

Fundamentos de Teoría de la Computación - 2020

- Horarios, aulas, plantel docente, material, inscripción:
 - **Teoría.** Lunes de 14 a 16 hs. Aula 10-B.
 - **Práctica.** Martes de 19 a 21 hs. Aula 10-A. Clase práctica frontal de media a una hora (fundamentalmente ejercicios), y luego atención de consultas. Ya hay esta semana.
 - **Clase comodín (plan de contingencias).** Viernes de 19 a 21 hs. Aula 10-B.
 - **Trabajos prácticos.** Semanales, para ejercitarse, no se entregan, se discuten con el plantel docente.
 - **Plantel docente:** Prof. Ricardo Rosenfeld (Parte 1) y Prof. Claudia Pons (Parte 2). JTP Leandro Mendoza. Ayudante Ilán Rosenfeld.
 - **Material.** Publicado en el sitio IDEAS (programa y contenidos, clases, bibliografía).
 - **Inscripción.** En el Sistema Guaraní y en el sitio IDEAS.

Fundamentos de Teoría de la Computación - 2020

- Estructura:

- **Parte 1. Computabilidad y Complejidad Computacional.** 7 clases teóricas y 7 clases prácticas.
- **Dictada.** 8 semanas, desde lunes 9-Mar hasta viernes 1-May (semana del 23-Mar no hay dictada).
Semana del 4-May: consultas. Examen parcial: lunes 11-May.
- **Parte 2. Lógica e Inteligencia Artificial.** 7 clases teóricas y 7 clases prácticas.
- **Dictada.** 7 semanas, desde lunes 18-May hasta viernes 3-Jul.
Semana del 6-Jul: consultas. Examen parcial: martes 14-Jul.
- **Receso Invernal.** 2 semanas, desde lunes 20-Jul hasta viernes 31-Jul.
- **Exámenes Finales (Promoción).** Lunes 3-Ago Parte 1, Martes 4-Ago Parte 2.
- **Fin del Cuatrimestre.** Sábado 8-Ago.
- **Aprobación de la Materia.** Se consideran los exámenes, y también el concepto por la participación en clase y la resolución de los trabajos prácticos.

Fundamentos de Teoría de la Computación - 2020

- Otros comentarios:

- No se obliga a asistir a clases.
- **De todos modos, se recomienda asistir y seguir día a día la materia, si no será dificultoso entenderla y aprobarla. Lo mismo aplica a la resolución de los trabajos prácticos.**
- Todo el material de la materia estará disponible progresivamente en el sitio IDEAS. Además, toda la bibliografía recomendada se encuentra en la Biblioteca.
- **Es necesario que el plantel docente cuente con las direcciones de email de todos los alumnos, como plan de contingencias, para distribuir el material e intercambiar consultas eventualmente, en caso de que IDEAS no esté disponible temporariamente.**
- Posibilidad de aprobar la materia optativa de la Licenciatura en Informática (Teoría de la Computación y Verificación de Programas), completando los contenidos faltantes de ésta.

Fundamentos de Teoría de la Computación - 2020

- [Alguna bibliografía \(digital, y el resto toda en la Biblioteca de la Facultad\):](#)

Básica

- R. Rosenfeld & J. Irazábal. 2013. *Computabilidad, Complejidad Computacional y Verificación de Programas*. EDULP. Edición digital.
- R. Rosenfeld & J. Irazábal. 2010. *Teoría de la Computación y Verificación de Programas*. McGraw Hill y EDULP.
- C. Pons, R. Rosenfeld & C. Smith. 2017. *Lógica para Informática*. EDULP. Edición digital.
- A. Hamilton. 1980. *Logic for Mathematicians*. Cambridge University Press.
- S. Russell & P. Norvig. 2010. *Artificial Intelligence. A Modern Approach*. Prentice Hall.

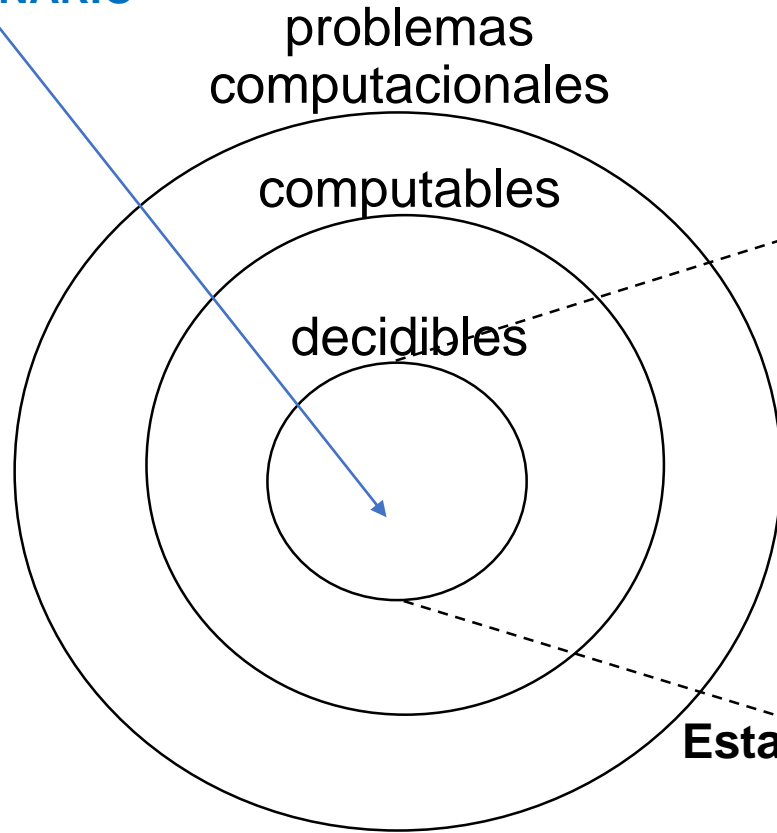
Complementaria

- Hopcroft & Ullman. 1979. *Introduction to Automata Theory, Language & Computation*. Prentice-Hall.
- S. Arora & B. Barak. 2007. *Computational Complexity: A Modern Approach*. Princeton Univ.
- C. Papadimitriou. 1995. *Computational Complexity*. Addison-Wesley.
- O. Goldreich. 2008. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press.
- Bovet & Crescenzi. *Introduction to the Theory of Complexity*. Prentice-Hall. 1994.
- C. Moore & S. Mertens. 2011. *The Nature of Computation*. Oxford University Press.
- N. Francez. 1992. *Program Verification*. Addison-Wesley (materia optativa).
- K. Apt & F. Olderog. 1997. *Verification of Sequential and Concurrent Programs*. Springer (materia optativa).

Parte 1. Computabilidad y Complejidad Computacional.

