tomando como ejemplo la aritmética. En ella podemos escribir fórmulas como

$$F \equiv \forall(n,x,y,z)((n > 2 \;\&\; x > 1 \;\&\; y > 1 \;\&\; z > 1) \rightarrow x^n + y^n \neq z^n)$$

donde x,n,y,z son números naturales. La fórmula F es el enunciado del último teorema de Fermat. La negación de F es

$$\overline{F} \equiv \exists(n,x,y,z)(n > 2 \;\&\; x > 1 \;\&\; y > 1 \;\&\; z > 1 \;\&\; x^n + y^n = z^n)$$

Las propiedades que, según Hilbert, debe cumplir el sistema axiomático son las siguientes:

- Independencia: los axiomas deben ser independientes entre sí.
- Consistencia: del sistema de axiomas no se deben seguir contradicciones, es decir, no pueden deducirse ambas F y $\overline{F}$.
- Completitud: los axiomas comprenden todo lo que se puede decir de la rama en cuestión. Para toda fórmula F o bien F o bien $\overline{F}$ se siguen de los axiomas. O dicho de otro modo, si añadimos un nuevo axioma no deducible de los anteriores, el sistema resultante es inconsistente.
- Decidibilidad: para toda fórmula F (y por tanto también para $\overline{F}$) debe existir un procedimiento finito que permita decidir si ésta se sigue de los axiomas, es decir, si es demostrable a partir de los axiomas.

Si la aritmética fuese consistente y completa sabríamos que, para toda fórmula F, o bien F o bien $\overline{F}$ es demostrable a partir de los axiomas. Si además fuese decidible sabríamos cuál de las dos es demostrable.

La cuestión sobre la decidibilidad del sistema axiomático es lo que se conoce como Entscheidungsproblem o problema de la decisión. Concretamente, el problema de la decisión está formulado en el capítulo *III* del libro y Hilbert se refiere a él como *"el principal problema de la lógica matemática"*.

También es interesante notar que este problema ya aparece apuntado entre sus célebres 23 problemas propuestos en el Congreso Internacional de Matemáticos de 1900. En particular, el décimo problema fue el de determinar si una ecuación diofántica es resoluble: *"dada una ecuación diofántica con cualquier número de incógnitas y con coeficientes enteros, establecer un proceso según el cual pueda determinarse, siguiendo un número finito de operaciones, si la ecuación tiene solución."*[1]

En 1929, en su tesis doctoral, Gödel demuestra que la lógica de primer orden es completa. Este resultado, conocido como el Teorema de Completitud de Gödel, no satisface las aspiraciones de Hilbert. La lógica matemática no existe en el vacío. Buena parte de su interés reside en que proporciona un marco y unos fundamentos para otras ramas de la matemática, como la aritmética, y aun quedan preguntas sin responder tales como si la lógica de primer orden sigue siendo completa si se le añaden los axiomas necesarios para definir la teoría de números.

Un año más tarde, en 1930, Gödel presenta su Primer Teorema de Incompletitud. En el sostiene que si a la lógica de primer orden se le añaden los axiomas necesarios para derivar la aritmética entonces el sistema resultante es incompleto. Es decir, podemos garantizar que existe una fórmula G tal que ni G ni su negación pueden deducirse de los axiomas [2]. De su primer Teorema de Incompletitud se deduce el segundo Teorema de Incompletitud de Gödel: una teoría consistente no puede demostrar su propia consistencia.

Sin embargo, los teoremas de incompletitud de Gödel no cierran el problema de la decisión. No descartan que, al menos, sea posible decidir si una fórmula es demostrable a partir de los axiomas.

De esto último se encarga Alan Turing en su artículo. En primer lugar caracteriza formalmente la idea de algoritmo como un método finito que se ejecuta automáticamente. Para ello define sus máquinas (que conoceremos como máquinas de Turing). A continuación diseña una máquina de propósito general capaz de ejecutar cualquier algoritmo almacenado en memoria (la Máquina Universal de Turing) y demuestra que existen problemas que no pueden resolverse algorítmicamente, es decir, mediante máquinas de Turing. Por último, propone una fórmula U sobre la que no es posible decidir, usando máquinas de Turing, si es demostrable, con lo que da una respuesta negativa al Entscheidungsproblem.

Al final del artículo hay un apéndice que hace referencia al trabajo de Alonzo Church. Alonzo fue un matemático estadounidense que, con unos pocos meses de antelación, había conseguido dar respuesta al problema de la decisión a través de medios totalmente distintos. Ambos habían enviado sus respuestas a Max Newman y, a pesar de que el trabajo de Church estaba en prensa, Newman consideró que el trabajo de Turing era tan original y brillante que recomendó su publicación junto con un apéndice donde se nombrasen los resultados de Church.

Capítulo 2

Máquinas de Turing

2.1. Introducción a las máquinas de Turing

Una máquina es un objeto creado con el objetivo de facilitar o realizar una tarea. En el caso de las máquinas de Turing, estas fueron diseñadas para computar secuencias de símbolos de forma automática. A pesar de que las nociones sobre qué es una máquina o qué es un proceso automático eran intuitivas, Turing se encontró con la dificultad de dar una formulación rigurosa y precisa que reflejase esa intuición. Para facilitar la comprensión de estos conceptos, Turing estableció una similitud entre lo que por entonces se conocía como computador humano (persona que realiza cálculos como sumas y multiplicaciones) y su idea de cómo debía ser un "proceso mecánico que se ejecutase automáticamente". Para realizar esta comparativa Turing tuvo en cuenta todos los elementos necesarios a la hora de realizar un cómputo tales como el papel en el que se escribe, los símbolos que se emplean o las normas que se siguen, y los transformó para crear, a partir de ellos, las máquinas de Turing.

Un contable o computador humano necesita de un papel para realizar sus cuentas. El papel puede ser cuadriculado y el contable puede disponer de una cantidad ilimitada de papel. La dimensión del papel no es esencial para el modelo, es decir, si queremos realizar una suma de dos números podemos escribir uno encima del otro o podemos poner ambos números seguidos en una única línea. Por tanto, sin pérdida de generalidad, podemos suponer que el papel es una cinta unidimensional infinita y dividida en secciones o "celdas", en cada una de las cuales se escribe un único símbolo.

El conjunto de símbolos empleados por el contable debe ser finito ya que si no lo fuese tendríamos tantos símbolos que algunos serían tan similares que podríamos confundirlos. Al conjunto de símbolos empleados lo llamó alfabeto y lo determinó con el símbolo $\sum$. Turing diferenció dos tipos de símbolos, los de primer tipo y los de segundo. Los de primer tipo eran aquellos con los que se operaba. Por ejemplo, en el caso de una suma de dos números enteros los símbolos de primer tipo serían los dígitos del 0 al 9, mientras que los símbolos de segundo tipo serían los empleados para marcar qué dígitos de los sumandos hemos considerado ya, apuntar las llevadas...

Dado que el contable realiza las cuentas en una cinta, si por ejemplo está sumando dos números muy grandes, este no puede observar de un único vistazo toda la cinta, es decir, un lector sólo puede ver simultáneamente una parte limitada del papel. Por tanto Turing supuso que, en cada instante de tiempo, la máquina solo puede leer una celda. Además, el contable no puede realizar grandes movimientos entre las celdas ya que al ser una cinta unidimensional muy larga podría equivocarse de celda. Por tanto, la máquina irá avanzando celda a celda hasta encontrar la casilla que busque en cada momento.

El contable debe seguir una serie de normas fijas. Según Turing, *"no tiene autoridad para desviarse de ellas en ningún momento"*. En el caso de la suma de dos números el contable debe seguir unos pasos tales como buscar el primer dígito del primer número, después buscar el primer dígito del segundo número, luego anotar el resultado de sumar dos dígitos.... Turing modelizó este comportamiento mediante lo que llamó *"estados mentales"* Como ocurre con los símbolos, los estados mentales deben ser finitos ya que, en otro caso, habría estados arbitrariamente próximos y sería imposible distinguirlos. Y según el estado y el símbolo que esté observando el contable, este realizará una instrucción u otra. A este último

proceso Turing lo denominó "transición". Cuando un contable está realizando una suma, una vez que tiene los dos dígitos a sumar, sabe que tiene que desplazar su mano hasta la derecha del papel y escribir el resultado de la suma. Sin embargo, si solo tiene uno de los dígitos tendrá que ir en busca del segundo. Es decir, en función del momento del proceso en que se encuentre el contable y del símbolo que haya leído realizará un movimiento u otro. Y lo mismo hará la máquina en función del estado mental en el que se encuentre y del símbolo que lea.

