Teoría de la Computación y Verificación de Programas

Introducción y Clase 1

Datos generales de la materia

Plantel docente
Ricardo Rosenfeld (teoría).
Leandro Mendoza, Pedro Dal Bianco, Jeremías Salsamendi (práctica).

Horarios y aulas
Teoría: martes de 19 a 21 hs. Aula 1-4.
Práctica: jueves de 19 a 21 hs. Aula 1-3.

Estructura de la materia y calendario
Parte 1. 4 clases de **computabilidad**.
Parte 2. 5 clases de **complejidad computacional**.
Parte 3. 4 clases de **verificación de programas**.
Exámenes promocionales: **25 de junio, 2 de julio, 11 de julio.**

Trabajos prácticos
Quincenales. Se recomienda trabajar en parejas.
No es obligación entregarlos. Pero se recomienda hacerlos para llegar bien entrenados al examen (¡y para aprender!). **Entregarlos otorga un bonus en la calificación final**.
Se consultan durante las clases prácticas, y durante toda la semana por medio de la plataforma **IDEAS.**

Material

Básico (Plataforma IDEAS)

Clases en ppt, más artículos de interés, más libro digital:

Rosenfeld & Irazábal. 2013. Computabilidad, Complejidad Computacional y Verificación de Programas. EDULP.

https://sedici.unlp.edu.ar/bitstream/handle/10915/27887/Documento_completo__.pdf?sequence=3

Complementario (Biblioteca)

Rosenfeld & Irazábal. 2010. Teoría de la Computación y Verificación de Programas. McGraw Hill y EDULP.

Hopcroft & Ullman. 1979. Introduction to Automata Theory, Language & Computation. Prentice-Hall.

Lewis & Papadimitriou. 1998. Elements of the Theory of Computation. Prentice-Hall

Sipser. 1997. Introduction to the Theory of Computation. PWS Publishing.

Arora & Barak. 2007. Computational Complexity: A Modern Approach. Princeton Univ.

Papadimitriou. 1995. Computational Complexity. Addison-Wesley.

Goldreich. 2008. Computational Complexity: A Conceptual Perspective. Cambridge University Press.

Bovet & Crescenzi. 1994. Introduction to the Theory of Complexity. Prentice-Hall.

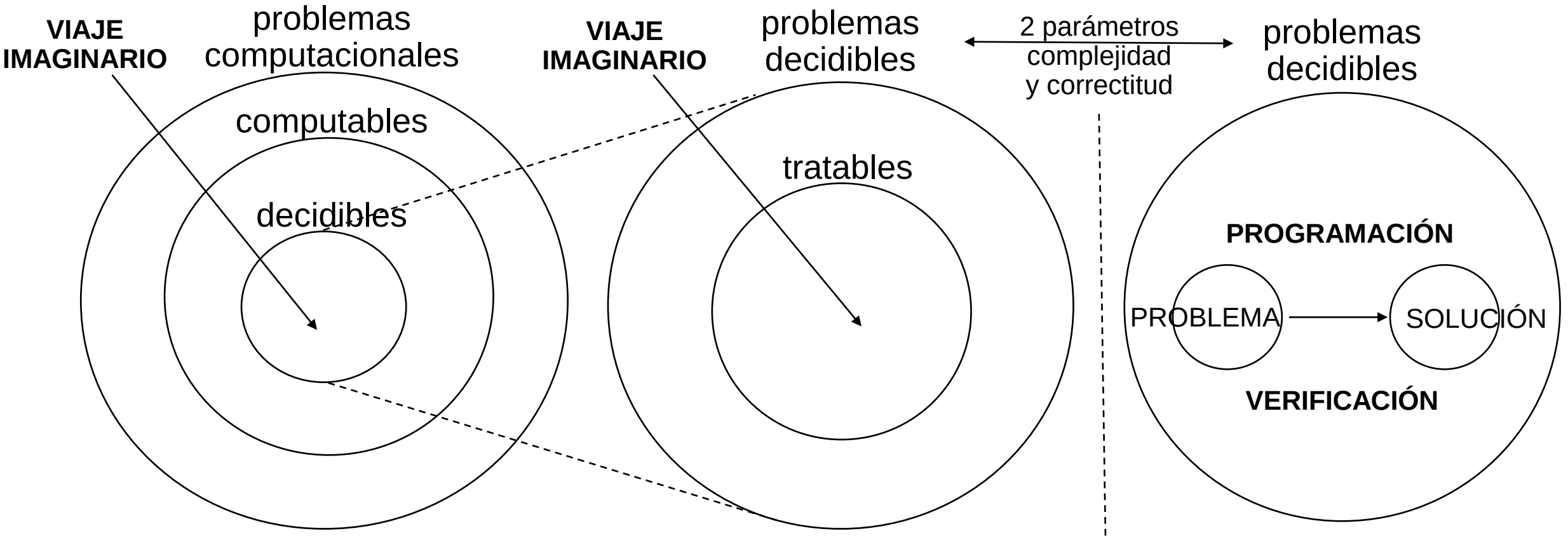
Moore & Mertens. 2011. The Nature of Computation. Oxford University Press.

