# Máquinas de Turing - Guía de Lectura

Enrique Romero
Teoria de la Computació
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

November 24, 2021

Este documento pretende ser una guía de lectura del material sobre *Máquinas de Turing* de la asignatura *Teoria de la Computació* (FIB, UPC).

## Introducción

Una Máquina de Turing (TM, Turing Machine) es un modelo de cálculo similar (en la forma) a los autómatas finitos (FA) y autómatas con pila (PDA):

1. Está formada por un conjunto de estados, con un estado inicial y un estado final
2. Las transiciones entre estados permiten "cambiar de estado"
3. Dispone de un mecanismo de memoria auxiliar, pero a diferencia de los PDA no es una pila, sino una cinta, por la que podremos movernos y escribir sin restricciones
4. Se puede asociar un lenguaje a una TM de manera similar a como se asocia un lenguaje a un FA o un PDA

La propiedad de disponer de una cinta como dispositivo auxiliar de memoria hace que las TM sean capaces de simular cualquier otro modelo de cálculo, y en particular proporcionan todas las funcionalidades que se pueden esperar de un algoritmo. Como consecuencia, podremos estudiar algunos problemas relacionados con algoritmos usando TM de manera "muy sencilla". Por ejemplo, de la misma forma que nos podemos preguntar "¿Todo lenguaje se puede reconocer con un FA?" o "¿Todo lenguaje se puede reconocer con un PDA?" nos podremos preguntar "¿Todo lenguaje se puede reconocer con una TM?", que es equivalente a preguntarse "¿Todo lenguaje se puede reconocer con un algoritmo?", que es un caso particular de la pregunta "¿Todo problema se puede resolver son un algoritmo?" Como ya veremos, la respuesta a esta pregunta es, igual que en el caso de FA y PDA, negativa, de manera que HAY PROBLEMAS QUE NO SE PUEDEN RESOLVER CON UN ALGORITMO.

## Definición de Máquina de Turing (libro TC-CI, pp. 9-11, vídeo 27)

DEFINICIÓ Una *màquina de Turing* és una estructura de la forma $(Q, \Sigma, \Gamma, \delta, q_0, q_F)$, on

$Q$ és un conjunt finit d'estats.

$\Sigma$ és un alfabet, l'alfabet d'entrada.

$\Gamma$ és un alfabet, l'alfabet de cinta, tal que $\Sigma \cup \{\mathtt{b}, \blacktriangleright\} \subseteq \Gamma$, on $\mathtt{b}, \blacktriangleright \notin \Sigma$.

$\delta$ és la funció de transició, $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathtt{e}, \mathtt{d}, \mathtt{n}\}$

$q_0$ és l'estat inicial ($q_0 \in Q$).

$q_F$ és l'estat final o acceptador ($q_F \in Q$).

Funcionamiento: ver documentación

## Lenguaje reconocido y función calculada por una TM (libro TC-CI, pp. 14-16, vídeo 28)

Una configuración de una TM es una descripción del estado de la máquina en un instante determinado. Se representa como una palabra $\alpha q \beta \in \Gamma^* Q \Gamma^*$ donde

1. La máquina se encuentra en el estado $q$
2. El contenido de la cinta es $\alpha\beta$ ($\beta$ no acaba en $\mathtt{b}$)
3. El cabezal de la cinta apunta al primer carácter de $\beta$ (o $\mathtt{b}$ si $\beta = \lambda$)

Con esta notación, la configuración inicial se representa como $\blacktriangleright\, q_0\, \omega$.

Se dice que una configuración es *terminal* cuando la máquina no puede cambiar de configuración. Esto puede pasar por dos motivos:

1. No hay ninguna transición definida para la combinación estado-símbolo de la configuración
2. La configuración es de la forma $qx$ (es decir, el cabezal apunta al primer símbolo de la cinta) y la transición que le corresponde intenta moverse hacia la izquierda (similar a cuando en un PDA eliminamos todos los símbolos de la pila)

Por otro lado, el cálculo de una TM con una entrada concreta $\omega$ puede parar (en una configuración terminal) o no (y en ese caso la TM va cambiando infinitamente de configuración sin pasar por ninguna configuración terminal). Usaremos la siguiente notación:

- $M(w)\uparrow$ significa que la TM $M$ amb entrada $w$ no s'atura.
- $M(w)\downarrow$ significa que la TM $M$ amb entrada $w$ s'atura.
- En cas que $M(w)\downarrow$, $M(w)$ representa el mot de $\Sigma^*$ corresponent a l'última configuració del càlcul de $M$ amb entrada $w$, contingut a la cinta i comprés entre el primer i el segon caràcter que no són de $\Sigma$. Si no s'ha modificat, $\blacktriangleright$ és el primer caràcter que no pertany a $\Sigma$.