Por tanto, una máquina está compuesta por una cinta infinita dividida en celdas, un cabezal de lectura y escritura que apunta a una única celda, un alfabeto finito y un conjunto de estados que, en función del estado en el que se encuentre la máquina y el símbolo que esté leyendo el cabezal, realizará unos determinados movimientos.

Demos ahora una definición formal de máquina de Turing:

2.2. Definición formal de las máquinas de Turing

Definición 1. Una máquina de Turing es una 5-tupla $(Q, \sum, \delta, q_0, F)$ donde

1. Q es un conjunto finito de estados
2. $\sum$ es un conjunto finito de símbolos llamado alfabeto.
3. δ: $Q \times \sum \to \sum \times \{L, R, N\} \times Q$ es la función transición.
4. $q_0 \in Q$ es el estado inicial.
5. $F \subseteq Q$ es el conjunto de estados finales.

A lo largo de este trabajo consideraremos la base binaria: los símbolos de primer tipo serán $\{0, 1\}$. Además, deberemos diferenciar las celdas pares de las impares. En las pares irán los símbolos de primer tipo (la secuencia a computar) y en las impares los símbolos auxiliares.

El dominio de la función transición δ es el par $(Q, \sum)$. A cada elemento del producto cartesiano $Q \times \sum$ le llamaremos configuración. Esto es lo que determina el movimiento de la máquina en cada instante. En cada momento estaremos en un estado concreto y leyendo un símbolo determinado, y en función de ellos la máquina escribirá un nuevo símbolo, realizará un único movimiento (es decir, se desplazará una posición a la izquierda, L, a la derecha, R, o no se moverá, N) y pasará a un estado nuevo.

El conjunto F de estados finales puede estar vacío. Un estado final determina cuándo la máquina deja de moverse, es decir, cuándo deja de realizar un cómputo.

Turing distinguió dos tipos de máquinas: las circulares y las no circulares. Las máquinas circulares son aquellas que no escriben más de un número finito de símbolos del primer tipo mientras que las no circulares escriben infinitos. Por ello el conjunto de estados finales de una máquina no circular es el vacío. A partir de esta caracterización de las máquinas definimos los siguientes conceptos:

Definición 2. Una secuencia se dice computable si puede ser calculada por una máquina no circular. Los dígitos de dicha secuencia se interpretan como la parte decimal, escrita en binario, de un número.

Y de aquí se deduce lo siguiente:

Definición 3. Un número es computable si existe una máquina de Turing no circular capaz de generar sus infinitos decimales.

2.3. Ejemplos

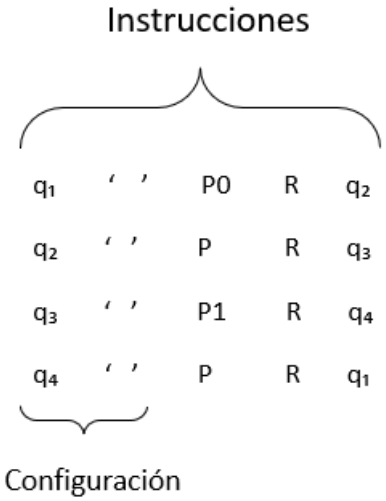
Veamos el ejemplo más sencillo de una máquina de Turing. Computemos la secuencia 010101010101... Esta secuencia es computable ya que existe una máquina no circular que la calcula y su expresión decimal es el número 1/3. Por tanto 1/3 es un número computable.

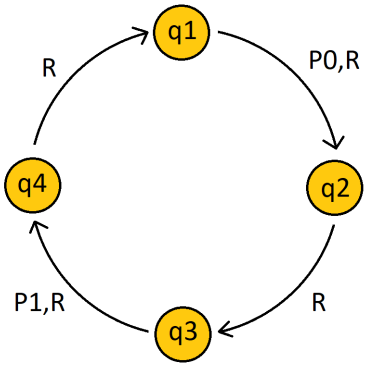
Para diseñar la máquina de Turing que compute dicha secuencia tenemos que definir todos los elementos que la componen. Por un lado tendremos el alfabeto que estará compuesto por los símbolos de primer tipo $0, 1$. En este caso no serán necesarios símbolos de segundo tipo o auxiliares. Tendremos cuatro estados q_1, q_2, q_3, q_4 y tendremos cuatro instrucciones que indicarán los movimientos de la máquina para cada configuración. Como queremos que la máquina no se detenga no tendremos estados finales.

Tendremos un estado inicial que será q_1. La cinta inicialmente estará vacía y el cabezal señalará a la primera celda, por tanto la configuración inicial será el par (q_1, celda vacía). Como la secuencia que queremos escribir comienza con un 0 le pediremos a la máquina que escriba un 0 y que a continuación se desplace una posición a la derecha. Una vez realizados estos movimientos la máquina pasará al siguiente estado que será q_2. Es decir, dada la configuración inicial (q_1, celda vacía) la función transición nos ha indicado qué elemento escribir, qué movimiento realizar y a qué estado pasar.

Ahora nos encontramos en la configuración (q_2, celda vacía). Como tenemos que respetar las celdas pares e impares la máquina no va a escribir ningún símbolo, se va a desplazar una posición a la derecha y pasará al siguiente estado, q_3. Ahora realizará la misma operación que al principio solo que escribiendo un 1 en lugar de un 0 y pasará al estado q_4. Por último, al estar en el estado q_4 y estar leyendo una celda vacía, como el cabezal se encuentra apuntando a una celda impar, la máquina no escribirá nada, se desplazará una posición a la derecha y pasará de nuevo al estado q_1.

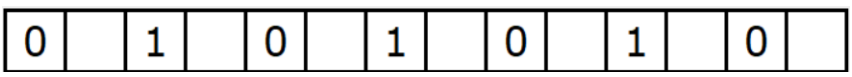
Podemos representar gráficamente el funcionamiento de la máquina con el siguiente diagrama: [1]





Como podemos observar se trata de un bucle a partir del cual, con cuatro estados, hemos conseguido computar la secuencia 0101010101... de la cual podríamos escribir infinitas cifras. Es decir, hemos creado un algoritmo o máquina de Turing no circular que nos computa una secuencia determinada.

La cinta tendría el siguiente aspecto: [2]



[1] P0 se describe como "print 0", P como "print espacio en blanco", P1 como "print 1" y R como "desplazarse una posición a la derecha".

[2] Al inicio de la cinta se suelen escribir dos símbolos consecutivos "@". Son para denotar el comienzo de la secuencia. Como la cinta inicialmente siempre está vacía tenemos que introducirlas nosotros. Para ello tenemos que crear un estado q' que escriba las dos "@" y que luego pase al estado $q0$. En este caso lo hemos omitido para facilitar la comprensión del problema.

El código realizado en python es el siguiente:

```
def q1(cinta, i):
    cinta[i] = '0'
    i = __R__(cinta,i)
    return i, q2

def q2(cinta, i):
    i = __R__(cinta,i)
    return i, q3

def q3(cinta, i):
    cinta[i] = '1'
    i = __R__(cinta,i)
    return i, q4

def q4(cinta, i):
    i = __R__(cinta,i)
    return i, q1
```

Cada función tiene dos parámetros y modeliza un estado. Cada vez que se invoca una función se ejecuta una transición y según el estado y el símbolo leído se realizaran distintas acciones. Por ejemplo, en $q1$ y en $q3$ se sobre escribe el contenido de la celda mientras que en $q2$ y en $q4$ el cabezal solo se desplaza una posición a la derecha. Cada función devuelve el índice de la siguiente celda a leer y el nuevo estado.

La función __R__ mueve el cabezal de lectura una posición a la derecha ampliando, si es preciso, el tamaño de la cinta. (A diferencia de en una máquina de Turing, la cinta de nuestro programa tiene una longitud inicial finita. Si es necesario la vamos ampliando).

En el apéndice del trabajo podemos encontrar tanto la descripción como el código y los diagramas de una máquina de Turing que computa un número irracional, de otra máquina que invierte una secuencia en binario y otra que suma dos números en binario.

