

TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE
IZTAPALAPA

INTEGRANTES:

CUANENEMI CUANALO MARIO ALBERTO	181080030
FERMIN CRUZ ERIK	181080007
GUTIERREZ ARELLANO RAFAEL	181080022
PEREZ ARMAS FAUSTO ISAAC	181080037

ISC-6AM

LENGUAJES Y AUTOMATAS I
M.C. ABIEL TOMÁS PARRA HERNÁNDEZ

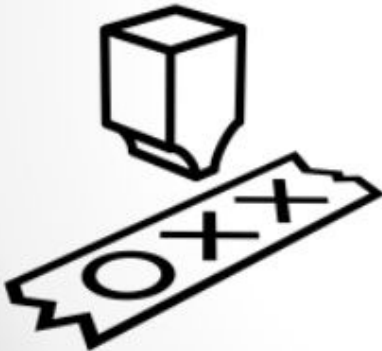
SEP 2020 / FEB 2021

ACTIVIDAD SEMANA 15

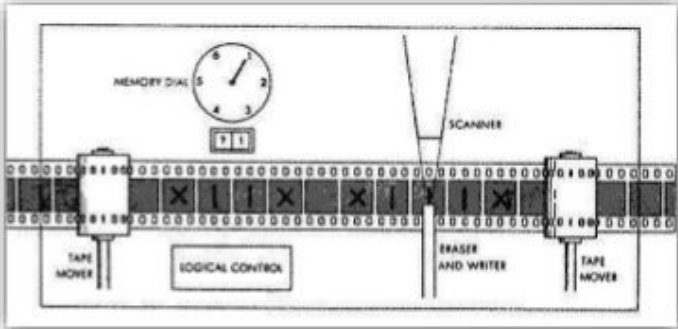
RESUMEN TEMA 8.1

Introducción a las máquinas de Turing

Máquinas de Turing



The diagram on the left shows a 3D perspective of a Turing Machine head positioned over a 2D representation of a tape. The tape is a long, narrow strip with a jagged right edge, containing the symbols '0', 'x', and 'x' from left to right.



The schematic diagram on the right illustrates the internal components of a Turing Machine. At the top center is a circular 'MEMORY DIAL' with a needle pointing to the number 1. Below it is a small rectangular box containing the number 11. A horizontal 'TAPE' runs through the center, marked with a sequence of symbols: 0, 1, x, 1, x, 1, x, 1, x, 0, 0, 0, 0, 0, 0, 0, 0. A 'SCANNER' is positioned above the tape, aligned with the first '0' on the right. Below the tape, there are two 'TAPE MOVER' mechanisms, one on the left and one on the right. A central box labeled 'LOGICAL CONTROL' is connected to the scanner and the tape movers. Below the scanner is a box labeled 'ERASER AND WRITER'. The entire machine is enclosed in a rectangular frame.

By :
Rosviannis Barreiro 14-1138
Listiany Agramonte 15-0737

Comenzaremos con una exposición informal, utilizando los conocimientos de la programación en C, para demostrar que existen problemas específicos que no se pueden resolver con una computadora. Estos problemas se conocen como “indecidibles”. Después presentaremos un venerable formalismo para computadoras: la máquina de Turing. Aunque una máquina de Turing no se parece

en nada a un PC, y sería enormemente ineficiente en el caso de que alguna empresa emprendedora las decidiera fabricar y vender, hace tiempo que la máquina de Turing se ha reconocido como un modelo preciso para representar lo que es capaz de hacer cualquier dispositivo físico de computación.

Problemas que las computadoras no pueden resolver.

El propósito de esta sección es proporcionar una introducción informal basada en la programación en C a la demostración de un problema específico que las computadoras no pueden resolver. El problema particular que vamos a abordar es si lo primero que imprime un programa C es hola, mundo. Aunque podríamos imaginar que la simulación del programa nos permitirá decir qué hace, en realidad tendremos que enfrentarnos a programas que tardan mucho tiempo en producir una respuesta. Este problema (no saber cuándo va a ocurrir algo, si es que alguna vez ocurre) es la causa última de nuestra incapacidad de decir lo que hace un programa. Sin embargo, demostrar formalmente que no existe un programa que haga una tarea establecida es bastante complicado, por lo que necesitamos desarrollar algunos mecanismos formales. En esta sección, proporcionamos el razonamiento intuitivo que hay detrás de las demostraciones formales.



La computadora es una máquina capaz de realizar y controlar a gran velocidad cálculos y procesos complicados que requieren una toma rápida de decisiones, no pueden pensar por sí mismas, no pueden resolver problemas ni tomar decisiones sin la intervención del hombre.

