

Tema 7: Máquinas de Turing

Serafín Moral

Universidad de Granada

Enero, 2015

- Introducción
- Máquinas de Turing
- Lenguajes recursivamente enumerables. Lenguajes recursivos
- Técnicas de construcción de Máquinas de Turing: memoria adicional, pistas múltiples, subrutinas
- Extensiones del concepto de MT
 - Movimientos estáticos
 - Máquinas multicinta
 - Máquinas no deterministas
- Limitaciones de las MT
 - Máquinas con cintas semiilimitadas
- Calculabilidad
 - Codificación de cadenas y MTs
 - El lenguaje diagonal
 - El lenguaje universal
 - Problemas indecidibles
 - Teorema de Rice

Enunciado

- Entrada: un programa y unos datos de entrada
- Salida: SI (cuando el programa termina para esos datos) y NO (cuando el programa cicla de forma indefinida para esos datos)

¿Por qué es difícil saber si un programa termina?

```
int exp(int i,n)
/* calcula i a la potencia n */
{
    int ans,j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    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");
            }
        total++;
    }
}
```

- El programa termina para una entrada n si y solo si existen números enteros positivos x, y, z tales que

$$x^n + y^n = z^n$$

- Ahora se sabe que no termina para $n > 2$ según el *último teorema de Fermat*, pero hace algunos años esto no se sabía.
- El teorema fue enunciado en 1637, pero no fue demostrado hasta 1995 por el matemático Andrew Wiles
- Se han necesitado más de 300 años para saber si un programa concreto termina, pero nosotros queremos un algoritmo que nos diga si cualquier programa termina!!
- Este es un problema indecidible. La teoría para demostrar esto la haremos usando Máquinas de Turing. La podríamos hacer usando programas en C, pero las demostraciones matemáticas serían más complejas.

Máquinas de Turing

Una **Máquina de Turing** (MT) es una séptupla $(Q, A, B, \delta, q_0, \#, F)$ en la que

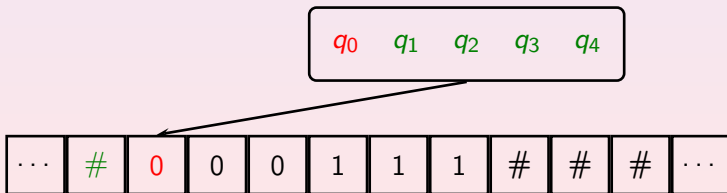
- Q es un conjunto finito de estados
- A es un alfabeto de entrada
- B es el alfabeto de símbolos de la cinta que incluye a A
- δ es la función de transición que asigna a cada estado $q \in Q$ y símbolo $b \in B$, el valor $\delta(q, b)$ que puede ser vacío (no definido) o una tripleta (p, c, M) donde $p \in Q, c \in B, M \in \{I, D\}$ donde I indica izquierda y D indica derecha.
- q_0 es el estado inicial
- $\#$ es un símbolo de $B \setminus A$ llamado símbolo blanco
- F es el conjunto de estados finales

Consideremos la MT

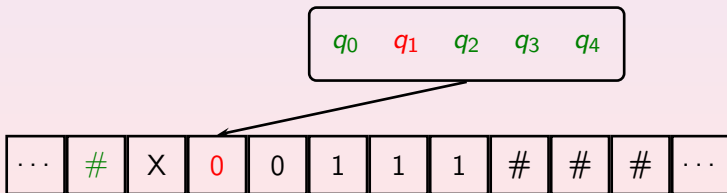
$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \#, \{q_4\})$ donde las transiciones no nulas son las siguientes:

$$\begin{array}{ll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) \\ \delta(q_1, 0) = (q_1, 0, D) & \delta(q_1, 1) = (q_2, Y, I) \\ \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) \end{array}$$

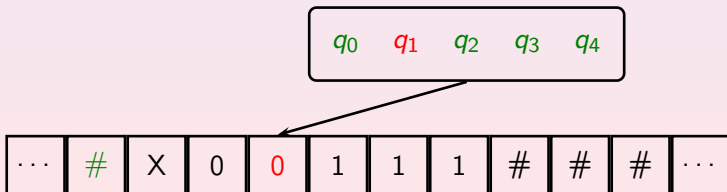
$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

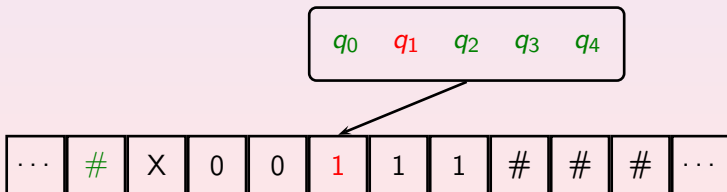


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



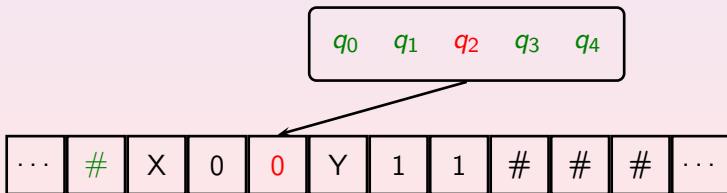
Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

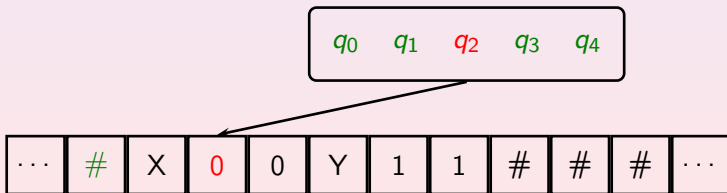


Funcionamiento

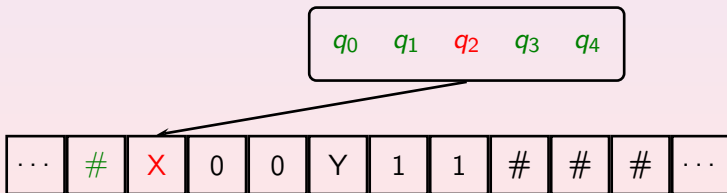
$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



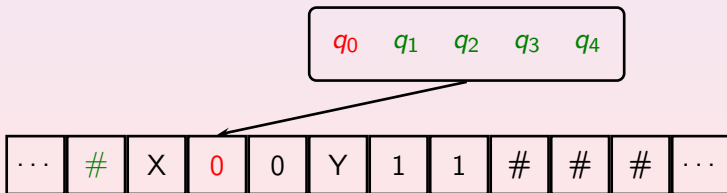
$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$



$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

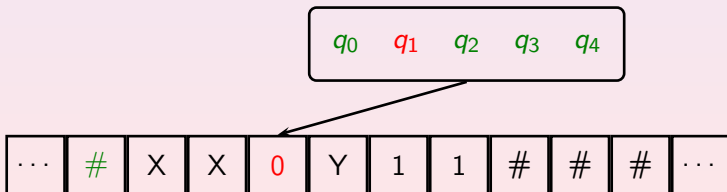


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



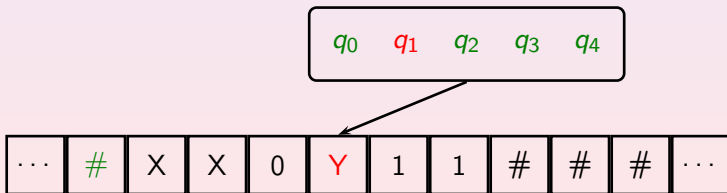
Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



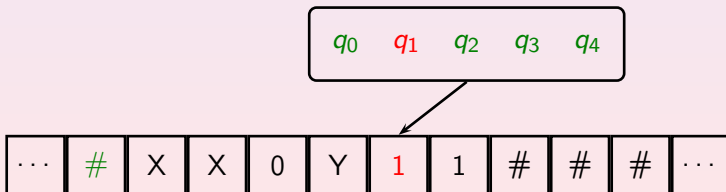
Funcionamiento

$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$

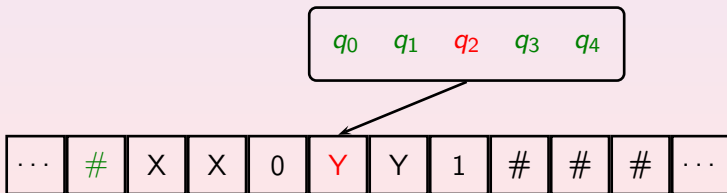


Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

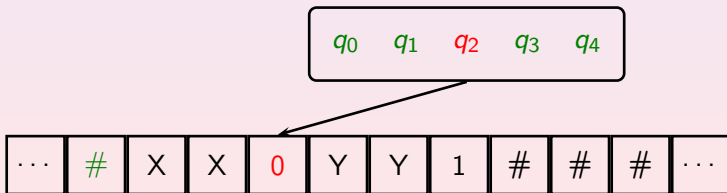


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

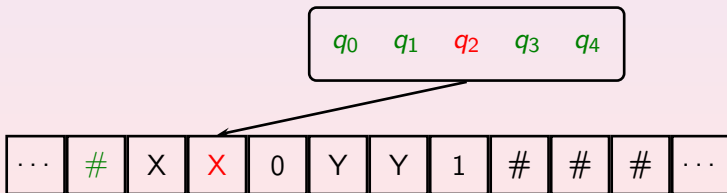


Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

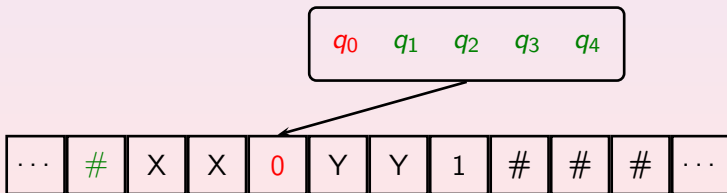


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



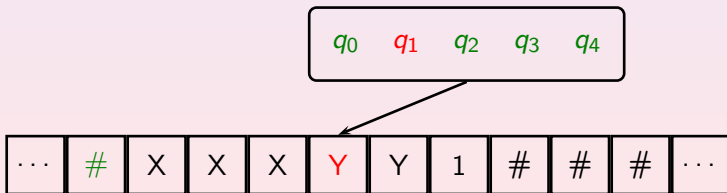
Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

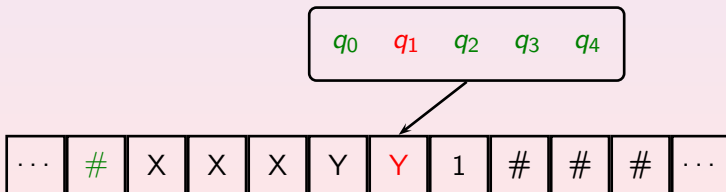


Funcionamiento

$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$

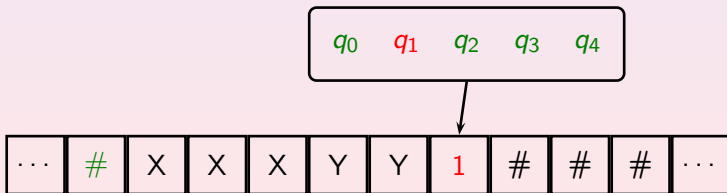


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

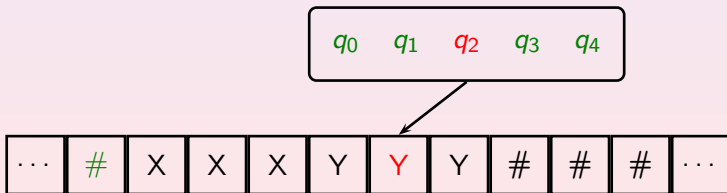


Funcionamiento

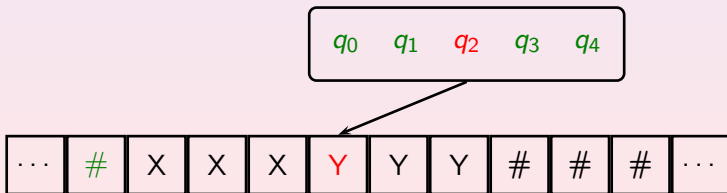
$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$