2.4. Tablas esqueleto

El ejemplo que acabamos de ver es bastante sencillo. Sin embargo, es fácil intuir que para máquinas más complejas habrá procesos más complicados y puede que estos se repitan varias veces a lo largo de la ejecución. Por ejemplo podemos pedirle a una máquina que compare secuencias, que encuentre un determinado símbolo todas las veces que aparezca en la cinta o que borre determinados elementos. Para no tener que reescribir cada una de estas órdenes cada vez que las necesitemos, recurrimos a las tablas esqueleto. Las tablas esqueleto no son mas que abreviaciones de un determinado proceso que nos van a ayudar a simplificar la programación de una máquina evitando un trabajo repetitivo. Además funcionan como generadoras de estados. Veamos un ejemplo de tabla esqueleto:

Supongamos que tenemos una máquina que ha escrito un número de símbolos tanto de primer tipo como de segundo. En un determinado momento queremos que nos encuentre en la cinta el símbolo $\mathfrak{a}$ situado más a la izquierda. Si lo encuentra la máquina pasará al estado $\mathfrak{C}$ y si no pasará a $\mathfrak{B}$. Por tanto tenemos un símbolo y dos estados. Para ello crearemos una tabla esqueleto que se llame *buscaprimero* y que tenga tres argumentos en los que pondremos $\mathfrak{C}$, $\mathfrak{B}$ y $\mathfrak{a}$. La tabla será la siguiente:

buscaprimero$(\mathfrak{a},\mathfrak{B},\mathfrak{C})$

$$f(\mathfrak{C},\mathfrak{B},\mathfrak{a})\begin{cases} @ & \text{L} & f_1(\mathfrak{C},\mathfrak{B},\mathfrak{a}) \\ not\ @ & \text{L} & f(\mathfrak{C},\mathfrak{B},\mathfrak{a}) \end{cases}$$

$$f_1(\mathfrak{C},\mathfrak{B},\mathfrak{a})\begin{cases} a & & \mathfrak{C} \\ not\ \mathfrak{a} & \text{R} & f_1(\mathfrak{C},\mathfrak{B},\mathfrak{a}) \\ None & \text{R} & f_2(\mathfrak{C},\mathfrak{B},\mathfrak{a}) \end{cases}$$

$$f_2(\mathfrak{C},\mathfrak{B},\mathfrak{a})\begin{cases} a & & \mathfrak{C} \\ not\ \mathfrak{a} & \text{R} & f_1(\mathfrak{C},\mathfrak{B},\mathfrak{a}) \\ None & \text{R} & \mathfrak{B} \end{cases}$$

Notar que *buscaprimero* ha generado tres estados distintos.

Veamos su funcionamiento así como un diagrama y su implementación en python:

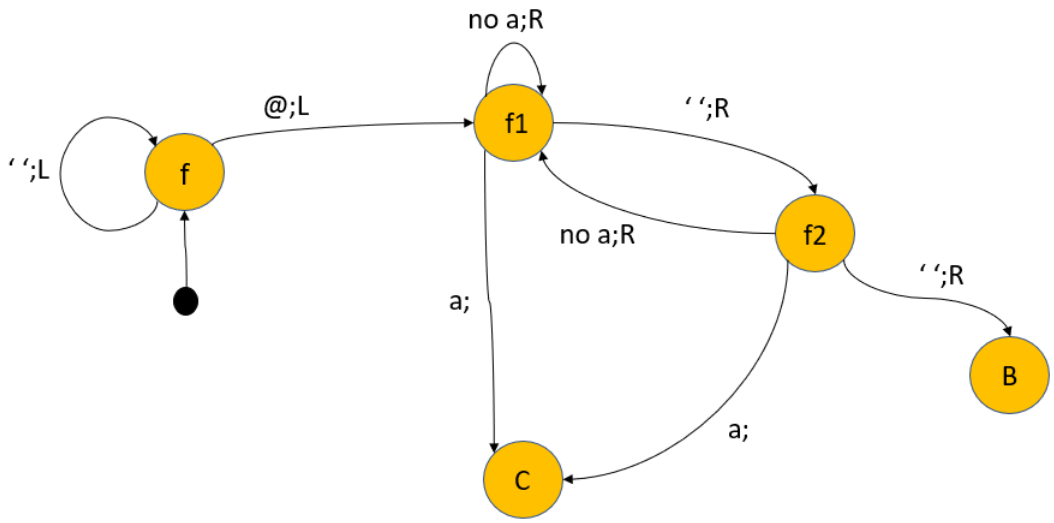
Comienza en el estado $f(\mathfrak{C},\mathfrak{B},\mathfrak{a})$. Como estamos buscando la primera aparición de $\mathfrak{a}$ el cabezal de lectura se desplazará hacia la izquierda hasta encontrar el símbolo que indica el inicio de la cinta. Una vez ahí la máquina recorrerá la cinta de izquierda a derecha buscando el símbolo $\mathfrak{a}$. Si lo encuentra pasará al estado $\mathfrak{C}$. Si no lo hace, llegará un momento en el que encuentre dos espacios vacíos consecutivos, lo que indica el final de la cinta escrita y que el carácter buscado no aparece en ella. Por tanto pasará al estado $\mathfrak{B}$.

Ahora supongamos que en la misma máquina queremos volver a realizar este proceso pero ahora queremos que nos busque el símbolo $\mathfrak{b}$ situado más a la izquierda y que si lo encuentra pase al estado $\mathfrak{E}$ y si no lo encuentra que pase a $\mathfrak{D}$. Lo único que tendríamos que hacer es llamar a *buscaprimero*$(\mathfrak{D},\mathfrak{E},\mathfrak{b})$.

Podemos entender la tabla esqueleto como un subalgoritmo: una sucesión de estados que realizan una función básica. O dicho de otro modo, como un generador de estados donde, en este caso, para cada posible combinación $(\mathfrak{C},\mathfrak{B},\mathfrak{a})$, genera tres estados distintos.

Es necesario recalcar la importancia de las tablas esqueleto ya que nos van a facilitar mucho la programación de cualquier máquina. En el siguiente capítulo veremos como estas son fundamentales para construir la Máquina Universal de Turing, la cual será imprescindible para dar respuesta al problema de la decisión.

El diagrama de la tabla esqueleto sería el siguiente:



El código implementado en python es este:

```
def buscaPrimero(C, B, a):
    def f(cinta, i):
        if cinta[i] == '@':
            i = i - 1
            q = f1
        else:
            i = i - 1
            q = f
        return i, q

    def f1(cinta, i):
        if cinta[i] == a:
            q = C
        elif cinta[i] == ' ':
            i = __R__(cinta, i)
            q = f2
        else:
            i = __R__(cinta, i)
            q = f1
        return i, q

    def f2(cinta, i):
        if cinta[i] == a:
            q = C
        elif cinta[i] == ' ':
            i = __R__(cinta, i)
            q = B
        else:
            i = __R__(cinta, i)
            q = f1
        return i, q

    return f
```

Capítulo 3

Máquina universal de Turing

Cada una de las máquinas de Turing que hemos visto hasta ahora realiza una función concreta (genera un número, invierte una cadena de 0 y 1, realiza una suma...). Es decir, cada máquina representa un algoritmo. Sin embargo Turing fue más allá y propuso una máquina de uso general, la máquina universal de Turing, capaz de ejecutar cualquier máquina. Con esto anticipó la existencia de los ordenadores, máquinas universales que ejecutan cualquier algoritmo que se cargue en la memoria.

En este capítulo nos centraremos en entender cómo funciona la máquina Universal de Turing y qué elementos la componen. Presentaremos las tablas esqueleto a partir de las cuales se generan los estados que determinan el comportamiento de la máquina universal y como paso previo veremos cómo se codifica una máquina M para introducirla en la cinta (la memoria) de la Máquina Universal.

3.1. Máquina Universal de Turing

3.1.1. Codificación de las instrucciones

Tomemos el ejemplo visto en el capítulo anterior. En él hemos generado una máquina de Turing que computa la secuencia 01010101... (a la que llamaremos máquina M) y lo hemos hecho a través de ciertas instrucciones. Esas instrucciones son las que determinan a una máquina de Turing. Por tanto, esto será lo que tengamos que introducir en la máquina universal. Pero como hemos dicho antes tendremos que traducirlas a un lenguaje que la máquina entienda. Para ello Turing ideó el siguiente sistema de codificación:

Para denotar un estado q_i tendremos que escribir una D seguida de la letra A repetida i veces. Por ejemplo, si tenemos el estado q_1 lo traduciremos como DA, y si tenemos el estado q_2 lo traduciremos como DAA. Los símbolos los denotó de la siguiente manera: espacio en blanco=D, 0=DC y 1=DCC. [1] Y para los desplazamientos denotó izquierda=L, derecha=R y no desplazarse=N.