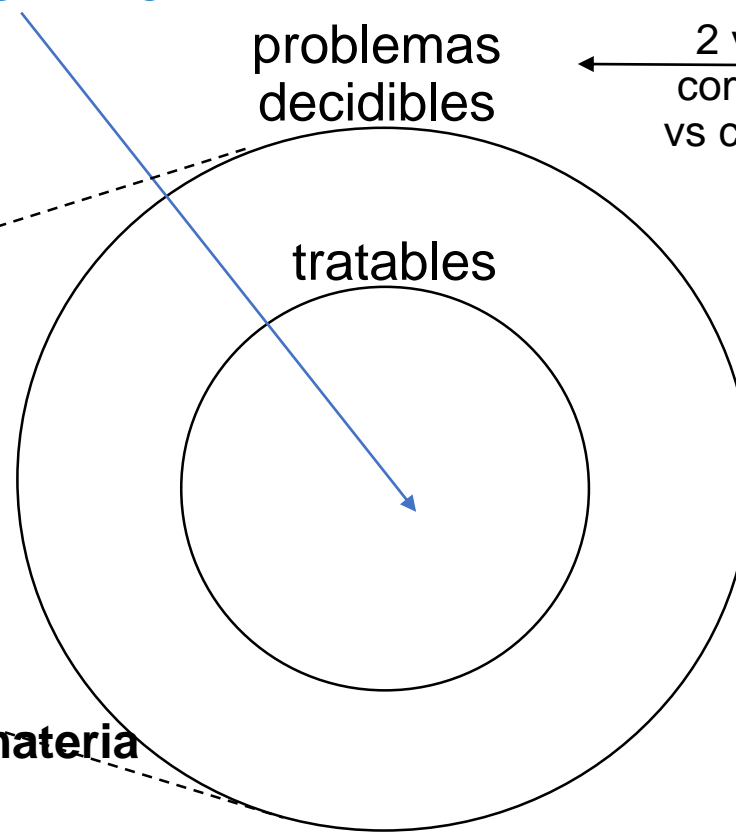
Nota: la parte de Verificación de Programas se ve en la materia optativa Teoría de la Computación y Verificación de Programas. En esta materia se dicta luego Lógica e Inteligencia Artificial (Parte 2, Prof. Pons), base para su estudio.

VIAJE
IMAGINARIO



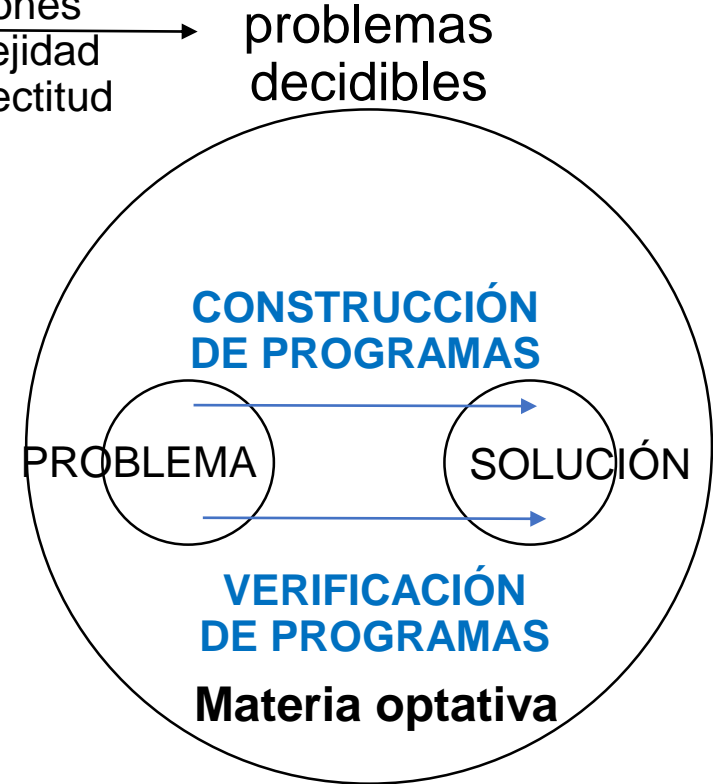
Mapa de la computabilidad (años 1930)
Máquinas de Turing

VIAJE
IMAGINARIO



Mapa de la complejidad
computacional (años 1960)
Máquinas de Turing

← 2 visiones
complejidad
vs correctitud →



Correctitud de programas
(años 1970)
Programas con While

Parte 1. Computabilidad y Complejidad Computacional.

Viaje imaginario al interior del universo de los problemas



Parte 1. Computabilidad y Complejidad Computacional.

- **Viaje imaginario**, desde los confines del universo de los problemas computacionales hacia su interior:
 - Problemas no computables, computables indecidibles, computables decidibles intratables, computables decidibles tratables.
- Desde otro punto de vista, **viaje complementario al ya emprendido en los primeros años de la Licenciatura**:
 - luego de “un viaje de ida” de 3 años, habiendo recorrido ítems tales como algorítmica (recursión, divide and conquer, greedy algorithms, programación dinámica, programación dual, reducción, etc.), programación, estructuras de datos, lenguajes de programación, testing, cierto nivel de profundidad en matemáticas discreta y continua, etc.,
 - ahora haremos “un viaje de vuelta” para volver a recorrerlos, pero con mayor madurez, desde una óptica más formal y abstracta, centrados en los **fundamentos de la Teoría de la Computación**. Consideraremos entre otros tópicos:
computabilidad, complejidad computacional, teoría de autómatas, lenguajes formales, gramáticas, lógica, inducción, diagonalización, autorreferencia, reducciones de problemas, teoría de grafos, teoría de conjuntos, combinatoria, números cardinales y ordinales, infinitos, etc.

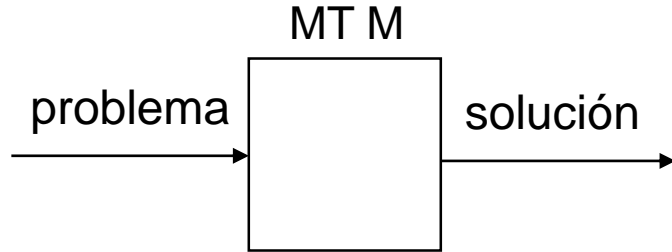
Parte 1. Computabilidad y Complejidad Computacional.

Algo de historia:

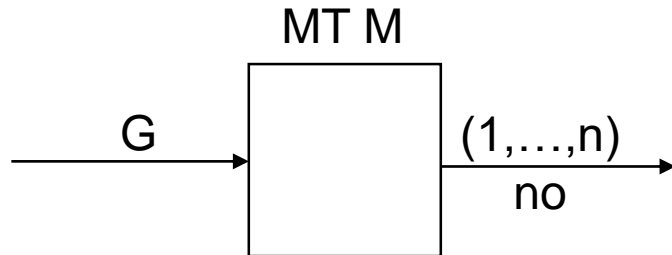
- Años 1900. Necesidad de **formalizar las matemáticas para evitar inconsistencias** (p.ej. la Paradoja de Russell en el marco de la Teoría de Conjuntos - obra de Frege -). En simultáneo, siguen los intentos por automatizar los cálculos con máquinas (iniciados siglos atrás por Pascal, Leibniz, Babbage, etc).
- Los formalistas. Hilbert. De la semántica a la sintaxis. La búsqueda de un esquema mecánico para **demostrar todas las verdades matemáticas**. El *Entscheidungsproblem* (el problema de decisión).
- **Primer fracaso.** Gödel. 1931. Teoría de Incompletitud. No existe un sistema consistente y completo para la aritmética. Hay verdades de la aritmética que no se pueden demostrar mecánicamente.
- **Segundo fracaso.** Turing y Church. 1936. Máquinas de Turing y Lambda Cálculo. Indecidibilidad. Hay enunciados acerca de los cuales no se puede establecer mecánicamente si son verdaderos o falsos.
- **Origen de la Teoría de la Computación.** “Lo computable”. Máquina de Turing. Tesis de Church-Turing. Otros modelos (funciones recursivas, gramáticas, Máquina de Post, etc). 1ras computadoras (EEUU, Inglaterra, años 1940). Complejidad (años 1960). Lógica para programación (años 1970).

Clase 1. Máquinas de Turing.