$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

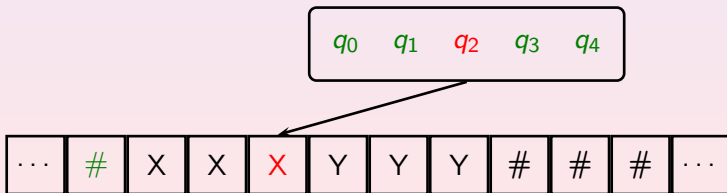


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

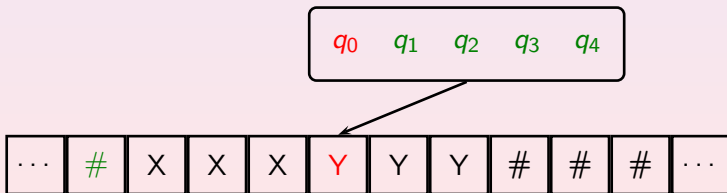


Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$

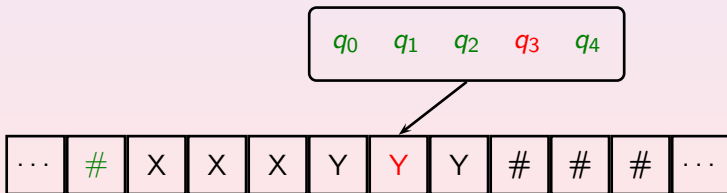


$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



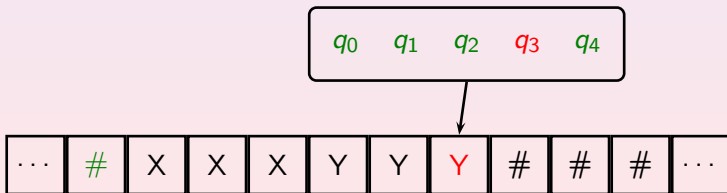
Funcionamiento

$\delta(q_0, 0) = (q_1, X, D)$ $\delta(q_0, Y) = (q_3, Y, D)$ $\delta(q_1, 0) = (q_1, 0, D)$
 $\delta(q_1, 1) = (q_2, Y, I)$ $\delta(q_1, Y) = (q_1, Y, D)$ $\delta(q_2, 0) = (q_2, 0, I)$
 $\delta(q_2, X) = (q_0, X, D)$ $\delta(q_2, Y) = (q_2, Y, I)$
 $\delta(q_3, Y) = (q_3, Y, D)$ $\delta(q_3, \#) = (q_4, \#, D)$



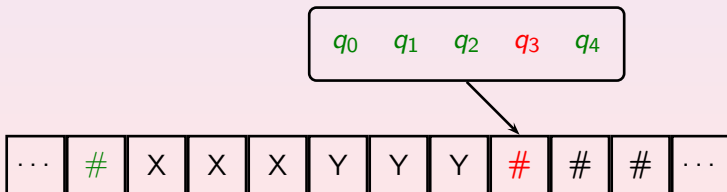
Funcionamiento

$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$



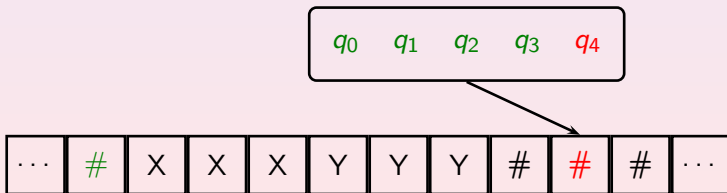
Funcionamiento

$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$



$$\begin{array}{lll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) & \delta(q_1, 0) = (q_1, 0, D) \\ \delta(q_1, 1) = (q_2, Y, I) & \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) & \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) & \end{array}$$

La palabra 000111 es aceptada



Lenguaje Aceptado por una MT

Lenguaje aceptado por una Máquina de Turing

Si M es una máquina de Turing, entonces el lenguaje aceptado es el conjunto de palabras $L(M)$ tales que comenzando con esa palabra en la cinta de entrada con el cabezal de lectura al principio de la palabra, entonces podemos llegar a una configuración en la que estamos en un estado final.

Definición: Recursivamente Enumerable

Un lenguaje $L \subseteq A^*$ se dice **recursivamente enumerable (e.r.)** si y solo si existe una máquina de Turing $M = (Q, A, C, \delta, q_0, \#, F)$ tal que $L(M) = L$.

Parada en las Máquinas de Turing

Una máquina **para** cuando en el estado actual y símbolo de la cinta no hay ninguna transición definida.

Cuando se llega a un estado final $q \in F$ podemos suponer que la Máquina de Turing para, es decir no hay ninguna transición definida.

Existe otro criterio de aceptación: una palabra es aceptada cuando la MT para. La clase de lenguajes aceptada por este criterio es también la clase de los lenguajes recursivamente enumerables.

Definición

Un lenguaje se dice **recursivo** si es aceptado por una MT que siempre para.

Un lenguaje recursivo es siempre recursivamente enumerable. Los lenguajes recursivos son aquellos cuyo problema de aceptación poder ser resuelto mediante un algoritmo.

En el caso de lenguajes recursivos, podemos suponer que hay dos tipos de estados finales: de aceptación y de rechazo. La máquina acepta cuando se llega a un estado de aceptación y rechaza cuando llega a un estado de rechazo.

Máquinas de Turing Calculadoras

- Una MT tiene siempre asociada una función definida en el conjunto de las entradas $D \subseteq A^*$ en las que la MT para. Si $u \in D$, entonces $f(u)$ es el contenido de la cinta cuando la MT termina (excluyendo los infinitos blancos a la derecha y a la izquierda de lo que hay escrito).
- Una función calculada por una MT se dice que es **parcialmente calculable**.
- Cuando la función está definida para todo A^* se dice **calculable total**.

Definición

Un problema **PROBLEMA**(x) ó **PROBLEMA** consta de:

- Un conjunto X de **entradas**. Un elemento $x \in X$ se llama una entrada.
- Un conjunto Y de **solución**. Un elemento $y \in Y$ se llama una solución.
- Una aplicación $R : X \times Y \rightarrow \{0,1\}$ que representa la relación que debe de existir entre las entradas y las soluciones. Si $R(x,y) = 1$ también escribiremos $R(x,y)$ simplemente y si $R(x,y) = 0$ escribiremos también $\neg R(x,y)$.

Problemas

Definición

Un problema **PROBLEMA**(x) ó **PROBLEMA** consta de:

- Un conjunto X de **entradas**. Un elemento $x \in X$ se llama una entrada.
- Un conjunto Y de **solución**. Un elemento $y \in Y$ se llama una solución.
- Una aplicación $R : X \times Y \rightarrow \{0,1\}$ que representa la relación que debe de existir entre las entradas y las soluciones. Si $R(x,y) = 1$ también escribiremos $R(x,y)$ simplemente y si $R(x,y) = 0$ escribiremos también $\neg R(x,y)$.

Ejemplo

Búsqueda de caminos en grafos dirigidos:

- Entradas X : conjunto formado por las tripletas (G, ns, nl) , donde G es un grafo dirigido, ns es un nodo de salida, nl es un nodo de llegada.
- Conjunto Y : lista de nodos (n_1, \dots, n_k) ó NO
- La relación que debe de haber entre las entradas y las soluciones es que $n_1 = ns, n_k = nl$ y todas las parejas (n_i, n_{i+1}) sean arcos de G . La solución es NO cuando no existe camino entre los dos nodos.

- Los problemas se resuelven mediante algoritmos.
- ¿Qué es un algoritmo? En esta asignatura vamos a considerar dos conceptos que son equivalentes:
 - Un programa en Python (Ph) bien escrito sintácticamente y con, al menos, una función definida, la primera de las cuales es la función principal (más adecuados para una resolución efectiva de problemas).
 - Una Máquina de Turing (MT) que definiremos en el siguiente tema (más adecuadas para el razonamientos teórico-matemáticos).
- Un algoritmo ALG **resuelve** un problema **PROBLEMA**(x) cuando el argumento de dicho algoritmo es un elemento $x \in X$ y **ALG**(x) es un $y \in Y$ tal que $R(x, y) = 1$.

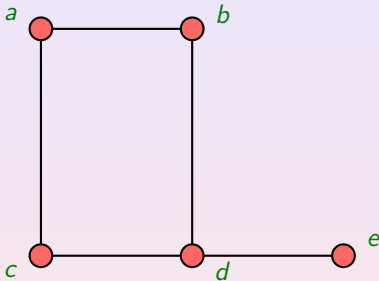
Problemas y su codificación

- Si queremos resolver un problema en un ordenador habrá que codificar las entradas y las soluciones en unos datos que pueda entender el ordenador.
- Nosotros vamos a suponer que las entradas y soluciones están siempre codificadas como palabras o cadenas de caracteres.
- Esto no supone ninguna restricción ya que cualquier tipo de entrada a un programa se puede representar mediante una cadena de caracteres, así como cualquier tipo de salida.
- También supondremos que hay sólo una palabra de entrada y una palabra de salida. Esto tampoco supone restricción ya que un conjunto de palabras se pueden codificar como una sola palabra en la que aparecen las palabras originales encadenadas con un separador para indicar el final de una y el comienzo de la siguiente.
- A partir de ahora, en términos generales hablaremos sólo sobre problemas en las que las entradas y las salidas son palabras sobre un alfabeto (**Problema Computacional**), aunque al hablar de un problema concreto hablaremos de grafos, números u otros elementos.

Problemas y su Codificación

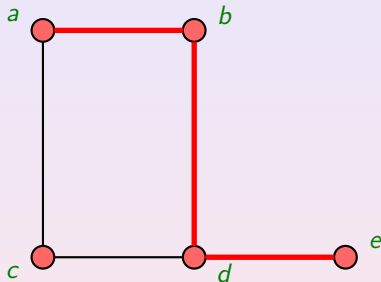
- Cualquier problema se transforma en un problema computacional con un sistema de codificación de las entradas y salidas.
- Puede haber palabras que no sean una codificación correcta de una entrada del problema original. En ese caso, supondremos que a esa entrada le corresponde una salida especial: 'NO'.
- En general, nuestro estudio teórico se hará sobre problemas computacionales, pero su resultado se aplicará también a problemas en general, ya que estos resultados no dependerán de la codificación elegida (siempre que ésta sea razonable).

Codificando Grafos



Grafo: a,b b,d c,d a,c d,e

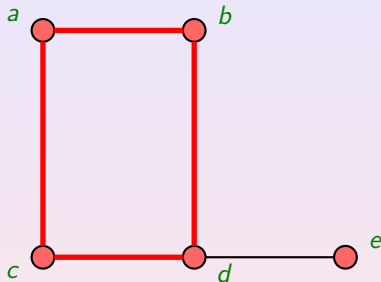
Codificando Grafos



Grafo: a,b b,d c,d a,c d,e

Camino: a,b,d,e

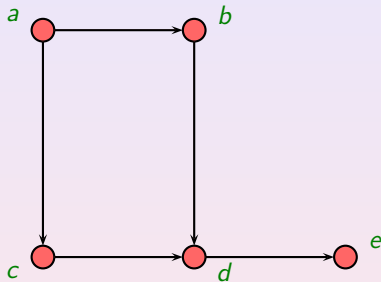
Codificando Grafos



Grafo: *a*,*b* *b*,*d* *c*,*d* *a*,*c* *d*,*e*

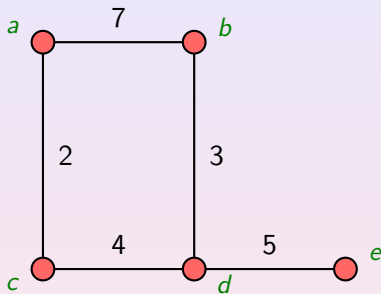
Ciclo: *a*,*b*,*d*,*c*

Codificando Grafos Dirigidos



Grafo dirigido: a,b b,d c,d a,c d,e

Codificando Grafos Con Pesos



Grafo con pesos: a,b,7 b,d,3 c,d,4 a,c,2 d,e,5

Tipos de Problemas

- **Problemas de Búsqueda:** Para una entrada x el problema consiste en encontrar una solución y tal que $R(x,y)$ cuando esta exista y decir 'NO' en caso contrario. Ejemplo: dado un grafo no dirigido encontrar un circuito hamiltoniano.
- **Problemas de Decisión:** Son aquellos en los que las soluciones son $Y = \{SI, NO\}$ y cada entrada x tiene una única solución. Ejemplo: dado un grafo determinar si tiene un circuito hamiltoniano.
- **Problemas de Optimización:** La solución optimiza (minimiza o maximiza) una función definida sobre un conjunto de soluciones factibles asociadas a la entrada. Ejemplo: el problema del viajante de comercio.
- **Problemas de función:** Cada entrada x tiene siempre una y sólo una solución. Por ejemplo, dado un número n calcular su cuadrado n^2 . A la solución que corresponde a la entrada x se le denota también por $PROBLEMA(x)$ (también en problemas de decisión).

