Modelos de Computación y Complejidad

Grado en Ingeniería Informática. Tecnologías Informáticas

Tema 5: Nociones básicas de Teoría de la Complejidad Computacional

David Orellana Martín
Mario de J. Pérez Jiménez

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática
Universidad de Sevilla

dorellana@us.es
marper@us.es
http://www.cs.us.es/~marper/

Curso 2021-2022

Objetivo general de la **Teoría de la Computabilidad**:

- ⋆ Dado un problema de decisión, determinar si existe un **procedimiento mecánico** que lo resuelve.

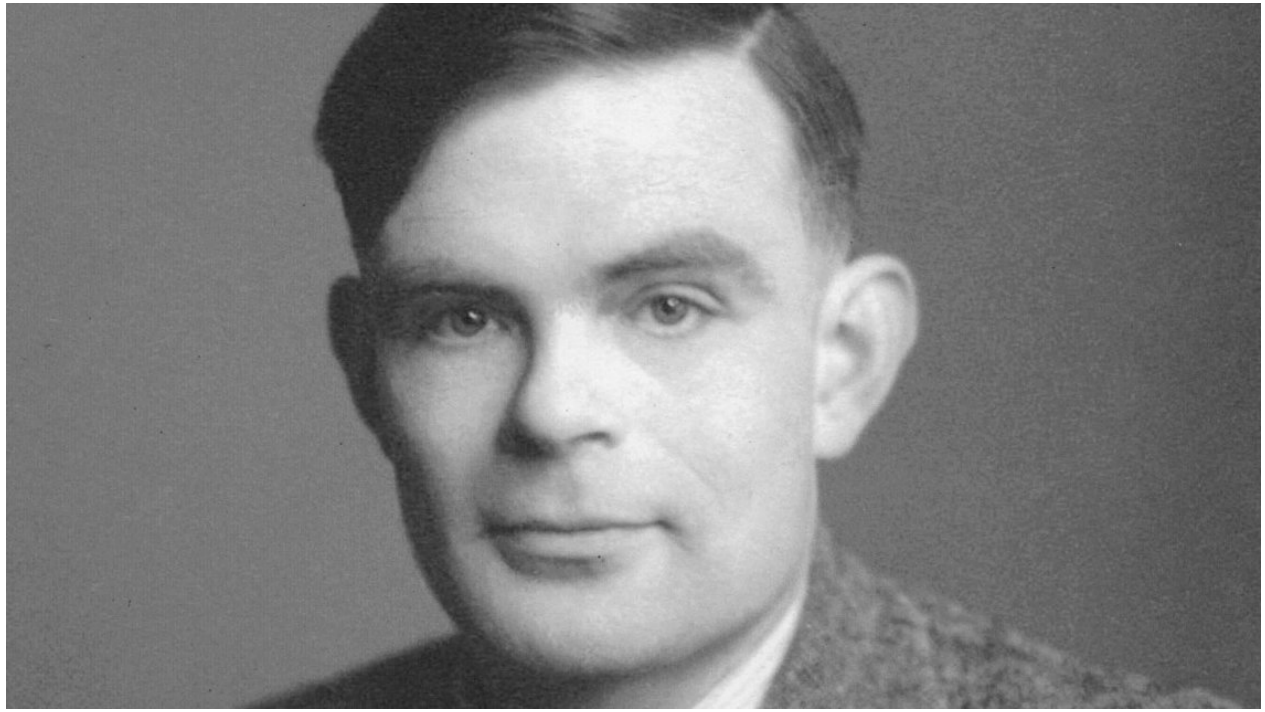
Objetivo general de la **Teoría de la Complejidad Computacional**:

- ⋆ Dado un problema de decisión, determinar si existe un **procedimiento mecánico** que lo resuelve **de manera eficiente**.

Objetivos específicos de este tema:

- ∗ **Máquinas de Turing (deterministas y no deterministas)**.
- ∗ **Medidas abstractas** de Complejidad.
- ∗ **Resolubilidad algorítmica** de problemas de decisión.
- ∗ **Reducibilidad en tiempo polinomial**.

Alan Turing (1912-1954)



- ⋆ Nació el 23 de junio de 1912 en una zona residencial de Londres (Maida Vale).
- ⋆ Pionero en la introducción de modelos de computación (mayo de 1936).
- ⋆ Tesis doctoral: **Systems of logic based on ordinals** (mayo de 1938, dirigida por A. Church).

- ⋆ Al día siguiente de la entrada de Gran Bretaña en la Segunda Guerra Mundial, fue "reclutado" en Bletchley Park (septiembre de 1939), donde se encontraba el Servicio Británico de Descifrado.
- ⋆ Papel decisivo en esa guerra, descifrando información codificada por los alemanes (para transmitir órdenes a sus naves que operaban en el Atlántico).
- ⋆ Los alemanes codificaban la información usando la máquina **Enigma**.

- ⋆ A finales de 1939, A. Turing diseñó una máquina **electromecánica** de propósito específico: la máquina **Bombe** (mejorando una máquina desarrollada en 1938 por investigadores polacos).

- ⋆ Se estima que la aportación de A. Turing propició que la guerra finalizara unos **dos años antes**, evitando la muerte de varios millones de personas (algunos expertos lo estiman en unos 14 millones).

En 1950, A. Turing formuló una pregunta muy interesante: **Can machines think?**[1]

Para responder a esta cuestión formuló ésta otra pregunta **Are there imaginable digital computers which would do well in the imitation game**?

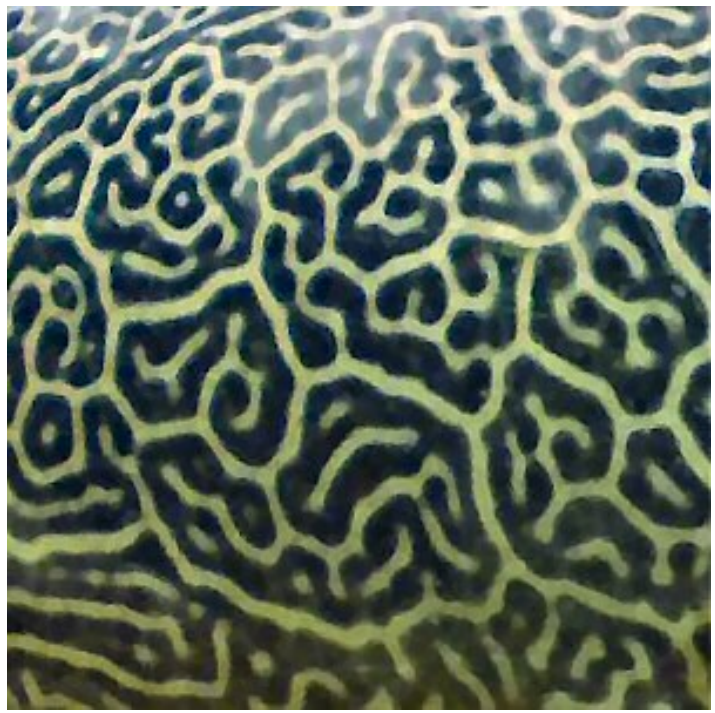
- ⋆ Conocido como **test de Turing**.
- ⋆ **Eugene Goostman** (desarrollado en 2001) una aplicación software "singular":

  - ∗ En 1950, A. Turing afirmó que en el año 2000 existirían máquinas capaces de mantener una conversación con una persona durante 5 m. usando mensajes de texto, sin que ésta supiera, con una probabilidad del 30%, si su interlocutor era o no una máquina.
  - ∗ Confundió a 45 de 155 personas encargadas de dictaminar si el "interlocutor" era una máquina.
  - ∗ Se considera la primera "máquina" que superó el test de Turing (en un polémico evento celebrado en 2014 por K. Warwick).

[1] A.M. Turing. Computing Machinery and Intelligence, **Mind**, Vol. **LIX**, Issue 236, October 1950, pp 433-460.

A. Turing es el pionero de una de las disciplinas científicas más relevante en la actualidad: la **Inteligencia Artificial**.

A partir de 1952, A.Turing se centró en el estudio de la **teoría de patrones**, aplicada al mecanismo de desarrollo de la forma[2] (**morfogénesis**).



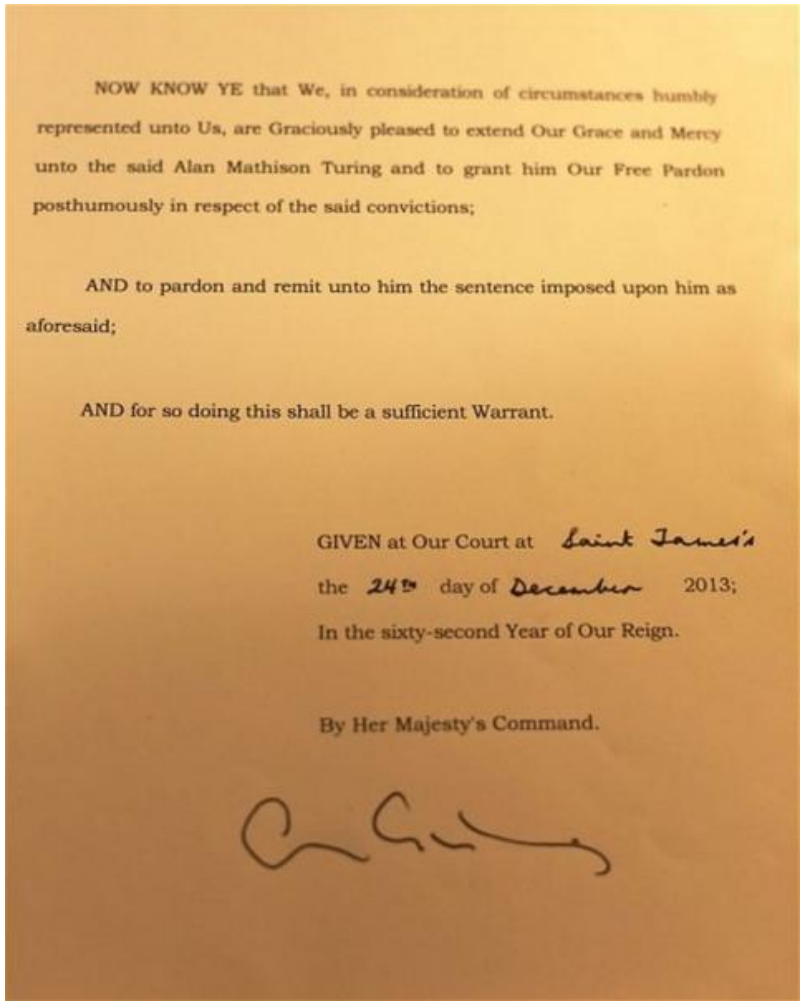
A. Turing conjeturó la existencia de **patrones regulares** en el sistema biológico animal, a la hora de implementar la morfogénesis.