Por tanto, en el ejemplo de la máquina que computa la secuencia 0101010101... teníamos cuatro bloques de instrucciones. Si los escribimos de forma consecutiva separándolos por puntos y comas tendríamos la siguiente secuencia, donde hemos denotado como _ a los espacios en blanco:

$$;q_1\_0Rq_2;q_2\_\_Rq_3;q_3,\_1Rq_4;q_4\_\_Rq_1$$

Ahora si sustituimos cada elemento por su correspondiente codificación obtendremos lo siguiente:

;DADDCRDAA;DAADDRDAAA;DAAADDCCRDAAAA;DAAAADDRDA

A esto se le llama **descripción estándar** de una máquina y lo denotaremos por $DE(M)$. Esta es la información codificada que se introduce en la máquina universal.

[1] Turing empleó una notación auxiliar donde al espacio en blanco lo denotó por S_0, al símbolo 0 por S_1 y al 1 por S_2. A partir del siguiente capítulo haremos uso de esta notación.

Además, si intercambiamos cada A por un 1, cada C por un 2, cada D por un 3, cada L por un 4, cada R por un 5, cada N por un 6 y cada ; por un 7 obtendremos el siguiente número:

31 3 32 5 311 7 311 3 3 5 3111 7 3111 3 322 5 31111 7 31111 3 3 5 31 7

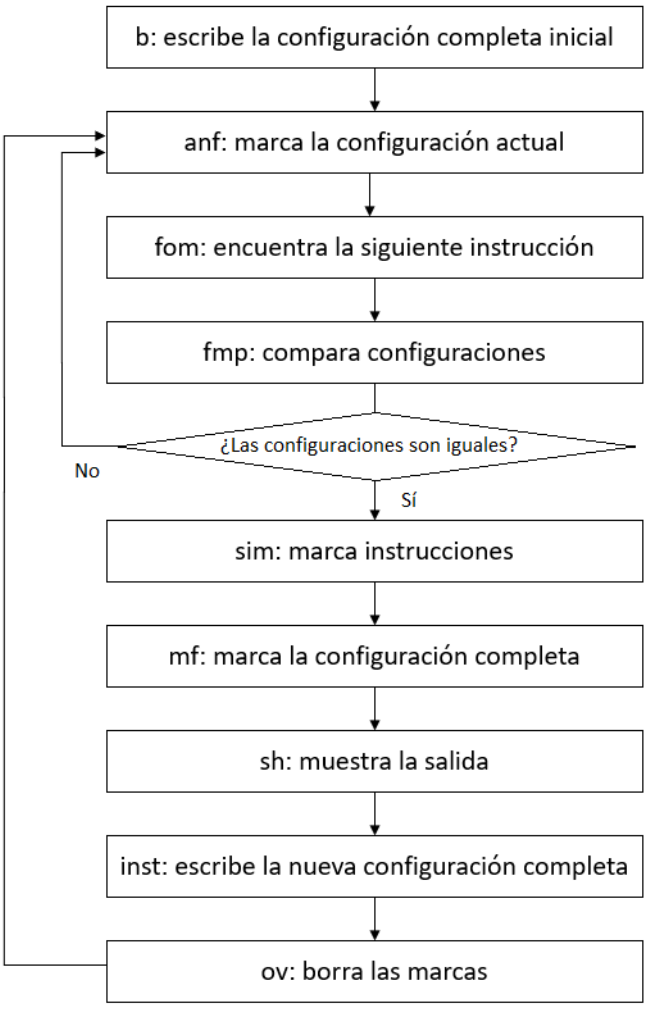
A este número se le llama **número de descripción** de la máquina de Turing M y lo denotaremos por $ND(M)$.

Este resultado es muy importante ya que de esta forma somos capaces de asociar a cada máquina de Turing un único entero positivo, lo cual nos será muy útil a la hora de demostrar varios resultados teóricos.

3.1.2. Funcionamiento de la máquina universal

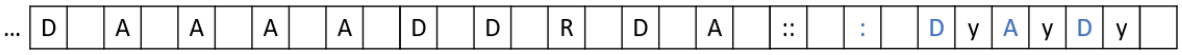
Ahora que sabemos cómo debemos introducir la información en la máquina Universal veamos como funciona. Para ello nos apoyaremos en el ejemplo de la máquina que computa la secuencia 0101010101...

Como hemos dicho antes la máquina universal está compuesta por distintas tablas esqueleto, en concreto por 9 tablas principales que se apoyan en otras tablas esqueleto auxiliares, lo que genera una gran cantidad de estados y configuraciones. En este capítulo nombraremos las 9 tablas principales, explicaremos qué hace cada una ellas y omitiremos las tablas auxiliares. En el apéndice se puede encontrar la programación completa de la máquina universal en Python.

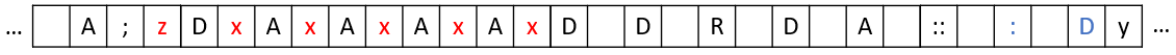


Como acabamos de ver a la máquina universal debemos introducirle la descripción estándar de la máquina que queremos simular. Una vez introducida la tabla esqueleto **b** se encarga de escribir ':' al final de la descripción estándar y a continuación escribe los símbolos DAD que corresponden con la configuración completa inicial. Esto es obvio ya que la configuración inicial de cualquier máquina va a ser el estado inicial q_1=DA junto con un espacio en blanco=D ya que inicialmente la cinta está vacía. Una vez realizados estos movimientos pasa a la siguiente tabla esqueleto.

anf se encarga de marcar la configuración actual con y. Entenderemos por marcar un símbolo por ejemplo con y a escribir en la celda impar situada a su derecha una y. Y por configuración actual entenderemos aquella situada a la derecha de los últimos dos puntos.

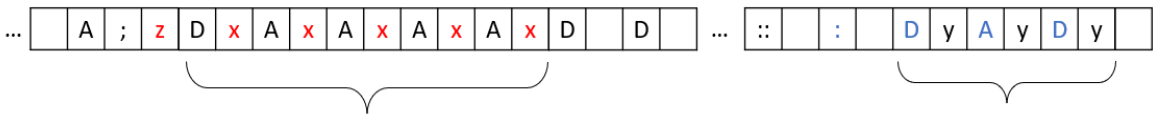


Una vez marcada la configuración actual la máquina tiene que encontrar aquel bloque de instrucciones que tenga la misma configuración. Para ello hay que revisar todas las instrucciones. De esto se encarga **fom**. Esta tabla esqueleto busca, de izquierda a derecha, la primera instrucción no marcada (al inicio todos estarán sin marcar). Como están separados por ; escribe después del ; una z para marcar la instrucción que va a estudiar, marca su configuración con una x y pasa al proceso de comparación.

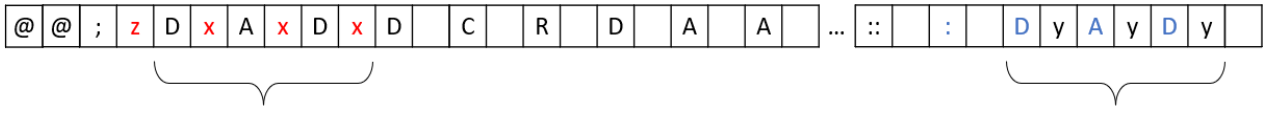


En este caso ha marcado la primera instrucción que ha encontrado ya que no estaba marcada.

Ahora la tabla **fmp** se encarga de comparar la configuración que ha marcado fom con la configuración marcada inicialmente. Si no son iguales borra todas las x e y y pasa de nuevo a la tabla anf para buscar el siguiente bloque de instrucciones no marcado. Notar que la z no se borra para indicar las instrucciones que ya hemos revisado. Si las configuraciones son iguales hemos encontrado el bloque de instrucciones a ejecutar y pasamos a la siguiente tabla.