Francez. 1992. Program Verification. Addison-Wesley.

Apt & Olderog. 1997. Verification of Sequential and Concurrent Programs. Springer.

Huth & Ryan. 2004. Logic in Computer Science. Cambridge University Press.

Introducción
viaje imaginario

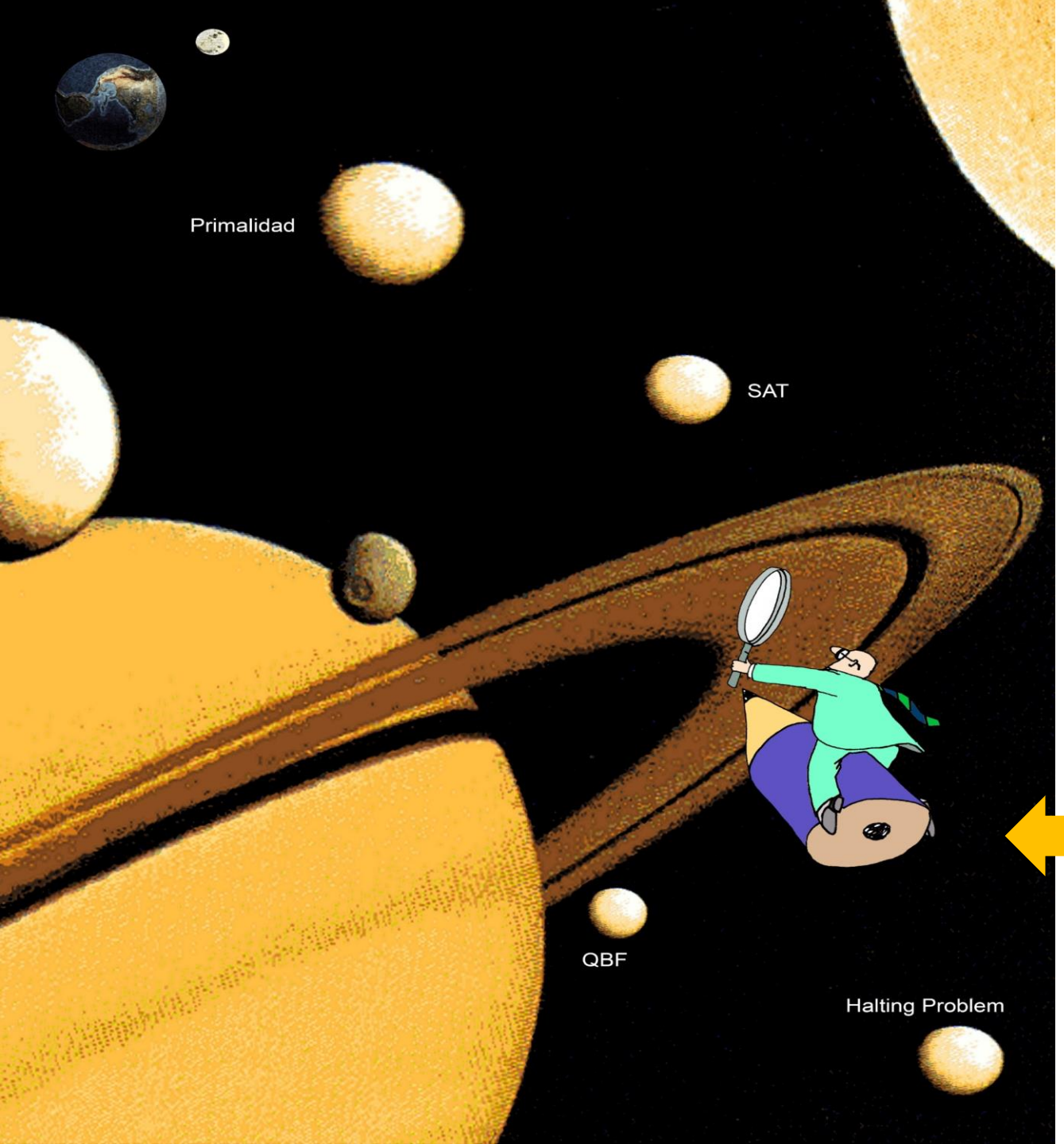


Computabilidad
(desde los años 1930)
Máquinas de Turing
clases 1 a 4

Complejidad Computacional
(desde los años 1960)
Máquinas de Turing
clases 5 a 9

Correctitud de Programas
(desde los años 1970)
Programas tipo Pascal
Clases 10 a 13

Matemática y Lógica (desde los años 1900)



Introducción
viajes de ida y vuelta

Viaje de ida (primeros años de la carrera)
Algoritmos
Estructuras de datos
Lenguajes de programación
Arquitecturas de computadoras
Paradigmas de programación
Matemáticas
Etc.