Ahora ya podemos definir el lenguaje reconocido y la función calculada por una TM:

DEFINICIÓ Un mot $w \in \Sigma^*$ és *acceptat* per una màquina de Turing $M$ si $M$ amb entrada $w$ s'atura i ho fa en l'estat acceptador.

A partir d'aquest concepte podem definir el de llenguatge reconegut per una TM.

DEFINICIÓ El *llenguatge reconegut o acceptat* per una TM $M$, representat per $\mathsf{L}(M)$, està format pel conjunt de mots acceptats per $M$. Donat un llenguatge $L$, diem que $M$ *reconeix o accepta* $L$ si $\mathsf{L}(M) = L$. Diem que un llenguatge $L$ és *enumerable recursivament* si existeix una TM que reconeix $L$.

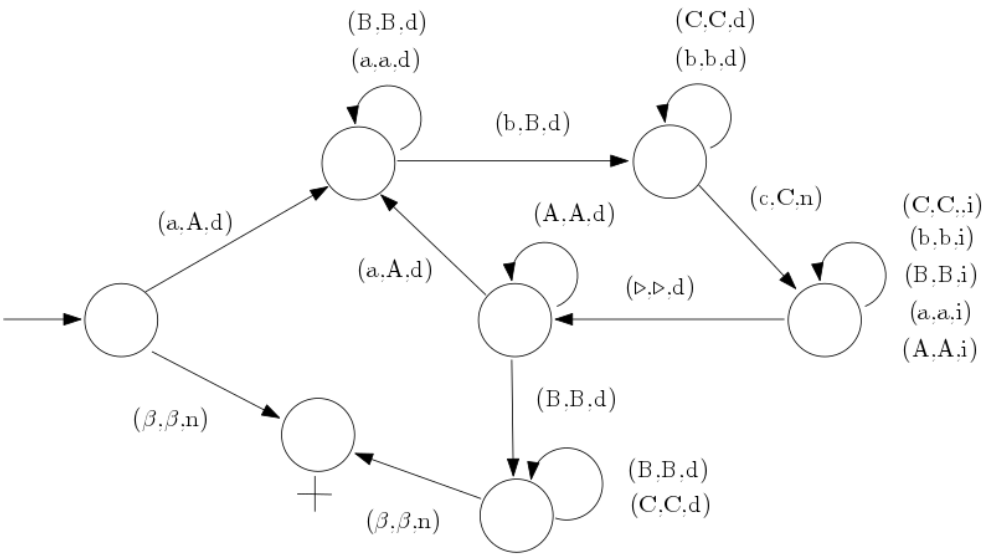
Quan una TM s'atura amb una entrada, podem extreure informació del que ha quedat escrit a la cinta i associar-la com a sortida. Això ens permet introduir els conceptes de funció computada i funció computable.

DEFINICIÓ La *funció computada* per una TM $M$, representada per $\mathsf{f}_M$, es defineix així: $\mathsf{f}_M(w)$ és $M(w)$ si $M(w)\downarrow$, i està indefinida en cas contrari. Donada una funció $f : \Sigma^* \longrightarrow \Sigma^*$ diem que $M$ *computa* $f$ si $f = \mathsf{f}_M$. Finalment, diem que una funció és *computable* si hi ha una TM $M$ per a la qual $f = \mathsf{f}_M$.

Y a partir de aquí, definir nuevas clases de lenguajes (y funciones) como hacíamos con FA y PDA:

1. Un lenguaje $L$ es **semi-decidible** (o **recursivamente enumerable**) si existe una maquina de Turing $M$ tal que $L(M) = L$
2. Un lenguaje $L$ es **decidible** (o **recursivo**) si existe una maquina de Turing $M$ tal que $L(M) = L$ y además $M$ para con todas las entradas $(\forall \omega \;\; M(\omega)\downarrow)$
3. Una función $f$ es **calculable** si existe una maquina de Turing $M$ tal que $f_M = f$

Ejemplo para $L = \{a^n b^n c^n \mid n \geq 0\}$




$$\mathsf{L} = \{a^n b^n c^n \mid n \geq 0\}$$

La notación entre lenguajes reconocidos y funciones calculadas es ligeramente diferente:

| Resultado de $M(\omega)$ | $L(M)$ | $f_M$ |
|---|---|---|
| $M(\omega)\downarrow$ en un estado final | $\omega \in L(M)$ | $f_M(\omega)\downarrow$ |
| $M(\omega)\downarrow$ en un estado no final | $\omega \notin L(M)$ | $f_M(\omega)\downarrow$ |
| $M(\omega)\uparrow$ | $\omega \notin L(M)$ | $f_M(\omega)\uparrow$ |

## Algunos Problemas sobre TM (libro TC-CI, pp. 20-21)

Aturada (HALT)
Donats un mot $w$ i una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, determinar si la màquina de Turing $M$ s'atura amb entrada $w$.