[2] A. Turing. The Chemical Basis of Morphogenesis. **Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences**, Vol. **237**, No 641. (Aug. 14, 1952), pp. 37-72.

A. Turing era homosexual, fue juzgado por ello, y condenado por **gross indecency** (1952). Aceptó la castración química para evitar la entrada en prisión.

El **final**: 7 de junio de 1954 (presunto suicidio, al haber mordido una manzana rociada de cianuro).

Su rehabilitación pública (algo tardía) se produjo el día de Nochebuena de 2013.



NOW KNOW YE that We, in consideration of circumstances humbly represented unto Us, are Graciously pleased to extend Our Grace and Mercy unto the said Alan Mathison Turing and to grant him Our Free Pardon posthumously in respect of the said convictions;

AND to pardon and remit unto him the sentence imposed upon him as aforesaid;

AND for so doing this shall be a sufficient Warrant.

GIVEN at Our Court at Saint James's
the 24th day of December 2013;
In the sixty-second Year of Our Reign.

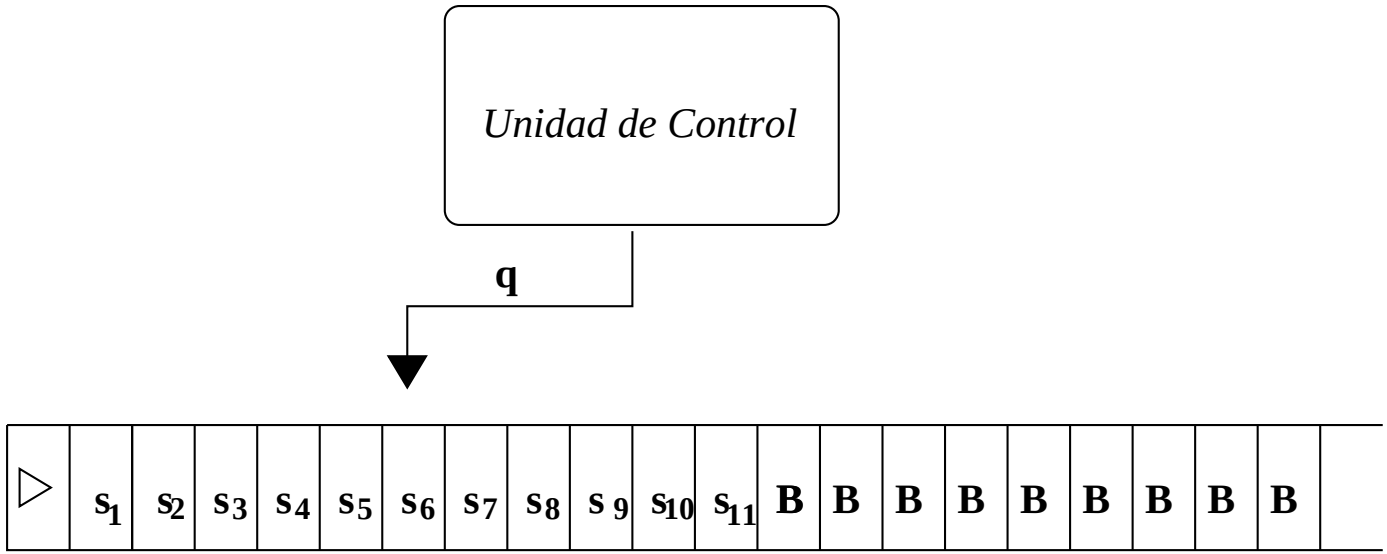
By Her Majesty's Command.

Máquinas de Turing deterministas: Sintaxis

Es una tupla, $M = (Q, q_0, F, \Sigma, B, \triangleright, \delta)$, en donde:

- Q es un alfabeto finito (**estados**).
- $q_0 \in Q$ (**estado inicial**).
- $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**).
- Σ es un alfabeto finito (*de la máquina*) tal que $\Sigma \cap Q = \emptyset$ (**símbolos**).
- $B \in \Sigma$ (**símbolo blanco**).
- $\triangleright \in \Sigma$ (**primer símbolo**), $\triangleright \neq B$.
- δ es una aplicación de $(Q - F) \times \Sigma$ en $Q \times \Sigma \times \{0, 1, -1\}$ (**función de transición**) tal que:
  - ⋆ Si $\delta(q, \triangleright) = (q', s', x)$ entonces $s' = \triangleright \wedge x = 1$.

Informalmente, una máquina de Turing determinista M consta de:

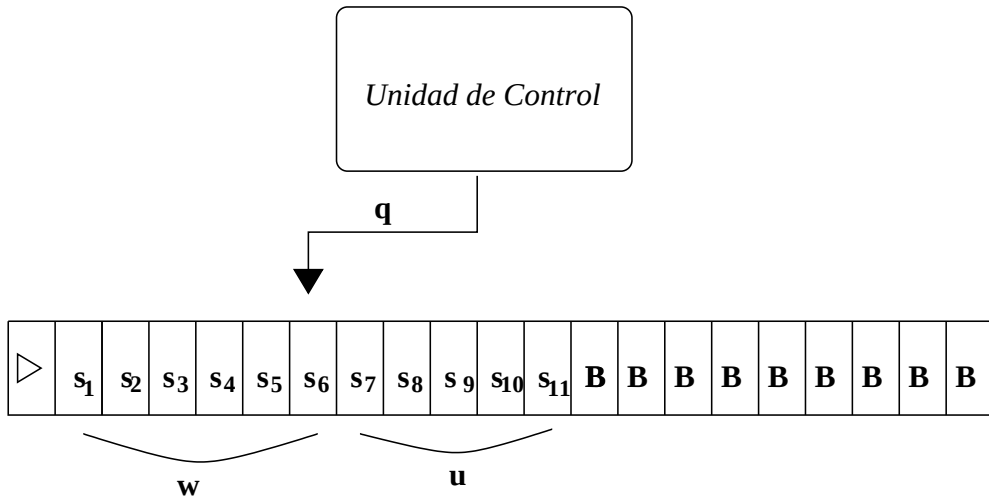


- ⋆ Una **cinta** con infinitas **casillas**/**celdas**, una de las cuales es la "primera".
- ⋆ Un **cabezal lector/escritor** que se puede desplazar a lo largo de la cinta.
- ⋆ En cada instante, la máquina se encuentra en un **estado**.

Si $\delta(q, s) = (q', s', x)$, diremos que M pasa de q a q', sustituye s por s' y **se mueve** según el valor de x.

Máquinas de Turing deterministas: Semántica

Configuración de M en un instante t: (q, w, u), siendo $q \in Q$, $w \in \Sigma^*$ y $u \in \Sigma^*$.



- Interpretación:
  - q es el estado actual en que se encuentra M.
  - $\triangleright wu$ es la cadena **escrita** en la cinta.
  - M *analiza* el último símbolo de la cadena $\triangleright w$.

Máquinas de Turing deterministas: Semántica

Sean M una MTD y (q, w, u) una configuración de M en un instante t.

- ⋆ Configuración de **parada**: $q \in F$.
- ⋆ Configuración de **aceptación**: $q = q_y$.
- ⋆ Configuración de **rechazo**: $q = q_n$.

Diremos que (q, w, u) es una **configuración inicial** de M **sii**

- El estado es q_0;
- El contenido de la cinta es del tipo $\triangleright uBBBBB\ldots$, siendo
  - ∗ $u \in (\Sigma - \{B\})^*$: cadena de entrada (I_u).
- El cabezal lector/escritor analiza la primera casilla.

Paso de transición/computación en una MTD

La función de transición δ de una MTD, M, permite "pasar" de un configuración en un instante t a la **configuración siguiente** en el instante $t+1$.

Sea (q, w, u) una configuración de M en un instante t.

Veamos cómo se calcula la configuración siguiente (q', w', u'):

- ⋆ Si $\delta(q, w_r) = (q', w'_r, -1)$ entonces:

$$(q, w_1 \dots w_{r-1} w_r, u_1 \dots u_s) \vdash_M (q', w_1 \dots w_{r-1}, w'_r u_1 \dots u_s)$$

- ⋆ Si $\delta(q, w_r) = (q', w'_r, +1)$ entonces

$$(q, w_1 \dots w_{r-1} w_r, u_1 \dots u_s) \vdash_M (q', w_1 \dots w_{r-1} w'_r u_1, u_2 \dots u_s)$$

- ⋆ Si $\delta(q, w_r) = (q', w'_r, 0)$ entonces

$$(q, w_1 \dots w_{r-1} w_r, u_1 \dots u_s) \vdash_M (q', w_1 \dots w_{r-1} w'_r, u_1 \dots u_s)$$

Computaciones en una MTD

Computación de una MTD con entrada $u \in (\Sigma - \{B\})^*$:

* Sucesión (finita o infinita) de configuraciones en donde I_u es el primer término y los restantes se obtienen del anterior mediante un paso de computación.
* Si la sucesión es finita (M llega a un estado final), la última configuración es de parada (M **para sobre** u: $M \downarrow u$)).
  * Si el estado es q_y: M **acepta** el dato u (escribiremos $M(u) = Y$).
  * Si el estado es q_n: M **rechaza** e dato u (escribiremos $M(u) = N$).
  * Si el estado es q_h: $M(u) = u' \in (\Sigma - \{B\})^*$, siendo $\triangleright u'BBBB...$ el contenido final de la cinta.
* Si la sucesión es infinita, diremos que M **no** *para* **sobre** u: $M \uparrow u$.

Nota interesante: Obsérvese que cualquier computación de una MTD ejecuta, al menos, **un paso de transición**.