Viaje de vuelta (últimos años, con más madurez)
Profundización de fundamentos de computación
Foco en algorítmica, eficiencia, verificación
- Autómatas
- Lenguajes formales
- Lógica, combinatoria, conjuntos, grafos
- Diagonalización
- Reducciones
- Pruebas
- Especificaciones formales
- Etc.

Introducción
un poco de historia

Comienzos del siglo XX
Hilbert. La búsqueda de un esquema mecánico para demostrar todas las verdades matemáticas.

1931
Hay verdades matemáticas que no se pueden demostrar mecánicamente. Gödel.

1936
Hay enunciados matemáticos que no se pueden decidir mecánicamente. Church, Turing, etc.
Origen de la computabilidad. Problemas computables y no computables. Máquinas de Turing. Otros modelos.

Desde los 1940
Computadoras. Von Neumann y Turing.

Desde los 1960
Complejidad computacional. Problemas fáciles y difíciles. Jerarquías temporal y espacial.
Máquinas aleatorias, máquinas cuánticas, inteligencia artificial (¿un paradigma distinto?).
Rabin, Cook, Levin, Shor, Grover, etc.

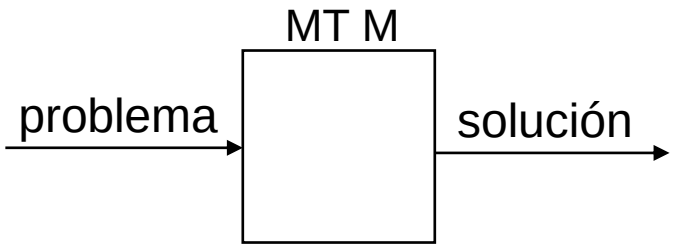
Desde los 1970
Correctitud de programas. Verificación formal. Hoare, Dijkstra, etc.

Clase teórica 1

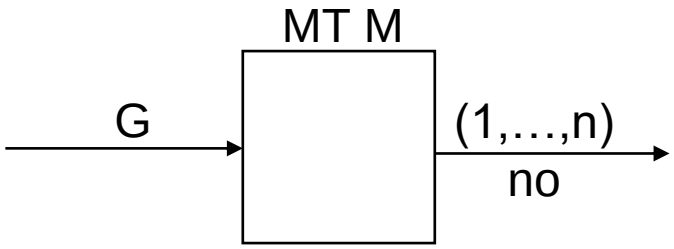
Máquinas de Turing (MT)

Máquina de Turing (MT)

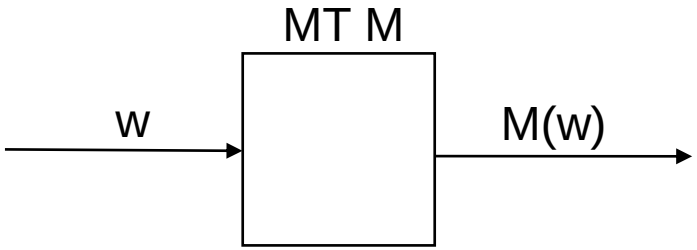
Modelo muy simple de una computadora, con el que estudiaremos computabilidad y complejidad computacional.



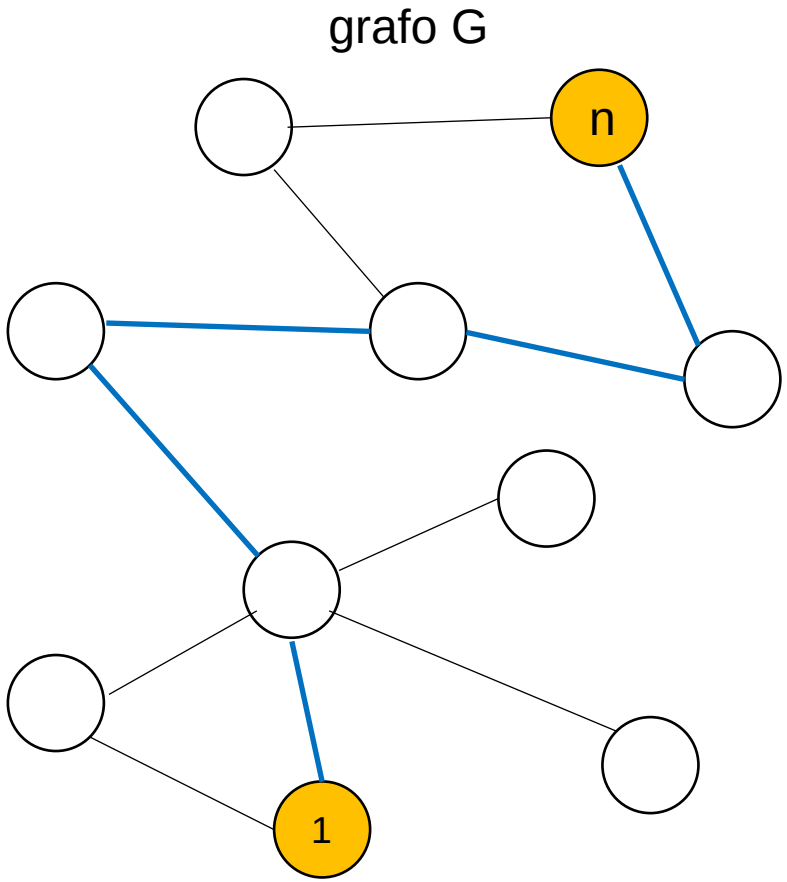
El **algoritmo** de M **devuelve** una solución al problema.