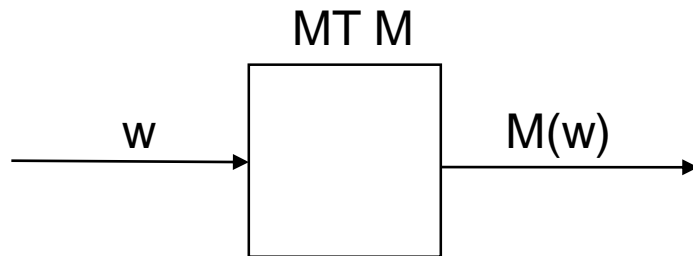
Modelo matemático simple de una computadora



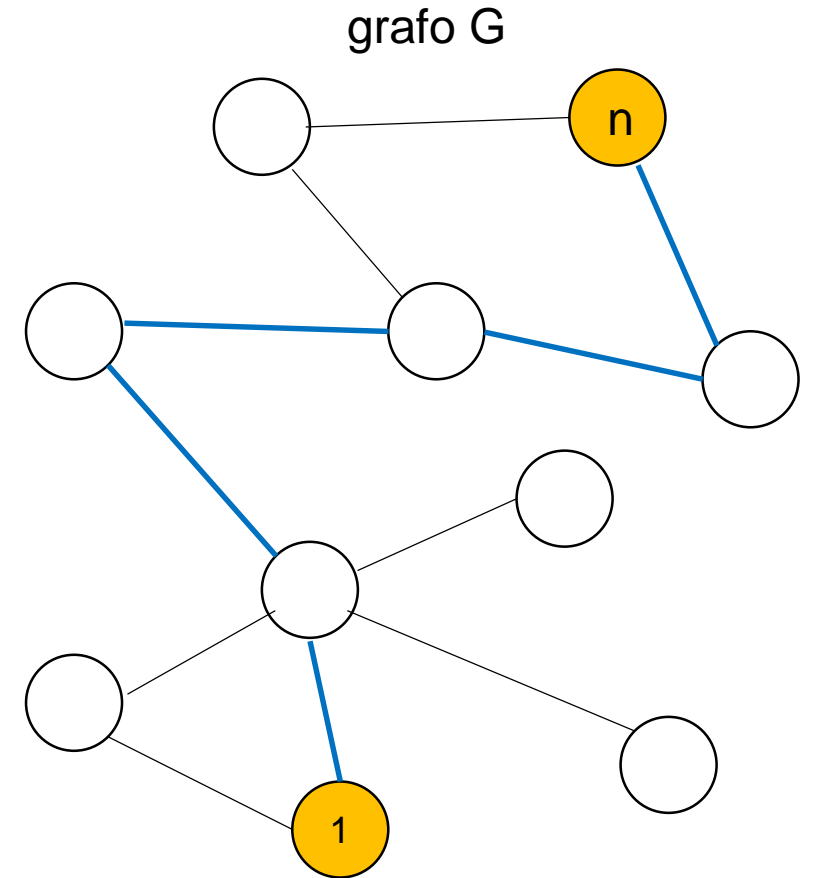
El **algoritmo** de M devuelve una solución al problema



Ejemplo: dado un grafo G , M devuelve, si existe, un camino en G del vértice 1 al vértice n , y si no existe, responde no

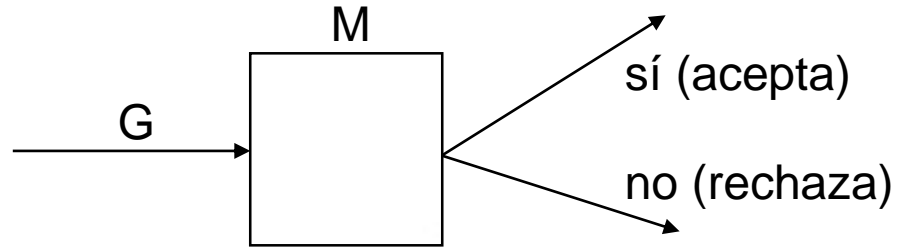


Genéricamente, w es el **input** y $M(w)$ es el **output**

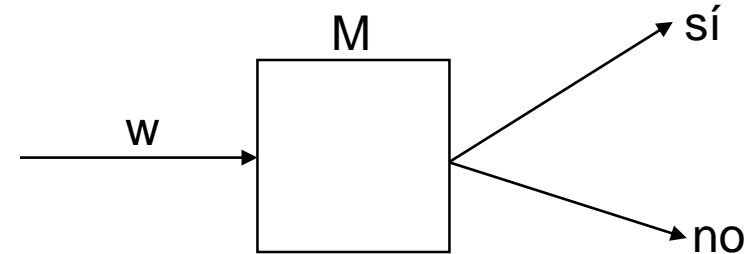


camino del vértice 1 al vértice n

Problema similar pero más simple: dado un grafo G , ¿existe en G un camino del vértice 1 al vértice n ? Este es un problema de **decisión**, no de **búsqueda**.



M acepta si y sólo si G tiene un camino del vértice 1 al vértice n



M **decide** a partir del input w

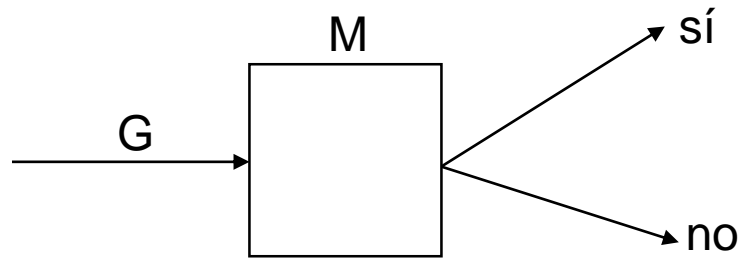
Otro ejemplo: dada una fórmula booleana φ , por ejemplo: $(x_1 \vee x_2) \wedge (x_3 \wedge x_4)$, ¿existe una asignación A de valores de verdad, que la satisface, es decir que la hace verdadera? Por ejemplo, $A1 = (V, F, V, V)$ satisface φ , y $A2 = (F, F, V, V)$ no.

De los problemas generales de **búsqueda** a los problemas particulares de **decisión**: estos últimos son más simples para estudiar la computabilidad y complejidad computacional (y veremos luego, también útiles para inferir sobre los de búsqueda).

Precisando la ventaja de enfocarnos en los problemas de decisión:

Logramos trabajar directamente con **lenguajes** (conjuntos de cadenas de símbolos), que es más sencillo. Volviendo al ejemplo anterior:

Problema del camino en un grafo G:



La MT M **acepta** todas las cadenas G que representan grafos con un camino del vértice 1 al vértice n

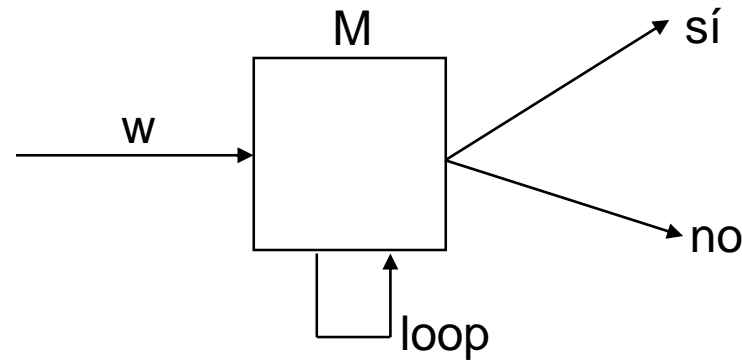
En otras palabras, M **acepta**, o **reconoce**, o **decide**, el lenguaje:

$L(M) = \{G \mid G \text{ es un grafo y } G \text{ tiene un camino del vértice 1 al vértice } n\}.$

Salvo indicación en contrario, trabajaremos con esta visión, de MT **reconocedora**. Así, problemas y lenguajes serán sinónimos en este contexto.