Versión Decisión de los problemas

Los problemas de decisión son especialmente importantes ya que son simples y es fácil razonar sobre ellos y además cualquier otro problema tiene asociado un problema de decisión:

- **Problemas de umbral para problemas de optimización:** Los mismos datos de un problema de optimización más un umbral K y ahora se pregunta si existe una solución de valor mayor o menor que el valor K según sea un problema de máximo o mínimo. Ejemplo: dado un caso del problema del viajante de comercio y un valor K determinar si existe un circuito de coste menor o igual que K .
- **Problemas de existencia para problemas de búsqueda:** Dado un x , determinar si existe un y tal que sea una solución de x .
- **Problemas de comprobación para problemas de función y búsqueda:** Dado x y una posible solución y determinar si y es una solución de x .

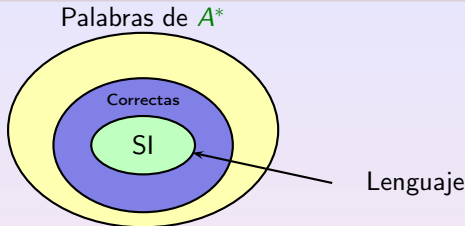
- **Camino mínimo:** Dado un grafo y dos nodos, encontrar un camino de longitud mínima entre estos nodos si este existe (problema de optimización).
- **Busqueda de caminos:** Dado un grafo y dos nodos, encontrar un camino entre ambos nodos, en caso de que exista, decir 'NO' en caso contrario.
- **Existencia de Caminos:** Dado un grafo y dos nodos, determinar si existe un camino entre ellos (problema de existencia).
- **Umbral del camino mínimo:** Dado un grafo, dos nodos, y un umbral K determinar si existe un camino entre estos nodos de longitud menor o igual a K (problema de umbral).
- **Problema de comprobación:** Dado un grafo, dos nodos, y un camino, determinar si es un camino que une esos dos nodos (problema de comprobación).

Un problema computacional de decisión en el que las entradas se codifican como palabras del alfabeto A con el lenguaje de las entradas que tienen respuesta 'SI':

$$L(\text{PROBLEMA}(x)) = \{x \in A^* : \text{PROBLEMA}(x) = 'SI'\}.$$

- Es decir los casos con respuesta **afirmativa**.
- Un lenguaje L sobre un alfabeto A , siempre define también un problema de decisión: dada $x \in A^*$ determinar si $x \in L$.

Problemas y Lenguajes



- En muchos casos, la teoría se desarrolla hablando de lenguajes.
- Cuando comenzamos con un problema genérico de decisión y lo transformamos en un problema computacional mediante una codificación de las entradas, las palabras que no son una codificación correcta de un ejemplo del problema se meten en el mismo 'saco' que los casos en los que la respuesta es 'NO'.
- La identificación de las codificaciones correctas (palabras que corresponden realmente a un ejemplo del problema) se considera que no es importante desde el punto de vista computacional y no se tendrá en cuenta.

Un problema de decisión + Una codificación \equiv Problema Computacional 'SI' 'NO'
 \equiv Problema Computacional 'SI' 'NO' \equiv Lenguaje

- Las definiciones sobre lenguajes se pueden transformar en definiciones sobre los problemas asociados. Los problemas cuyos lenguajes son recursivos se llaman **decidibles** o **calculable** y los que no lo son **indecidibles** o **no calculables**. Esta terminología se aplica también a los lenguajes.
- Los problemas cuyos lenguajes son r.e. se llaman **semidecidibles** o **parcialmente calculables**.
- La identificación de las codificaciones correctas (palabras que corresponden realmente a un ejemplo del problema) se considera que no es importante desde el punto de vista computacional (en todos los ejemplos el reconocimiento de una entrada correcta se puede realizar en tiempo lineal).

Problema Contrario

Dado un problema de decisión **PROBLEMA**, el problema contrario de **PROBLEMA**, es el problema **C – PROBLEMA** que intercambia las salidas 'SI' y 'NO' o de forma más precisa

$\text{PROBLEMA}(x) = 'SI' \Leftrightarrow \text{C – PROBLEMA}(x) = 'NO'$.

Problemas contrarios y lenguajes

El lenguaje asociado al problema contrario de **PROBLEMA** es el lenguaje complementario del lenguaje del lenguaje asociado a **PROBLEMA**:

$$L(\text{C – PROBLEMA}) = \overline{L(\text{PROBLEMA})}$$

Problema Contrario

Dado un problema de decisión **PROBLEMA**, el problema contrario de **PROBLEMA**, es el problema **C – PROBLEMA** que intercambia las salidas 'SI' y 'NO' o de forma más precisa

PROBLEMA(x) = 'SI' \Leftrightarrow **C – PROBLEMA**(x) = 'NO'.

Problemas contrarios y lenguajes

El lenguaje asociado al problema contrario de **PROBLEMA** es el lenguaje complementario del lenguaje del lenguaje asociado a **PROBLEMA**:

$$L(\text{C – PROBLEMA}) = \overline{L(\text{PROBLEMA})}$$

Nota: Cuando hablamos de problemas genéricos, esto no es del todo exacto: ya que las codificaciones incorrectas estarán englobadas con el 'SI' en el lenguaje complementario o problema contrario, pero nosotros consideraremos que esta propiedad también se aplica a los problemas genéricos sin tener esto en cuenta: en realidad no importa como se traten las codificaciones incorrectas, ya que estas son siempre fáciles de detectar.

Ejemplo: Isomorfismo de subgrafos

- **X:** Datos

Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$

- **Y:** 'SI', 'NO' (Problema de Decisión)

Relación: Respuesta 'SI' corresponde a ¿Contiene G_1 un subgrafo isomorfo a G_2 ?

Es decir, existe un subconjunto $V' \subseteq V_1$ y un subconjunto de aristas $E' \subseteq E_1$ tal que $E' \subseteq V' \times V'$ y existe una aplicación biyectiva $f : V_2 \rightarrow V'$ de tal manera que se verifica

$$(u, v) \in E_2 \Leftrightarrow (f(u), f(v)) \in E'$$

Ejemplo: restar números en unario

- Vamos a diseñar una MT que resta números en unario.
- Para dos números naturales $n, m \in \mathbb{N}$ calcula $f(n, m)$ que es igual a $n - m$ si $n \geq m$ y 0 si $n < m$.
- La entrada debe de ser $0^n 10^m$ y la salida debe de ser una confiración en la que en la cinta esté $0^{f(n, m)}$ rodeado de blancos. No nos preocupa cual es la salida si la entrada no es correcta (no corresponde a dos series de ceros separadas por un 1).
- La MT será
$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1\}, \{0, 1, \#\}, \delta, q_0, \#, q_6)$$

Restar números en unario: función de transición

δ viene dada por la siguiente tabla:

Estado	0	1	#
q_0	$(q_1, \#, D)$	$(q_5, \#, D)$	—
q_1	$(q_1, 0, D)$	$(q_2, 1, D)$	—
q_2	$(q_3, 1, I)$	$(q_2, 1, D)$	$(q_4, \#, I)$
q_3	$(q_3, 0, I)$	$(q_3, 1, I)$	$(q_0, \#, D)$
q_4	$(q_4, 0, I)$	$(q_4, \#, I)$	$(q_6, 0, D)$
q_5	$(q_5, \#, D)$	$(q_5, \#, D)$	$(q_6, \#, D)$
q_6	—	—	—

Programación de Máquinas de Turing: recordando símbolos

- Una MT puede diseñarse para que recuerde un símbolo del alfabeto de trabajo (o del alfabeto de entrada).
- Por ejemplo, si queremos que se recuerde un símbolo de B cuando está en el estado q , entonces basta con cambiar el estado q por las parejas de estados $[q, b]$ donde $b \in B$.
- A menudo queremos que se recuerde un símbolo en cualquier estado y entonces el conjunto de estados sería el conjunto de las parejas $Q' \times B$ formadas por un elemento $q \in Q'$ y un símbolo $b \in B$.
- Escribir los estados de esta forma ayuda a comprender el significado de los mismos.

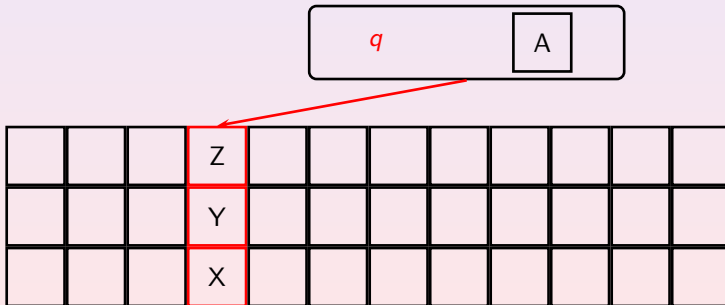
Vamos a hacer una máquina de Turing que reconozca el lenguaje $01^* + 10^*$: la Máquina tiene que recordar el primer símbolo leído y después comprobar que nunca más aparezca.

La Máquina es $M = (Q, \{0, 1\}, \{0, 1, \#\}, \delta, [q_0, \#], \{[q_1, \#]\})$ donde

- $Q = \{q_0, q_1\} \times \{0, 1, \#\}$
- Las posibles transiciones de δ son:
 - 1 $\delta([q_0, B], a) = ([q_1, a], a, D)$ para $a = 0$ o $a = 1$.
 - 2 $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, D)$, donde \bar{a} es el complementario de a (esto es, $\bar{a} = 1$ si $a = 0$ y $\bar{a} = 0$ si $a = 1$).
 - 3 $\delta([q_1, a], \#) = ([q_1, \#], \#, D)$

Programación de Máquinas de Turing: pistas múltiples

- A menudo es útil pensar que la MT tiene una cinta con varias pistas: en lugar de tener una sola casilla, disponemos de varias casillas en cada posición donde poder escribir un símbolo.
- Tener dos cintas equivale a suponer que el alfabeto de trabajo está formado por los elementos de $B \times B$ y tener k cintas a suponer que el alfabeto de trabajo es B^k .



Una forma de utilizar las pistas múltiples es imaginar que una pista se usa para los datos y otra para poner una marca. Vamos a diseñar una MT que acepte el lenguaje $L = \{wcw \mid w \in \{0,1\}^+\}$.

La MT tiene los siguientes elementos

$M = (Q, A, B, \delta, [q_1, \#], [\#, \#], \{[q_9, \#]\})$ donde

- $Q = \{q_1, q_2, \dots, q_9\} \times [0, 1]$ (podemos recordar 0, 1.
- $B = \{0, 1, c, \#\} \times \{\#, *\}$
- $A = \{0, 1, c\}$. 0 se identifica con $[0, \#]$ y 1 se identifica con $[1, \#]$
- La función de transición δ se especifica en la siguiente pantalla.