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.

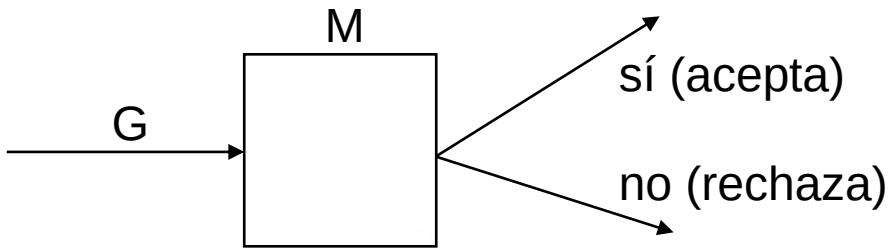


Genéricamente, w es la **entrada** y M(w) es la **salida.**

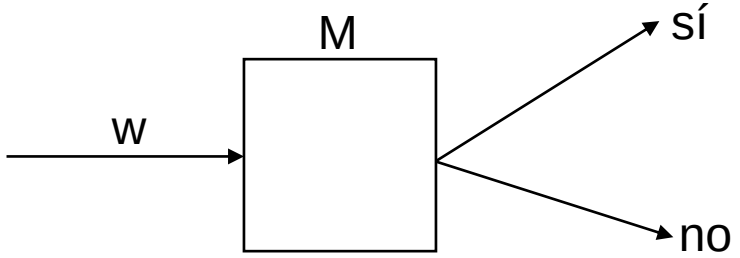


Grafo con un camino del vértice 1 al vértice n. En este caso la MT M lo genera.

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** como el que vimos antes. La salida es “sí” o “no”.

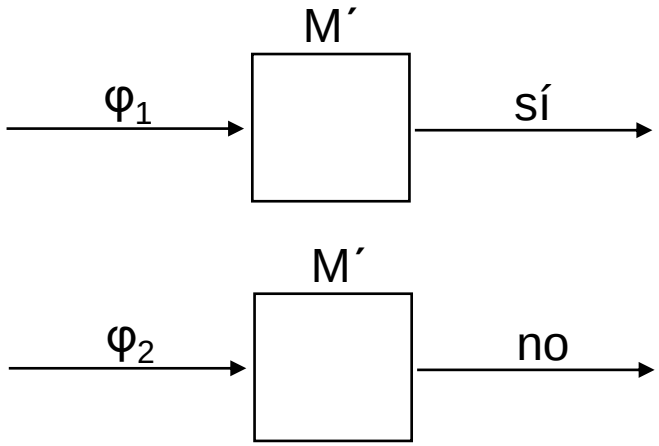


M **acepta** G sii G tiene un camino del vértice 1 al vértice n.



Genéricamente, M **acepta** o **rechaza** w.

Otro ejemplo:
Dada una fórmula booleana φ,
¿existe una asignación A de valores de verdad que la satisface?
Por ejemplo,
Si $\varphi_1 = (x_1 \vee x_2) \wedge (x_3 \wedge x_4)$, A_1 = (V, F, V, V) la satisface.
Si $\varphi_2 = x_1 \wedge \neg x_1$, no existe A_2 que la satisfaga.

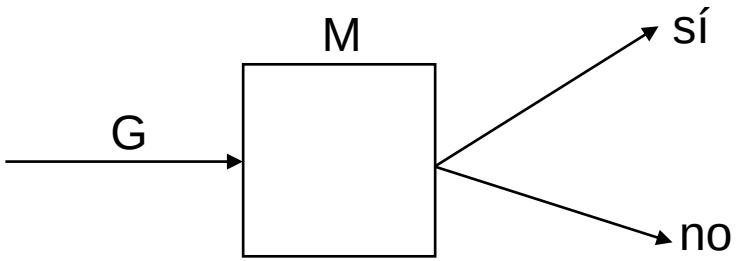


Omitiremos en general los problemas de **búsqueda**, trabajaremos con los problemas de **decisión:**
son más simples para estudiar la computabilidad y la complejidad computacional.