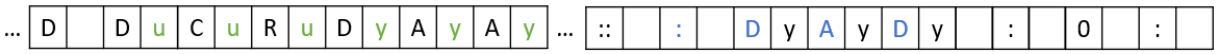
En el caso de la máquina que estamos estudiando, como la primera vez que ejecutamos fmp las configuraciones a comprar son distintas, borraremos las x, dejaremos la z e iremos en busca de la siguiente instrucción. En concreto en este caso repetiremos el procedimiento hasta llegar a la primera instrucción donde encontraremos la misma configuración. Esto tiene sentido ya que la primera vez que ejecutamos la máquina realizamos las instrucciones del estado inicial y este está al inicio de la cinta.



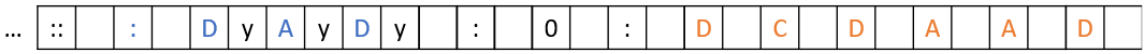
sim se va a encargar de marcar la configuración actual de la instrucción con espacios en blanco, es decir, borra las x. Además, va a marcar la instrucción a realizar (que comprende el símbolo a escribir y el movimiento a realizar) con u y el estado final con y.

Una vez realizado esto **ml** se encarga de volver a la última configuración completa (que encontraremos después del último ':') y marca el símbolo inmediatamente anterior al estado q_i con una x, los símbolos anteriores con una v y los posteriores al estado y al símbolo con una w. Además pone ':' al final de la configuración completa. En este caso no tenemos símbolos que marcar ya que la configuración completa inicial consta de muy pocos elementos.

Una vez marcada la configuración completa **sh** se encarga de examinar la instrucción marcada para ver si la máquina realiza alguna salida. Si esta lo hace, es decir, si la instrucción ordena escribir un 0 o un 1 **sh** se encarga de escribirlo al final de la cinta, después de los dos puntos que marcan el final de la última configuración completa.



Ahora **inst** se encarga de escribir la nueva configuración completa al final de la cinta, después de los últimos ':'. La nueva configuración completa incluirá los símbolos escritos en la cinta, el nuevo estado y el símbolo que estemos leyendo en la celda actual. Notar que en función de si la instrucción indica desplazarse una posición a la derecha, izquierda o no moverse se leerá un símbolo distinto.



Por último, **ov** se encarga de borrar todas las marcas que queden en la cinta y vuelve al estado anf para volver a repetir el proceso.

La salida completa tras varias iteraciones sería la siguiente:

@	@	;		D		A		D		D		C		R		D		A		A		;		D		A		A		
D		D		R		D		A		A		A		;		D		A		A		A		D		D	u	C	u	
C	u	R		D	y	A	y	A	y	A	y	A	y	;		D		A		A		A		A		D		D		
R		D		A		::		:		D		A		D		:		0		:		D		C		D		A		
A		D		:		D		C		D		D		A		A		A		D		:		1		:		D		
C		D		D		C		C		D		A		A		A		A		D		:		D		C		D		
D		C		C		D		D		A		D		:		0		:		D		C		D		D		C		
C		D		D		C		D		A		A		D		:		D		C		D		D		C		C		
D		D		C		D		D		A		A		A		D		:		1		:		D		C		D		
D		C		C		D		D		C		D		D		C		C		D		A		A		A		A		
D		:		D		C		D		D		C		C		D		D		C		D		D		C		C		
D		D		A		D		:		0		:		D		C		D		D		C		C		D		D		
C		D		D		C		C		D		D		C		D		A		A		D		:		D		C		

Interpretar esta salida puede ser un poco confuso así que vamos a desglosarla para entenderla mejor. Reescribiremos el código en función de los estados q_1, q_2, q_3, q_4 (DA,DAA,DAAA,DAAAA respectivamente) y de los símbolos ' ','0','1' (D,DC,DCC). Además omitiremos las celdas impares para tener más espacio.

La cinta inicialmente contiene los dos símbolos "@", la descripción estándar de la máquina que se va a ejecutar (en este caso sus cuatro instrucciones) y la configuración inicial:

@	@					
;	q1		0	R	q2	
;	q2			R	q3	
;	q3		1	R	q4	
;	q4			R	q1	::
:	q1					

A continuación la Máquina Universal va generando configuraciones completas. Las once primeras tienen el siguiente aspecto:

@	@											
;	q1		0	R	q2							
;	q2			R	q3							
;	q3		1	R	q4							
;	q4			R	q1	::						
:	q1		:	0								
:	0	q2										
:	0		q3		:	1						
:	0		1	q4								
:	0		1		q1		:	0				
:	0		1		0	q2						
:	0		1		0		q3		:	1		
:	0		1		0		1	q4				
:	0		1		0		1		q1		:	0
:	0		1		0		1		0	q2		
:	0		1		0		1		0		q3		:	1

3.2. Resultados teóricos

Para dar respuesta el Entscheidungsproblem Turing necesitó demostrar previamente algunos resultados teóricos y lo hizo apoyándose en el sistema conceptual que había creado. En este apartado veremos estos resultados y los completaremos.

En primer lugar vamos a introducir un resultado en el que nos apoyaremos para demostrar el primer teorema:

Teorema 3.1. *Sea B un conjunto no vacío. B es numerable si existe una inyección entre B y los números naturales.[7]*

A partir de este teorema demostramos lo siguiente:

Teorema 3.2. *Los números computables son numerables. Además, son infinitos.*

Demostración. Para probar que los números computables son numerables demostraremos que existe una aplicación inyectiva entre el conjunto de los números computables y los naturales.

Sea c un número computable. c estará generado por, al menos, una máquina de Turing (en general habrá varias máquinas distintas que generen el mismo número). Sea entonces M_c el conjunto de máquinas de Turing que generan c, $M_c = \{M|\ M$ genera c$\}$. Toda máquina $M \in M_c$ tiene un número de descripción. Tomemos N_c como el menor número de descripción de entre las máquinas de M_c, $N_c = min_{M \in M_c}(ND(M))$.

La función $f : c \to N_c$, que a cada número computable le asocia el menor número de descripción de todas las máquinas que lo computan, es la aplicación que buscábamos. En efecto es una función con dominio el conjunto de los números computables y recorrido $\mathbb{N}$. Además, si c y d son números computables, $f(c) = f(d) \Rightarrow c = d$ ya que si dos máquinas de Turing tienen el mismo número de descripción entonces tendrán las mimas instrucciones y generarán el mismo número.Por tanto la aplicación f es inyectiva.

Por otro lado veamos que son infinitos. Para todo $n \in \mathbb{N}$ el número racional 2^{-n} es computable ya que es sencillo construir una máquina de Turing no circular que lo calcule: basta con escribir n-1 ceros seguidos de un 1 y posteriormente infinitos ceros. Por tanto hemos encontrado un conjunto infinito de números computables.

□

De este resultado se deduce que podemos escribir los números computables como una secuencia. Sea pues A la secuencia de los números computables y sea α_n el n-ésimo número de dicha secuencia. Definimos la función $\phi_m(\alpha_n)$ como el m-ésimo dígito de α_n y definimos el número β como $\phi_n(\beta) = 1 - \phi_n(\alpha_n)$. Tenemos el siguiente resultado:

Teorema 3.3. *β no es computable.*

Demostración. Para todo n se cumple que $\phi_n(\beta)$ y $\phi_n(\alpha_n)$ son distintos. Es decir, β no está incluido en la secuencia de números computables. □

Del resultado anterior se sigue el siguiente teorema:

Teorema 3.4. *No existe un algoritmo, es decir, una máquina de Turing tal que dado un número natural m determine si este es el número de descripción de una máquina de Turing no circular.*