La condició $M(w)\downarrow$ es pot expressar com

$$\boxed{\exists\, q \in Q\ \exists\, \alpha, \beta \in \Gamma^*\ \exists\, a \in \Gamma \quad \blacktriangleright q_0 w \overset{*}{\underset{M}{\vdash}} \alpha q a \beta \wedge (q, a) \notin \mathsf{Dom}\,\delta}$$

Pertinença (PERT)
Donats un mot $w$ i una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, determinar si la màquina de Turing $M$ accepta $w$.

La propietat $w \in \mathsf{L}(M)$ és pot expressar com

$$\boxed{\exists \alpha, \beta \in \Gamma^* \quad \blacktriangleright q_0 w \overset{*}{\underset{M}{\vdash}} \alpha q_F \beta}$$

Equivalència (TM-EQUIV)
Donades dues màquines de Turing $M$ i $N$, determinar si reconeixen el mateix llenguatge.

La propietat $\mathsf{L}(M) = \mathsf{L}(N)$ és equivalent a

$$\boxed{\forall w \in \Sigma^* \quad w \in \mathsf{L}(M) \Longleftrightarrow w \in \mathsf{L}(N)}$$

## Máquinas de Turing y Algoritmos (libro TC-CI, pp. 31-46, vídeos 29-30)

Como hemos comentado, las TM son capaces de simular cualquier otro modelo de cálculo, y en particular proporcionan todas las funcionalidades que se pueden esperar de los algoritmos:

1. **Los esquemas algorítmicos básicos se pueden implementar con TM** (pp. 31-39):
   (a) Composición secuencial
   (b) Composición condicional
   (c) Composición iterativa
   (d) Uso de valores constantes
2. **Las TM se pueden representar como cadenas de caracteres** sobre un cierto alfabeto (pp. 39-41), de manera que:
   (a) La representación cumple los requerimientos de una buena codificación
   (b) Tiene sentido definir el lenguaje formado por todas las TM e intentar resolver problemas que se refieren a subconjuntos de TM que cumplan una determinada propiedad (por ejemplo, HALT, PERT y TM-EQUIV)
   (c) Tiene sentido que una TM sea la entrada a otra TM (como hace un compilador o un intérprete, por ejemplo)
3. Como consecuencia del punto anterior, **podemos asignar un número natural a cada codificación de una TM de manera biyectiva**:
   (a) Todo número natural tiene asignado una TM, y esta asignación se puede hacer con una TM
   (b) Toda TM tiene asignado un número natural, y esta asignación se puede hacer con una TM; El número natural asignado a una TM se llama **"número de Gödel"**, y a la función que asigna números de Gödel a partir una TM $m$ la llamaremos $gödel(m)$
   (c) Notación:
      i. $M_x$ es la TM de número de Gödel $x$
      ii. $L_x$ es el lenguaje reconocido por la TM de número de Gödel $x$
      iii. $\varphi_x$ es la función calculada por la TM de número de Gödel $x$
4. Esta misma idea se puede aplicar a las palabras definidas sobre cualquier alfabeto, de manera que, a partir de ahora, **podremos suponer que tenemos las TM representadas como números naturales y las entradas a las TM también representadas como números naturales**

   Por tanto, podemos reformular los problemas sobre TM de una manera mucho más concisa:

$$\texttt{HALT} = \{\langle x, y\rangle \mid M_x(y) \downarrow\}$$
$$\texttt{PERT} = \{\langle x, y\rangle \mid y \in L_x\}$$
$$\texttt{TM-EQUIV} = \{\langle x, y\rangle \mid L_x = L_y\}$$

5. **Existen TM que simulan el comportamiento de cualquier otra TM con cualquier entrada** (pp. 42-45):
   (a) Máquina universal: Con entrada $\langle x, y\rangle$, simula $M_x(y)$
   (b) Máquina reloj: Con entrada $\langle x, y, t\rangle$, simula $t$ pasos de $M_x(y)$

Todos estos resultados justifican la utilización de las TM como modelo formal de algoritmo:

**Tesi de Church-Turing**
La noció intuïtiva d'algorisme es correspon amb la de màquina de Turing.

Asumiendo la tesis de Church-Turing y todas las propiedades mencionadas anteriormente, a partir de ahora podremos trabajar con TM "a alto nivel" (es decir, no hará falta explicitar hasta el último detalle de las operaciones que ya sepamos que se pueden hacer con TM). A modo de ejemplo, podremos escribir una TM de la siguiente forma:

*Input* $x$
*If* $x$ "es múltiplo de 3" *then*
  *Compute* $y = M_x(x/3)$
  *output* $y$
*else*
  $z = 3 * x$
  *Simulate* $x$ *steps of* $M_z(x)$
  *If* "la simulación no acaba en estado final" *then*
    *loop*
  *else*
    *output* $x$

Esta manera de trabajar es mucho más cercana a la que ya estamos acostumbrados a trabajar con algoritmos.

## Indecidibilidad de Lenguajes (libro TC-CI, pp. 49-63, vídeos 32-33)

Las consecuencias de la tesis de Church-Turing son muy importantes, porque permiten estudiar propiedades de los algoritmos utilizando un modelo formal bien definido, sobre el que podremos aplicar razonamientos lógicos. Uno de los resultados más importantes es la existencia de lenguajes indecidibles (equivalente a la existencia de lenguajes no regulares o no incontextuales), que, como se ha comentado en la Introducción, es equivalente a decir que HAY PROBLEMAS QUE NO SE PUEDEN RESOLVER CON UN ALGORITMO.

El primer lenguaje que vamos a demostrar que no es decidible es el siguiente:

$$\mathtt{K} = \{x \mid M_x(x) \downarrow\}$$

El lenguaje K es el lenguaje de las TM que paran cuando reciben como entrada el número natural correspondiente a su propia codificación. Es muy fácil ver que K es semi-decidible, porque la siguiente TM reconoce K (aunque no es de parada segura):

*Input* $x$
*Simulate* $M_x(x)$
*accept*

El problema es cómo saber cuándo $M_x(x)$ no para:

1. ¿Simulando $M_x(x)$? Seguro que no, porque no para
2. ¿Inspeccionando el código? Podría ser, pero ¿cómo podemos saber todas las (posiblemente infinitas) configuraciones en las que la TM se puede encontrar?

**Teorema.** K no es decidible.

La demostración se puede hacer por reducción al absurdo. La idea es suponer que K es decidible y llegar a una contradicción que es consecuencia de la existencia de una máquina de parada segura que reconoce K.

*Demostración.* Supongamos que K es decidible, y sea $M_{\mathtt{K}}$ la máquina de parada segura que decide K. Eso quiere decir que, para toda entrada $y$,

1. $M_{\mathtt{K}}(y)$ siempre acepta o rechaza (parada segura)
2. $M_{\mathtt{K}}(y)$ acepta si y sólo si $M_y(y) \downarrow$ (es decir, $M_{\mathtt{K}}(y)$ rechaza si y sólo si $M_y(y) \uparrow$)

Ahora podemos construir la TM siguiente, a la que llamaremos $N$:

*Input* $z$
*If* $M_{\mathtt{K}}(z)$ *accepts then*
  *loop*
*stop*

En primer lugar, fijémonos que $N$ es realmente una TM (es decir, que todas las operaciones que están indicadas se pueden implementar con el modelo de TM):

1. La instrucción condicional se puede implementar con una TM, como hemos visto anteriormente
2. Comprobar si $M_{\mathtt{K}}(z)$ acepta o no se puede implementar con una TM, porque
   (a) $M_{\mathtt{K}}$ existe (por hipótesis)
   (b) Podemos simular el comportamiento de cualquier TM con cualquier entrada (máquina universal)
3. Tanto *loop* como *stop* se pueden implementar muy fácilmente con TM

Además, dado que $M_{\mathtt{K}}$ es de parada segura, tenemos que

1. $N(z)$ para si y sólo si $M_{\mathtt{K}}(z)$ rechaza
2. $N(z)$ entra en bucle si y sólo si $M_{\mathtt{K}}(z)$ acepta

Ahora ya estamos en condiciones de acabar la demostración. Sea $n$ el número de Gödel de $N$. La pregunta que nos llevará a contradicción es "¿$n$ pertenece a K?"