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 problema del camino en un grafo G:



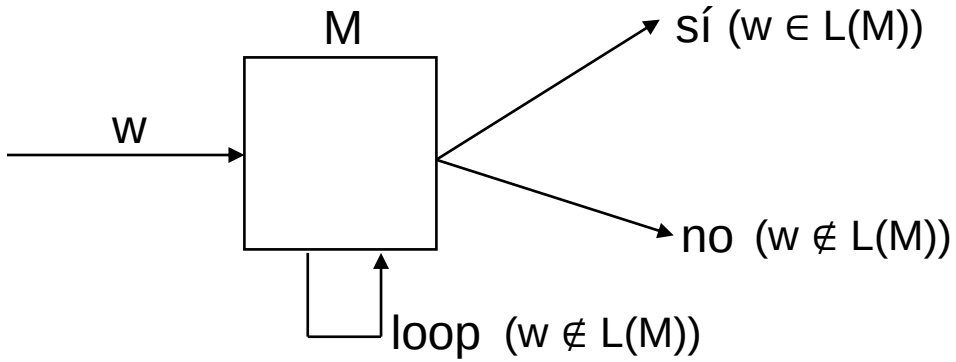
La MT M **acepta** (o r**econoce**) el lenguaje de las cadenas G que representan grafos con un camino del vértice 1 al vértice n.

En otras palabras, la MT M **acepta el lenguaje L(M)** = $\{G_1, G_2, G_3, \ldots\}$,
que tiene todas las cadenas G_i que representan grafos con un camino del vértice 1 al vértice n.

Hasta nuevo aviso entonces, **problemas y lenguajes serán sinónimos.**

¡Inconveniente!:

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

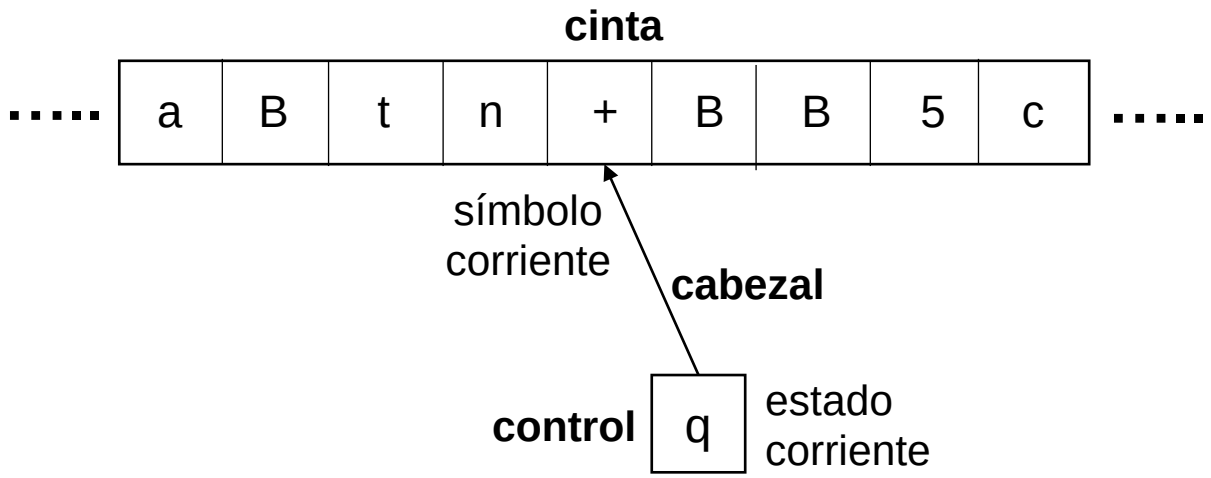


Es decir, para algunos problemas **NO existen MT que siempre paran**.

Es el caso de los problemas **indecidibles:**

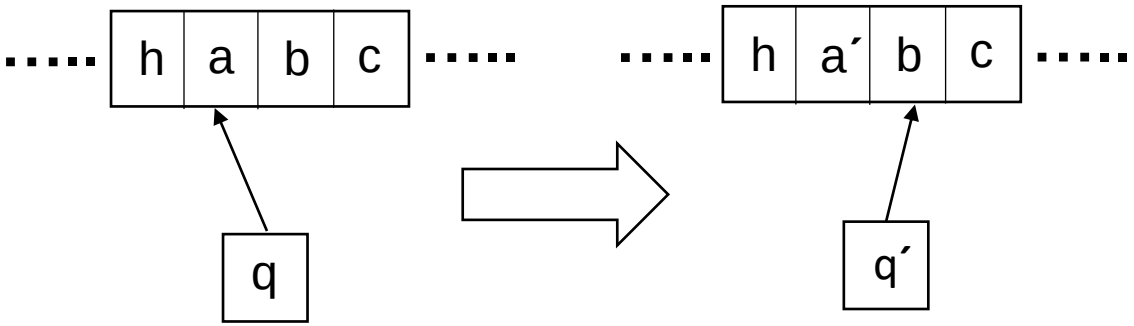
Toda MT M que pretenda resolver un problema de esta clase cumplirá que en algunos casos negativos,
es decir en algunos casos de entradas w que no pertenezcan al lenguaje L(M),
en lugar de responder “no” en tiempo finito,
la máquina **“loopeará”, no terminará, no emitirá respuesta alguna.**