Ejemplo: Función de Transición

a y b pueden ser 0,1.

$$\delta([q_1, \#], [a, \#]) = ([q_2, a], [a, *], D)$$

$$\delta([q_2, a], [c, \#]) = ([q_3, a], [c, \#], D)$$

$$\delta([q_3, a], [a, \#]) = ([q_4, \#], [a, *], I)$$

$$\delta([q_4, \#], [c, \#]) = ([q_5, \#], [c, \#], I)$$

$$\delta([q_6, \#], [a, \#]) = ([q_6, \#], [a, \#], I)$$

$$\delta([q_5, \#], [a, *]) = ([q_7, \#], [a, *], D)$$

$$\delta([q_8, \#], [a, *]) = ([q_8, \#], [a, *], D)$$

$$\delta([q_2, a], [b, \#]) = ([q_2, a], [b, \#], D)$$

$$\delta([q_3, a], [b, *]) = ([q_3, a], [b, *], D)$$

$$\delta([q_4, \#], [a, *]) = ([q_4, \#], [a, *], I)$$

$$\delta([q_5, \#], [a, \#]) = ([q_6, \#], [a, \#], I)$$

$$\delta([q_6, \#], [a, *]) = ([q_1, \#], [a, *], D)$$

$$\delta([q_7, \#], [c, \#]) = ([q_8, \#], [c, \#], D)$$

$$\delta([q_8, \#], [\#, \#]) = ([q_9, \#], [\#, \#], D)$$

Subrutinas

- Una **subrutina** en una MT es un conjunto de estados que realiza una acción concreta.
- En este conjunto de estados habrá un estado inicial y otro estado que sirve como estado de retorno.
- No se añade ninguna funcionalidad nueva, sólo es una forma de organizar los estados de una MT agrupando aquellos que realizan una tarea concreta y suponiendo que siempre podemos movernos a ese conjunto de estados.
- La MT no tiene un sistema de llamadas que permita saber a qué posición y en qué estado hay que volver.
- La posición se puede recordar con una pista adicional y un símbolo extra que indique la casilla en la que se tiene que posicionar.
- El estado se puede determinar haciendo varias copias del último estado de la subrutina, una para cada estado al que haya que volver. El número de copias es finito.
- Siempre que hagamos un conjunto de estados para una tarea determinada, por ejemplo, desplazar el contenido de todas las casillas a partir de la posición actual un lugar a la derecha, supondremos que esta tarea siempre la podemos hacer en una MT sin necesidad de especificar los estados.

Ejemplo

- Vamos a diseñar una MT que multiplique números en binario escritos en base 1: la MT comenzará con una cadena de la forma $0^m 10^n$ en la cinta, y terminará con 0^{mn} al final. No nos preocupamos si la MT tienen una entrada mal escrita. Finalizará con algo en la cinta que no tendrá sentido, en general.
- En etapas sucesivas, la cinta va a contener cadenas de la forma $0^i 10^n 10^{kn}$ donde $i + k = m$ para valores de $k = 1, \dots, m$.
- En un paso básico se cambia el primer cero del primer grupo por un blanco y se añaden n ceros al último grupo: se pasa de $0^i 10^n 10^{kn}$ a $0^{i-1} 10^n 10^{(k+1)n}$.
- Finalmente la subcadena $10^n 1$ se sustituye por blancos.

Ejemplo: Subrutina

Vamos a diseñar un conjunto de estados que copia n ceros al final de la cinta cuando está situada justo al principio de la serie de n ceros. Termina en la misma posición de la cinta en el estado q_5 . Tiene los siguientes estados y estructura:

$$\delta(q_1, 0) = (q_2, X, D)$$

$$\delta(q_2, 0) = (q_2, 0, D) \quad \delta(q_2, 1) = (q_2, 1, D)$$

$$\delta(q_2, \#) = (q_3, 0, I)$$

$$\delta(q_3, 0) = (q_3, 0, I) \quad \delta(q_3, 1) = (q_3, 1, I)$$

$$\delta(q_3, X) = (q_1, X, D) \quad \delta(q_1, 1) = (q_4, 1, I)$$

$$\delta(q_4, X) = (q_4, 0, I) \quad \delta(q_4, 1) = (q_5, 1, D)$$

Estructura del Programa

- Pasamos de la configuración en la que hay $0^m 10^n$ en la cinta y estamos colocados al principio de esta palabra a la configuración en la que en la cinta hay $0^m 10^n 1$ y estamos colocados al principio de la palabra.

$$\begin{aligned}\delta(q_0, 0) &= (q_0, 0, D) & \delta(q_0, 1) &= (q_0, 1, D) \\ \delta(q_0, \#) &= (q_{13}, 1, I) & \delta(q_{13}, 0) &= (q_{13}, 0, I) \\ \delta(q_{13}, 1) &= (q_{13}, 1, I) & \delta(q_{13}, \#) &= (q_{14}, \#, D)\end{aligned}$$

- Miramos si hay un 0 al principio, lo sustituimos por un blanco, #, y nos ponemos en situación para copiar n ceros al final.

$$\begin{aligned}\delta(q_{14}, 0) &= (q_6, \#, D) \\ \delta(q_6, 0) &= (q_6, 0, D) & \delta(q_6, 1) &= (q_1, 1, D)\end{aligned}$$

Estructura del Programa (II)

- Después de hacer la copia, estamos en q_5 y desde este estado, volvemos a la posición inicial si hay más 0 en la primera serie o finalizamos si ya no hay más ceros (q_{12} es el estado final de parada):

$$\delta(q_5, 0) = (q_7, 0, I)$$

$$\delta(q_7, 1) = (q_8, 1, I)$$

$$\delta(q_8, 0) = (q_9, 0, I)$$

$$\delta(q_8, \#) = (q_{10}, \#, D)$$

$$\delta(q_9, 0) = (q_9, 0, I)$$

$$\delta(q_9, \#) = (q_{14}, \#, D)$$

$$\delta(q_{10}, 1) = (q_{11}, \#, D)$$

$$\delta(q_{11}, 0) = (q_{11}, \#, D) \quad \delta(q_{11}, 1) = (q_{12}, \#, D)$$

- Rediseñar las MT de los ejercicios anteriores introduciendo las técnicas de programación de almacenamiento de símbolo, multipista o subrutinas.
- Diseñar una subrutina que desplace todos los símbolos desde la posición actual a la derecha, dejando un espacio en dicha posición en el que se pueda escribir un carácter. Debe de terminar con la cabeza de lectura en la misma posición en la que se terminó.
- Escribir una subrutina que comience en una posición con un cero y se mueva a la derecha de todos los ceros hasta que alcance un uno o un blanco. Se suponen que en la cinta solo hay caracteres del conjunto $\{0, 1, \#\}$. Si se comienza con un carácter que es distinto de cero, la MT debe de pararse. Utilizar dicha rutina para escribir una MT que acepte todas las cadenas de ceros y unos, que no tengan dos unos consecutivos.

Variaciones de la MT básica

- Extensiones:

- **MT que se pueden quedar en la misma posición en un paso:** no es necesario moverse a la izquierda o derecha y pueden quedarse en el mismo sitio (S).
- **MT con múltiples cintas:** hay distintas cintas en las que se puede leer o escribir y una cabeza de lectura para cada una de ellas.
- **MT no deterministas:** hay distintas transiciones que puede realizar una MT en una configuración dada.

- Limitaciones:

- **MT con cintas semiilimitadas:** la cinta de la MT es ilimitada sólo por la derecha y en la izquierda tiene un tope.

Ninguna de estas modificaciones cambiará la potencialidad de las MT.

MT que se pueden quedar en la misma posición en un paso

- En estas MT se supondrá que $\delta(q, b)$ puede ser vacío (no definido) o una tripleta (p, c, M) donde $p \in Q, c \in B, M \in \{I, D, S\}$. El símbolo S indica que el cabezal de lectura no se mueve en ninguna dirección y permanece en el mismo sitio.
- Esto no supone ninguna potencialidad adicional, ya que si $\delta(q, b) = (p, c, S)$, esto lo podemos simular con un nuevo estado r_p por cada estado p de la MT con estos movimientos haciendo, $\delta(q, b) = (r_p, d, D)$ y desde todos los estados r_p lo único que se puede hacer es movernos a la izquierda: $\delta(r_p, d) = (p, d, I)$ para todo estado r_p y todo símbolo d de B .

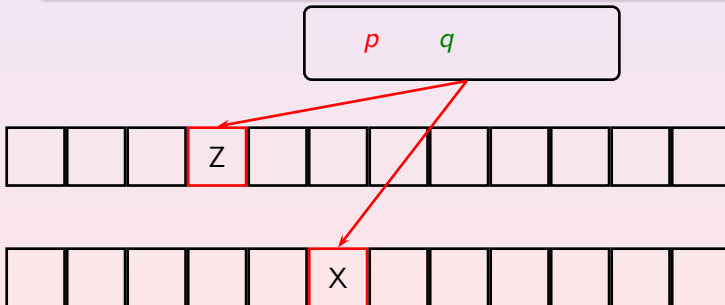
MT con Múltiples Cintas

- En estas MTs suponemos que existen k cintas ilimitadas en las que leer y escribir.
- Las diferencias entre múltiples pistas y múltiples cintas son:
 - Múltiples pistas no supone una modificación de la definición de MT, simplemente es una forma de visualizar la cinta de una MT en la que el alfabeto de trabajo es un producto cartesiano $B = B' \times B'$, es decir cada símbolo está formado por una pareja de símbolos básicos. El tener múltiples cintas si implicará una modificación de la definición. Ahora a cada (q, b_1, \dots, b_k) , δ le podrá asignar un vector $(p, c_1, \dots, c_k, M_1, \dots, M_k)$.
 - Cuando hay múltiples cintas, el cabezal de lectura podrá estar en una posición distinta en cada cinta. Por eso hay que especificar qué movimiento M_i hay que realizar en cada cinta i , además de lo que se ve en cada cinta b_i y lo que se escribe en cada una c_i . Suponemos que el movimiento de cada cinta puede ser $\{I, D, S\}$.

MT con Múltiples Cintas

Configuración

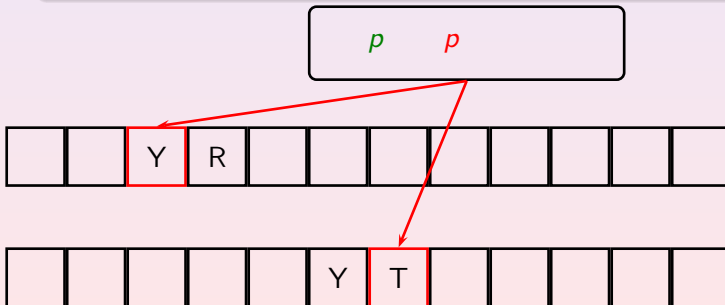
Será un vector $(q, u_1, w_1, u_2, w_2, \dots, u_k, w_k)$, donde q es el estado en el que está la MT, u_i es la parte de la palabra que hay a la izquierda del cabezal de lectura de la cinta i y w_i la parte de la palabra que hay en la cinta i a partir del cabezal de lectura de esa cinta hacia la derecha (incluyendo el símbolo que se lee en ese momento).



MT con Múltiples Cintas

Configuración

Será un vector $(q, u_1, w_1, u_2, w_2, \dots, u_k, w_k)$, donde q es el estado en el que está la MT, u_i es la parte de la palabra que hay a la izquierda del cabezal de lectura de la cinta i y w_i la parte de la palabra que hay en la cinta i a partir del cabezal de lectura de esa cinta hacia la derecha (incluyendo el símbolo que se lee en ese momento).



Lenguaje Aceptado y Función Calculada

Lenguaje Aceptado

El aceptado por una MT con múltiples cintas es el conjunto de palabras u tales que empezando en una configuración en la que en la primera cinta está la palabra u y el resto de las cintas son vacías y el cabezal de lectura de la primera cinta está en el primer símbolo de u y en cualquier casilla de las otras cintas termina en un estado de aceptación.