1. Supongamos que $n$ pertenece a K. Entonces

$$n \in \mathtt{K} \Rightarrow M_n(n)\downarrow \Rightarrow N(n)\downarrow \Rightarrow M_{\mathtt{K}}(n) \text{ rechaza } \Rightarrow M_n(n)\uparrow \Rightarrow n \notin \mathtt{K}$$

   La primera implicación es por definición de K
   La segunda implicación es porque $n$ es el número de Gödel de $N$ ($M_n = N$)
   La tercera implicación es por la construcción de $N$ (la entrada $z$ tiene valor $n$)
   La cuarta implicación es porque estamos suponiendo que K es decidible, y $M_{\mathtt{K}}$ es la máquina de parada segura que decide K
   La quinta implicación es por definición de K
   Así pues, tenemos $n \in \mathtt{K} \Rightarrow n \notin \mathtt{K}$. CONTRADICCIÓN, que viene de suponer que $M_{\mathtt{K}}$ es la máquina de parada segura que decide K

2. Si suponemos que $n$ no pertenece a K también llegamos a contradicción, con la misma cadena de razonamientos:

$$n \notin \mathtt{K} \Rightarrow M_n(n)\uparrow \Rightarrow N(n)\uparrow \Rightarrow M_{\mathtt{K}}(n) \text{ acepta } \Rightarrow M_n(n)\downarrow \Rightarrow n \in \mathtt{K}$$

   La primera implicación es por definición de K
   La segunda implicación es porque $n$ es el número de Gödel de $N$ ($M_n = N$)
   La tercera implicación es por la construcción de $N$ (la entrada $z$ tiene valor $n$)
   La cuarta implicación es porque estamos suponiendo que K es decidible, y $M_{\mathtt{K}}$ es la máquina de parada segura que decide K
   La quinta implicación es por definición de K
   Así pues, tenemos $n \notin \mathtt{K} \Rightarrow n \in \mathtt{K}$. CONTRADICCIÓN, que viene de suponer que $M_{\mathtt{K}}$ es la máquina de parada segura que decide K

La no decidibilidad de K tiene, como consecuencia, que su complementario, $\bar{\mathtt{K}}$, no es semi-decidible

**Corolario.** $\bar{\mathtt{K}}$ no es semi-decidible.

*Demostración.* El teorema del complementario (libro TC-CI, p. 62) afirma que si $L$ y $\bar{L}$ son semi-decidibles, entonces $L$ es decidible. Ya sabemos que K es semi-decidible. Si $\bar{\mathtt{K}}$ fuese semi-decidible, entonces aplicando el teorema del complementario tendríamos que K sería decidible. CONTRADICCIÓN, que viene de suponer que $\bar{\mathtt{K}}$ es semi-decidible.

Ahora que ya sabemos que hay lenguajes indecidibles, podemos usarlo para demostrar que hay otros lenguajes que tampoco son decidibles.

**Corolario.** HALT no es decidible.

Recordemos que

$$\texttt{HALT} = \{\langle x, y\rangle \mid M_x(y) \downarrow\}$$

La idea de la demostración es, como en el caso de K, suponer que es decidible y llegar a contradicción. La diferencia es que ahora, en lugar de demostrarlo directamente, la contradicción la buscaremos deduciendo que, si HALT fuese decidible, entonces K sería decidible, y ya sabemos que no lo es.

*Demostración.* Supongamos que HALT es decidible, y sea $M_{\texttt{HALT}}$ la máquina de parada segura que decide HALT. Eso quiere decir que, para toda entrada $\langle x, y\rangle$,

1. $M_{\texttt{HALT}}(x, y)$ siempre acepta o rechaza (parada segura)
2. $M_{\texttt{HALT}}(x, y)$ acepta si y sólo si $M_x(y) \downarrow$
   $(= M_{\texttt{HALT}}(x, y)$ rechaza si y sólo si $M_x(y) \uparrow)$

Ahora podemos construir la TM siguiente, a la que llamaremos $N$:

*Input* $x$
*If* $M_{\texttt{HALT}}(x, x)$ *accepts then*
  *accept*
*reject*

Entonces,

1. $N$ es de parada segura, porque $M_{\texttt{HALT}}$ lo es
2. $N(x)$ acepta si y sólo si $M_{\texttt{HALT}}(x, x)$ acepta (es decir, si $x \in \texttt{K}$)
3. $N(x)$ rechaza si y sólo si $M_{\texttt{HALT}}(x, x)$ rechaza (es decir, si $x \notin \texttt{K}$)

Es decir, $N$ es una TM de parada segura que reconoce K, CONTRADICCIÓN, que viene de suponer que HALT es decidible.