Descripción formal de una MT



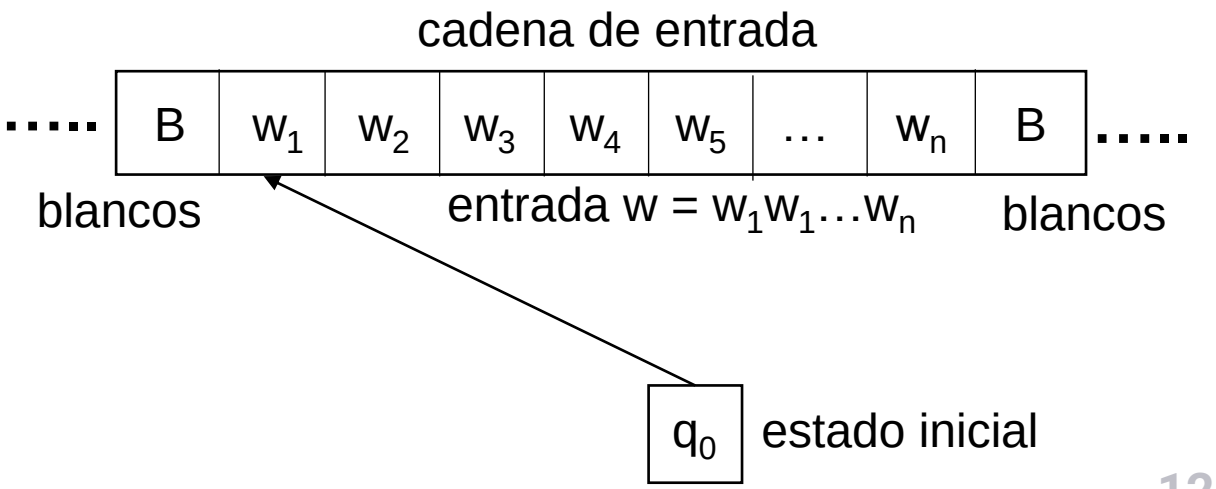
Tesis de Church-Turing
¡Todo lo computable puede ser computado por una Máquina de Turing!

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



un paso de la MT

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



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

- Q es el conjunto de **estados** de M.

- Σ es el **alfabeto** admitido por la cinta de M.

  Σ incluye al símbolo blanco B.

  Las cadenas de entrada no admiten blancos.

- q_0 es el **estado inicial** de M.

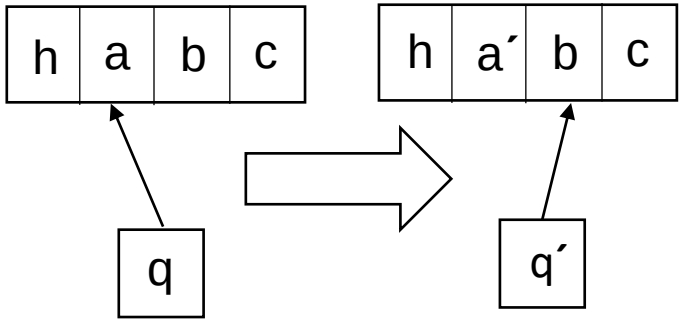
- q_A y q_R son los **estados finales** de aceptación y rechazo de M, respectivamente.

- δ es la **función de transición** de M (la especificación de su comportamiento):

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

Dado un estado corriente de Q, y un símbolo corriente de Σ,
la máquina pasa eventualmente a un nuevo estado de Q, o a q_A, o a q_R,
modifica eventualmente el símbolo corriente de Σ,
y se mueve un lugar a la derecha (R), a la izquierda (L), o no se mueve (S).
Se detiene si en algún momento alcanza el estado q_A o el estado q_R.

Ejemplo de transición
δ(q, a) = (q´, a', R)



Ejemplo de construcción de una MT

L = {$a^n b^n$ | n ≥ 1}. Es decir, L = {ab, aabb, aaabbb, …}. Queremos construir una MT M que acepte L.

1. **Idea General.** Un posible algoritmo podría ser ir marcando el primer símbolo *a*, el primer símbolo b, el segundo símbolo *a*, el segundo símbolo b, y así hasta detectar al final si las cantidades son iguales, o si por la mitad aparecen otros símbolos, o si el orden de los *a* y los b no es el correcto.

   Por ejemplo, supongamos la entrada **aaaaabbbbb**. M se comportaría de la siguiente manera:

   aaaaabbbbb

   αaaaabbbbb

   αaaaaβbbbb

   ααaaaβbbbb

   ααaaaββbbb

   ……..