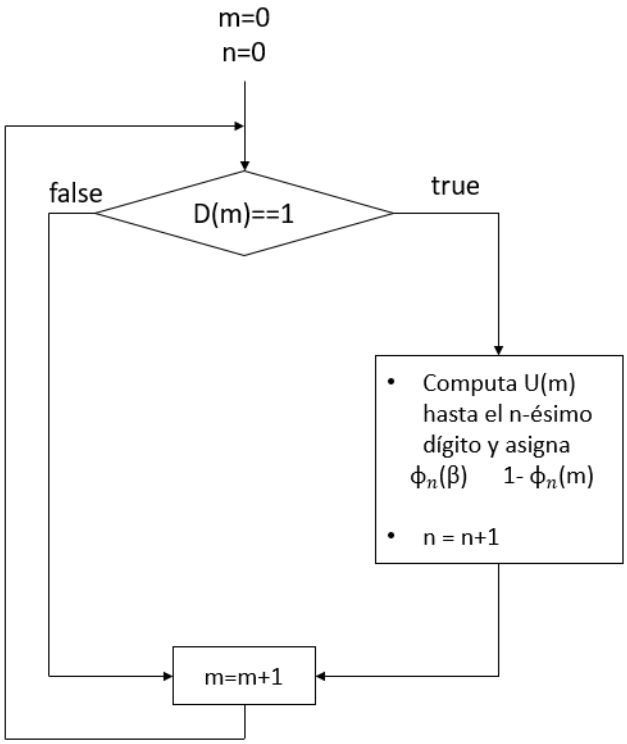
Vamos a ver dos demostraciones distintas de este teorema: una de ellas mediante la construcción de β y otra mediante la construcción de β', cuyo n-ésimo dígito es $\phi_n(\alpha_n)$. En ambos casos y en los resultados siguientes utilizaremos el método de reducción al absurdo para probar los teoremas. Además de ello, haremos uso del método de reducción. Una reducción consiste en transformar un problema en otro de forma que una solución al segundo problema sea válida para resolver el primero. Si somos capaces de demostrar que dos problemas son equivalentes, es decir, que el primero se reduce al segundo, y podemos dar una solución al segundo problema entonces tendremos una respuesta para el primero. [6]

Demostración 1: Probaremos que si existe dicha máquina entonces podríamos computar β.

Supongamos que existe una máquina de Turing, llamémosla D, tal que, al introducirle un entero positivo m, es capaz de determinar si dicho número es el número de descripción de una máquina de Turing no circular. Si lo es, $D(m)$ devuelve un 1 y sino devuelve un 0. Entonces a partir de ella podemos crear un algoritmo para calcular β. El algoritmo tendría un funcionamiento similar al siguiente:

- Inicializamos dos contadores m=0 y n=0 que recorran los números naturales.
- Introducimos el número m en la máquina D y esta decidirá si m es el número de descripción de una máquina no circular.
    - Si $D(m) == 1$ introducimos dicho número en la Máquina Universal de Turing (U), ejecutamos dicha máquina y calculamos la secuencia que computa. Tomamos el n-ésimo dígito y calculamos $\phi_n(\beta)$, el n-ésimo dígito de β. Por último, añadimos una unidad a m y a n.
    - Si $D(m) == 0$ añadimos una unidad a m.

Gráficamente el algoritmo se vería así:



Por tanto podríamos construir una máquina de Turing que computase β. Esto contradice el teorema 3.3 ya que hemos visto que β no es computable, por tanto no existe dicha máquina.

□

Demostración 2: Supongamos de nuevo que existe la máquina D. Combinando las máquinas D y U construimos una nueva máquina a la que llamamos H y que computa la secuencia β'. Recordemos que β' se define como $\phi_n(\beta') = \phi_n(\alpha_n)$. Probaremos que si existe D, H es una máquina no circular.

El funcionamiento de la máquina H es análogo al de la presentada en la demostración anterior. Se trata de un proceso iterativo donde cada iteración se realiza en un tiempo finito. En primer lugar la máquina D, que por hipótesis existe, evalúa cada entero positivo m y devuelve su resultado en un tiempo finito. Si $D(m) == 0$ la iteración ha terminado. Si $D(m) == 1$ entonces m es el número de descripción de una máquina no circular, m se introduce en U y esta calcula los n primeros dígitos en un tiempo finito.

Supongamos que llevamos escritos los n primeros dígitos de β', es decir, hemos encontrado n máquinas de Turing no circulares. Como existen infinitas máquinas no circulares (recordemos que hemos probado que los números computables son infinitos, por tanto habrá infinitas máquinas de Turing) siempre existirá un m'>m tal que m' sea el número de descripción de una máquina no circular. Alcanzar m' y computar su n+1 dígito (el n+1 dígito de β) tomará un tiempo finito. Siempre vamos a ser capaces de escribir el siguiente dígito. Por tanto, H es no circular.

Sea ahora K el número de descripción de H y n_k el número de dígitos de β calculados. Al llegar a la K-ésima iteración $D(K)$ devolverá un 1 ya que H es una máquina no circular y U ejecutará la máquina H para calcular sus dígitos. Esto es, volveremos a empezar el proceso de estudiar todos los números naturales m a partir de 0. Al llegar a m=K, U volverá a ejecutar la máquina H desde el inicio. La máquina H se invoca a sí misma recursivamente cada vez que llega a m=k y nunca llegará a calcular el dígito $n_k + 1$, luego H es una máquina circular. Esto nos lleva a una contradicción. Esta proviene de suponer que la hipótesis de partida (la existencia de D) es cierta, luego no existe una máquina D capaz de determinar si un número natural es el número de descripción de una máquina no circular.

□

El resultado que acabamos de probar es fundamental para demostrar el siguiente teorema el cual tiene una aplicación directa a la hora de dar una respuesta al problema de la decisión.

Teorema 3.5. *No existe una máquina ε tal que, dado un número de descripción m de una máquina cualquiera M, determine si M escribe alguna vez un determinado símbolo, por ejemplo un 0.*

Para abordar esta demostración primero veremos que si existe dicha máquina ε entonces existirá un método de determinar si una máquina M escribe un número infinito de 0 y veremos como esto nos lleva a una contradicción.

Demostración. Supongamos que existe la máquina ε y sea M una máquina cualquiera. Si M escribe alguna vez un 0 entonces $\varepsilon(M)$ devuelve un 1 y si M no escribe nunca un un 0 entonces $\varepsilon(M)$ devuelve un 0. Definimos M_1 como la máquina que computa la misma secuencia que M excepto en la posición donde aparece el primer 0, donde M_1 escribirá un $\overline{0}$. Por ejemplo, si M computa la secuencia

$$ABA01AAB0010AB...$$

M_1 computará la secuencia

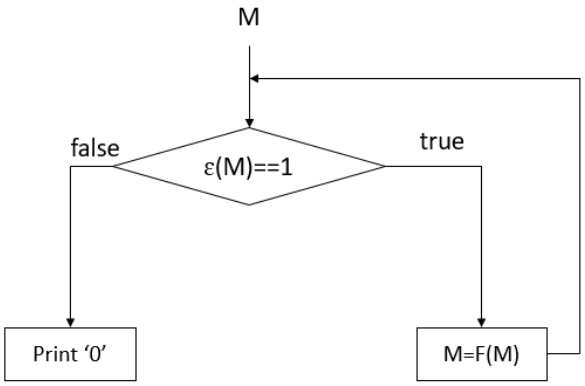
$$ABA\overline{0}1AAB0010AB...$$

Definimos del mismo modo M_2 como la máquina que escribe la misma secuencia que M excepto en las dos primeras posiciones donde aparece un 0. En este caso la secuencia que escribe M_2 será

$$ABA\overline{0}1AAB\overline{0}010AB...$$

y así sucesivamente con M_i.

Sea F una máquina tal que, al introducirle la descripción estándar de M devuelva M_1. Usaremos F para ir generando consecutivamente las descripciones estándar de las máquinas M_1, M_2...: $M_1 = F(M), M_2 = F(M_1) = F(F(M))$... Si combinamos F con ε obtenemos una máquina G que funciona de la siguiente forma: supongamos que tenemos la descripción estándar de nuestra máquina M. ε se encarga de testearla y si $\varepsilon(M) == 1$ es cierto eso quiere decir que M escribe alguna vez un 0. En ese caso no realizamos ningún cambio en la cinta y le pedimos a F que nos genere la descripción estándar de M_1 y volvemos a repetir el proceso. Por otro lado si $\varepsilon(M) == 1$ es falso, eso quiere decir que M no escribe ningún 0. En ese caso escribimos un 0 en la cinta. Notar que si M_i no escribe ningún 0 entonces M_j con j>i $\forall i, j \in \mathbb{N}$ tampoco va a escribir ningún 0, por tanto no es necesario seguir con el proceso. Veamos una representación de G:



Ahora usemos la máquina ε para comprobar la máquina G. Si ε detecta que G nunca escribe un 0 quiere decir que M escribe infinitos 0 y $\forall i$, M_i va a escribir algún 0. En caso contrario, si G escribe algún

0 quiere decir que hay algún i tal que M_i deja de escribir 0 y por tanto M va a escribir un número finito de 0. Es decir, hemos encontrado un método para determinar si M escribe infinitos 0 o no.