**Corolario.** El lenguaje $\texttt{NONEMPTY} = \{x \mid \exists y\ M_x(y)\downarrow\}$ no es decidible.

Aunque lo podríamos demostrar de manera similar al caso anterior, esta vez usaremos una estrategia ligeramente diferente, cuya idea la podremos generalizar al concepto de reducción, que veremos en la siguiente (y última) sección. La idea general seguirá siendo suponer que es decidible para llegar a contradicción construyendo una TM de parada segura para K.

*Demostración.* Consideremos la siguiente familia de TM, a la que llamaremos $R[x]$:

*Input* $z$
*Simulate* $M_x(x)$
*stop*

Observemos que:

1. La $x$ no es una variable local, de manera que para cada $x_0$ tenemos una TM diferente $R[x_0]$
2. Fijado $x_0$, el comportamiento de $R[x_0]$ es el mismo para todas las posibles entradas $z$:
   (a) Si $x_0 \in \texttt{K}$, entonces $R[x_0](z)\downarrow$ para cualquier $z$
   (b) Si $x_0 \notin \texttt{K}$, entonces $R[x_0](z)\uparrow$ para cualquier $z$

   O, de manera equivalente,

   (a) Si $x_0 \in \texttt{K}$, entonces $g\ddot{o}del(R[x_0]) \in \texttt{NONEMPTY}$
   (b) Si $x_0 \notin \texttt{K}$, entonces $g\ddot{o}del(R[x_0]) \notin \texttt{NONEMPTY}$

Ahora ya podemos acabar la demostración. Sea $f(x) = g\ddot{o}del\,(R[x])$, que es una función que cumple:

1. Es calculable, existe una TM que la calcula (ver Teorema s-m-n, pp. 71-74)
2. Es total, porque está definida para todo $x$ (aquí es clave darse cuenta de que $f(x)$ simplemente es el número de Gödel, no se va a simular ninguna TM)
3. $M_{f(x)} = R[x]$, de manera que
   (a) Si $x \in \texttt{K}$, entonces $f(x) \in \texttt{NONEMPTY}$
   (b) Si $x \notin \texttt{K}$, entonces $f(x) \notin \texttt{NONEMPTY}$

Supongamos que NONEMPTY es decidible, y sea $M_{\texttt{NONEMPTY}}$ la máquina de parada segura que decide NONEMPTY. Entonces la siguiente TM, a la que llamaremos $N$:

*Input* $x$
*Compute* $y = f(x)$
*If* $M_{\texttt{NONEMPTY}}(y)$ *accepts then*
  *accept*
*reject*

es una TM que cumple que

1. Es de parada segura (tanto $f$ como $M_{\texttt{NONEMPTY}}$ son TM de parada segura)
2. Acepta una entrada $x$ si y sólo si $x \in \texttt{K}$:

$$N(x)\ \textit{acepta} \Leftrightarrow M_{\texttt{NONEMPTY}}(f(x))\ \textit{acepta} \Leftrightarrow f(x) \in \texttt{NONEMPTY} \Leftrightarrow x \in \texttt{K}$$

y por tanto K es decidible. CONTRADICCIÓN, que viene de suponer que NONEMPTY es decidible.

## **Reducciones** (libro TC-CI, pp. 67-71, vídeos 32-33)

Las reducciones son una herramienta que nos permitirán demostrar la indecidibilidad (o no semi-decidibilidad) de lenguajes sin tener que construir explícitamente un argumento como los de la sección anterior. El resultado será que si podemos "reducir" un lenguaje $A$ a otro lenguaje $B$, entonces podremos establecer una relación entre la (no) decidibilidad (o semi-decidibilidad) de $A$ y la de $B$. La idea que hay detrás de la definición de reducción es esencialmente la misma que ya hemos visto en el último ejemplo de la sección anterior.

DEFINICIÓ Donats dos llenguatges $A$ i $B$ sobre un alfabet $\Sigma$, diem que $A$ es *redueix* (o *és reductible*) a $B$, i ho representem per $A \leqslant_{\mathrm{m}} B$, quan existeix una funció $f : \Sigma^* \longrightarrow \Sigma^*$ total i computable tal que, per a tot $w \in \Sigma^*$,

$$w \in A \Longleftrightarrow f(w) \in B.$$

La funció $f$ s'anomena *funció de reducció*.