¡Inconveniente!:

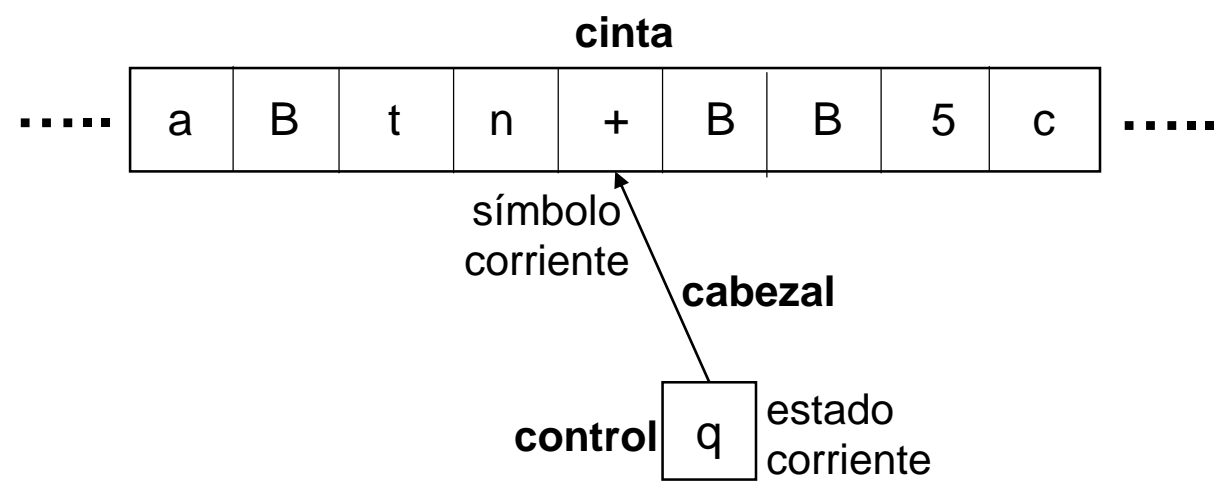
Veremos que el caso más general de MT corresponde a este esquema:



Es decir, para algunos problemas **NO existen MT que siempre paran.**

Es el caso de los problemas **indecidibles**: toda MT M que pretenda resolver un problema así cumplirá que en algunos casos negativos, es decir en que $w \notin L(M)$, en lugar de responder no M “loopeará”, no terminará, no emitirá respuesta alguna.

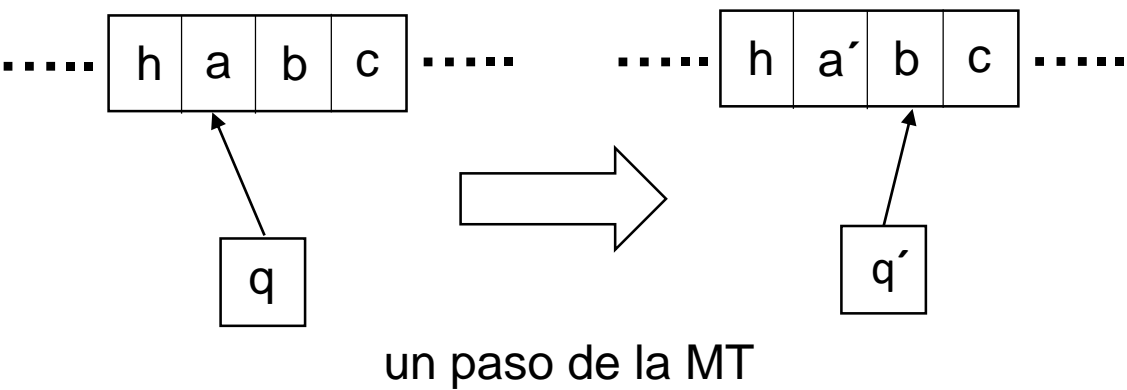
Descripción formal de una Máquina de Turing



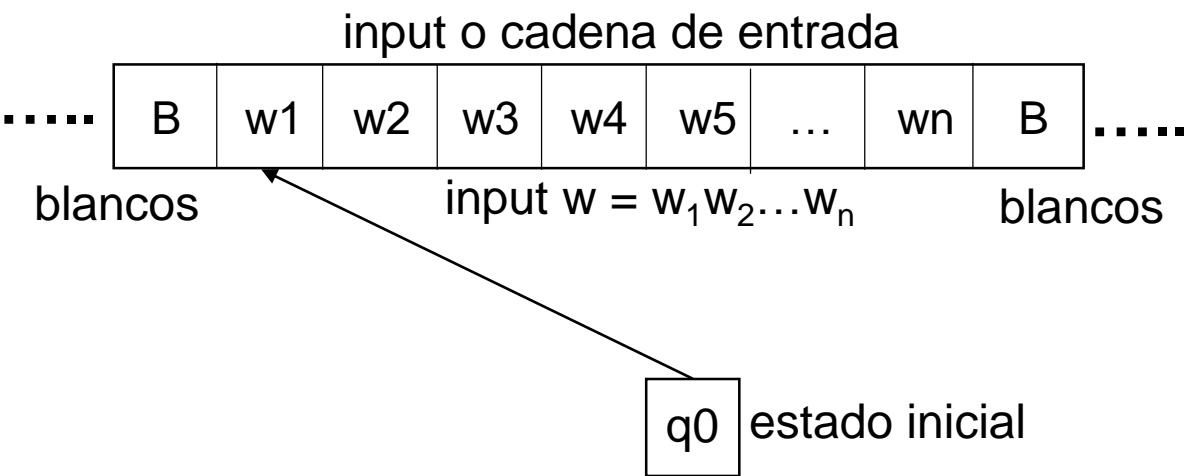
Tesis de Church-Turing

¡Todo lo computable puede ser resuelto por una Máquina de Turing!

En un paso, una MT puede modificar el símbolo corriente, el estado corriente, y moverse un lugar a derecha o izquierda



Al inicio, el input se delimita por blancos, y el cabezal apunta al 1er símbolo de la izquierda



Formalmente, una MT M es una tupla $(Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$:

- Q es el conjunto de estados de M
- Σ es el alfabeto de los inputs de M
- Γ es el alfabeto de las cadenas de la cinta de M

Nota: podría omitirse en la descripción de la MT M , el alfabeto Σ de los inputs.

Se cumple: $\Sigma \subseteq \Gamma$. Convención: $B \in (\Gamma - \Sigma)$, siendo B el símbolo blanco.

- q_0 es el estado inicial de M
- q_A y q_R son los estados finales de aceptación y rechazo de M , respectivamente
- δ es la función de transición de M (la que especifica el comportamiento de M):

$$\delta : Q \times \Gamma \rightarrow (Q \cup \{q_A, q_R\}) \times \Gamma \times \{L, R, S\}$$

Dado un estado corriente y un símbolo corriente, la máquina pasa a un nuevo estado (o el mismo), que puede ser final, a un nuevo símbolo (o el mismo), y se mueve un lugar a derecha, izquierda o se queda en la misma celda.