MTD de decisión y de cálculo de funciones

MTD, M, que decide un lenguaje L sobre el alfabeto $\Sigma - \{B\}$ si se verifica:

- $\star$ Si $u \in L$, entonces $M(u) = Y$ (el estado de la configuracion de parada es q_y).
- $\star$ Si $u \notin L$, entonces $M(u) = N$ (el estado de la configuracion de parada es q_n).

MTD, M, que calcula una función parcial f de $(\Sigma - \{B\})^*$ en Σ^*: se verifica

- $\star$ $M \downarrow u$ sii $f(u) \downarrow$, para cada $u \in (\Sigma - \{B\})^*$.
- $\star$ Si $M \downarrow u$, entonces $M(u) = f(u)$, para cada $u \in (\Sigma - \{B\})^*$.

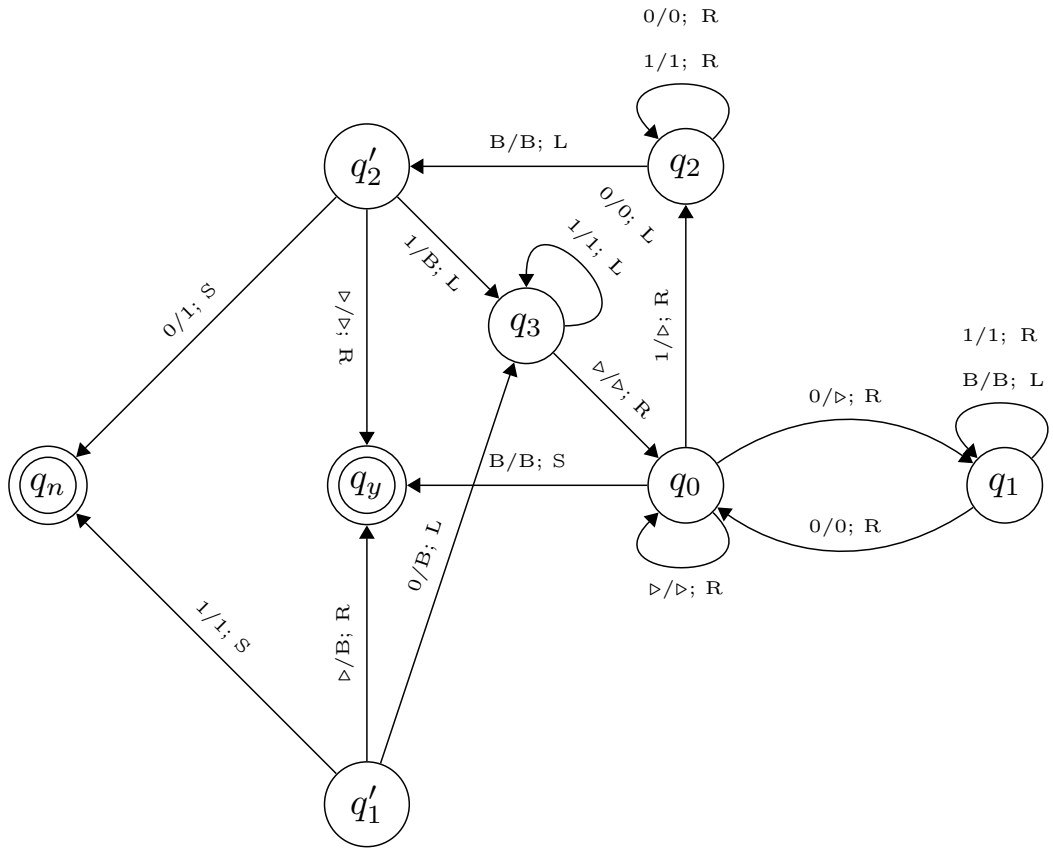
Una MTD de decisión

$Q = \{q_0, q_1, q_2, q_1', q_2', q_3, \mathsf{q_y}, \mathsf{q_n}\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_1, \triangleright, 1)$
$(q_0, 1)$	$(q_2, \triangleright, 1)$
(q_0, B)	$(\mathsf{q_y}, B, 0)$
$(q_1, 0)$	$(q_1, 0, 1)$
$(q_1, 1)$	$(q_1, 1, 1)$
(q_1, B)	$(q_1', B, -1)$
$(q_2, 0)$	$(q_2, 0, 1)$
$(q_2, 1)$	$(q_2, 1, 1)$
(q_2, B)	$(q_2', B, -1)$
$(q_1', \triangleright)$	$(\mathsf{q_y}, B, 1)$
$(q_1', 0)$	$(q_3, B, -1)$
$(q_1', 1)$	$(\mathsf{q_n}, 1, 0)$
$(q_2', \triangleright)$	$(\mathsf{q_y}, \triangleright, 1)$
$(q_2', 0)$	$(\mathsf{q_n}, 1, 0)$
$(q_2', 1)$	$(q_3, B, -1)$
$(q_3, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_3, 0)$	$(q_3, 0, -1)$
$(q_3, 1)$	$(q_3, 1, -1)$

Una MTD de decisión

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_1, \triangleright, 1)$
$(q_0, 1)$	$(q_2, \triangleright, 1)$
(q_0, B)	$(q_y, B, 0)$
$(q_1, 0)$	$(q_1, 0, 1)$
$(q_1, 1)$	$(q_1, 1, 1)$
(q_1, B)	$(q_1', B, -1)$
$(q_2, 0)$	$(q_2, 0, 1)$
$(q_2, 1)$	$(q_2, 1, 1)$
(q_2, B)	$(q_2', B, -1)$
$(q_1', \triangleright)$	$(q_y, B, 1)$
$(q_1', 0)$	$(q_3, B, -1)$
$(q_1', 1)$	$(q_n, 1, 0)$
$(q_2', \triangleright)$	$(q_y, \triangleright, 1)$
$(q_2', 0)$	$(q_n, 1, 0)$
$(q_2', 1)$	$(q_3, B, -1)$
$(q_3, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_3, 0)$	$(q_3, 0, -1)$
$(q_3, 1)$	$(q_3, 1, -1)$



$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_1, \triangleright, 1)$
$(q_0, 1)$	$(q_2, \triangleright, 1)$
(q_0, B)	$(q_y, B, 0)$
$(q_1, 0)$	$(q_1, 0, 1)$
$(q_1, 1)$	$(q_1, 1, 1)$
(q_1, B)	$(q_1', B, -1)$
$(q_2, 0)$	$(q_2, 0, 1)$
$(q_2, 1)$	$(q_2, 1, 1)$
(q_2, B)	$(q_2', B, -1)$
$(q_1', \triangleright)$	$(q_y, B, 1)$
$(q_1', 0)$	$(q_3, B, -1)$
$(q_1', 1)$	$(q_n, 1, 0)$
$(q_2', \triangleright)$	$(q_y, \triangleright, 1)$
$(q_2', 0)$	$(q_n, 1, 0)$
$(q_2', 1)$	$(q_3, B, -1)$
$(q_3, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_3, 0)$	$(q_3, 0, -1)$
$(q_3, 1)$	$(q_3, 1, -1)$

Ejecutemos dicha MTD con dato de entrada **1 0**.

1. $\overset{q_0}{\triangleright}\ 1\ 0\ B$
2. $\triangleright\ \overset{q_0}{1}\ 0\ B$
3. $\triangleright\ \triangleright\ \overset{q_2}{0}\ B$
4. $\triangleright\ \triangleright\ 0\ \overset{q_2}{B}$
5. $\triangleright\ \triangleright\ \overset{q_2'}{0}\ B$
6. $\triangleright\ \triangleright\ \overset{q_n}{1}\ B$

Por tanto, la computación **M(1 0)** es de parada y devuelve **NO**.

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_1, \triangleright, 1)$
$(q_0, 1)$	$(q_2, \triangleright, 1)$
(q_0, B)	$(q_y, B, 0)$
$(q_1, 0)$	$(q_1, 0, 1)$
$(q_1, 1)$	$(q_1, 1, 1)$
(q_1, B)	$(q_1', B, -1)$
$(q_2, 0)$	$(q_2, 0, 1)$
$(q_2, 1)$	$(q_2, 1, 1)$
(q_2, B)	$(q_2', B, -1)$
$(q_1', \triangleright)$	$(q_y, B, 1)$
$(q_1', 0)$	$(q_3, B, -1)$
$(q_1', 1)$	$(q_n, 1, 0)$
$(q_2', \triangleright)$	$(q_y, \triangleright, 1)$
$(q_2', 0)$	$(q_n, 1, 0)$
$(q_2', 1)$	$(q_3, B, -1)$
$(q_3, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_3, 0)$	$(q_3, 0, -1)$
$(q_3, 1)$	$(q_3, 1, -1)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\triangleright\ \overset{q_0}{0}\ 1\ 0\ B$
2. $\triangleright\ 0\ \overset{q_0}{1}\ 0\ B$
3. $\triangleright\ \triangleright\ \overset{q_1}{1}\ 0\ B$
4. $\triangleright\ \triangleright\ 1\ \overset{q_1}{0}\ B$
5. $\triangleright\ \triangleright\ 1\ 0\ \overset{q_1}{B}$
6. $\triangleright\ \triangleright\ 1\ \overset{q_1'}{0}\ B$
7. $\triangleright\ \triangleright\ \overset{q_3}{1}\ B\ B$
8. $\triangleright\ \triangleright\ 1\ B\ B$
9. $\triangleright\ \triangleright\ \overset{q_0}{1}\ B\ B$
10. $\triangleright\ \triangleright\ \triangleright\ \overset{q_2}{B}\ B$
11. $\triangleright\ \triangleright\ \overset{q_2'}{\triangleright}\ B\ B$
12. $\triangleright\ \triangleright\ \triangleright\ \overset{q_y}{B}\ B$

Por tanto, la computación M(010) es de parada y devuelve **SÍ**.

Ejemplos de MTDs que deciden lenguajes

Sea $\Gamma = \{a\}$ un alfabeto.

Ejemplo 1: El lenguaje $L_1 = \{\epsilon\}$ cuyo único elemento es la cadena vacía.