Del mismo modo podemos comprobar si M escribe un número infinito de 1, y combinando ambos procesos hemos encontrado una forma de saber si una máquina escribe un número infinito de símbolos de primer tipo. Y esto es equivalente a haber encontrado un método para determinar si, a partir de su número de descripción, una máquina es no circular, lo cual contradice el teorema anterior. Por tanto no existe la máquina ε.

□

De este resultado es deducible el siguiente teorema:

Teorema 3.6. *No existe un algoritmo capaz de determinar si una máquina escribe infinitos 0.*

Capítulo 4

El problema de la decisión

Una vez entendida la teoría introducida por Turing y vistos los resultados teóricos tenemos las herramientas suficientes para dar respuesta al problema de la decisión. Sin embargo, antes debemos familiarizarnos con ciertas nociones de la lógica proposicional.

La lógica proposicional es un sistema formal formado por proposiciones y operadores que, combinados, forman proposiciones más complejas. Emplearemos las siguientes propiedades:

Lema 1. Expresiones lógicas.

1. $(X \to Z) \to (X \& Y \to Z \& Y)$
2. $[(X \to Y) \& (Y \to Z)] \to (X \to Z)$
3. $X \to (Y \to Z)$ equivale a $(X \to Y) \to (X \to Z)$
4. $(X \to Z) \to (X \& Y \to Z)$

Recordemos que el operador lógico & significa "y", v significa "o" y $\to$ nos indica, para cada proposición, un antecedente y un consecuente.

4.1. El problema de la decisión

Si recordamos el programa de Hilbert, este comprendía las propiedades de independencia, consistencia, completitud y decidibilidad. Gödel demostró que existen proposiciones $\mathfrak{U}$ tales que ni $\mathfrak{U}$ ni -$\mathfrak{U}$ son demostrables, lo que supuso la imposibilidad de dar una prueba de la completitud del sistema formado por la lógica de primer orden junto a los axiomas de la aritmética. Turing sin embargo fue más allá y demostró que no existe un método general que nos diga si dada una fórmula $\mathfrak{U}$ esta es demostrable en dicho sistema. Y lo hizo de la siguiente forma:

Para cada máquina M construiremos una fórmula lógica $\mathfrak{U}_n(M)$ y veremos que decidir si dicha fórmula es demostrable o no se reduce a decidir si la máquina M escribe un 0 alguna vez. Y como el segundo problema no es decidible el primero tampoco lo será.

Antes de definir la fórmula $\mathfrak{U}_n(M)$ introduzcamos algunas nociones necesarias:

Por un lado necesitaremos las siguientes funciones proposicionales:

- $R_S(x,y)$: en la configuración completa x el símbolo en la celda y es S
- $I(x,y)$: en la configuración completa x la celda escaneada es y
- $K_{q_m}(x)$: en la configuración completa x el estado es q_m
- $F(x,y)$: y es el sucesor inmediato de x

Además, definiremos también las siguientes funciones:

- $r(n,m)$: índice del m-ésimo símbolo de la n-ésima configuración completa
- $i(n)$: número de celda leída en la n-ésima configuración
- $k(n)$: índice del estado de la n-ésima configuración

A partir de ellas generaremos la siguiente expresión lógica,

$$(x,y,x',y')\{K_{q_i}(x)\&I(x,y)\&R_{S_j}(x,y)\&F(x,x')\&F(y',y) \rightarrow$$

$$K_{q_l}(x')\&I(x',y')\&R_{S_k}(x',y)\ \&(z)[F(y',z)v(R_{S_j}(x,z) \rightarrow R_{S_j}(x',z))])\}$$

la cual abreviaremos por Inst$\{q_iS_jS_kLq_l\}$ y cuya interpretación es la siguiente:
para todo x, y, x', y', si en la x-ésima configuración completa el estado es q_i, además en esa misma configuración estamos leyendo la celda y-ésima y el carácter leído es S_j y x' es el sucesor inmediato de x e y es el sucesor inmediato de y' entonces en la configuración completa x' ocurrirá lo siguiente:

- Estaremos en el estado q_l
- La celda escaneada será y$'$
- En la celda y-ésima aparecerá el símbolo S_k
- para toda celda z distinta de y el símbolo en la siguiente configuración completa es el mismo.

De la misma forma se construyen Inst$\{q_iS_jS_kRq_l\}$ y Inst$\{q_iS_jS_kNq_l\}$.

Ahora dada una máquina M transformaremos todas sus instrucciones en expresiones de la forma Inst$\{q_iS_jS_kRq_l\}$, Inst$\{q_iS_jS_kLq_l\}$ o Inst$\{q_iS_jS_kNq_l\}$ y tomaremos su suma lógica, que no es mas que su conjunción. A la expresión resultante la llamaremos Des(M).
Ahora sí, la fórmula $\mathfrak{U}_n(M)$ propuesta por Turing será la siguiente:

$$(\exists u)\ [N(u)\&(x)(N(x) \rightarrow (\exists x')F(x,x'))\&(y,z)(F(y,z) \rightarrow N(y)\&N(z))$$

$$\&(y)R_{S_0}(u,y)\&I(u,u)\&K_{q_1}$$

$$\&Des(M)]$$

$$\rightarrow (\exists s)(\exists t)[N(s)\&N(t)\&R_{S_1}(s,t)]$$

Abreviaremos $[N(u)\&...\&Des(M)]$ por $A(M)$.
En esta fórmula $\mathfrak{U}_n(M)$ el antecedente $A(M)$ consta de tres bloques principales: por un lado, la conjunción de los axiomas de Peano (primera línea) que permiten definir los números naturales. N es una función proposicional. $N(x)$ devuelve "true" si x es un entero y "false" en caso de que no lo sea. u es el primer número natural, en nuestro caso el 0, a partir del cual se construyen los demás aplicando la función F. Notar que falta la noción de unicidad. Esto es una errata del artículo de Turing pero que no impide el correcto desarrollo del problema. Después, en la segunda línea, está la configuración inicial de la máquina M. Por último, en la tercera, está la descripción de la máquina M (la conjunción de sus instrucciones). El consecuente, situado a la derecha del operador $\rightarrow$ significa: en alguna configuración completa de la máquina M aparece un 0 en la cinta.
Ahora nuestro cometido es ver si somos capaces de determinar si $\mathfrak{U}_n(M)$ es demostrable. Como hemos mencionado en el capítulo anterior, la reducción nos permite obtener la solución a un problema a

través de otro. En nuestro caso vamos a reducir nuestro problema de determinar si $\mathfrak{U}_n(M)$ es demostrable al problema de determinar si el símbolo 0 aparece en la cinta en alguna configuración de la máquina M. Como en el capítulo anterior hemos dado una respuesta a este segundo problema tendremos una respuesta directa al Enstcheidungsproblem.

Veamos pues que ambos enunciados son equivalentes, es decir:

- Si 0 aparece en la cinta en alguna configuración completa de M entonces $\mathfrak{U}_n(M)$ es demostrable.
- Si $\mathfrak{U}_n(M)$ es demostrable entonces 0 aparece en la cinta en alguna configuración completa de M.

Lema 2. Si 0 aparece en la cinta en alguna configuración completa de M entonces $\mathfrak{U}_n(M)$ es demostrable.

Demostración. Supongamos que estamos en la n-ésima configuración de la máquina M. Los símbolos de las celdas de la cinta serán $S_{r(n,0)}, S_{r(n,1)}, S_{r(n,2)} ... S_{r(n,n)}$ seguidos de infinitos espacios en blanco ya que como estamos en la configuración n-ésima como mucho habremos llegado hasta la celda n. Sea $i(n)$ la celda escaneada y $q_{k(n)}$ el estado en el que nos encontremos. Entonces podemos formar la siguiente proposición que describe la n-ésima configuración completa y que abreviaremos por CC_n:

$$R_{S_{r(n,0)}}(u^{(n)}, u) \& R_{S_{r(n,1)}}(u^{(n)}, u') \& R_{S_{r(n,2)}}(u^{(n)}, u'') \& ... \& R_{S_{r(n,n)}}(u^{(n)}, u^{(n)})$$

$$\& I(u^n, u^{(i(n))}) \& K_{q_{k(n)}}$$

$$\&(y)(F(y,u') \mathrm{v} F(u,y) \mathrm{v} F(u',y) \mathrm{v} ... \mathrm{v} F(u^{(n-1)}, y) \mathrm{v} R_{S_0}(u^{(n)}, y))$$

Tomamos u como el elemento inicial de forma que u' es el sucesor inmediato de u, u'' es el sucesor inmediato de u' y así sucesivamente hasta $F(u^{(n-1)}, u^{(n)})$. Abreviaremos $F(u,u') \& F(u',u'') \& ... \& F(u^{(r-1)}, u^{(r)})$ por $F^{(r)}$. Lo que vamos a probar a continuación es que la proposición $A(M) \& F^{(n)} \to CC_n$ es demostrable y la denotaremos por CF_n. Lo haremos por el método de inducción.