L representa el movimiento a izquierda, R a derecha, S el no movimiento

Así, en el ejemplo anterior se usó: $\delta(q, a) = (q', a', R)$

(Mini) repaso sobre lenguajes

- Σ es el alfabeto o conjunto de símbolos:

$$\Sigma = \{w_1, w_2, w_3, \dots\}$$

- Σ^* es el lenguaje o conjunto de cadenas de símbolos generado a partir de Σ :

$$\Sigma^* = \{\lambda, w_1, w_2, w_3, \dots, w_1w_1, w_1w_2, w_1w_3, \dots, w_1w_1w_1, w_1w_1w_2, \dots\}$$

Σ^* es **infinito**. Sus cadenas son **finitas**. λ es la **cadena vacía**.

- Todo lenguaje L que consideraremos será un subconjunto de Σ^* , siendo Σ un único alfabeto que tomaremos como referencial, universal:

$L \subseteq \Sigma^*$. Expresado de otra manera: $L \in P(\Sigma^*)$.

$P(\Sigma^*)$ es el conjunto de partes de Σ^* , es decir, el conjunto de todos sus subconjuntos.

- Operaciones típicas entre lenguajes: $L_1 \cap L_2$, $L_1 \cup L_2$, $L_1 - L_2$, L^C , $L_1 \cdot L_2$, etc.

Ejemplo con MT

$L = \{a^n b^n \mid n \geq 1\}$. Es decir, $L = \{ab, aabb, aaabbb, \dots\}$

Queremos construir una MT M que acepte (o decida) L , es decir tal que $L(M) = L$

1. Idea General. Un posible algoritmo podría ser:

aaaaabbbbbb

α aaaaabbbbbb

α aaaa β bbbb

$\alpha\alpha$ aaa β bbbb

$\alpha\alpha$ aaa $\beta\beta$ bbb

.....

$\alpha\alpha\alpha\alpha\alpha\beta\beta\beta\beta$

2. Construcción.

La correspondiente MT $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ sería así:

- $Q = \{q_0, q_a, q_b, q_L, q_H\}$

q_a : M busca una a q_b : M busca una b q_L : M vuelve q_H : no hay más a

- $\Sigma = \{a, b\}$

- $\Gamma = \{a, b, \alpha, \beta, B\}$

Nota: en la construcción que presentamos a continuación hemos optado por diferenciar q_0 de q_a . Podría pensarse una manera en que sean el mismo estado.

Función de transición δ

1. $\delta(q_0, a) = (q_b, \alpha, R)$
2. $\delta(q_a, a) = (q_b, \alpha, R)$
3. $\delta(q_a, \beta) = (q_H, \beta, R)$
4. $\delta(q_b, a) = (q_b, a, R)$
5. $\delta(q_b, b) = (q_L, \beta, L)$
6. $\delta(q_b, \beta) = (q_b, \beta, R)$
7. $\delta(q_L, a) = (q_L, a, L)$
8. $\delta(q_L, \alpha) = (q_a, \alpha, R)$
9. $\delta(q_L, \beta) = (q_L, \beta, L)$
10. $\delta(q_H, \beta) = (q_H, \beta, R)$
11. $\delta(q_H, B) = (q_A, B, S)$

Todos los casos omitidos en la descripción son de rechazo.
Por ejemplo: $\delta(q_0, b) = (q_R, b, S)$.

Forma alternativa de describir la función de transición δ

	a	b	α	β	B
q_0	q_b, α, R				
q_a	q_b, α, R			q_H, β, R	
q_b	q_b, a, R	q_L, β, L		q_b, β, R	
q_L	q_L, a, L		q_a, α, R	q_L, β, L	
q_H				q_H, β, R	q_A, B, S

Las celdas en blanco representan los casos de rechazo (estado q_R)

3. Prueba de $L(M) = L$.

- Si $w \in L$, entonces w tiene la forma $a^n b^n$, con $n \geq 1$.

Entonces, por cómo está definida la función de transición δ , claramente a partir de w **la MT M acepta w** , es decir que $w \in L(M)$.

(Esto no es formal. Formalmente, esta parte podría probarse por inducción, primero considerando $n = 1$ y luego $n > 1$.)

- Si $w \notin L$, entonces M no tiene la forma $a^n b^n$, con $n \geq 1$.

Entonces se cumple que **la MT M rechaza w** , es decir que $w \notin L(M)$:

- Si $w = \lambda$, M rechaza porque no está definido en δ el par (q_0, B) .
- Si w empieza con b , M rechaza porque no está definido en δ el par (q_0, b) .
- Etc.
- Queda como ejercicio completar la prueba.

Notación

Por ejemplo, volviendo al problema $L = \{a^n b^n \mid n \geq 1\}$:

$$q_0 a a a b b b \vdash_M \alpha q_b a a b b b \vdash_M \alpha \alpha q_b a b b b \vdash_M \dots$$

El estado corriente precede al símbolo corriente. El símbolo \vdash_M representa un paso de M.

$$q_0 a a a b b b \vdash_M^* \alpha \alpha \alpha \beta \beta \beta q_A$$

En varios pasos (representados por el símbolo \vdash_M^*) se pasa de $q_0 a a a b b b$ a $\alpha \alpha \alpha \beta \beta \beta q_A$

Genéricamente, las MT arrancan con la configuración:

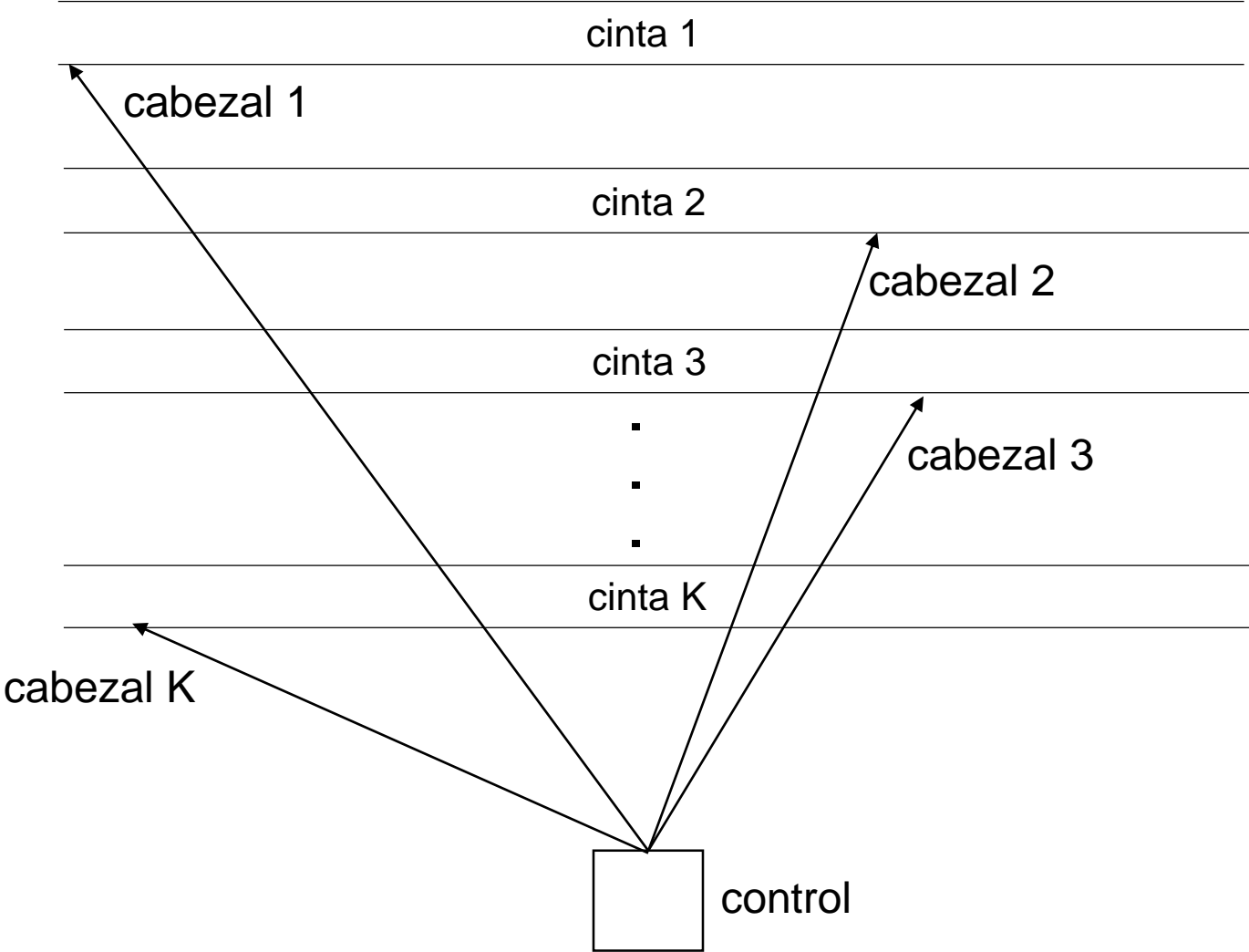
$$q_0 w_1 w_2 w_3 \dots w_n$$

y si paran, terminan en una configuración:

$$x q_A y \text{ o bien } x q_R y, \text{ siendo } x \text{ e } y \text{ cadenas}$$

Otro caso del ejemplo anterior: $q_0 a \vdash_M \alpha q_b \vdash_M \alpha q_R$, y así: $q_0 a \vdash_M^* \alpha q_R$

Máquinas de Turing con varias cintas (MT con K cintas)



Características generales de las MT con K cintas:

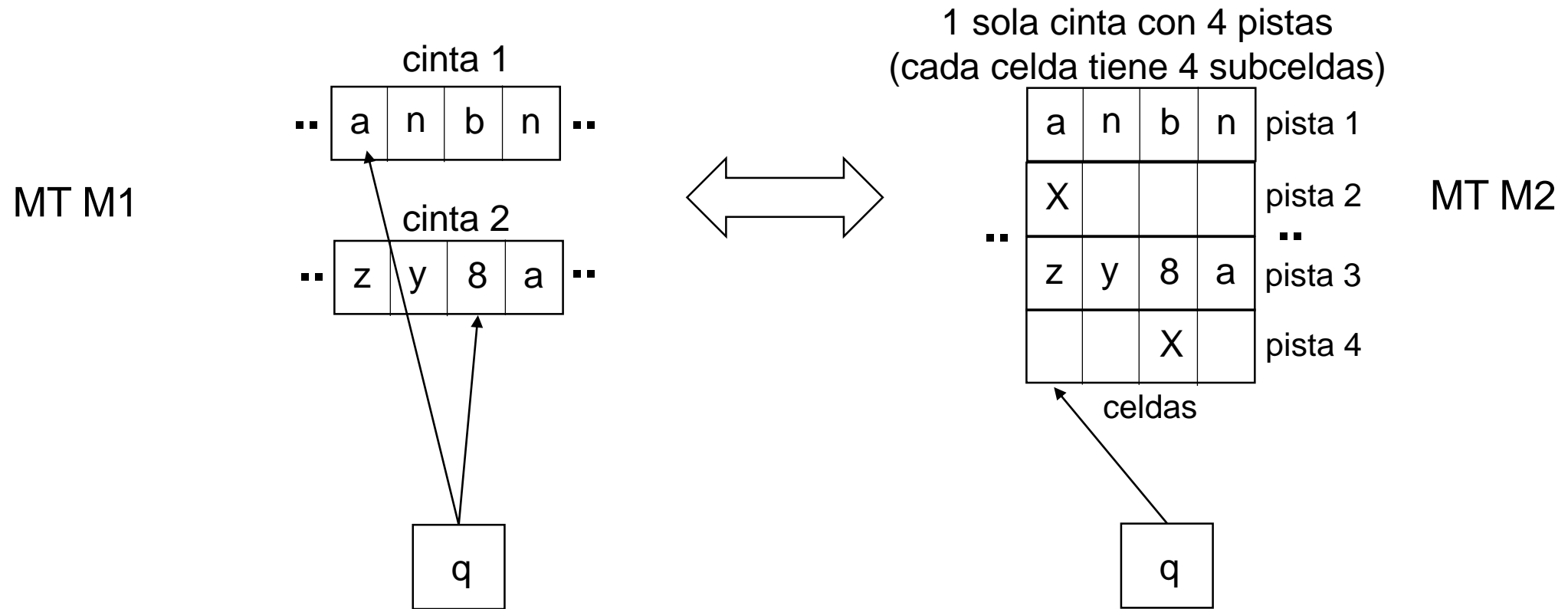
- En cada cinta la MT se comporta de manera independiente
- La cinta 1 contiene el input
- La MT se comporta como siempre según su función de transición δ , que ahora:
 - En cada paso lee un estado y una K-tupla de símbolos (los apuntados por los cabezales 1 a K)
 - Modifica eventualmente el estado
 - Modifica cero, uno o más símbolos
 - Se mueve independientemente en cada cinta (derecha, izquierda o nada)

Ahora tenemos: $\delta: Q \times \Gamma^K \rightarrow (Q \cup \{q_A, q_R\}) \times (\Gamma \times \{L, R, S\})^K$

Por ejemplo, para el caso de 3 cintas, abreviando algunos paréntesis:

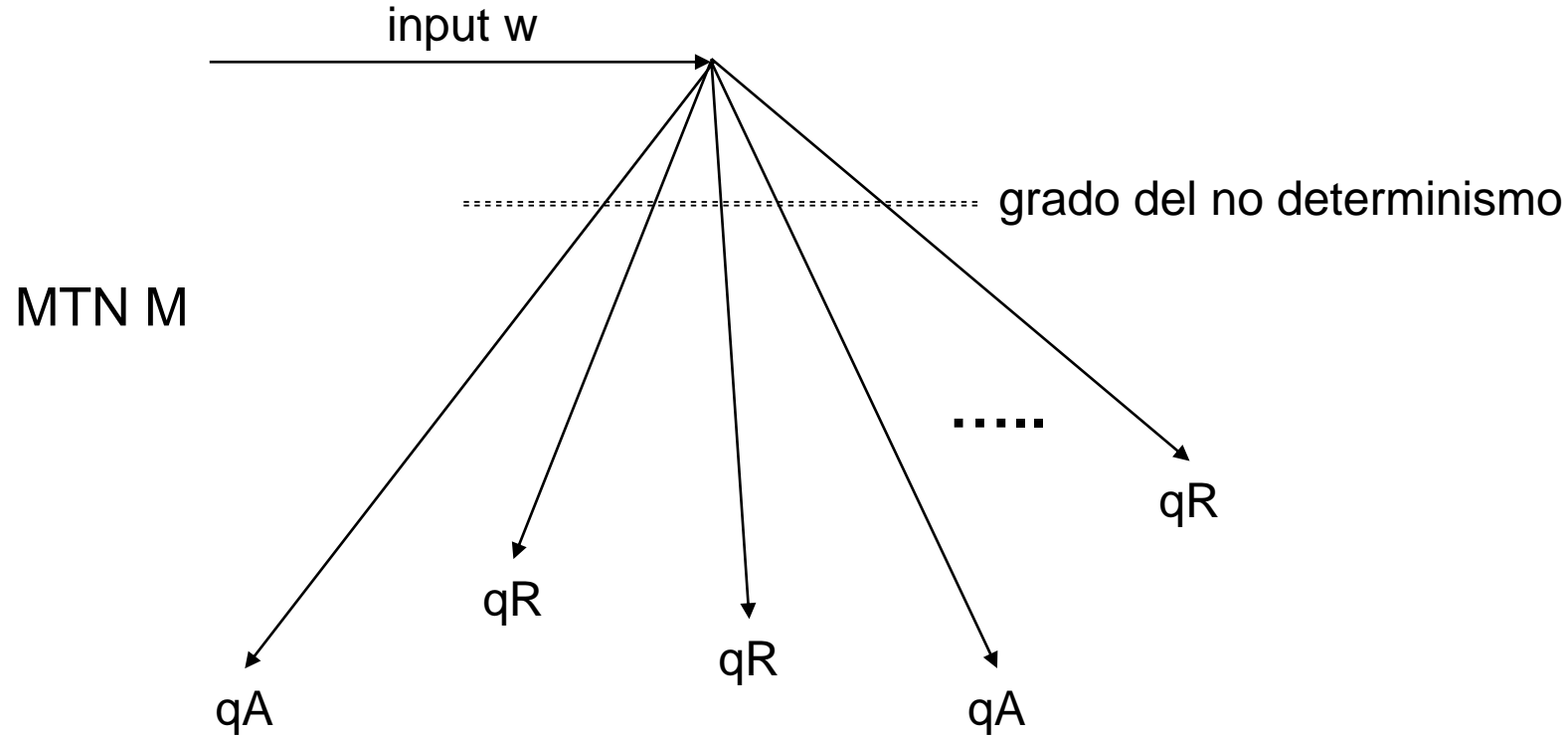
$$\delta(q, (a, b, c)) = (q', (a, R), (b', L), (c', S))$$