Aunque no lo hemos dicho explícitamente, en la sección anterior ya hemos visto algunos ejemplos:

1. $\mathtt{K} \leqslant_m \mathtt{HALT}$ con $f(x) = \langle x, x \rangle$
2. $\mathtt{K} \leqslant_m \mathtt{NONEMPTY}$ con $f(x) = g\ddot{o}del(R[x])$

Ahora ya podemos demostrar el resultado principal sobre reducciones:

**Teorema.** Sean $A$ y $B$ dos lenguajes tales que $A \leqslant_m B$. Entonces:

1. Si $B$ es decidible, entonces $A$ es decidible
2. Si $B$ es semi-decidible, entonces $A$ es semi-decidible

*Demostración.* La idea de la demostración es exactamente la misma que la demostración de que `NONEMPTY` no es decidible, aunque más sencilla porque nos podemos ahorrar algunos pasos que ya están incluidos en la definición de reducción.

1. Si $B$ es decidible, entonces $A$ es decidible:
   Sea $M_B$ la TM de parada segura que reconoce $B$
   Sea $M_f$ la TM que calcula la reducción de $A$ a $B$ (que también es de parada segura porque es total)
   Entonces, la siguiente TM es de parada segura y reconoce $A$:

   *Input* $x$
   *Simulate* $y = M_f(x)$
   *If* $M_B(y)$ *accepts then*
   *accept*
   *reject*

   Es de parada segura porque tanto $M_f$ como $M_B$ son de parada segura
   Reconoce $A$ porque $x \in A \Leftrightarrow y = f(x) \in B$
2. Si $B$ es semi-decidible, entonces $A$ es semi-decidible:
   La demostración es análoga al caso anterior, con la misma construcción. Simplemente en esta caso hay que tener en cuenta que $M_B$ no es de parada segura, y por tanto la TM construida tampoco lo será, pero sí que reconoce $A$

Por último, para demostrar la no (semi-) decidibilidad de un lenguaje podemos usar el contrarrecíproco del teorema anterior:

**Corolario.** Sean $A$ y $B$ dos lenguajes tales que $A \leqslant_m B$. Entonces:

1. Si $A$ no es decidible, entonces $B$ no es decidible
2. Si $A$ no es semi-decidible, entonces $B$ no es semi-decidible

Ahora podemos volver a demostrar que `HALT` y `NONEMPTY` son indecidibles usando reducciones:

1. $\texttt{K} \leqslant_m \texttt{HALT}$ con $f(x) = \langle x, x \rangle$
   Como `K` no es decidible, entonces `HALT` no es decidible
2. $\texttt{K} \leqslant_m \texttt{NONEMPTY}$ con $f(x) = gödel(R[x])$
   Como `K` no es decidible, entonces `NONEMPTY` no es decidible

Y también podemos hacer nuevas demostraciones de manera mucho más fácil, como por ejemplo:

1. El problema 3 del RACSO: $\{p \mid \exists y : M_p(y) \uparrow\}$ no es semi-decidible, porque podemos construir una reducción $\bar{\texttt{K}} \leqslant_m \{p \mid \exists y : M_p(y) \uparrow\}$ con $f(x) = gödel(R_3[x])$, donde $R_3[x]$ es la TM siguiente:

   *Input y*
   *Simulate* $M_x(x)$
   *stop*

2. El problema 2 del RACSO: $\{p \mid \forall y : M_p(y) \downarrow\}$ no es semi-decidible, porque podemos construir una reducción $\bar{\texttt{K}} \leqslant_m \{p \mid \forall y : M_p(y) \downarrow\}$ con $f(x) = gödel(R_2[x])$, donde $R_2[x]$ es la TM siguiente:

   *Input y*
   *If* $M_x(x)$ *stops in y steps*
   *loop*
   *stop*

**Conjuntos de índices y Teorema de Rice** (libro TC-CI, pp. 76-79, vídeo 33).
El teorema de Rice es un caso particular de aplicación de las reducciones, que en algunos casos (cuando el lenguaje es un conjunto de índices) puede dar lugar a demostraciones más sencillas de la no decidibilidad y/o la no semi-decidibilidad de lenguajes:

$A$ és un conjunt d'índexs $\iff \forall x, y\ (x \in A\ \wedge\ \varphi_x = \varphi_y\ \implies\ y \in A)$

TEOREMA 4.6 (Teorema de Rice) *Sigui $A$ un conjunt d'índexs. $A$ és recursiu si i només si $A = \varnothing$ o $A = \mathbb{N}$.*

COROL·LARI 4.7 *Sigui $A$ un conjunt d'índexs no recursiu. Si existeix $x \in A$ tal que $\mathsf{Dom}(\varphi_x) = \varnothing$, aleshores $A$ no és enumerable recursivament.*

## Guía General de Resolución de Problemas de Indecidibilidad

En esta sección intentaremos dar un esquema general para la resolución de problemas sobre indecidibilidad de lenguajes. **NO ES LA ÚNICA MANERA DE HACERLO, PERO NOS PUEDE AYUDAR SI NO SABEMOS POR DÓNDE EMPEZAR**.