La siguiente MTD **decide** el lenguaje L_1: $Q = \{q_0, q_y, q_n\}$; $\Sigma = \{a, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
(q_0, a)	$(q_n, a, 0)$
(q_0, B)	$(q_y, B, 0)$

Ejemplo 2: El lenguaje $L_2 = \emptyset$ (**lenguaje vacío**).

La siguiente MTD **decide** el lenguaje L_2: $Q = \{q_0, q_y, q_n\}$; $\Sigma = \{a, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_n, \triangleright, 1)$

Ejemplos de MTDs que deciden lenguajes

Ejemplo 3: El lenguaje $L_3 = \{u \in \{0,1\}^* \ : \ |u| = 4\}$.

La siguiente MTD **decide** el lenguaje L_3:

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_y, q_n\}; \ \Sigma = \{a, B, \triangleright\}$$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
(q_0, B)	$(q_n, B, 0)$
$(q_0, 0)$	$(q_1, 0, 1)$
$(q_0, 1)$	$(q_1, 1, 1)$
(q_1, B)	$(q_n, B, 0)$
$(q_1, 0)$	$(q_2, 0, 1)$
$(q_1, 1)$	$(q_2, 1, 1)$
(q_2, B)	$(q_n, B, 0)$
$(q_2, 0)$	$(q_3, 0, 1)$
$(q_2, 1)$	$(q_3, 1, 1)$
(q_3, B)	$(q_n, B, 0)$
$(q_3, 0)$	$(q_4, 0, 1)$
$(q_3, 1)$	$(q_4, 1, 1)$
(q_4, B)	$(q_y, B, 0)$
$(q_4, 0)$	$(q_n, B, 0)$
$(q_4, 1)$	$(q_n, B, 0)$

Una MTD que calcula la función vacía

$Q = \{q_0, q_1, q_h\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_1, \triangleright, 1)$
$(q_1, 0)$	$(q_0, 0, -1)$
$(q_1, 1)$	$(q_0, 1, -1)$
(q_1, B)	$(q_0, B, -1)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\overset{q_0}{\triangleright}\ 0\ 1\ 0\ B$
2. $\triangleright\ \overset{q_1}{0}\ 1\ 0\ B$
3. $\overset{q_0}{\triangleright}\ 0\ 1\ 0\ B$
4. $\triangleright\ \overset{q_1}{0}\ 1\ 0\ B$

Y así, se "entra" en un *bucle infinito*.

Por tanto, la computación M(010) **no** es de **parada**.

Una MTD que calcula la función identidad en $\mathbb{N}$

$Q = \{q_0, q_h\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_h, \triangleright, 1)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\overset{q_0}{\triangleright}$ 0 1 0 B
2. $\triangleright$ $\overset{q_h}{0}$ 1 0 B

Por tanto, la computación **M(0 1 0) es de parada** y **devuelve 0 1 0**.

Una MTD que calcula la función constante igual a 1

$Q = \{q_0, q_h\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 0, 1)$
(q_0, B)	$(q_h, 1, 0)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\overset{q_0}{\triangleright}\ 0\ 1\ 0\ B\ B$
2. $\triangleright\ \overset{q_0}{0}\ 1\ 0\ B\ B$
3. $\triangleright\ 0\ \overset{q_0}{0}\ 0\ B\ B$
4. $\triangleright\ 0\ 0\ \overset{q_0}{0}\ B\ B$
5. $\triangleright\ 0\ 0\ 0\ \overset{q_0}{B}\ B$
6. $\triangleright\ 0\ 0\ 0\ \overset{q_h}{1}\ B$

Por tanto, la computación M(0 1 0) **es de parada** y **devuelve 1**.

Una MTD que calcula el producto por 2 de un número en binario

El caso $x = 0$ es trivial. Sea $x = a_n \cdot 2^n + \ldots a_1 \cdot 2^1 + a_0 \cdot 2^0$, con $a_n \geq 1$. Entonces $2 \cdot x = a_n \cdot 2^{n+1} + \ldots a_1 \cdot 2^2 + a_0 \cdot 2^1$. Un esquema algorítmico para hallar $2 \cdot x$:

`Entrada:` un número natural $x = a_n \ldots a_1 a_0$ en binario

`devolver` $a_n \ldots a_1\, a_0\, 0$

Consideremos la siguiente MTD, M: $Q = \{q_0, q_h\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 1, 1)$
(q_0, B)	$(q_h, 0, 0)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\triangleright\ \overset{q_0}{0}\ 1\ 0\ B\ B$
2. $\triangleright\ 0\ \overset{q_0}{1}\ 0\ B\ B$
3. $\triangleright\ 0\ 1\ \overset{q_0}{0}\ B\ B$
4. $\triangleright\ 0\ 1\ 0\ \overset{q_0}{B}\ B$
5. $\triangleright\ 0\ 1\ 0\ B\ \overset{q_h}{B}$
6. $\triangleright\ 0\ 1\ 0\ \overset{q_h}{B}\ B$

Por tanto, la computación **M(0 1 0)** es de parada y devuelve **0 1 0 0**.

Una MTD que calcula la división entera por 2 de un número en binario

El caso $x = 0$ es trivial. Sea $x = a_n \cdot 2^n + \ldots a_1 \cdot 2^1 + a_0 \cdot 2^0$, con $a_n \geq 1$. Entonces $\lfloor \frac{x}{2} \rfloor = a_n \cdot 2^{n-1} + \ldots a_1 \cdot 2^0$. Un esquema algorítmico para hallar $\lfloor \frac{x}{2} \rfloor$:

`Entrada:` un número natural $x = a_n \ldots a_1\, a_0$ en binario

`devolver` $a_n \ldots a_1$

Consideremos la siguiente MTD, M: $Q = \{q_0, q_1, q_h\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 1, 1)$
(q_0, B)	$(q_1, B, -1)$
$(q_1, 0)$	$(q_h, B, 0)$
$(q_1, 1)$	$(q_h, B, 0)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\overset{q_0}{\triangleright}\ 0\ 1\ 0\ B$
2. $\triangleright\ \overset{q_0}{0}\ 1\ 0\ B$
3. $\triangleright\ 0\ \overset{q_0}{1}\ 0\ B$
4. $\triangleright\ 0\ 1\ \overset{q_0}{0}\ B$
5. $\triangleright\ 0\ 1\ 0\ \overset{q_0}{B}$
6. $\triangleright\ 0\ 1\ \overset{q_1}{0}\ B$
7. $\triangleright\ 0\ 1\ \overset{q_h}{B}\ B$

Por tanto, la computación **M(0 1 0)** es de parada y devuelve **0 1**.

Un esquema algorítmico para calcular el sucesor de un número en binario

El caso $x = 0$ es trivial. Sea $x = a_n \cdot 2^n + \ldots a_1 \cdot 2^1 + a_0 \cdot 2^0$, con $a_n \geq 1$. Entonces $x + 1 = a_n \cdot 2^{n-1} + \ldots a_1 \cdot 2^0 + 1$.

Un esquema algorítmico para hallar $x + 1$:

```
Entrada: un número natural x = a_n ... a_1 a_0 en binario
    si a_n = ... = a_1 = a_0 = 1 entonces
        b_{n+1} ⟵ 1
        para j = 0 hasta n hacer
            b_j ⟵ 0
        devolver b_{n+1} b_n ... b_1 b_0
    si no
        k ⟵ mín {j | 0 ≤ j ≤ n ∧ a_j = 0}
        b_k ⟵ 1
        para j = 0 hasta k − 1 hacer
            b_j ⟵ 0
        para j = k + 1 hasta n hacer
            b_j ⟵ a_j
        devolver b_n ... b_1 b_0
```

MTD's que resuelven problemas

¿Cómo **resolver un problema** X, mediante una MTD?

- Si X es un problema de decisión,
  - Una **MTD resuelve** X sii *decide* el lenguaje L_X asociado a X.
- Si X es un problema abstracto identificado por una aplicación f_X cuyo dominio es del conjunto de instancias:
  - Una **MTD resuelve** X sii *calcula* la función f_X.

Resolviendo problemas de decisión mediante MTDs

Problema de decisión: Sea $k \in \mathbb{N}, k \geq 2$. Para cada $x \in \mathbb{N}$, determinar si x es múltiplo de k.

Esquema algorítmico 1 para resolver el problema:

```
Entrada: un número natural x ∈ ℕ (en el sistema decimal)
         hallar la división euclidea de x entre k
           sea r el resto de esa división
         si r = 0 entonces
           devolver sí
         si no
           devolver no
```

¿Qué ingredientes se necesita para diseñar una MTD que "implemente" este esquema?

Resolviendo problemas de decisión mediante MTDs

Problema de decisión: Sea $k \in \mathbb{N}, k \geq 2$. Para cada $x \in \mathbb{N}$, determinar si x es múltiplo de k.

Observación importante:

* El caso $x = 0$ tiene una solución trivial.
* Si $x = a_n \cdot k^n + \ldots + a_1 \cdot k^1 + a_0 \cdot k^0 \geq 1$ está expresado en base k (recuérdese que $a_j \in \{0, \ldots, k-1\}$ para $0 \leq j \leq n$) , entonces x es múltiplo de k **sii** $a_0 = 0$.

Esquema algorítmico 2 para resolver el problema:

```
Entrada: x = a_n · k^n + ... + a_1 · k^1 + a_0 · k^0 ≥ 1 expresado en base k
         si a_0 = 0 entonces
           devolver sí
         si no
           devolver no
```

Resolviendo problemas de decisión mediante MTDs

Problema de decisión: Sea $k \in \mathbb{N}, k \geq 2$. Para cada $x \in \mathbb{N}$, determinar si x es múltiplo de k.

Diseño de una MTD que "implementa" el esquema algorítmico 2 y resuelve el problema (la entrada es una cadena que representa el número $x \geq 1$ expresado en base k):

Sean $Q = \{q_0, q_h\}$; $\Sigma = \{0, 1, \ldots, k-1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 1, 1)$
$\ldots$	$\ldots$
$(q_0, k-1)$	$(q_0, k-1, 1)$
(q_0, B)	$(q_1, B, -1)$
$(q_1, 0)$	$(q_y, 0, 0)$
$(q_1, 1)$	$(q_n, 1, 0)$
$\ldots$	$\ldots$
$(q_1, k-1)$	$(q_n, k-1, 0)$

Resolviendo problemas de decisión mediante MTDs

Problema de decisión: Para cada número natural $x \geq 1$, representado en el sistema binario, determinar si es un múltiplo de 4.