El primer programa en C con el que se encuentran los estudiantes que leen el clásico texto de Kernighan y Ritchie.¹ Es fácil descubrir que este programa imprime hola, mundo y termina. Este programa es tan transparente que se ha convertido en una práctica habitual presentar los lenguajes mostrando cómo escribir un programa que imprime hola, mundo en dichos lenguajes.

```
main()
{
    printf("hola, mundo\n");
}
```

Fermin Cruz Erik

El propósito de la teoría de los problemas indecidibles es proporcionar una guía a los programadores sobre lo que se puede o no conseguir a través de la programación.

La razón de ello es que mientras que los problemas indecidibles normalmente suelen resultar obvios y habitualmente no se intentan resolver, los problemas intratables se presentan continuamente. Además, a menudo dan lugar a pequeñas modificaciones de los requisitos o a soluciones heurísticas. Por tanto, el diseñador se enfrenta con frecuencia a tener que decidir si un problema es o no intratable, y qué hacer si lo es.

Una ventaja de la máquina de Turing sobre los programas como representación de lo que se puede calcular es que la máquina de Turing es lo suficientemente simple como para que podamos representar su configuración de manera precisa, utilizando una notación sencilla muy similar a las descripciones instantáneas de un autómata a pila.

Con la notación de la máquina de Turing, demostraremos que ciertos problemas, que aparentemente no están relacionados con la programación, son indecidibles.

Gutierrez Arellano Rafael

Problemas que las computadoras no pueden resolver

El propósito de esta sección es proporcionar una introducción informal basada en la programación en C a la demostración de un problema específico que las computadoras no pueden resolver. El problema particular que vamos a abordar es si lo primero que imprime un programa C es hola, mundo.

```
main()
{
    printf("hola, mundo\n");
}
```

Sin embargo, existen otros programas que también imprimen hola, mundo; a pesar de que lo que hacen no es ni mucho menos evidente. Mostramos otro programa que imprime hola, mundo. Toma una entrada n , y busca las soluciones enteras positivas de la ecuación $x^n + y^n = z^n$. Si encuentra una, imprime hola, mundo. Si nunca encuentra los enteros x , y y z que satisfacen la ecuación, continúa buscando de forma indefinida y nunca imprime hola, mundo.

```
int exp(int i, n)
/* calcula i a la potencia n */
{
```

```
    int ans, j; ans = 1;
    for (j=1; j<=n; j++) ans *= j; return(ans);
}
main ()
{
    int n, total, x, y, z; scanf("%d", &n);
    total = 3; while (1)
    {
        for (x=1; x<=total-2; x++)
            for (y=1; y<=total-x-1; y++)
            {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hola, mundo\n");
            }
        total++;
    }
}
```

Parece probable que, si los matemáticos tardaron 300 años en resolver una pregunta acerca de un único programa de 22 líneas, entonces el problema general de establecer si un determinado programa, para una entrada dada, imprime hola, mundo tiene que ser realmente complicado. De hecho, cualquiera de los problemas que los matemáticos todavía no han podido solucionar puede transformarse en una pregunta de la forma “¿imprime este programa, con esta entrada, el texto hola, mundo?”. Por tanto, sería totalmente extraordinario que consiguiéramos escribir un programa que examinara cualquier programa P y la entrada I para P , y estableciera si P , ejecutado para la entrada I , imprime o no hola, mundo. Demostraremos que tal programa no existe.

¿Por qué tienen que existir problemas indecidibles? Aunque es complicado demostrar que un problema específico, tal como el “problema de hola-mundo” que acabamos de ver, es indecidible, es bastante sencillo ver, mediante cualquier sistema que implique programación, por qué casi todos los problemas tienen que ser indecidibles. Recuerde que hemos establecido que un “problema” puede definirse como una cuestión acerca de si una cadena pertenece a un lenguaje. El número de lenguajes diferentes sobre cualquier alfabeto de más de un símbolo es no numerable. Es decir, no hay ninguna manera de asignar enteros a los lenguajes tal que todo lenguaje tenga asignado un entero y todo entero esté asignado a un lenguaje. Por otro lado, los programas que tienen cadenas finitas construidas con un alfabeto finito (normalmente un subconjunto del alfabeto ASCII) son contables. Es decir, podemos ordenarlos de acuerdo con su longitud, y los programas con la misma longitud, podemos ordenarlos lexicográficamente. Así, podemos hablar del primer programa, el segundo programa y, en general, del programa i -ésimo para

cualquier entero i . Como resultado, sabemos que existen infinitos menos programas que problemas. Si elegimos un lenguaje al azar, casi seguro que será un problema indecidible. La única razón por la que la mayoría de los problemas parecen ser decidibles es porque rara vez estaremos interesados en problemas elegidos al azar. En lugar de ello, tendemos a buscar problemas sencillos y bien estructurados, que a menudo son decidibles. Sin embargo, incluso entre los problemas de nuestro interés que pueden definirse de manera clara y sucinta, nos encontraremos con muchos que son indecidibles; el problema de hola-mundo es uno de ellos.