Aunque no es tan intuitivo, una MT con K cintas **no tiene más potencia computacional** que una MT con 1 cinta (es decir, sin considerar el tiempo, lo que hace una MT con K cintas lo puede hacer una MT con 1 cinta). Por ejemplo:



Idea general para la simulación: uso de **cintas con pistas**. En el ejemplo, 2 cintas se simulan con 1 cinta con 4 pistas. Lo que sí, la MT M2 tardará más que la MT M1 (orden cuadrático): h pasos de M1 se simulan con unos $4 + 8 + 12 + \dots + 4h = 4 \cdot (1 + 2 + 3 + \dots + h) = 4 \cdot h \cdot (h+1)/2 = O(h^2)$ pasos de M2 (se verá luego).

Máquinas de Turing no determinísticas (MTN)



- En lugar de una **función** de transición δ , M tiene una **relación** de transición Δ , es decir que para un mismo par (q, a) , la máquina puede responder de más de una manera. Por ejemplo:
$$\Delta(q, a) = \{(q_1, a_1, L), (q_2, a_2, R), (q_3, a_3, S)\}$$
- La relación de transición se define así: $\Delta: Q \times \Gamma \rightarrow P((Q \cup \{q_A, q_R\}) \times \Gamma \times \{L, R, S\})$
- Una MTN acepta si y sólo si **al menos** una computación acepta

Las MTN tampoco introducen mayor potencia computacional (es decir, sin considerar el tiempo, lo que hace una MTN lo puede hacer una MT estándar o determinística, es decir una MTD):

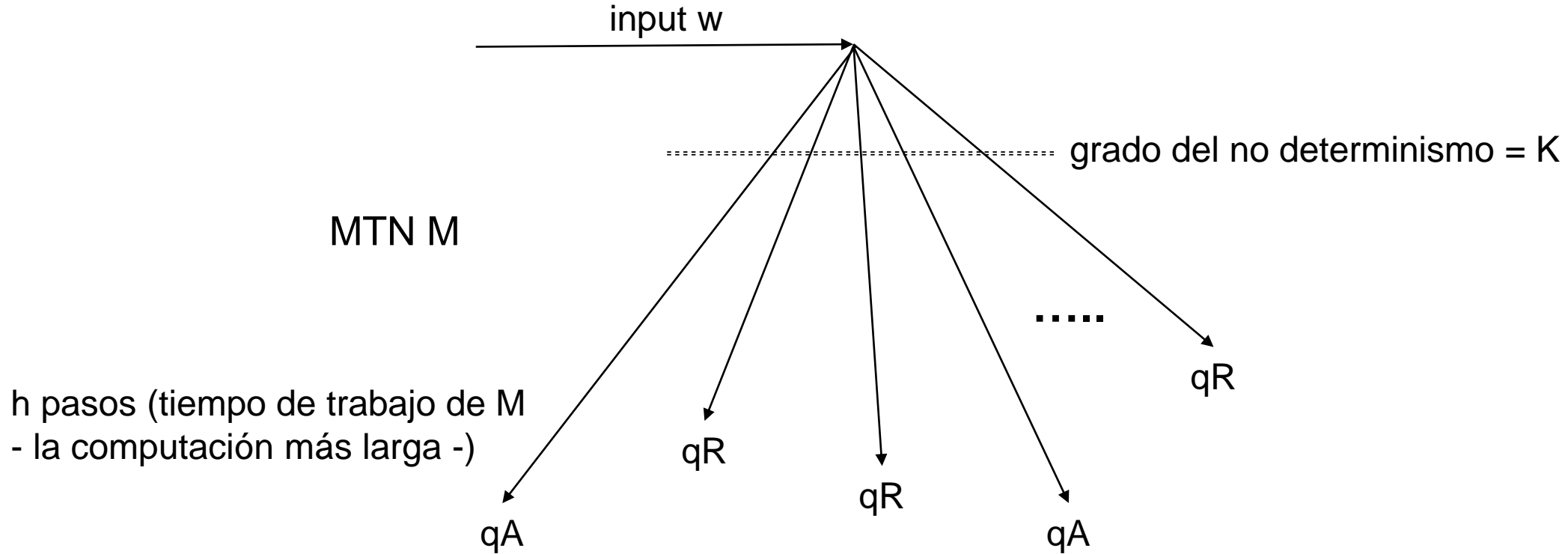
La simulación de una MTN con una MTD consiste en recorrer una a una las computaciones de la MTN y aceptar si y sólo si se encuentra una computación que acepta.

Para evitar el problema de las computaciones que no paran, el recorrido debe ser **a lo ancho** (BFS), no a lo largo (DFS).

Las MTN no son máquinas “reales”, se usan más que nada para **abreviar las descripciones de las MT**. Veremos su uso fundamentalmente en la parte de complejidad computacional.

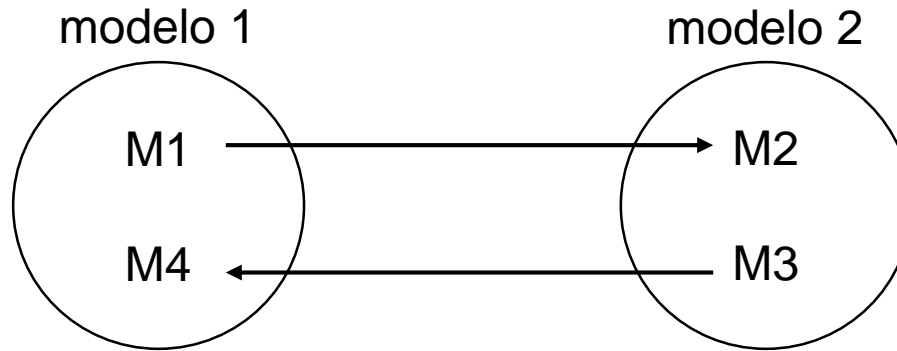
Vimos que simular una MT con K cintas mediante una MT con 1 cinta, hace que el tiempo de ejecución se eleve a un **orden cuadrático**. En el caso de las MTN el retardo es mayor, es **exponencial**, como se muestra a continuación:

Tiempo de retardo para simular una MTN con una MT estándar o MTD



Si el grado de la relación Δ es K , y la computación más larga tiene h pasos, entonces la simulación debe recorrer **a lo sumo K^h computaciones de a lo sumo h pasos cada una**, es decir un **número exponencial de pasos con respecto a h** , siendo h el tiempo **en el que trabaja M** . Por ejemplo, si $K = 3$ y $h = 10$, la simulación deberá considerar a lo sumo 3^{10} computaciones de a lo sumo 10 pasos cada una.