   αααααβββββ

   En este caso, la MT M debe aceptar la entrada. Casos de rechazo serían: aabbb, aab, bbaa, aacbb, etc.

2. Construcción de la MT

Definición de la MT $M = (Q, \Sigma, \delta, q_0, q_A, q_R)$:

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

q_0 : estado inicial

q_a : M busca una *a*

q_b : M busca una b

q_L : M vuelve a buscar otra *a*

q_H : no hay más *a*

Alfabeto $\Sigma = \{a, b, \alpha, \beta, B\}$

Ayuda memoria del algoritmo con la entrada aaabbb:

aaabbb
αaabbb
αaaβbb
ααaβbb
ααaββb
αααββb
αααβββ

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. $\delta(q_H, B) = (q_A, B, S)$

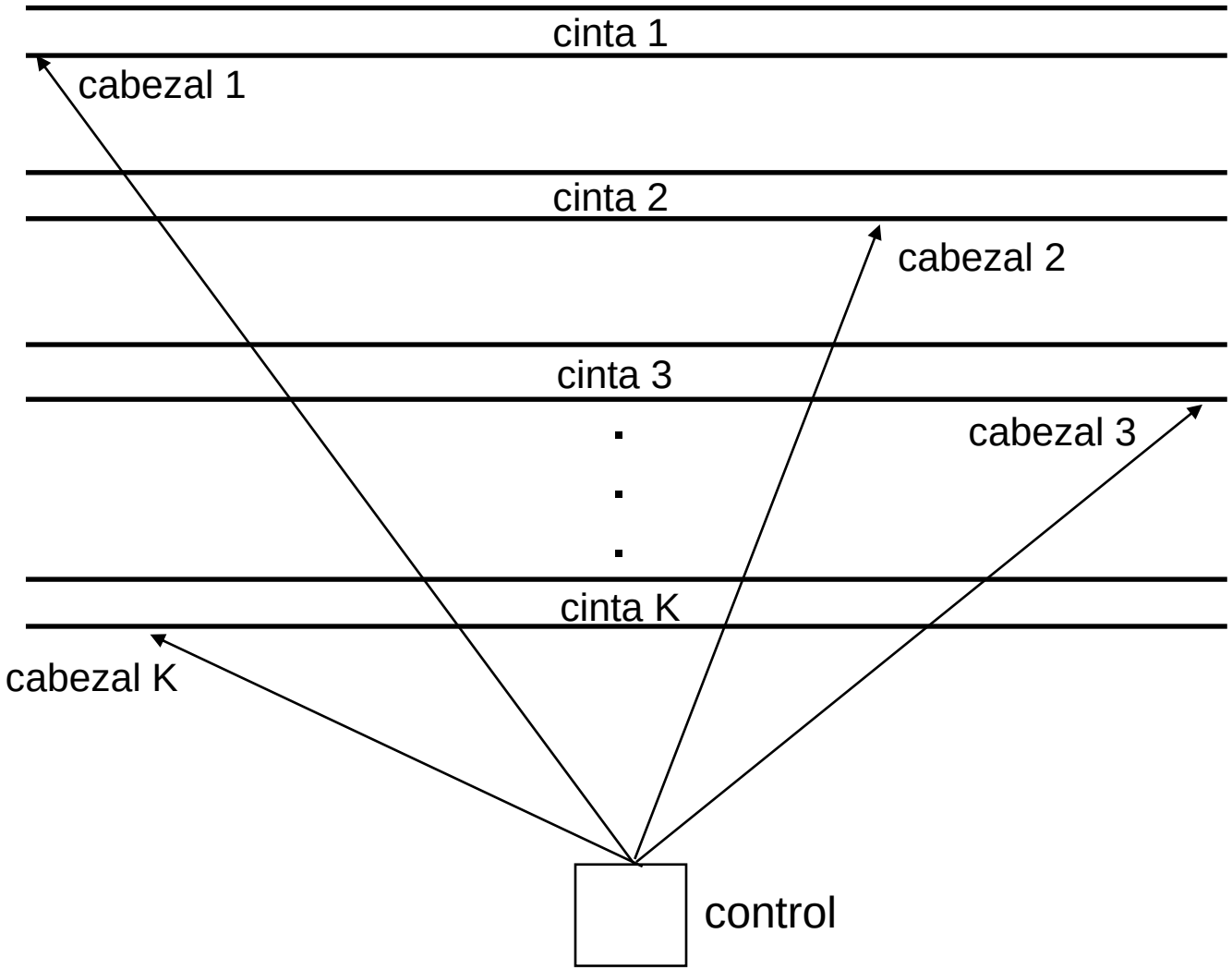
Todos los pares omitidos corresponden a rechazos, no se especifican para abreviar la descripción. Por ejemplo: $\delta(q_0, B)$, $\delta(q_0, b)$, $\delta(q_b, B)$, $\delta(q_H, b)$, etc.