Función Calculada

La función parcial f es calculada por una MT si para cada palabra u para la que está definida $f(u)$, si la MT comienza con u en la primera cinta y el resto de las cintas vacías y el cabezal de lectura de la primera cinta está en el primer símbolo de u (configuración inicial) se llega a una configuración en la que estamos en un estado final y el contenido de la última cinta es $f(u)$. Si $u \notin D$ entonces la máquina no para.

Equivalencia

Está claro que si un lenguaje es aceptado por una MT con una cinta puede hacerlo por una MT con varias cintas. El recíproco también es cierto.

Teorema

Todo lenguaje aceptado por una MT con varias cintas es recursivamente enumerable.

Supongamos que M es la MT con k cintas, vamos a simular su funcionamiento con una MT N con una sola cinta.

Equivalencia

Está claro que si un lenguaje es aceptado por una MT con una cinta puede hacerlo por una MT con varias cintas. El recíproco también es cierto.

Teorema

Todo lenguaje aceptado por una MT con varias cintas es recursivamente enumerable.

Supongamos que M es la MT con k cintas, vamos a simular su funcionamiento con una MT N con una sola cinta.

La MT N tendrá una cinta con $2k$ pistas. En cada par de pistas se simula una cinta de M . En una de las pistas se coloca un símbolo especial $*$ en el lugar en el que se encuentre el cabezal de la MT de k pistas, en la otra pista se coloca el contenido de la cinta de M .

Equivalencia

Está claro que si un lenguaje es aceptado por una MT con una cinta puede hacerlo por una MT con varias cintas. El recíproco también es cierto.

Teorema

Todo lenguaje aceptado por una MT con varias cintas es recursivamente enumerable.

Supongamos que M es la MT con k cintas, vamos a simular su funcionamiento con una MT N con una sola cinta.

La MT N tendrá una cinta con $2k$ pistas. En cada par de pistas se simula una cinta de M . En una de las pistas se coloca un símbolo especial $*$ en el lugar en el que se encuentre el cabezal de la MT de k pistas, en la otra pista se coloca el contenido de la cinta de M .

La MT N almacena en la unidad de control los k símbolos que contiene M . Para ello comienza a revisar la cinta de izquierda a derecha y cada vez que encuentra un símbolo lo almacena en su correspondiente lugar de la unidad de control. Lleva un contador donde empieza en 0 y se aumenta en 1 cada vez que encuentra un símbolo hasta llegar a k (el valor de k es fijo y se puede almacenar).

Equivalencia

Está claro que si un lenguaje es aceptado por una MT con una cinta puede hacerlo por una MT con varias cintas. El recíproco también es cierto.

Teorema

Todo lenguaje aceptado por una MT con varias cintas es recursivamente enumerable.

Supongamos que M es la MT con k cintas, vamos a simular su funcionamiento con una MT N con una sola cinta.

La MT N tendrá una cinta con $2k$ pistas. En cada par de pistas se simula una cinta de M . En una de las pistas se coloca un símbolo especial $*$ en el lugar en el que se encuentre el cabezal de la MT de k pistas, en la otra pista se coloca el contenido de la cinta de M .

La MT N almacena en la unidad de control los k símbolos que contiene M . Para ello comienza a revisar la cinta de izquierda a derecha y cada vez que encuentra un símbolo lo almacena en su correspondiente lugar de la unidad de control. Lleva un contador donde empieza en 0 y se aumenta en 1 cada vez que encuentra un símbolo hasta llegar a k (el valor de k es fijo y se puede almacenar).

Una vez hecho esto, tiene todos los elementos para realizar la transición de la MT de k cintas, para ello se va colocando en cada una de las posiciones señaladas de cada una de las cintas y realiza la transición correspondiente, escribiendo el símbolo que corresponda y moviendo la señal de posición del cabezal de lectura.

Equivalencia

Está claro que si un lenguaje es aceptado por una MT con una cinta puede hacerlo por una MT con varias cintas. El recíproco también es cierto.

Teorema

Todo lenguaje aceptado por una MT con varias cintas es recursivamente enumerable.

Supongamos que M es la MT con k cintas, vamos a simular su funcionamiento con una MT N con una sola cinta.

La MT N tendrá una cinta con $2k$ pistas. En cada par de pistas se simula una cinta de M . En una de las pistas se coloca un símbolo especial $*$ en el lugar en el que se encuentre el cabezal de la MT de k pistas, en la otra pista se coloca el contenido de la cinta de M .

La MT N almacena en la unidad de control los k símbolos que contiene M . Para ello comienza a revisar la cinta de izquierda a derecha y cada vez que encuentra un símbolo lo almacena en su correspondiente lugar de la unidad de control. Lleva un contador donde empieza en 0 y se aumenta en 1 cada vez que encuentra un símbolo hasta llegar a k (el valor de k es fijo y se puede almacenar).

Una vez hecho esto, tiene todos los elementos para realizar la transición de la MT de k cintas, para ello se va colocando en cada una de las posiciones señaladas de cada una de las cintas y realiza la transición correspondiente, escribiendo el símbolo que corresponda y moviendo la señal de posición del cabezal de lectura.

Los estados de aceptación de N son aquellos estados que corresponden a los estados de aceptación de M .

Simulación: complejidad en tiempo

Teorema

El tiempo empleado por la MT N del teorema anterior en simular n pasos de la MT de k cintas M es $O(n^2)$.

Para la demostración, basta tener en cuenta:

- Después de n movimientos de la MT M la diferencia entre las posiciones de los distintos cabezales de lectura es, a lo más, $2n$. Al principio podemos suponer que todos están en la misma posición y, en cada paso, se alejan a lo más en dos posiciones (si dos cabezales se mueven en distintas direcciones).
- Para ver los contenidos de las distintas casillas en las cintas en la máquina N nos hacen falta $2n$ pasos como máximo.
- Ahora nos movemos de derecha a izquierda y cada vez que encontramos un marcador de posicionamiento, realizamos la transición correspondiente. Esta transición, a lo más, necesita 2 movimientos (para llevarlo a cabo y volver a la posición en la que estábamos). Esto implica $2n + 2k$ movimientos.
- Si le sumamos a los $2n$ iniciales, hacen $4n + 2k$ por cada movimiento, así que los n movimientos implicarán como máximo $n(4n + 2k)$ que es de orden $O(n^2)$, teniendo en cuenta que k es constante.

Máquinas de Turing No Deterministas

Definición

Una **Máquinas de Turing No Determinista (MTND)** tiene la misma definición que una MT con la única diferencia que ahora $\delta(q, a)$ puede ser un conjunto finito de tripletas

$$\{(q_1, b_1, M_1), \dots, (q_k, b_k, M_k)\}$$

Cálculo

El cálculo asociado a una MTND se define de forma similar a una MT. Ahora en una configuración en la que está en el estado q y ve a en la cinta puede evolucionar con cualquiera de las tripletas (q_i, b_i, M_i) : puede ir a q_i escribir b_i y hacer el movimiento M_i para $i = 1, \dots, k$.

Lenguaje Aceptado

Es el conjunto de todas las palabras aceptadas.

Acepta una palabra cuando para la configuración inicial asociada a la palabra, existe una sucesión de movimientos posibles que permiten llegar a un estado de aceptación y parar. No importa que haya otras computaciones posibles que no lleguen a un estado de aceptación.

Equivalencia MTND y MT

Toda MT es una MTND por lo que todo lenguaje recursivamente enumerable es también aceptado por una MTND. También se da el inverso.

Teorema

Si un lenguaje L es aceptado por una MTND, entonces es recursivamente enumerable.

Suponemos m el número máximo de opciones en la MTND.

- Utilizamos una MT con dos cintas.
- En la primera tenemos una sucesión de configuraciones (q, u, v) separadas por un símbolo \square , también existe una marca $*$ en la *configuración activa*.
- Inicialmente hay sólo una configuración: la configuración inicial.
- En cada momento se coge la configuración activa, se copia en la cinta auxiliar, nos vamos al final de la primera cinta y se realizan todas las transiciones posibles sobre la configuración de la cinta auxiliar colocándolas al final de la primera cinta.
- Se busca la configuración marcada y se pasa a procesar la siguiente.
- Si en algún momento sale una configuración con un estado de aceptación terminamos.
- El procedimiento hace una búsqueda en anchura exhaustiva de todos los posibles cálculos de la MTND. Si en uno se acepta, lo encontrará.

Equivalencia MTND y MT. Relación entre el tiempo de las dos máquinas

- Si para una entrada la MTND tarda un tiempo n en la determinista tardará un tiempo d^n .

Cintas semiilimitadas

Podemos suponer que las cintas son ilimitadas sólo por la derecha y que la palabra de entrada se escribe a partir de la primera casilla. Para ello, vamos a demostrar que todo lenguaje se puede aceptar sin escribir a la izquierda de la primera casilla que ocupa la palabra de entrada, lo que equivale a suponer que no existe cinta a la izquierda de la palabra de entrada.

Teorema

Todo lenguaje aceptado por una MT M_2 es también aceptado por una MT M_1 con las siguientes restricciones:

- 1 M_1 nunca escribe el espacio en blanco $\#$
- 2 La cabeza de M_1 nunca se mueve hacia la izquierda de su posición actual.

La primera condición es muy fácil de conseguir añadiendo un nuevo símbolo a M_1 que es otro espacio en blanco $\#'$.

Si M_2 escribe un espacio en blanco $\delta_2(q, a) = (p, \#, M)$, entonces M_1 escribe el nuevo blanco: $\delta_1(q, a) = (p, \#', M)$. Después, cada transición $\delta_1(q, \#')$ se hace idéntica a $\delta_2(q, \#)$.

Cintas semiilimitadas

Para la segunda condición, una cinta ilimitada por ambos extremos se simula con una cinta ilimitada sólo por la derecha con dos pistas y la siguiente estructura:

X_0	X_1	X_2								
\triangleright	X_{-1}	X_{-2}								

Si $M_2 = (Q_2, A, B_2, \delta_2, q_2, \#, F_2)$ entonces $M_1 = (Q_1, A \times \{\#\}, B_1, \delta_1, q_0, [\#, \#], F_1)$ donde

- Los estados de M_1 son $\{q_0, q_1\} \cup (Q_2 \times \{S, I\})$. Los estados q_0 y q_1 sirven para preparar la cinta de entrada (por ejemplo, poner el tope \triangleright en la pista inferior). En los otros estados tenemos que especificar el valor S (pista superior) o I (pista inferior) además del estado.
- Los símbolos de trabajo de M_1 son $B_2 \times B_2$, es decir todas las parejas de símbolos de trabajo de M_2 . Cada símbolo $a \in A_2$ de M_2 se identifica con el símbolo $[a, \#]$ de M_1 . Además en B_1 están todas las parejas $[b, \triangleright]$ donde $b \in B_2$. Este símbolo \triangleright se usa como tope para saber que estamos en el extremo izquierdo de la cinta.

Cintas semiilimitadas

- $\delta_1(q_0, [a, \#]) = (q_1, [a, \triangleright], D)$ para cualquier $a \in B_2$
- $\delta_1(q_1, [a, \#]) = ([q_2, S], [a, \#], I)$ (nos movemos a la izquierda y decimos que estamos arriba)
- Si $\delta_2(q, a) = (p, b, M)$, entonces para todo $c \in B_2$
 - 1 $\delta_1([q, S], [a, c]) = ([p, S], [b, c], M)$
 - 2 $\delta_1([q, I], [c, a]) = ([p, I], [c, b], \overline{M})$, donde \overline{M} es el movimiento de sentido opuesto a M .
- Si $\delta_2(q, a) = (p, b, D)$ entonces
$$\delta_1([q, I], [a, \triangleright]) = \delta_1([q, S], [a, \triangleright]) = ([p, S], [b, \triangleright], D)$$

Si estamos al principio de la cinta, siempre se supone que el símbolo activo es el superior. Allí se escribe y si nos movemos a la derecha el superior es el activo.
- Si $\delta_2(q, a) = (p, b, I)$ entonces
$$\delta_1([q, I], [a, \triangleright]) = \delta_1([q, S], [a, \triangleright]) = ([p, I], [b, \triangleright], D)$$

Si estamos al principio de la cinta, siempre se supone que el símbolo activo es el superior. Allí se escribe y si nos movemos a la izquierda el inferior es el activo.