Obsérvese que un número natural $x = a_n \cdot 2^n + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 \geq 1$, en **binario**, es un **múltiplo de 4 sii** los dos "ultimos" dígitos (a_0, a_1) son iguales a 0.

- Si $x = a_n \cdot 2^n + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 \geq 1$ es múltiplo de 4, entonces 4 ha de dividir a $a_1 \cdot 2^1 + a_0 \cdot 2^0$. Por tanto, $a_0 = a_1 = 0$.
- Si $x = a_n \cdot 2^n + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 \geq 1$, con $a_1 = a_0 = 0$, entonces $x = a_n \cdot 2^n + \ldots + a_2 \cdot 2^2$ y, por tanto, es múltiplo de $2^2 = 4$.

Consideremos la siguiente MTD, M: $Q = \{q_0, q_1, q_y, q_n\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 1, 1)$
(q_0, B)	$(q_1, B, -1)$
$(q_1, 1)$	$(q_n, 0, 0)$
$(q_1, 0)$	$(q_2, 0, -1)$
$(q_2, 0)$	$(q_y, 0, 0)$
$(q_2, 1)$	$(q_n, 0, 0)$

Nota: Si el número x está expresado en **base 4**, entonces la multiplicidad por 4 está caracterizada por la condición $\boxed{a_0 = 0}$.

Resolviendo problemas de decisión mediante MTDs

Problema de decisión: Dado un número natural $x \geq 1$, representado en el sistema binario, determinar si es un número par.

Obsérvese que un número natural $x = a_n \cdot 2^n + \ldots a_1 \cdot 2^1 + a_0 \cdot 2^0 \geq 1$, es **par sii** el dígito de las "unidades" (a_0) es 0.

Consideremos la siguiente MTD, M: $Q = \{q_0, q_1, q_y, q_n\}$; $\Sigma = \{0, 1, B, \triangleright\}$

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$
$(q_0, 0)$	$(q_0, 0, 1)$
$(q_0, 1)$	$(q_0, 1, 1)$
(q_0, B)	$(q_1, B, -1)$
$(q_1, 0)$	$(q_y, 0, 0)$
$(q_1, 1)$	$(q_n, 0, 0)$

Ejecutemos dicha MTD con dato de entrada **0 1 0**.

1. $\triangleright \overset{q_0}{0}\ 1\ 0\ B$
2. $\triangleright\ \overset{q_0}{0}\ 1\ 0\ B$
3. $\triangleright\ 0\ \overset{q_0}{1}\ 0\ B$
4. $\triangleright\ 0\ 1\ \overset{q_0}{0}\ B$
5. $\triangleright\ 0\ 1\ 0\ \overset{q_0}{B}$
6. $\triangleright\ 0\ 1\ \overset{q_1}{0}\ B$
7. $\triangleright\ 0\ 1\ \overset{q_y}{0}\ B$

Por tanto, la computación **M(0 1 0)** es de parada y devuelve **SÍ**.

Potencia computacional de la MTDs

Recuérdese que, de acuerdo con la **tesis de Church-Turing**, el modelo de las MTDs es **universal.**

* Es decir, todo problema abstracto que sea resoluble en un modelo de computación arbitrario, también es resoluble en el modelo de las MTDs.

Además, se puede probar que el modelo de computación `GOTO` tiene la misma potencia computacional que el modelo de las MTDs.

* En consecuencia, el modelo de computación `GOTO` también es **universal**.

Máquinas de Turing deterministas con $k \geq 1$ cintas

Consta de k cintas, infinitas a la derecha con primera casilla/celda.

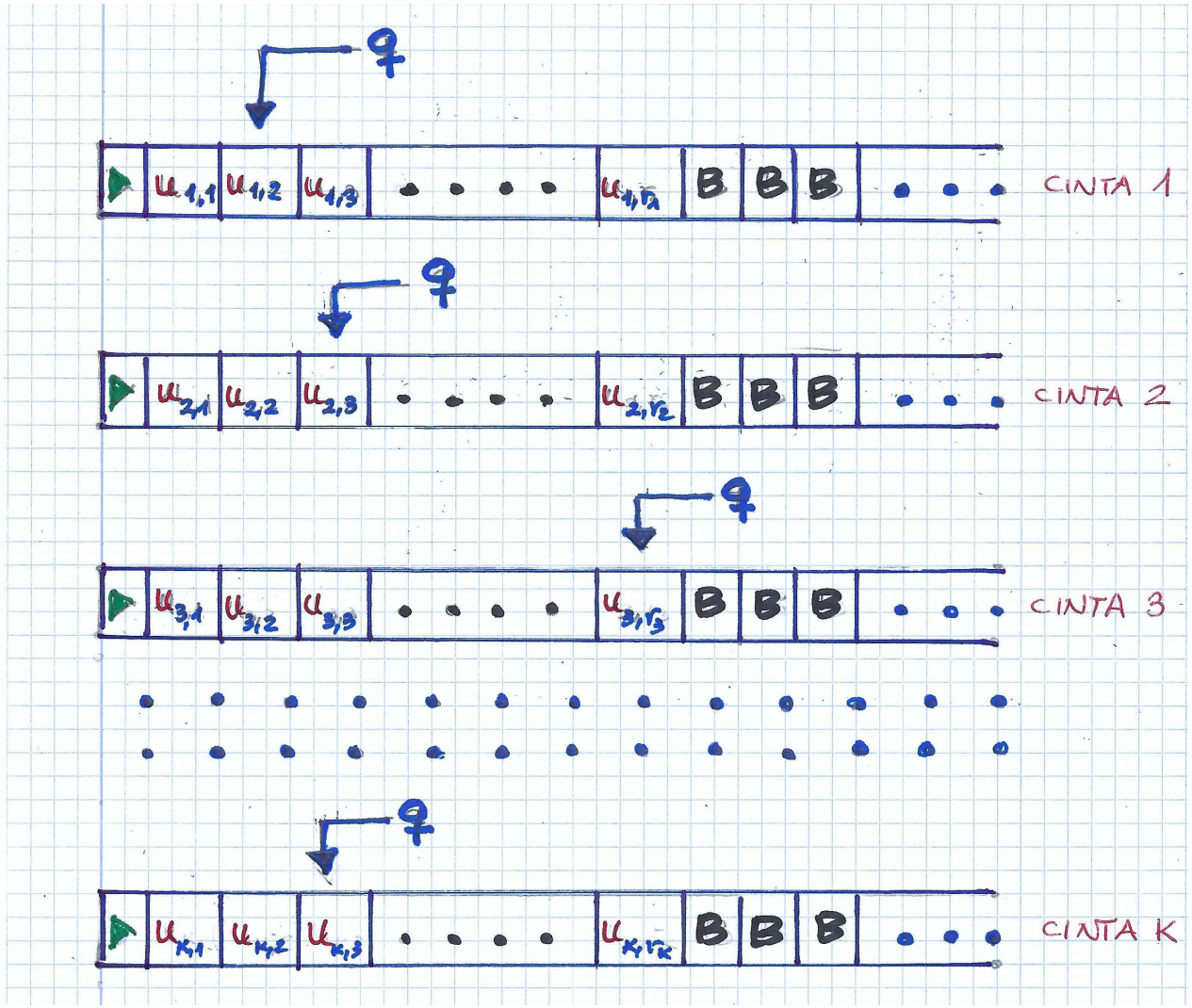
* Las cintas se pueden considerar **enumeradas** del 1 al k.

En cada instante, la máquina está en **un estado**.

Cada cinta tiene un cabezal lector/escritor que, en cada instante:

- Analiza una casilla.
- Puede reescribir sobre la casilla.
- Puede cambiar de estado (el **mismo** en todas las cintas).
  * Todos los cabezales están "**sincronizados**".
- Se puede desplazar: $0, +1, -1$.

Máquina de Turing determinista con k cintas



Máquina de Turing determinista con k cintas

Para realizar una computación con entrada $u \in (\Sigma - \{B\})^*$:

- Se registra la entrada $u \in (\Sigma - \{B\})^*$ en la primera cinta.
- Las restantes cintas están en blanco.
- Inicialmente, todos los cabezales inspeccionan la primera celda.
- Inicialmente, la máquina está en el estado q_0.
- La función de transición **actúa**, simultáneamente, sobre cada cinta.

$$\delta(q, w^{(1)}, \ldots, w^{(k)}) = (q', w'^{(1)}, x^{(1)}, \ldots, w'^{(k)}, x^{(k)})$$

En donde $x^{(1)}, \ldots, x^{(k)} \in \{0, +1, -1\}$.

La **computación** será de **parada** si, a lo largo de las transiciones, la máquina llega a un **estado final** (y el resultado estará codificado en la cinta k-ésima).

Las MTDs con k cintas se pueden considerar como una extensión de las MTDs ordinarias (que tienen una sóla cinta).

Con esa generalización, ¿se ha ganado **potencia computacional**?

Teorema : Para cada MTD, M, con $k \geq 1$ cintas existe una MTD, M', con *una cinta* tal que $M(w) = M'(w)$, para cada entrada w.

¿Qué **peaje se ha de pagar** para "pasar" de una MTD con k cintas a una MTD con una sóla cinta que sea "equivalente" a ella?

Máquinas de Turing no deterministas

Es una tupla, $M = (Q, q_0, F, \Sigma, B, \triangleright, \delta)$, en donde:

- Q es un alfabeto finito (**estados**).
- $q_0 \in Q$ (**estado inicial**).
- $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**).
- Σ es un alfabeto finito (*de la máquina*) tal que $\Sigma \cap Q = \emptyset$ (**símbolos**).
- $B \in \Sigma$ (**símbolo blanco**).
- $\triangleright \in \Sigma$ (**primer símbolo**), $\triangleright \neq B$.
- δ es una aplicación de $(Q - F) \times \Sigma$ en $\mathbf{P}(Q \times \Sigma \times \{0, 1, -1\}) - \{\emptyset\}$ (**función de transición**) tal que:
  - Si $(q', s', x) \in \delta(q, \triangleright)$ entonces $s' = \triangleright \wedge x = 1$.