Sea n=0. Entonces la expresión CC_0 se traduce en

$$(y) R_{S_0}(u,y) \& I(u,u) \& K_{q_1}$$

que quiere decir que toda celda y en la configuración completa inicial u está vacía, que estamos mirando la primera celda y que nos encontramos en la configuración inicial q_1. Esto no es otra cosa que la descripción inicial de la fórmula $\mathfrak{U}_n(M)$ descrita en $A(M)$. Por otro lado, como estamos en la configuración inicial tenemos el primer elemento u pero no tenemos su siguiente y podemos omitir F^0. Por tanto, $A(M) \to CC_0$ es cierto y CF_0 es demostrable.

Supongamos entonces que la fórmula es cierta hasta n y veamos si se cumple para n+1. Es decir, tenemos que ver que $CF_n \to CF_{n+1}$. Como tenemos distintos casos a tener en cuenta vamos a suponer que en la configuración que vamos a estudiar la máquina se va a desplazar a la izquierda.

Supongamos que tenemos los siguientes valores de los índices $b = r(n, i(n))$, $d = r(n+1, i(n+1))$, $a = k(n)$, $c = k(n+1)$. Entonces $Des(M)$ deberá incluir la siguiente instrucción $\mathrm{Inst}\{q_a S_b S_d L q_c\}$. Por tanto es obvio que

$$Des(M) \to Inst\{q_a S_b S_d L q_c\}$$

y como $Des(M)$ está incluido en $A(M)$ es claro que

$$A(M) \to Des(M) \to Inst\{q_a S_b S_d L q_c\}$$

y aplicando la expresión lógica 1.1 del lema 1 tenemos

$$A(M)\&F^{(n+1)} \rightarrow Inst\{q_aS_bS_dLq_c\}\&F^{(n+1)} \tag{4.1}$$

Pero además, la instrucción Inst$\{q_aS_bS_dLq_c\}$ construye la configuración completa CC_{n+1} a partir de la configuración CC_n ya que partiendo del estado de la configuración n-ésima y del símbolo que esté leyendo nos indica qué símbolo escribir, qué movimiento realizar y a qué estado pasar. Por tanto si añadimos $F^{(n+1)}$ a la instrucción tendremos que

$$Inst\{q_aS_bS_dLq_c\}\&F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1}) \tag{4.2}$$

es demostrable y aplicando la expresión lógica 1. 2 a (4.1) y (4.2) tenemos que

$$A(M)\&F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1}) \tag{4.3}$$

es demostrable, y aplicando 1.3 a (4.3) obtenemos que

$$(A(M)\&F^{(n+1)} \rightarrow CC_n) \rightarrow (A(M)\&F^{(n+1)} \rightarrow CC_{n+1}) \tag{4.4}$$

Ahora por hipótesis de inducción se cumple que $A(M)\&F^{(n)} \rightarrow CC_n$, y por la expresión lógica 1.4

$$A(M)\&F^{(n)}\&F(u^{(n)},u^{(n+1)}) \rightarrow CC_n$$

que es lo mismo que

$$A(M)\&F^{(n+1)} \rightarrow CC_n \tag{4.5}$$

Y por último, aplicando del lema 1 la expresión 1.2 a la hipótesis de inducción, a (4.5) y a (4.4) tenemos que si

$$(A(M)\&F^{(n)} \rightarrow CC_n) \rightarrow (A(M)\&F^{(n+1)} \rightarrow CC_n)$$

y

$$(A(M)\&F^{(n+1)} \rightarrow CC_n) \rightarrow (A(M)\&F^{(n+1)} \rightarrow CC_{n+1})$$

entonces

$$(A(M)\&F^{(n)} \rightarrow CC_n) \rightarrow (A(M)\&F^{(n+1)} \rightarrow CC_{n+1})$$

que es lo mismo que

$$CF_n \rightarrow CF_{n+1} \tag{4.6}$$

Es decir, CF_n es demostrable para todo n.

Ahora recordemos que por hipótesis estamos suponiendo que el 0 aparece en la cinta en alguna configuración completa. Es decir, existen dos enteros N y K para los que CC_N tendrá como uno de

sus símbolos $R_{S_1}(u^{(N)}, u^{(K)})$. Si recordamos, CC_N contiene la descripción de la N-ésima configuración completa, incluyendo el símbolo que hay en cada una de las celdas. Es decir, $CC_N \rightarrow R_{S_1}(u^{(N)}, u^{(K)})$.

Además acabamos de demostrar que $CF_n \equiv A(M)\&F^{(n)} \rightarrow CC_n$ es demostrable, por tanto

$$A(M)\&F^{(N)} \rightarrow CC_N \tag{4.7}$$

Notar que como estamos en la configuración N-ésima como mucho habremos llegado hasta la celda N luego K $\leq$ N.

Además, tenemos que

$$(\exists u)A(M) \rightarrow (\exists u)(\exists u')...(\exists u^N)(A(M)\&F^N)$$

luego

$$(\exists u)A(M) \rightarrow (\exists u)(\exists u')...(\exists u^N)R_{S_1}(u^{(N)}, u^{(K)})$$

por tanto

$$(\exists u)A(M) \rightarrow (\exists u^N)(\exists u^K)R_{S_1}(u^{(N)}, u^{(K)})$$

es decir,

$$(\exists u)A(M) \rightarrow (\exists s)(\exists t)R_{S_1}(s,t)$$

y $\mathfrak{U}_n(M)$ es demostrable.

□

Veamos ahora el segundo lema:

Lema 3. Si $\mathfrak{U}_n(M)$ es demostrable entonces 0 aparece en la cinta en alguna configuración completa de M.

Demostración. Si transformamos las sentencias proposicionales de la fórmula $\mathfrak{U}_n(M)$ obtendremos una proposición verdadera. Por tanto si $\mathfrak{U}_n(M)$ es demostrable entonces S_1 aparecerá en alguna configuración completa de la máquina M. □

Por tanto hemos probado la equivalencia de los lemas y estamos en condiciones de dar una respuesta al problema de la decisión.

Supongamos que el Entscheidungsproblem tiene solución. Entonces existirá un proceso mecánico capaz de determinar si la fórmula $\mathfrak{U}_n(M)$ es demostrable. Pero por la equivalencia de los lemas hemos visto que determinar si $\mathfrak{U}_n(M)$ es demostrable es lo mismo que determinar si el 0 aparece en alguna configuración completa de la máquina M. Sin embargo, por el teorema 3.4 visto en el capítulo anterior, sabemos que esto no es posible y llegamos a una contradicción. Por tanto, el problema de la decisión no tiene solución.

Bibliografía

[1] DAVID HILBERT AND WILHELM ACKERMANN, *Principles of Mathematical Logic*, 2.ª ed., ISBN. 0821820249, Chelsea Publishing Company, 1950.

[2] KURT GÖDEL, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, Monatshefte für mathematik und physik journal, volume 38, pages 173–198, Springer, 1931.

[3] ALONZO CHURCH, *An unsolvable problem of elementary number theory*, American journal of mathematics, volume 58, number 2, pages 345–363, JSTOR, 1936.

[4] *Hilbert's Tenth Problem page*, `https://logic.pdmi.ras.ru/Hilbert10/`

[5] ALAN TURING, *On computable numbers, with an application to the entscheidungsproblem*, J. of math, 1936

[6] MICHAEL SIPSER, *Introduction to the Theory of computation*, segunda edición, ISBN 0-534-95097-3, PWS Publishing, 2006

[7] JAMES R. MUNKRES, *Topology*, segunda edición, ISBN 10: 1-292-02362-7, Pearson Education Limited, 2014

[8] MARK PRIESTLY, *A science of Operations*, ISBN 978-1-84882-555-0, Springer, 2011