- Cuando usamos cintas semiilimitadas, supondremos desde el principio que tenemos el símbolo \triangleright a la izquierda de cada una de las cintas que se usen y que si se llega a ese símbolo inmediatamente nos vamos hacia la derecha en el próximo movimiento.
- Esto no supone ninguna limitación, ya que toda MT se puede simular con una con estas restricciones: siempre podemos incluir ese símbolo al principio de la palabra de entrada, desplazando todos los símbolos una casilla a la derecha y después como hemos visto siempre se puede simular la MT con una de cintas semiilimitadas en las que nunca se pasa a la izquierda de la palabra de entrada.

Palabras y Números

Supongamos un alfabeto $A = \{a_1, \dots, a_n\}$. Podemos establecer una correspondencia biyectiva entre las palabras sobre este alfabeto y los números naturales. Supongamos $w = a_{i_k} \dots a_{i_1}$, entonces el número de w que denotaremos como $N(w)$ es $\sum_{j=1}^k i_j \cdot n^{j-1}$, siendo $N(\varepsilon) = 0$.

Ejemplo

Si $A = \{0, 1, 2\}$, $N(\varepsilon) = 0$, $N(0) = 1$, $\langle 1 \rangle = 2$, $N(2) = 3$, $N(202) = 3 + 1 \times 3 + 3 \times 3^2$

Dado un alfabeto $A = \{a_1, \dots, a_n\}$, Si m es un número natural, siempre se puede encontrar una cadena, que denotaremos como $C(m)$ o como w_m cuya codificación sea m . Esto se puede conseguir de la siguiente forma,

- Si $m = 0$, $C(m) = \varepsilon$
- Si $m > 0$, sea

$$i = \begin{cases} R(m, n) & \text{si } n \text{ no divide a } m \\ n & \text{si } n \text{ divide a } m \end{cases}$$

$$p = \begin{cases} \lfloor m/n \rfloor & \text{si } n \text{ no divide a } m \\ \lfloor m/n \rfloor - 1 & \text{si } n \text{ divide a } m \end{cases}$$

donde $R(m, n)$ es el resto de la división entera de m entre n y $\lfloor m/n \rfloor$ es la división entera de m entre n .

Entonces $C(m) = C(p)a_i$.

Distintos alfabetos

Si tenemos dos alfabetos A y B podemos establecer una aplicación biyectiva entre las palabras de ambos alfabetos $f : A^* \rightarrow B^*$:

- Sea $w \in A^*$. entonces la codificamos como número $N_A(w)$ y la decodificamos como palabra de B^* : $f(w) = C_B(N_A(w))$

Si tenemos dos alfabetos A y B podemos establecer una aplicación biyectiva entre las palabras de ambos alfabetos $f : A^* \rightarrow B^*$:

- Sea $w \in A^*$. entonces la codificamos como número $N_A(w)$ y la decodificamos como palabra de B^* : $f(w) = C_B(N_A(w))$

Es suficiente trabajar sobre un sólo alfabeto. Usualmente, lo haremos sobre el alfabeto $\{0,1\}$ (para desarrollar la teoría) aunque lo haremos sobre alfabetos más amplios (ASCII) para un caso práctico.

Orden Total en las Palabras de un Alfabeto

- Si $A = \{a_1, \dots, a_n\}$ nosotros siempre vamos a considerar el siguiente orden total en sus palabras:
 - $u_1 \leq u_2$ si y solo si se da una de las siguientes condiciones:
 - 1 $u_1 = u_2$
 - 2 $|u_1| < |u_2|$
 - 3 $|u_1| = |u_2|$ y u_1 precede a u_2 en orden alfabético, teniendo en cuenta que $a_1 < a_2 < \dots < a_n$. Es decir si $u_1 = a_{r_1} \dots a_{r_i} a_{s_1} \dots a_{s_m}$ y $u_2 = a_{r_1} \dots a_{r_i} a_{l_1} \dots a_{l_m}$ y $s_1 < l_1$ (el primer símbolo en el que son distintas las palabras es menor en u_1 que en u_2).
- El orden alfabético como tal no es operativo ya que no podemos establecer un ciclo infinito que recorra todas las palabras: hay infinitas palabras que empiezan por a_1 antes de las palabras que empiezan por a_2 y nunca se llegaría a ellas.

Codificando Máquinas de Turing

Se le puede asignar a cada MT sobre un alfabeto $\{0,1\}$ una cadena y un número natural. Para ello, suponemos

- Los estados son $\{q_1, \dots, q_k\}$. El estado inicial es q_1 y hay un único estado final q_2 (esto siempre se puede conseguir).
- Los símbolos de B son $\{a_1, a_2, \dots, a_m\}$ donde a_1 es 0, a_2 es 1 y a_3 es el símbolo blanco.
- Al movimiento izquierda le asignamos un 1 y al de la derecha un 2. Este número será $u(M)$.

La codificación de la MT se realiza de la siguiente forma:

- Cada transición $\delta(q_i, a_j) = (q_k, a_l, M)$ se codifica como $0^i 10^j 10^k 10^l 10^{u(M)}$.
- Todas las transiciones se van añadiendo a la codificación separadas por 11.

Una vez calculada la cadena $w = \langle M \rangle$, podemos calcular su número $N(\langle M \rangle)$ con el alfabeto $\{0,1\}$, según el procedimiento que hemos visto para asignar números a palabras. Este número también se denotará como $N(M)$.

Cada número natural corresponderá a una MT, o corresponderá a una cadena sin sentido alguno. Sea $MT(n)$ la MT correspondiente al número n o 'Nula' si no hay MT asociada al número n . También denotaremos como M_n a la MT con número n .

Codificando Máquinas de Turing y Entradas

- Para codificar una MT M y una palabra w sobre el alfabeto $\{0,1\}$, podemos codificar M como hemos visto y después añadir 111 seguido de w , dando lugar a la cadena $\langle M, w \rangle$

El lenguaje de diagonalización

Vamos a definir un lenguaje L_d sobre $\{0,1\}$ que no es r.e. Este lenguaje se conoce como **lenguaje de diagonalización**.

Lenguaje de Diagonalización

Sea $w \in \{0,1\}^*$, $w \in L_d$ si y solo si la MT M cuya codificación es $w = \langle M \rangle$ no acepta w .

Se interpreta que si w no es un MT correcta, entonces representa una MT con un sólo estado y ninguna transición (siempre rechaza).

Si w_0, w_1, w_2, \dots son todas las palabras de $\{0,1\}^*$ ordenadas, entonces

		Palabras				
		w_0	w_1	w_2	w_3	...
Máquinas	w_0	0	0	0	0	...
	w_1	1	0	1	0	...
	w_2	1	1	0	0	...
	w_3	0	1	0	1	...

El lenguaje de diagonalización

Vamos a definir un lenguaje L_d sobre $\{0,1\}$ que no es r.e. Este lenguaje se conoce como **lenguaje de diagonalización**.

Lenguaje de Diagonalización

Sea $w \in \{0,1\}^*$, $w \in L_d$ si y solo si la MT M cuya codificación es $w = \langle M \rangle$ no acepta w .

Se interpreta que si w no es un MT correcta, entonces representa una MT con un sólo estado y ninguna transición (siempre rechaza).

Si w_0, w_1, w_2, \dots son todas las palabras de $\{0,1\}^*$ ordenadas, entonces

		Palabras					
		w_0	w_1	w_2	w_3	...	
Máquinas	w_0	0	0	0	0	...	
	w_1	1	0	1	0	...	
	w_2	1	1	0	0	...	
	w_3	0	1	0	1	...	
	
	
		L_d	1	1	1	0	...

Teorema

L_d no es r.e.

Supongamos una MT M que acepta L_d . Dicha MT estará definida sobre el alfabeto $\{0,1\}$. Dicha máquina acepta palabras w tales que la MT cuya codificación es w no acepta la palabra w .

Sea w_M la codificación de la MT M : $w_M = R(M)$.

- Si $w_M \in L_d$ entonces M acepta w_M y la MT cuya codificación es w_M acepta la palabra w_M y, por la definición de L_d , $w_M \notin L_d$.
- Si $w_M \notin L_d$ entonces M no acepta w_M y la MT cuya codificación es w_M no acepta la palabra w_M y, por la definición de L_d , $w_M \in L_d$.

Con lo que la existencia de una MT M que acepte L_d nos lleva a una contradicción.

Problema de Diagonalización

A la versión problema de decisión del lenguaje de diagonalización le llamaremos **Problema de Diagonalización** y lo notaremos como **DIAGONAL(M)**.

Consiste en determinar si dada una máquina de Turing M , entonces M no acepta cuando se lee a ella misma.

Recordatorio: Problemas y Lenguajes

Dada u , ¿es $u \in L$?

Lenguaje L

Problema Decisión **PROBLEMA**

Conjunto x , es tal que **PROBLEMA**(x) = SI

Máquina Turing Acept.

Aceptar \equiv Responder SI

Algoritmo SI/NO

- Lenguaje recursivamente enumerable \equiv Problema **semidecidible**
 - Existe una MT que acepta las palabras del lenguaje. Para las palabras del lenguaje la MT puede rechazar o ciclar.
 - Existe un algoritmo que responde correctamente las entradas cuya salida es 'SI', para las entradas de 'NO' el algoritmo puede decir 'NO' o ciclar
- Lenguaje recursivo \equiv Problema **decidible**
 - Existe una MT que acepta las palabras del lenguaje y rechaza las palabras que no son del lenguaje (nunca cicla).
 - Existe un algoritmo que responde correctamente las entradas cuya salida es 'SI' y aquellas cuya salida es 'NO' (nunca cicla).

Complementarios de los lenguajes recursivos y r.e.

Si $L \subseteq A^*$, el complementario de L respecto a A^* se denotará como \bar{L} .

Teorema

Si L es un lenguaje recursivo, entonces \bar{L} también lo es.

Demostración

Si L es recursivo, entonces es aceptado por una MT que siempre para. Se construye \bar{M} con las siguientes características:

- Los estados de aceptación de M se convierten en estados de rechazo de \bar{M} donde no se realizan nuevas transiciones.
- \bar{M} tiene un nuevo estado r que es de aceptación y que no tiene transiciones definidas.
- Para cada estado p de M que no sea de aceptación y para cada símbolo de la cinta $a \in B$ para el que no haya definida una transición, se añade $\delta(p, a) = (r, a, D)$.

Está claro que \bar{M} acepta el lenguaje \bar{L} y siempre termina, ya que M lo hace.

Teorema

Si L y \bar{L} son ambos r.e., entonces L es recursivo.

Demostración

Sea M_1 la MT que acepta L y M_2 la MT que acepte \bar{L} . Vamos a construir una MT M con dos cintas: en una funciona como M_1 y en la otra como M_2 .

M es una máquina que funciona como M_1 y M_2 a la vez (como en el autómata producto). El conjunto de estados de M es el producto $Q_1 \times Q_2$ donde Q_1 es el conjunto de estados de M_1 y Q_2 el de M_2 . En cada paso, se pasa al estado que corresponde según M_1 y al estado que corresponde según M_2 .

La MT termina cuando una de las dos máquinas termina.

Los estados de aceptación de M son las parejas, en las que el primer elemento es un estado de aceptación de M_1 .

Está claro que M acepta L y siempre termina, ya que toda palabra $u \in A^*$ está en L o en \bar{L} . En el primer caso, M_1 termina y en el segundo lo hace M_2 . Por lo tanto M siempre termina.

El lenguaje L_d no es r.e. Su complementario podría ser r.e., pero nunca recursivo. De hecho, es r.e.