MTD versus MTND

Es una tupla $M = (Q, q_0, F, \Sigma, B, \triangleright, \delta)$:

Máquinas de Turing deterministas

- $\star$ Q es un alfabeto finito (**estados**).
- $\star$ $q_0 \in Q$ (**estado inicial**).
- $\star$ $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**).
- $\star$ Σ es un alfabeto (*de la máquina*), $\Sigma \cap Q = \emptyset$.
- $\star$ $B \in \Sigma$ (**símbolo blanco**).
- $\star$ $\triangleright \in \Sigma$ (**primer símbolo**), $\triangleright \neq B$.
- $\bullet$ $\delta : (Q - F) \times \Sigma \to Q \times \Sigma \times \{0, 1, -1\}$

Máquinas de Turing no deterministas

- $\star$ Q es un alfabeto finito (**estados**).
- $\star$ $q_0 \in Q$ (**estado inicial**).
- $\star$ $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**).
- $\star$ Σ es un alfabeto (*de la máquina*), $Q \cap \Sigma = \emptyset$.
- $\star$ $B \in \Sigma$ (**símbolo blanco**).
- $\star$ $\triangleright \in \Sigma$ (**primer símbolo**), $\triangleright \neq B$.
- $\bullet$ $\delta : (Q - F) \times \Sigma \to \mathcal{P}(Q \times \Sigma \times \{0, 1, -1\}) - \{\emptyset\}$

Sobre la función de transición en las MTNDs

Supongamos que en una MTND se tiene $\delta(q, s) = \{(q', s', x'), (q'', s'', x'')\}$.

Sea $C_t = (q, w, u)$ una configuración de la MTND tal que **s** es el último símbolo de la cadena **w** (el cabezal está analizando ese símbolo).

* Un paso de computación se obtendría "eligiendo" **(q',s',x')** como "imagen" de $\delta(q, s)$.

  De esta manera obtendríamos **una** configuración siguiente C'_{t+1} de C_t.

* Otro paso de computación se obtendría "eligiendo" **(q",s",x")** como "imagen" de $\delta(q, s)$.

  De esta manera obtendríamos **otra** configuración siguiente C''_{t+1} de C_t

En consecuencia, en este caso la configuración C_t tendria dos **configuraciones siguientes**: C'_{t+1} y C''_{t+1}.

Máquinas de Turing no deterministas

Los conceptos de configuración y computacion en MTNDs se definen de manera similar a como se hizo con las MTDs.

* Una configuración de una MTND puede tener más de una configuración siguiente.

* Dada una MTND, M, y un dato de entrada u pueden existir muchas computaciones de M sobre u: hablaremos del **árbol de computaciones** de $M(u)$.
  - Los nodos de ese árbol son configuraciones.
  - La raíz es la configuración inicial asociada a la cadena u.
  - Cada configuración en el árbol que no sea la inicial, es una siguiente de otra configuración.
  - Cada rama maximal del árbol es una computación.

* Toda MTD es, en particular, una MTND.
* Una MTND de decisión es aquella cuyos estados finales son $\{q_y, q_n\}$.

Resolución de problemas de decisión por MTNDs

MTND, M, que **decide** un lenguaje L sobre el alfabeto $\Sigma - \{B\}$: para cada $u \in (\Sigma - \{B\})^*$ se verifica que

- $u \in L$ si y sólo si **existe, al menos,** una computación de $M(u)$ tal que para y devuelve **YES** (**computación de aceptación**).

Una **MTND resuelve un problema de decisión** X sii decide el lenguaje L_X asociado a X.

- Este concepto de resolubilidad por MTNDs **no** es, propiamente, mecánico.

Teorema : Si una **MTND** decide un lenguaje L entonces existe otra **MTD** con *tres cintas* que también decide L.

¿Qué **peaje se ha de pagar** para "pasar" de una MTND que decide L a una MTD que también decide L?

Medidas abstractas de complejidad

Se trata de formalizar el concepto de **recursos computacionales.**

Al igual que se hizo con los programas GOTO, las MTDs se pueden enumerar a través de unos códigos: $\{M_e : e \in \mathbb{N}\}$.

* $\varphi_e^{(k)}$: función de aridad k calculada por la MTD, M_e, de código e.

Definición: Una **medida de complejidad computacional** es un conjunto $\{f_e : e \in \mathbb{N}\}$ de funciones 1–arias sobre $\mathbb{N}$, tal que:

(a) $\text{dom}(f_e) = \text{dom}(\varphi_e^{(1)})$, para cada $e \in \mathbb{N}$.

(b) El predicado $\theta(e, x, y) \equiv f_e(x) = y$, es computable.

Las condiciones (a) y (b) se denominan **axiomas de Blum**.

Proposición 1: *Si* $\{f_e : e \in \mathbb{N}\}$ *es una medida de complejidad, entonces* <u>todas</u> *las funciones* f_e *son computables.*

* En efecto: veamos que la gráfica de f_e es un conjunto **r.e.**, para cada $e \in \mathbb{N}$.

  Para ello, describamos la correspondiente relación de pertenencia:

$$(x, y) \in G(f_e) \iff f_e(x) = y \iff \theta(e, x, y) = 1$$

  Luego la gráfica de f_e es un conjunto **r.e.** y, por tanto, la función f_e será computable.

■

Proposición 2: *Sea* $\{f_e : e \in \mathbb{N}\}$ *un conjunto de funciones 1–arias sobre* $\mathbb{N}$. *Son equivalentes:*

(a) *El predicado* $\theta(e,x,y) \equiv f_e(x) = y$ *es computable.*

(b) *El predicado* $\theta'(e,x,y) \equiv f_e(x) < y$ *es computable.*

(c) *El predicado* $\theta''(e,x,y) \equiv f_e(x) \leqslant y$ *es computable.*

Demostración:

⋆ $\boxed{(a) \Rightarrow (b)}$ Téngase presente que:

$$\theta'(e,x,y) \equiv f_e(x) < y \equiv \exists z < y\ (f_e(x) = z) \equiv \exists z < y\ (\theta(e,x,z))$$

⋆ $\boxed{(b) \Rightarrow (c)}$ Téngase presente que:

$$\theta''(e,x,y) \equiv f_e(x) \leq y \equiv f_e(x) < y+1 \equiv \theta'(e,x,y+1)$$

⋆ $\boxed{(c) \Rightarrow (a)}$ Téngase presente que:

$$\begin{aligned}\theta(e,x,y) \equiv f_e(x) = y &\equiv [y = 0 \wedge f_e(x) \leq y] \vee [y \neq 0 \ \wedge f_e(x) \leq y \ \wedge \neg\,(f_e(x) \leq y-1)] \\ &\equiv [y = 0 \wedge \theta''(e,x,y)] \vee [y \neq 0 \wedge \theta''(e,x,y) \wedge \neg\,\theta''(e,x,y-1)]\end{aligned}$$

■

El TIEMPO como medida abstracta de complejidad

En el marco de las MTDs se define la medida **tiempo** como sigue:

* Es el conjunto $\{t_e : e \in \mathbb{N}\}$ de funciones 1–arias sobre $\mathbb{N}$ tal que:

$$t_e(x) = \begin{cases} \textbf{número de pasos} \text{ en la computación } M_e(x), & \text{si } M_e(x)\downarrow \\ \uparrow, & \text{si } M_e(x)\uparrow \end{cases}$$

Proposición 3: El **tiempo** es una medida abstracta de complejidad.

Demostración: Veamos que la sucesión $\{t_e : e \in \mathbb{N}\}$ satisface los axiomas de Blum.

El primer axioma se verifica por la propia definición de la sucesión.

Para establecer que se verifica el segundo axioma será suficiente (por la proposición 2) probar que el predicado $\theta''(e,x,y) \equiv t_e(x) \leq y$ es computable.

Vamos a describir un esquema algorítmico (que se puede simular por una MTD) tal que, para cada dato de entrada **x**, si $t_e(x) \leq y$ entonces devuelve 1 y, caso contrario, devuelve 0.

El TIEMPO como medida abstracta de complejidad

Consideremos el siguiente esquema algorítmico:

Entrada: una terna (e, x, y) de números naturales.
si $y = 0$ entonces
devolver **0**
si no
$t \to 1$;
mientras $t \leq y$ hacer
hallar la configuración C_t en el instante t de la computación $M_e(x)$;
si C_t es de parada entonces
devolver **1**
si no
$t \longleftarrow t + 1$
devolver **0**

■

El ESPACIO como medida abstracta de complejidad

En el marco de las MTDs se define la medida **espacio** como sigue:

- Es el conjunto $\{s_e : e \in \mathbb{N}\}$ de funciones 1–arias sobre $\mathbb{N}$ tal que:

$$s_e(x) = \begin{cases} \max\{|x|, \text{número celdas } \textbf{inspeccionadas} \text{ en la computación } M_e(x)\}, & \text{si } M_e(x)\downarrow \\ \uparrow, & \text{si } M_e(x)\uparrow \end{cases}$$

Se demuestra que el conjunto $\{s_e : e \in \mathbb{N}\}$ de funciones 1–arias sobre $\mathbb{N}$ satisface los axiomas de Blum (es una medida abstracta de complejidad).

El segundo axioma se establece probando que $\theta''(e,x,y) \equiv s_e(x) \leq y$ es un predicado computable. Para ello, téngase presente que:

- Es finito el número de configuraciones distintas de $M_e(x)$ tales que, a partir de la celda $(y+1)$-ésima, todos los símbolos son B.
- Si en la computación $M_e(x)$ existen configuraciones C_t y $C_{t'}$ tales que $t \neq t'$ y $C_t = C_{t'}$, entonces $M_e(x)$ no es de parada.

Un conjunto de funciones que NO es una medida de complejidad

Proposición 3: *El conjunto de* **todas** *las* **funciones computables** *1-arias* $\{\varphi_e^{(1)} \mid e \in \mathbb{N}\}$ **no es** *una medida de complejidad.*

Demostración: Veamos que **no** se verifica el segundo axioma de Blum.