Perez Armas Fausto Isaac

El propósito de la teoría de los problemas indecidibles no es sólo establecer la existencia de tales problemas (una idea excitante por sí misma desde el punto de vista intelectual) sino proporcionar también una guía a los programadores sobre lo que se puede o no conseguir a través de la programación. La teoría también tiene un gran impacto práctico, como veremos en el Capítulo 10, al tratar problemas que aunque sean decidibles, requieren mucho tiempo para ser resueltos. Estos problemas, conocidos como “problemas intratables”, suelen plantear una mayor dificultad al programador y al diseñador de sistemas que los problemas indecidibles. La razón de ello es que mientras que los problemas indecidibles normalmente suelen resultar obvios y habitualmente no se intentan resolver, los problemas intratables se presentan continuamente. Además, a menudo dan lugar a pequeñas modificaciones de los requisitos o a soluciones heurísticas. Por tanto, el diseñador se enfrenta con frecuencia a tener que decidir si un problema es o no intratable, y qué hacer si lo es. Necesitamos herramientas que nos permitan determinar cuestiones acerca de la indecidibilidad o intratabilidad todos los días. La tecnología presentada en la Sección 8.1 resulta útil para cuestiones que tratan con programas, pero no se puede trasladar fácilmente a problemas en otros dominios no relacionados.

Una ventaja de la máquina de Turing sobre los programas como representación de lo que se puede calcular es que la máquina de Turing es lo suficientemente simple como para que podamos representar su configuración de manera precisa, utilizando una notación sencilla muy similar a las descripciones instantáneas de un autómata a pila. En cambio, aunque los programas en C tienen un estado, que implica a todas las variables en cualquier secuencia de llamadas a función que se realice, la notación para describir estos estados es demasiado compleja como para poder realizar demostraciones formales comprensibles. Con la notación de la máquina de Turing, demostraremos que ciertos problemas, que aparentemente no están relacionados con la programación, son indecidibles.

Notación para la máquina de Turing Podemos visualizar una máquina de Turing. La máquina consta de una unidad de control, que puede encontrarse en cualquiera de

un conjunto finito de estados. Hay una cinta dividida en cuadrados o casillas y cada casilla puede contener un símbolo de entre un número finito de símbolos

Inicialmente, la entrada, que es una cadena de símbolos de longitud finita elegidos del alfabeto de entrada, se coloca en la cinta. Las restantes casillas de la cinta, que se extiende infinitamente hacia la izquierda y la derecha, inicialmente almacenan un símbolo especial denominado espacio en blanco.

El espacio en blanco es un símbolo de cinta, pero no un símbolo de entrada, y pueden existir también otros símbolos de cinta además de los símbolos de entrada y del espacio en blanco. Existe una cabeza de la cinta que siempre está situada en una de las casillas de la cinta. Se dice que la máquina de Turing señala dicha casilla. Inicialmente, la cabeza de la cinta está en la casilla más a la izquierda que contiene la entrada. Un movimiento de la máquina de Turing es una función del estado de la unidad de control y el símbolo de cinta al que señala la cabeza. En un movimiento, la máquina de Turing:

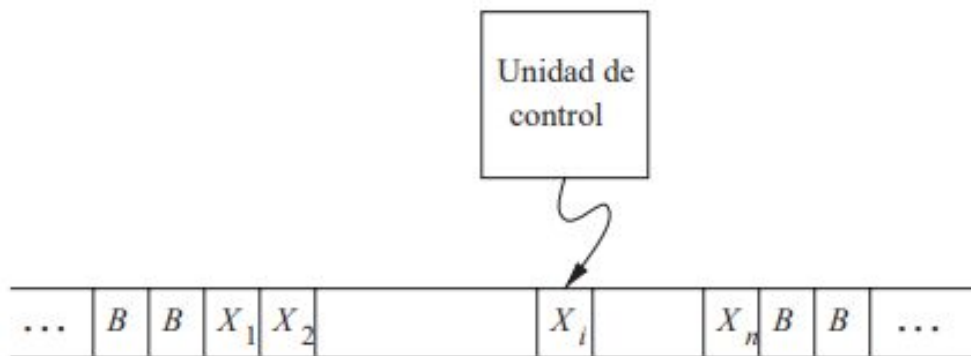


Figura 8.8. Una máquina de Turing.