El Lenguaje Universal

Definición

El **lenguaje universal** L_U es el conjunto de todas $\langle M, w \rangle$ donde M tiene como alfabeto de entrada $\{0,1\}$, w está definida sobre el mismo alfabeto y tales que la MT M acepta la cadena w .

Problema Universal

A la versión problema de decisión del lenguaje universal le llamaremos **Problema Universal** y lo notaremos como $UNIVERSAL(M.w)$.

Consiste en determinar si dada una máquina de Turing M y una entrada w , entonces M acepta w .

Teorema

El lenguaje L_U es r.e.

La idea básica es construir una MT M_u que lea la codificación $R(M, w)$ simule la MT M sobre la entrada w y termine cuando termina M , aceptando si lo hace M . Esta MT se llama Máquina de Turing Universal. Está claro que si hace la simulación, aceptará cuando M acepta w . M_u contiene varias cintas:

- En la primera contiene la codificación de M y w
- En la segunda contiene lo que sería la cinta de M para la entrada w . Un símbolo $a_i \in B$ se representa como 0^i y los distintos símbolos se separan por un 1.
- En la tercera cinta representa el estado de M . El estado q_i se representa mediante 0^i
- La cuarta cinta se utiliza para cálculos auxiliares

Demostración (cont.)

- 1 Primero M_u examina M y w para asegurarse de que la entrada es correcta.
- 2 Inicializa la segunda cinta con w . El 0 se codifica como 01. El 1 se codifica como 001. Cada vez que introduzca un blanco lo tendrá que hacer como 0001. Todo de acuerdo con los códigos vistos.
- 3 Inicializa la tercera cinta con 0 que corresponde con el estado inicial (suponemos que es q_1).
- 4 Simula los movimientos. Tendrá que localizar en la primera cinta $0^i 10^j 10^k 10^l 10^m$ donde q_i es el estado en el que se encuentra, $0^j 1$ es lo que se ve en la segunda cinta. Si no lo encuentra para. Entonces debe de ejecutar el movimiento correspondientes:
 - 1 Cambiar el la cinta 3, el estado a 0^k
 - 2 Sustituir 0^j en la cinta 2 por 0^l
 - 3 Hacer el movimiento en la cinta 2, según sea m ($m = 1$ a la izquierda, $m = 2$ a la derecha).
- 5 Si M pasa a un estado de aceptación (el estado q_2), entonces M_u para y acepta.

Indecibilidad del Lenguaje Universal

Teorema

El lenguaje universal L_U no es recursivo.

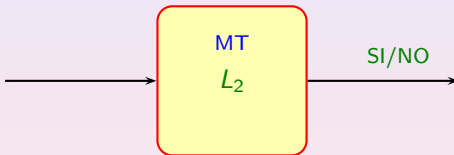
Demostración

Por reducción al absurdo.

- Si L_U fuese recursivo, entonces $\overline{L_U}$ también sería recursivo.
- Si M es una MT que aceptase $\overline{L_U}$, construiríamos la siguiente MT, M' :
 - Si M' lee w entonces, convierte w en $w111w$ (como la codificación de (w, w)).
 - Entonces pasa a funcionar como M , aceptando si M acepta.
 - M' acepta w si y solo sí, M acepta $w111w$. Es decir $w111w \notin L_U$. Esto es la MT cuya codificación es w no acepta la palabra w . Esto es equivalente a que $w \in L_d$.
- Hemos construido una MT que acepta L_d , en contra de lo que sabemos: L_d no es r.e.

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se **reduce** al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \rightarrow B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

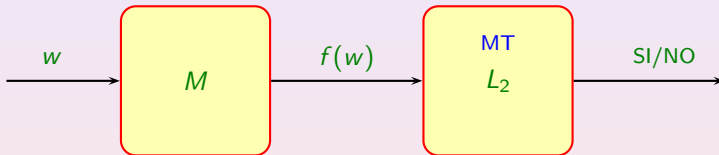


Teorema

Si L_1 se reduce a L_2 , entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2 .

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se **reduce** al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \rightarrow B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

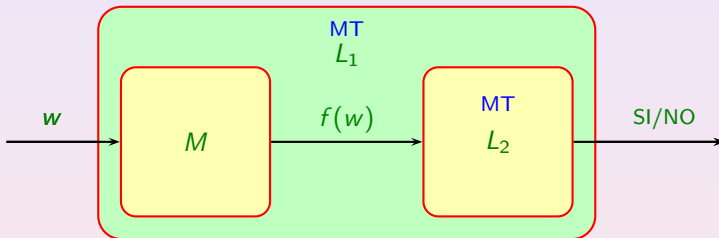


Teorema

Si L_1 se reduce a L_2 , entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2 .

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se **reduce** al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \rightarrow B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

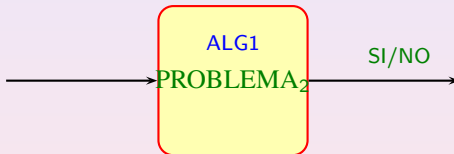


Teorema

Si L_1 se reduce a L_2 , entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2 .

Reducción (en términos de problemas)

Sean **PROBLEMA₁** y **PROBLEMA₂** problemas de decisión, entonces decimos que **PROBLEMA₁** se **reduce** a **PROBLEMA₂** si existe un algoritmo **ALG(*w*)** que siempre para y calcula una función ***f(w)*** tal que para toda entrada ***w*** a **PROBLEMA₁**, tenemos que **PROBLEMA₂** produce la misma respuesta para la entrada ***f(w) = ALG(w)***.

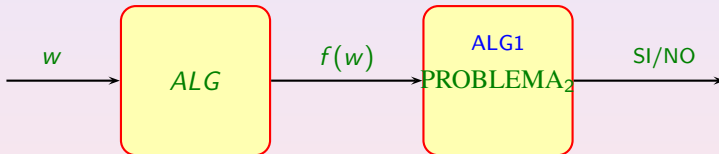


Una **reducción** de **PROBLEMA₁** a **PROBLEMA₂** es un algoritmo **ALG(*x*)** de las entradas de **PROBLEMA₁** a las entradas de **PROBLEMA₂** de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(\text{ALG}(x))$$

Reducción (en términos de problemas)

Sean **PROBLEMA₁** y **PROBLEMA₂** problemas de decisión, entonces decimos que **PROBLEMA₁** se **reduce** a **PROBLEMA₂** si existe un algoritmo **ALG(*w*)** que siempre para y calcula una función ***f*(*w*)** tal que para toda entrada ***w*** a **PROBLEMA₁**, tenemos que **PROBLEMA₂** produce la misma respuesta para la entrada ***f*(*w*) = ALG(*w*)**.

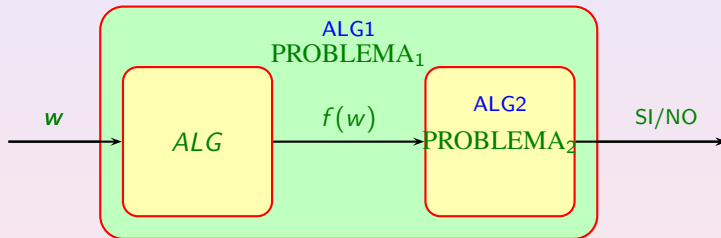


Una **reducción** de **PROBLEMA₁** a **PROBLEMA₂** es un algoritmo **ALG(*x*)** de las entradas de **PROBLEMA₁** a las entradas de **PROBLEMA₂** de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(\text{ALG}(x))$$

Reducción (en términos de problemas)

Sean **PROBLEMA₁** y **PROBLEMA₂** problemas de decisión, entonces decimos que **PROBLEMA₁** se **reduce** a **PROBLEMA₂** si existe un algoritmo **ALG(w)** que siempre para y calcula una función **f(w)** tal que para toda entrada **w** a **PROBLEMA₁**, tenemos que **PROBLEMA₂** produce la misma respuesta para la entrada **f(w) = ALG(w)**.



Una **reducción** de **PROBLEMA₁** a **PROBLEMA₂** es un algoritmo **ALG(x)** de las entradas de **PROBLEMA₁** a las entradas de **PROBLEMA₂** de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(\text{ALG}(x))$$

Reducción (en términos de problemas)

Teorema

Si **PROBLEMA₁** se reduce a **PROBLEMA₂**, entonces si **PROBLEMA₁** es indecidible, también lo es **PROBLEMA₂** y si **PROBLEMA₁** no es semidecidible, tampoco lo es **PROBLEMA₂**.

Sea **ALG(w)** el algoritmo de la reducción de **PROBLEMA₁** a **PROBLEMA₂**. Supongamos que **ALG2(x)** es un algoritmo que hace a **PROBLEMA₂** semidecidible (decidible), entonces el algoritmo:

ALG1(x)

w = ALG(x)

Return (ALG2(w))

hará al problema **PROBLEMA₁** semidecidible (decidible).

Por lo tanto si **PROBLEMA₁** no es semidecidible (decidible), tampoco lo puede ser **PROBLEMA₂**.

MTs que aceptan el lenguaje vacío

Definimos los siguientes lenguajes sobre el alfabeto $\{0,1\}$:

- L_e conjunto de palabras w tales que la MT M sobre $\{0,1\}$ cuya coficación es w no acepta ninguna palabra ($L(M) = \emptyset$).
- L_{ne} conjunto de palabras w tales que la MT M sobre $\{0,1\}$ cuya coficación es w acepta alguna palabra ($L(M) \neq \emptyset$).

Teorema

L_{ne} es r.e.

Demostración

Hay una MT no determinista M que acepta L_{ne} :

- M lee w que codifica una MT N , entonces con el algoritmo no determinista genera una palabra de entrada u .
- Después pone a simular N sobre u con la MT universal M_u .
- Si M_u para y acepta, entonces M acepta w .

- VACIO(M) es la versión de problema de L_e : dada una MT M , determinar si acepta el lenguaje vacío.
- C-VACIO(M) es la versión de problema de L_{ne} : dada una MT M , determinar si acepta un lenguaje distinto del vacío.

MTs que aceptan el lenguaje vacío

Teorema

L_{ne} no es recursivo

Demostración

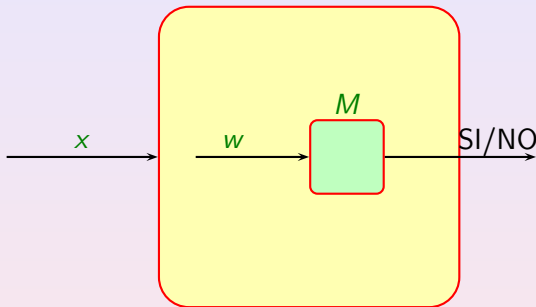
Vamos a demostrarlo usando los problemas asociados. Vamos a reducir UNIVERSAL(M, w) and C-VACIO(M'). Eso consiste en un algoritmo ALG(M, w) que calcula una MT M' de tal manera que UNIVERSAL(M, w) tenga la misma solución que C-VACIO(M'). Este funciona de la siguiente forma:

- Supongamos una entrada (M, w) vamos a construir una MT M' que funciona de la siguiente forma.
- M' ignora su entrada x y coloca en la cinta de entrada w . Si la longitud de w es n , esto se puede hacer con n estados. Cada estado q_i escribe el símbolo i de (M, w) y se mueve a la derecha. Después pasaría a un nuevo estado en el que borra lo que quede de x .
- M' se mueve a la izquierda hasta el primer símbolo de w .
- M' pasa al estado inicial de M con w y funciona como M para w .
- La salida de M' es la misma que la de M para w .

Está claro que M acepta w si y solo si M' acepta alguna palabra. De hecho $L(M') = A^*$ si M acepta w y $L(M') = \emptyset$ si M no acepta w .

Demostración Gráfica y Consecuencia

Dada una entrada (M, w) la reducción construye la siguiente MT:



Teorema

L_e no es r.e. (si lo fuese, entonces L_{ne} sería recursivo)

Propiedades de los r.e.

Propiedad lenguaje r.e. \leftrightarrow Propiedad de los lenguajes de las MTs

Una propiedad de los lenguajes r.e. se identifica con el problema de saber si el lenguaje de una MT verifica esa propiedad.