- ⋆ Si el predicado $\theta(e, x, y) \equiv (\varphi_e^{(1)}(x) = y)$ fuese computable, entonces se considera la función auxiliar g de $\mathbb{N}$ en $\mathbb{N}$ definida como sigue:

$$g(x) = \begin{cases} 0 & \text{si } \varphi_x^{(1)}(x)\downarrow \\ \uparrow & \text{si } \varphi_x^{(1)}(x)\uparrow \end{cases}$$

Se tiene que $(x, y) \in G(g) \iff \varphi_x^{(1)}(x)\downarrow \wedge y = 0 \iff x \in \mathcal{K} \wedge y = 0$

Es decir, $G(g)$ sería un conjunto **r.e.** y, por tanto, la función g sería computable.

Entonces, existiría $e' \in \mathbb{N}$ tal que $g = \varphi_{e'}$. Luego, se verificaría:

$$\left[\theta(e', x, 0) = 1\right] \iff \left[\varphi_{e'}^{(1)}(x) = 0\right] \iff \left[g(x) = 0\right] \iff \left[x \in \mathcal{K}\right]$$

Esto implicaría que $\mathcal{K}$ sería computable. Lo que es una **contradicción**.

■

Consecuencias de los axiomas de Blum

Teorema de **recursividad relativa**.

- Dadas dos medidas de complejidad arbitrarias, existe una relación entre ellas (expresada a través de lo que se denomina un **factor de escala**).

Teorema de existencia de **problemas intratables**.

- Respecto de cualquier medida de complejidad, existe, al menos, un problema tal que cualquier MTD que lo resuelve utiliza una cantidad exponencial de recursos.

Teorema de **aceleración** (Blum).

- Respecto de cualquier medida de complejidad, existe, al menos, un problema que carece de una MTD **óptima** (en función de los recursos) que lo resuelve.

Complejidad inherente a un problema, respecto de una medida

Dado un problema abstracto, se puede definir la **complejidad inherente** al mismo (respecto de una medida), como sigue:

* Se consideran todas las soluciones mecánicas del problema.
* Se calcula el "coste" de cada solución a partir de la medida considerada.
* Se elige una solución que utilice una cantidad **mínima** de recursos.
* La **complejidad inherente** del problema será la cantidad de recursos que necesita esa solución.

Ahora bien, de acuerdo con el teorema de aceleración de Blum, existirían problemas abstractos que **carecerían** de complejidad inherente.

Clases de complejidad computacional, respecto de una medida

Conjunto de problemas que pueden ser resueltos por procedimientos mecánicos que usan una cantidad de recursos computacionales, respecto de esa medida, acotada superiormente por una cierta función.

Una **clase de complejidad** estará especificada por:

- ⋆ Un **modelo** de computación y, en su caso, un **modo** de computación.
- ⋆ Los **recursos** a contabilizar (medida de complejidad).
- ⋆ Una **función** computable (**cota superior** de los recursos permitidos).

Coste en tiempo de una MTD

Sea M una **MTD**.

Definición: Si $\mathcal{C} = (C_u, C_1, \ldots, C_k)$ es una computación de parada de M con entrada $u \in (\Sigma \setminus \{B\})^*$, diremos que $k = t(\mathcal{C})$ es el **tiempo de ejecución** de la computación $\mathcal{C}$.

Definición: El **coste en tiempo** de M es la función parcial $t_M : \mathbb{N} - \to \mathbb{N}$ definida así:

$$t_M(n) = \begin{cases} \text{máx}\{\text{tiempo de ejecución de } M(u) : u \in (\Sigma \setminus \{B\})^* \wedge |u| = n\}, & \text{si existe dicho máximo} \\ \text{No definida}, & \text{si no existe dicho máximo} \end{cases}$$

Definición: Diremos que M **trabaja/tiene coste** en **tiempo polinomial** si existe un polinomio $p(n)$ tal que $\exists c > 0 \; \exists n_0 \; \forall n \; (n \geq n_0 \;\Rightarrow\; t_M(n) \leq c \cdot p(n))$.

Información que proporciona el coste en tiempo de una MTD

Para cada número natural $n \in \mathbb{N}$, el valor $\mathrm{t_M(n)}$ se calcula como sigue:

- Se consideran todas las cadenas de entrada $u_1, u_2, \ldots, u_r$ de tamaño n.
- Se consideran las computaciones $M(u_1), M(u_2), \ldots M(u_r)$.
  - Si alguna de esas computaciones no es de parada, entonces $\mathrm{t_M(n)} \uparrow$.
  - Caso contrario, $\mathrm{t_M(n)}$ es el **mayor** de los tiempos de ejecución de las computaciones $M(u_1), M(u_2), \ldots M(u_r)$.

Información que proporciona el coste en tiempo de una MTD

Por tanto, si $t_M(n) = k$ entonces se tiene lo siguiente:

- ⋆ Las computaciones de M con entrada cualquier cadena de longitud n, siempre son de parada.
- ⋆ Para cada cadena u_j de longitud n, **basta simular**, a lo sumo, k pasos de la computación $M(u_j)$ para conocer el **resultado** de la misma.
- ⋆ Si M es una **MTD** de **decisión**, entonces para saber si M **acepta** o **rechaza** una cadena u_j de longitud n, **basta simular**, a lo sumo, k pasos de la computación $M(u_j)$.

Ejemplo de ilustración:
Si M es una MTD de decisión ¿**cómo se interpreta que** $t_M(27) = 1594$?

- ⋆ 1594 es el máximo número de pasos que habría que simular, de la computación de M con entrada cualquier cadena de longitud 27, a fin de saber si M **acepta** o **rechaza** dicha cadena.

Ejemplo de una MTD que trabaja en tiempo lineal

Una MTD, M, trabaja en tiempo lineal **sii** existen $a, b \in \mathbb{N}$ tal que:

- $\star$ Para cada $n \in \mathbb{N}$, existe $n_0 \in \mathbb{N}$ tal que para cadena u de longitud $n \geq n_0$, el tiempo de ejecución de la computación $M(u)$ es, a lo sumo, $a \cdot n + b$.

Vamos a diseñar una MTD que trabaja en tiempo $3 \cdot n + 4$.

Alfabeto de estados: $Q = \{q_0, q_1, q_2, q_h\}$. Alfabeto de la máquina $\Sigma = \{0, 1, B, \triangleright\}$

Idea del diseño:

(a) Con el estado q_0 vamos a "recorrer" la cadena de entrada (de izquierda a derecha) hasta llegar al primer símbolo B ($n + 1$ **pasos**).

(b) Retrocedemos un lugar, cambiando al estado q_1 (1 **paso**).

(c) Con el estado q_1 vamos a "recorrer" la cadena (de derecha a izquierda) hasta llegar al primer símbolo $\triangleright$ (n **pasos**).

(d) Avanzamos un lugar, cambiando al estado q_2 (1 **paso**).

(e) Con el estado q_2 volvemos a "recorrer" la cadena de entrada (de izquierda a derecha) hasta llegar al primer símbolo B (n **pasos**).

(f) Finalmente pasamos del estado estado q_2 al estado q_h (1 **paso**).

Número total de pasos: $3 \cdot n + 4$.

Ejemplo de una MTD que trabaja en tiempo lineal

Función de transición que captura la idea:

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{0, 1, -1\}$	
$(q_0, \triangleright)$	$(q_0, \triangleright, 1)$	} (a)
$(q_0, 0)$	$(q_0, 0, 1)$	
$(q_0, 1)$	$(q_0, 1, 1)$	
(q_0, B)	$(q_1, B, -1)$	(b)
$(q_1, 0)$	$(q_1, 0, -1)$	} (c)
$(q_1, 1)$	$(q_1, 1, -1)$	
$(q_1, \triangleright)$	$(q_2, \triangleright, 1)$	(d)
$(q_2, 0)$	$(q_2, 0, 1)$	} (e)
$(q_2, 1)$	$(q_2, 1, 1)$	
(q_2, B)	$(q_h, B, 0)$	(f)

Ejemplo de una MTD que trabaja en tiempo lineal

Una traza de la computación M(110):

1. $\overset{q_0}{\triangleright}\ 1\ 1\ 0\ B$
2. $\triangleright\ \overset{q_0}{1}\ 1\ 0\ B$
3. $\triangleright\ 1\ \overset{q_0}{1}\ 0\ B$
4. $\triangleright\ 1\ 1\ \overset{q_0}{0}\ B$
5. $\triangleright\ 1\ 1\ 0\ \overset{q_0}{B}$
6. $\triangleright\ 1\ 1\ \overset{q_1}{0}\ B$
7. $\triangleright\ 1\ \overset{q_1}{1}\ 0\ B$
8. $\triangleright\ \overset{q_1}{1}\ 1\ 0\ B$
9. $\overset{q_1}{\triangleright}\ 1\ 1\ 0\ B$
10. $\triangleright\ \overset{q_2}{1}\ 1\ 0\ B$
11. $\triangleright\ 1\ \overset{q_2}{1}\ 0\ B$
12. $\triangleright\ 1\ 1\ \overset{q_2}{0}\ B$
13. $\triangleright\ 1\ 1\ 0\ \overset{q_2}{B}$
14. $\triangleright\ 1\ 1\ 0\ \overset{q_h}{B}$

Ejemplo de una MTD que trabaja en tiempo lineal

Coste en tiempo de una MTND de decisión

Sea M una **MTND** de **decisión**.

Definición: Si $\mathcal{C} = (C_u, C_1, \ldots, C_k)$ es una computación de parada de M con entrada $u \in (\Sigma \setminus \{B\})^*$, diremos que $k = t(\mathcal{C})$ es el **tiempo de ejecución** de la computación $\mathcal{C}$.

Definición: El **coste en tiempo** de M es la función total $t_M : \mathbb{N} \to \mathbb{N}$ definida así:

$$t_M(n) = \textbf{máx}\ \{t^*_M(u) :\ u \in (\Sigma \setminus \{B\})^* \wedge |u| = n\}$$

en donde: $t^*_M(u) = \textbf{min}\ \{t(\mathcal{C}) \mid\ \mathcal{C}$ es **computación de aceptación** de $u\}$ (**si existe** dicho mínimo). Caso contrario, $t^*_M(u) = 0$.