Dos MT son equivalentes (computacionalmente) si reconocen el mismo lenguaje. **Dos modelos de MT son equivalentes** si dada una MT de un modelo existe una MT equivalente del otro. Ya vimos los casos de las MT con 1 y K cintas, y las MTD y MTN.



Otros modelos equivalentes a las MT estudiadas son las MT con **cintas semi-infinitas**, MT con cintas de **2 dimensiones**, MT con **2 cintas y un solo estado**, etc. También hay varios modelos computacionales equivalentes a las MT, como las **máquinas RAM**, los **autómatas celulares**, los **circuitos booleanos**, el **lambda cálculo**, las **funciones recursivas parciales**, las **gramáticas**, los **programas Java**, etc. Todo esto refuerza la **Tesis de Church-Turing**.

Ejemplo con MT con K cintas

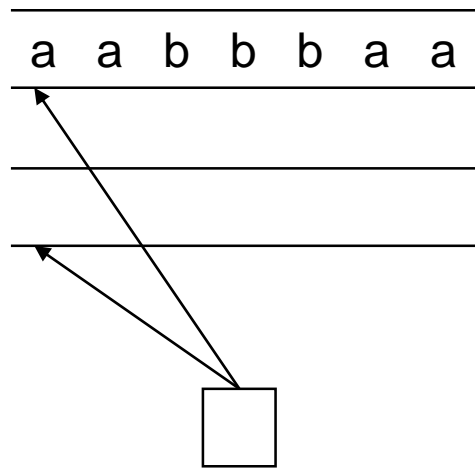
Sea $L = \{w \mid w \in \{a, b\}^* \text{ y } w \text{ es un palíndromo o "capicúa"}\}$

w es un palíndromo o "capicúa" si $w = w^R$, siendo w^R la cadena inversa de w . Por ejemplo, si $w = abb$, entonces $w^R = bba$.

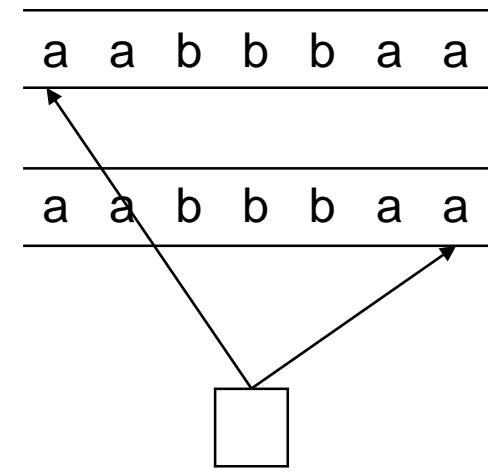
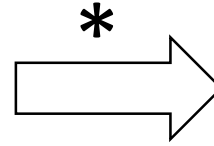
Queremos construir una MT M que acepte L .

Idea general: MT con 2 cintas, que hace:

1. Copiar el input, de la cinta 1 a la cinta 2
2. Volver a apuntar al 1er símbolo de la cinta 1 (dejar como está el 2do cabezal)
3. Comparar los símbolos apuntados (al comienzo el 1ro de la cinta 1 y el último de la cinta 2):
 - Si son distintos, rechazar
 - Si son iguales y blancos, aceptar
 - En otro caso moverse un lugar a derecha en la cinta 1, un lugar a izquierda en la cinta 2, y volver al paso (3)



Configuración inicial



Situación luego del paso 2 (en realidad, conjunto de pasos 2) de la MT

Si el input mide n símbolos, la MT hace unos $3n$ pasos (**tiempo lineal**): unos n pasos para copiar la cinta 1 en la 2, unos n pasos para volver a ubicar el cabezal de la cinta 1 al comienzo, y unos n pasos para comparar los símbolos de las 2 cintas.

Usando una cinta, en cambio, **el tiempo es cuadrático**. Por ejemplo, ir a derecha e izquierda tachando primero el 1er y el último símbolo, luego el 2do y el anteúltimo, luego el 3ro y el antepenúltimo, etc., lo que tarda:

$n + (n - 1) + (n - 2) + (n - 3) + \dots + 2 + 1 = n.(n+1)/2 = O(n^2)$ pasos.

Función de transición δ de la MT con 2 cintas

q_0 : copia de cinta 1 a 2; q_1 : reposicionamiento en cinta 1; q_2 : comparación de las 2 cintas
(como siempre, las celdas en blanco son casos de rechazo de la MT)

	a, a	a, b	a, B	b, a	b, b	b, B	B, a	B, b	B, B
q_0			q_0 , a, R, a, R			q_0 , b, R, b, R			q_1 , B, L, B, L
q_1	q_1 , a, L, a, S	q_1 , a, L, b, S		q_1 , b, L, a, S	q_1 , b, L, b, S		q_2 , B, R, a, S	q_2 , B, R, b, S	q_2 , B, S, B, S
q_2	q_2 , a, R, a, L	q_R , a, S, b, S		q_R , b, S, a, S	q_2 , b, R, b, L				q_A , B, S, B, S

Ejemplo sencillo de uso de MTN

Construir una MTN que acepte todas las cadenas iniciadas por un símbolo de cabecera h , seguido por cero o más símbolos a , o por cero o más símbolos b :

1. $\Delta(q_0, h) = \{(q_a, h, R), (q_b, h, R)\}$
2. $\Delta(q_a, a) = \{(q_a, a, R)\}$
3. $\Delta(q_a, B) = \{(q_A, B, S)\}$
4. $\Delta(q_b, b) = \{(q_b, b, R)\}$
5. $\Delta(q_b, B) = \{(q_A, B, S)\}$

Queda como ejercicio construir una MT estándar (es decir determinística, o MTD) para reconocer el mismo lenguaje.

Ejemplo con la visión de MT calculadora (problema general, de búsqueda)

Construir una MT que **reste** 2 números representados en notación unaria

Por ejemplo, dado el input 11111101111, obtener el output 11 ($6 - 4 = 2$)

Idea general: tachar el primer 1 antes del 0, luego el primer 1 después del 0, luego el segundo 1 antes del 0, y así hasta tachar al final el 0. **La construcción queda como ejercicio.**

Comentario: en este caso, además del estado final, interesa el contenido final de la cinta.

Ejemplo con la visión de MT generadora (quizá se vea después)

Construir una MT que genere **todas** las cadenas de la forma $a^n b^n$, con $n \geq 1$

Idea general (**la construcción queda como ejercicio**):

(1) $i := 1$

(2) imprimir i veces a , imprimir i veces b , e imprimir separador

(3) $i := i + 1$ y volver a (2)

Comentario: se prueba que las visiones de MT reconocedora y generadora son equivalentes (quizá se pruebe después).