Es una problema de decisión del tipo: Dada una MT M , ¿verifica el lenguaje $L(M)$ la propiedad P ?

Ejemplo

Propiedades

Saber si el lenguaje de una MT es finito.

No es una propiedad de los r.e. saber si una MT tiene más de 10 estados.

Definición

Una propiedad de los lenguajes r.e. se dice **trivial** si para toda MT su lenguaje aceptado no verifica la propiedad o para toda MT su lenguaje aceptado siempre verifica la propiedad.

Teorema de Rice

Toda propiedad no trivial sobre los lenguajes r.e. es indecidible

Demostración

Llamemos NOTRIVIAL(M) a dicha propiedad no trivial y supongamos una propiedad no trivial y supongamos que el lenguaje vacío aceptado por la MT M_e no verifica la propiedad que otro lenguaje L aceptado por la MT M_L si verifica la propiedad.

Vamos a reducir el problema UNIVERSAL(M, w) a esta propiedad. Supongamos (M, w) una MT y su entrada construimos M' de la siguiente forma (se supone que tiene una entrada x en la primera cinta):

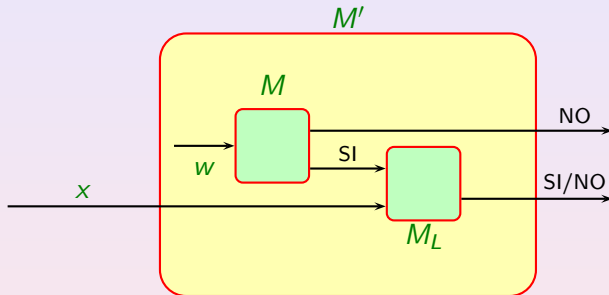
- M' tiene dos cintas. En la segunda coloca a w y empieza trabajando sobre esta cinta con las mismas transiciones de M . Si termina y no acepta, entonces M' no acepta.
- Si M termina y acepta con w , entonces empieza a funcionar como M_L sobre la entrada x en la primera cinta y tiene la misma salida que M_L .

Está claro que si M acepta w , entonces el lenguaje de M' es L y verifica la propiedad. Si M no acepta w , entonces el lenguaje de M' es vacío y no verifica la propiedad. Con esto se acaba la reducción y la propiedad no es decidible.

Si el lenguaje vacío acepta la propiedad se haría la misma transformación con la propiedad complementaria, demostrando que no es decidible y, por lo tanto, la propiedad original tampoco lo es.

Teorema de Rice. Gráfica de la Reducción

Dada una entrada para el problema universal (M, w) se construye la siguiente MT M'



Está claro $\text{UNIVERSAL}(M, w)$ tiene respuesta 'SI', entonces M acepta w , entonces M' acepta L y la respuesta de $\text{NOTRIVIAL}(M')$ es también 'SI'.

Si $\text{UNIVERSAL}(M, w)$ tiene respuesta 'NO', entonces M no acepta w , y esta máquina M' acepta \emptyset y la respuesta de $\text{NOTRIVIAL}(M')$ es también 'NO'.

Como $\text{UNIVERSAL}(M, w)$ no es decidable, $\text{UNIVERSAL}(M, w)$ tampoco lo es.

Otros Problemas Indecidibles

El Problema de las Correspondencias de Post (POST(A1,A2))

Tenemos un alfabeto de referencia A , y dos listas con la misma longitud $B_1 = w_1, \dots, w_k$, $B_2 = u_1, \dots, u_k$ de palabras sobre A . El problema es determinar si existe una secuencia no vacía de enteros i_1, \dots, i_m tales que $w_{i_1} \dots w_{i_m} = u_{i_1} \dots u_{i_m}$.

Podemos pensar en cada pareja (w_i, u_i) como un bloque de construcción:
 $w_i u_i$.

La especificación del problema nos da un conjunto de bloques disponibles. Por ejemplo:

$abba \quad babb \quad abb$

Otros Problemas Indecidibles

El Problema de las Correspondencias de Post (POST(A1,A2))

Tenemos un alfabeto de referencia A , y dos listas con la misma longitud $B_1 = w_1, \dots, w_k$, $B_2 = u_1, \dots, u_k$ de palabras sobre A . El problema es determinar si existe una secuencia no vacía de enteros i_1, \dots, i_m tales que $w_{i_1} \dots w_{i_m} = u_{i_1} \dots u_{i_m}$.

Podemos pensar en cada pareja (w_i, u_i) como un bloque de construcción: $w_i u_i$.

La especificación del problema nos da un conjunto de bloques disponibles. Por ejemplo:

$abba \quad babb \quad abb$

En este caso, la respuesta es afirmativa. Secuencia: $1, 3, 1, 1, 3, 2, 2$

$abbaabbabbaabbaabbbabbbabb \rightarrow$
 $abbaabbabbbabb$
 $abbaabbabbbabb$

El Problema de las Correspondencias de Post Modificado (POSTM(A1,A2))

PCP Modificado

Tenemos un alfabeto de referencia A , y dos listas con la misma longitud $B_1 = w_1, \dots, w_k$, $B_2 = u_1, \dots, u_k$ de palabras sobre A tales que $u_i, w_j \neq \varepsilon$ y un entero i . El problema es determinar si existe una secuencia no vacía de enteros i_1, \dots, i_m tales que $i_1 = i$ y $w_{i_1} \dots w_{i_m} = u_{i_1} \dots u_{i_m}$.

La única diferencia es que ahora nos dicen el bloque por el que necesariamente hay que comenzar y que las palabras son no vacías.

El Problema de las Correspondencias de Post Modificado

Teorema

El problema de las correspondencias de Post modificado es indecidible si A tiene al menos 2 símbolos.

Si A tiene dos símbolos, podemos codificar palabras de cualquier alfabeto, de manera que sea un homomorfismo.

Vamos a reducir el lenguaje universal a este problema.

Vamos a suponer una MT M y una palabra w . La pregunta es si $w \in L(M)$. Vamos a construir un problema de correspondencias de Post modificado con la misma solución.

El alfabeto que vamos a considerar para el alfabeto del problema de las correspondencias es el alfabeto de la MT, más el conjunto de estados, más un separador que no esté en los conjuntos anteriores $*$

Si $w = a_1 \dots a_n$

Introducimos el bloque $**q_0 a_1 \dots a_n*$

Demostración (Cont.)

Si $w = a_1 \dots a_n$

Introducimos el bloque $**q_0a_1 \dots a_n*$, que será el bloque de comienzo.

Por cada transición $\delta(q, a) = (q', b, D)$, introducimos el bloque $qabq'$

Por cada transición $\delta(q, a) = (q', b, I)$, introducimos el bloque $cqaq'cb$ para cada c del alfabeto de la MT

Para cada símbolo a de la MT y $*$ introducir aa

Los bloques $*\#*$, $**\#$

Demostración (Cont.)

Para cada $q \in F$, añade los bloques $aq q$, $qa q$,
y el bloque $q**$.

La idea de la reducción es representar en los bloques el cálculo que realiza la MT para la palabra de entrada w escribiendo el historial de sus configuraciones.

Cada configuración (q, u, v) se representa por la cadena uqv y las distintas configuraciones se separan por $*$.

En la parte de abajo se lleva una configuración de ventaja respecto a la parte de arriba.

Cuando llegamos a un estado de aceptación, la parte de abajo empieza a disminuir símbolo a símbolo paso a paso y copiándose arriba, hasta que quede abajo $\dots * q*$ y arriba $\dots * qa*$ ó $\dots * aq*$ donde q es el estado final. Entonces podemos completar con el último bloque aceptación.

El PCP es indecidible

Teorema

El problema de las correspondencias de Post es indecidible

Se reduce el problema de las correspondencias de Post modificado al problema de las correspondencias de Post.

Supongamos que nos dan un conjunto de bloques $u_1 v_1, u_2 v_2, \dots, u_n v_n$ y que el primero necesariamente es el bloque 1 (no hay ninguna pérdida de generalidad al suponer que el obligatorio para comenzar va primero).

Construimos el siguiente problema de Post con dos símbolos nuevos: \blacklozenge , \blacksquare y con los bloques:

$\blacklozenge u_1 \blacklozenge v_1 \blacklozenge, \blacklozenge u_1 v_1 \blacklozenge, \blacklozenge u_2 v_2 \blacklozenge, \dots, \blacklozenge u_n v_n \blacklozenge, \blacklozenge \blacksquare \blacksquare$

Donde, si $u = a_1 a_2 \dots a_n$, se entiende que

$\blacklozenge u = \blacklozenge a_1 \blacklozenge a_2 \dots \blacklozenge a_n, \quad u \blacklozenge = a_1 \blacklozenge a_2 \dots \blacklozenge a_n \blacklozenge, \quad \blacklozenge u \blacklozenge = \blacklozenge a_1 \blacklozenge a_2 \dots \blacklozenge a_n \blacklozenge$

Es fácil comprobar que los dos problemas son equivalentes.

Problemas sobre Gramáticas

Suponemos que G , G_1 y G_2 son gramáticas independientes del contexto dadas y R es un lenguaje regular.

- Saber si $L(G_1) \cap L(G_2) = \emptyset$.
- Determinar si $L(G) = T^*$, donde T es el conjunto de símbolos terminales.
- Comprobar si $L(G_1) = L(G_2)$.
- Determinar si $L(G_1) \subseteq L(G_2)$.
- Determinar si $L(G_1) = R$.
- Comprobar si $L(G)$ es regular.
- Determinar si G es ambigua.
- Conocer si $L(G)$ es inherentemente ambiguo.
- Comprobar si $L(G)$ es determinista.

Teorema

Saber si una gramática independiente del contexto es ambigua (AMBIGUA(G)) es indecidible.

Se reduce el problema de las correspondencias de Post. Supongamos el alfabeto A y los bloques $u_1 v_1, \dots, u_k v_k$.

Sea $B = A \cup \{b_1, \dots, b_k\}$ donde $b_i \notin A$ y construimos la siguiente gramática:

$$S \rightarrow C|D$$

$$C \rightarrow u_i C b_i | u_i b_i, \quad i = 1, \dots, k$$

$$D \rightarrow v_i D b_i | v_i b_i, \quad i = 1, \dots, k$$

La solución al problema de las correspondencias de Post es equivalente a que la gramática sea ambigua.

Demostración

Supongamos el alfabeto A y los bloques $u_1 v_1, \dots, u_k v_k$.

Sea $B = A \cup \{b_1, \dots, b_k\}$ donde $b_i \notin A$ y construimos la siguiente gramática:

$S \rightarrow C|D$, $C \rightarrow u_i C b_i | u_i b_i$, $i = 1, \dots, k$, $D \rightarrow v_i D b_i | v_i b_i$, $i = 1, \dots, k$,

Se basa en lo siguiente:

- Las palabras generadas por la gramática son las generadas a partir de C más las generadas a partir de D
- Las palabras generadas a partir de C son de la forma $u_{i_1} \dots u_{i_l} b_{i_l} \dots b_{i_1}$, donde $i_j \in \{1, \dots, k\}$. Sólo hay una forma de generar una de estas palabras a partir de C .
- Las palabras generadas a partir de D son de la forma $v_{i_1} \dots v_{i_l} b_{i_l} \dots b_{i_1}$, donde $i_j \in \{1, \dots, k\}$. Sólo hay una forma de generar una de estas palabras a partir de D .
- La gramática es ambigua cuando una misma palabra u es generada a partir de C y a partir de D , es decir cuando $u = u_{i_1} \dots u_{i_l} b_{i_l} \dots b_{i_1} = v_{i'_1} \dots v_{i'_l} b_{i'_l} \dots b_{i'_1}$. Esto solo ocurre si $l' = l$ e $i_j = i'_j, \forall j$. Es decir cuando existen (i_1, \dots, i_l) tal que $u_{i_1} \dots u_{i_l} = v_{i_1} \dots v_{i_l}$, es decir cuando el PCP tiene solución.