Definición: M **trabaja/tiene coste** en **tiempo polinomial** si existe un polinomio $p(n)$ tal que $\exists c > 0\ \exists n_0\ \forall n\ (n \geq n_0 \to t_M(n) \leq c \cdot p(n))$.

Información que proporciona el coste en tiempo de una MTND de decisión

Para cada número natural $n \in \mathbb{N}$, el valor $\mathrm{t_M(n)}$ se calcula como sigue:

- Se consideran todas las cadenas de entrada $u_1, \ldots, u_r$ de tamaño n.
- Se consideran todos los **árboles de computaciones** de $M(u_1), \ldots, M(u_r)$.
- De cada árbol de computaciones **se elige**, si existe, **una computación** $\mathcal{C}_j$, $1 \leq j \leq r$, que sea de **aceptación** y tenga **menor "longitud"**.
- A su vez, de esas computaciones $\mathcal{C}_j$ se selecciona una que tenga mayor "longitud": ese valor será, por definición, $\mathrm{t_M(n)}$.

Información que proporciona el coste en tiempo de una MTND de decisión

Por tanto, si $t_M(n) = k$ entonces se verifica:

* Caso $k = 0$ Para cada cadena u de longitud n, $t^*_M(u) = 0$ y, por tanto, **ninguna** de las computaciones de M con entrada u será de aceptación.

  En este caso, M **no acepta** ninguna cadena de longitud n.

* Caso $k > 0$ Para cada cadena u de longitud n, basta simular, a lo sumo, k pasos de todas las computaciones del árbol de $M(u)$ para decidir si M **acepta o no** dicha cadena.

  - Si en esa simulación **aparece** alguna computación de aceptación, entonces M **aceptará** esa cadena.
  - Si en esa simulación **no aparece** ninguna computación de aceptación, se debe a que una tal computación **no existe** en el árbol de $M(u)$. Por tanto, M **no aceptará** esa cadena.

Es decir, si se conoce el coste en tiempo de una MTND de **decisión**, entonces **aceptar o rechazar** una cadena por medio de una tal máquina es, en cierto sentido, un proceso mecánico.

Ejemplo de ilustración: Si M es una MTND de **decisión** ¿**cómo se interpreta que** $t_M(27) = 1594$?

Sea u cualquier cadena de entrada de M cuyo tamaño es 27.

Si en el árbol de computaciones de $M(u)$ existe alguna computación de aceptación, entonces su longitud ha de ser menor o igual que 1594.

Por tanto, al simular, a lo sumo, 1594 pasos de transición de cualquier computación del árbol de $M(u)$ puede suceder dos cosas:

- Que hayamos encontrado una computación de aceptación.
  - En este caso, **aceptamos la cadena u**.
- Que **no** hayamos encontrado una computación de aceptación.
  - En este caso, **no aceptamos la cadena u**.

Conclusión: 1594 es el máximo número de pasos que se ha de simular de todas las computaciones de M con entrada cualquier cadena de longitud 27, a fin de saber si M **acepta** o **no** dicha cadena.

Resolubilidad algorítmica de problemas de decisión

Sea $X = (\Sigma_X, E_X, \theta_X))$ un problema de decisión.

Definición: Una MTD de decisión, M, **resuelve algorítmicamente** el problema X **sii** M **decide el lenguaje** $L_X = \{u \in \Sigma_X \mid \theta_X(u) = 1\}$ asociado a X.

- Para cada $u \in \Sigma^*$: si $u \in L_X$ (es decir, $\theta_X(u) = 1$) entonces $M(u)$ **es una computación de aceptación**, y si $u \notin L_X$ (es decir, $\theta_X(u) = 0$) entonces $M(u)$ **es una computación de rechazo**.

Definición: Una MTND de decisión, M, **resuelve algorítmicamente** el problema X **sii** M **decide el lenguaje** $L_X = \{u \in \Sigma_X \mid \theta_X(u) = 1\}$ asociado a X.

- Para cada $u \in \Sigma^*$: $u \in L_X$ (es decir, $\theta_X(u) = 1$) **sii** en el árbol de computaciones $M(u)$, **existe**, al menos, **una computación de aceptación**.

Definición: El problema X es **resoluble algorítmicamente en tiempo polinomial** si existe una MTD o una MTND que **trabaja en tiempo polinomial** y, además, **decide el lenguaje** L_X asociado a X.

Reducibilidad en tiempo polinomial

Sean $X_1 = (\Sigma_{X_1}, E_{X_1}, \theta_{X_1})$ y $X_2 = (\Sigma_{X_2}, E_{X_2}, \theta_{X_2})$ dos problemas de decisión.

Se trata de formalizar la **dificultad comparativa** de la resolubilidad algorítmica de X_1 respecto de la de X_2 (X_1 es "**más fácil**" que X_2, o bien X_2 es "**más difícil**" que X_1).

Definición: Diremos que X_1 es **reducible en tiempo polinomial** a X_2 (y notaremos $X_1 \leqslant^p X_2$) **sii** existe una función total g de E_{X_1} en E_{X_2} tal que:

- $\star$ g es computable en tiempo polinomial.
- $\star$ Para cada $u \in E_{X_1}$: $u \in L_{X_1} \Leftrightarrow g(u) \in L_{X_2}$ $(\theta_{X_1}(u) = 1 \Leftrightarrow \theta_{X_1}(g(u)) = 1)$.

Diremos que g es una **reducción polinomial** de X_1 a X_2, y notaremos $g : X_1 \leqslant^p X_2$.

Proposición 4: *Si X_1 es un problema de decisión, entonces $X_1 \leqslant^p X_1$* (**reflexividad**).

Demostración: La aplicación identidad g en E_{X_1} ($g(u) = u$, para cada $u \in E_{X_1}$) es computable en tiempo polinomial y, además, verifica: $u \in L_{X_1}$ **sii** $g(u) = u \in L_{X_1}$.

Es decir, g es una reducibilidad en tiempo polinomial de X_1 en sí mismo. ■

Proposición 5: *Sean X_1, X_2, X_3 problemas de decisión tales que $X_1 \leqslant^p X_2$ y $X_2 \leqslant^p X_3$. Entonces $X_1 \leqslant^p X_3$* (**transitividad**).

Demostración:
Sean $g : X_1 \leqslant^p X_2$ y $h : X_2 \leqslant^p X_3$ sendas reducibilidades en tiempo polinomial.

Entonces. la composición $f = h \circ g$ es una función total de E_{X_1} en E_{X_3}.

Puesto que g y h son funciones computables en tiempo polinomial, resulta que f será, asímismo, computable en tiempo polinomial.

Además, para cada $u \in E_{X_1}$ se verifica lo siguiente:

$$\boxed{u \in L_{X_1}} \overset{g\ r.t.p.}{\Longleftrightarrow} g(u) \in L_{X_2} \overset{h\ r.t.p.}{\Longleftrightarrow} h(g(u)) \in L_{X_3} \Longleftrightarrow \boxed{f(u) \in L_{X_3}}$$

Por tanto, f es una reducibilidad en tiempo polinomial de X_1 a X_3.

■

Definición: (*Equivalencia en tiempo polinomial*)
Diremos que un problema de decisión X_1 es **equivalente en tiempo polinomial** a un problema de decisión X_2 **sii** $X_1 \leqslant^p X_2$ y $X_2 \leqslant^p X_1$.

La equivalencia en tiempo polinomial entre problemas de decisión formaliza el concepto de tener la **misma dificultad comparativa**. Se trata de una **relación de equivalencia** (reflexiva, simétrica y transitiva).

Un ejemplo de reducibilidad en tiempo polinomial

Sea $G = (V, E)$ un grafo no dirigido y $V_1 \subseteq V$

- ⋆ Diremos que V_1 es un **clique** de G **sii** cada par de vértices distintos de V_1 están conectados por una arista.
- ⋆ Diremos que V_1 es un **recubrimiento de vértices** de G **sii** para cada arista de G, al menos uno de sus extremos pertenece al conjunto V_1.

El problema **CLIQUE**: **Dado un grafo no dirigido $G = (V, E)$ y un número natural $k \leq |V|$, determinar si existe en G un clique de tamaño k.**

El problema del **recubrimiento de vértices**: **Dado un grafo no dirigido $G = (V, E)$ y un número natural $k \leq |V|$, determinar si existe en G un recubrimiento de vértices de tamaño k.** Notaremos este problema por **VC**.

Un ejemplo de reducibilidad en tiempo polinomial

Proposición 6: CLIQUE $\leq^p$ VC.

Demostración:

Sea $F : E_{\text{CLIQUE}} \longrightarrow E_{\text{VC}}$ la función definida como sigue:

- ⋆ Si $G = (V, E)$ es un grafo no dirigido entonces se considera $G' = (V', E')$, siendo $V' = V$ y $E' = \{(x, y) \in V \times V \mid x \neq y \wedge \{x, y\} \notin E\}$.
- ⋆ $F(G, k) = (G', k')$, siendo $k' = |V| - k$.

Obviamente, F es una función computable en tiempo polinomial.

Veamos que para cada $(G, k) \in E_{\text{CLIQUE}}$: $(G, k) \in L_{\text{CLIQUE}}$ **sii** $(G', k') \in L_{\text{VC}}$.

Es decir, G posee un clique de tamaño k **sii** G' posee un r.v. de tamaño k'.

- ⋆ Sea V_1 un **clique** de G de tamaño k. Entonces $V_1' = V - V_1$ es un conjunto de tamaño k'. Además, V_1' es un **r.v.** de G' ya que, de lo contrario, existiría una arista $\{x, y\} \in E'$ tal que $x \notin V_1'$ e $y \notin V_1'$; es decir, $x \in V_1$, $y \in V_1$, $x \neq y$, pero $\{x, y\} \notin E$. Lo que **contradice** que V_1 sea un **clique** de G.
- ⋆ Sea V_2 un **r.v.** de G' de tamaño $k' = |V| - k$. Entonces $V_2' = V - V_2$ es un conjunto de tamaño k. Ademas, V_2' es un **clique** de G ya que, de lo contrario, existirían nodos distintos x, y de V_2' tales que $\{x, y\} \notin E$; es decir, $x \notin V_2$, $y \notin V_2$, $x \neq y$, pero $\{x, y\} \in E'$. Lo que **contradice** que V_2 sea un **r.v.** de G'.

■