Supongamos que nos piden que clasifiquemos un lenguaje $L$ como decidible, semi-decidible pero no decidible o no semi-decidible.

El esquema general podría seguir los siguientes pasos:

1. Argumentación intuitiva, en el que deberíamos contestar (intuitivamente) a las siguientes preguntas:

   (Ls) ¿Existe un algoritmo para detectar cuando una entrada pertenece a $L$? ¿Por qué? En caso afirmativo, ¿cómo sería?

   (Ln) ¿Existe un algoritmo para detectar cuando una entrada pertenece al complementario de $L$? ¿Por qué? En caso afirmativo, ¿cómo sería?

2. Razonamiento formal, en el que, dependiendo de las respuestas a las preguntas anteriores, buscaremos la manera de justificarlo formalmente:

   (a) Si las respuestas a (Ls) y (Ln) son las dos afirmativas, entonces nuestra intuición nos dice que el lenguaje $L$ es decidible, y para demostrarlo deberíamos intentar construir una TM de parada segura que lo reconozca (a partir de las respuestas a "En caso afirmativo, ¿cómo sería?")

   (b) Si la respuesta a (Ls) es afirmativa pero la respuesta a (Ln) es negativa, entonces nuestra intuición nos dice que el lenguaje $L$ es semi-decidible pero no decidible, y para demostrarlo deberíamos intentar

      i. Construir una TM (que no será de parada segura) para reconocer $L$ y demostrar así que es semi-decidible

      ii. Construir una reducción desde un lenguaje indecidible (por ejemplo $\mathtt{K} \leqslant_m L$) para demostrar que no es decidible o usar el teorema de Rice

   (c) Si la respuesta a (Ls) es negativa, entonces nuestra intuición nos dice que el lenguaje $L$ no es semi-decidible, y para demostrarlo deberíamos intentar construir una reducción desde un lenguaje no semi-decidible (por ejemplo $\overline{\mathtt{K}} \leqslant_m L$) para demostrar que no es semi-decidible o usar el teorema de Rice

Siguiendo con un ejemplo conocido, supongamos que nos piden que clasifiquemos el lenguaje

$$\texttt{NONEMPTY} = \{x \mid \exists y \; M_x(y) \downarrow\}$$

como decidible, semi-decidible pero no decidible o no semi-decidible.

El esquema general aquí se aplicaría así:

1. Argumentación intuitiva

   (Ls) ¿Existe un algoritmo para detectar cuando una entrada pertenece a NONEMPTY? ¿Por qué? En caso afirmativo, ¿cómo sería?
   Creo que sí que es posible, porque podemos buscar entre todas las posibles entradas $y$ si alguna para (de manera controlada, simulando $M_x$ por pasos), y en ese caso aceptar

   (Ln) ¿Existe un algoritmo para detectar cuando una entrada pertenece al complementario de NONEMPTY? ¿Por qué? En caso afirmativo, ¿cómo sería?
   Creo que no es posible, porque si no existe ninguna entrada $y$ para la que $M_x$ pare no podremos encontrarla a base de simulaciones de $M_x$ (precisamente porque no para) y no parece que haya ninguna otra propiedad del lenguaje (que sea finito, por ejemplo) que permita detectarlo sin tener que hacer simulaciones de $M_x$

2. Razonamiento formal, en el que intentaremos demostrar que NONEMPTY es semi-decidible pero no decidible

   (a) Demostración de que NONEMPTY es semi-decidible, construyendo una TM que reconoce NONEMPTY:

   *Input* $x$
   $t = 0$
   found = *false*
   *while* not found
   $t = t + 1$
   *for* $y \in \{1, \ldots, t\}$
   *Simulate* $t$ *steps of* $M_x(y)$
   *If* "la simulación acaba en estado final" *then*
   found = *true*
   *accept*

   **Ejercicio**: Justificar que la TM anterior reconoce NONEMPTY

   (b) Demostración de que NONEMPTY es indecidible, construyendo una reducción desde K como ya habíamos visto, $f(x) = g\ddot{o}del\,(R[x])$, donde $R[x]$ es la familia de TM definida como:

   *Input* $y$
   *Simulate* $M_x(x)$
   *stop*

   **Ejercicio**: Justificar nuevamente que $f(x)$ es una reducción válida para $\texttt{K} \leqslant_m \texttt{NONEMPTY}$

   **Ejercicio**: Demostrar que NONEMPTY es indecidible usando el teorema de Rice