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).

Otro modelo de MT: MT con varias cintas



Características generales de una MT con K cintas:

- La primera cinta contiene la entrada.
- La MT **en un solo paso** puede modificar el estado corriente, los símbolos corrientes de todas las cintas, y moverse distinto en cada cinta. Más en detalle:

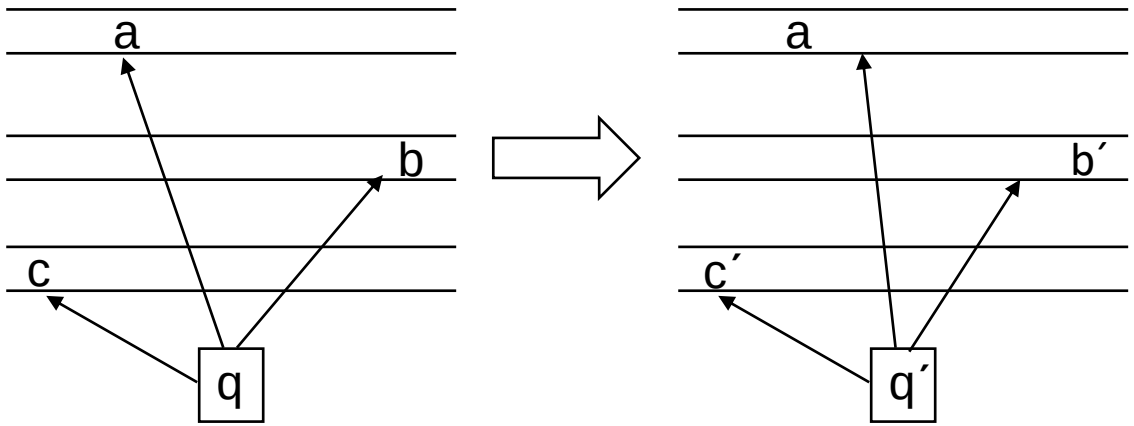
Teorema

Dada una MT M_1 con K cintas, existe una MT M_2 equivalente, es decir que acepta el mismo lenguaje, con 1 cinta.
Comentario: si M_1 acepta una cadena en h pasos, M_2 lo hace en unos h^2 pasos.

1. En un paso lee un estado y una K-tupla de símbolos (los apuntados por los cabezales 1 a K).
2. Modifica eventualmente el estado corriente.
3. Modifica eventualmente los símbolos corrientes de las cintas.
4. Se mueve independientemente en cada cinta (derecha, izquierda o no se mueve).

- Por ejemplo, para el caso de 3 cintas, una transición podría ser:

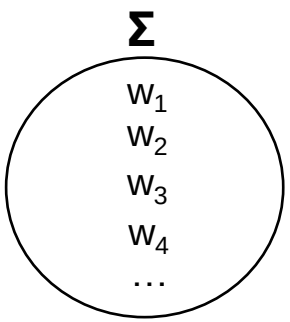
  δ(q, (a, b, c)) = (q', (a, R), (b', L), (c´, S))



Anexo de la clase teórica 1

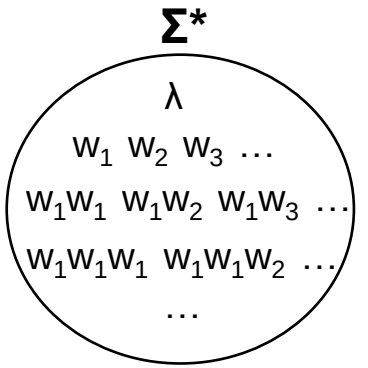
Máquinas de Turing (MT)

Acerca de los lenguajes



- Σ es un **alfabeto** o conjunto de símbolos.

  $\Sigma = \{w_1, w_2, w_3, w_4, \ldots\}$



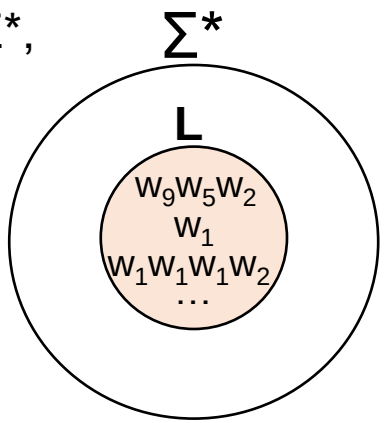
- Σ* es el **lenguaje** o conjunto de cadenas de símbolos generado a partir de Σ.

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

  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 **universal.**

  $L \subseteq \Sigma^*$.



Por ejemplo, las cadenas a^nb^n, los grafos con un camino de 1 a n, las fórmulas booleanas satisfactibles, etc.

- **Operaciones** típicas entre lenguajes:

  Intersección: $L_1 \cap L_2$

  Unión: $L_1 \cup L_2$

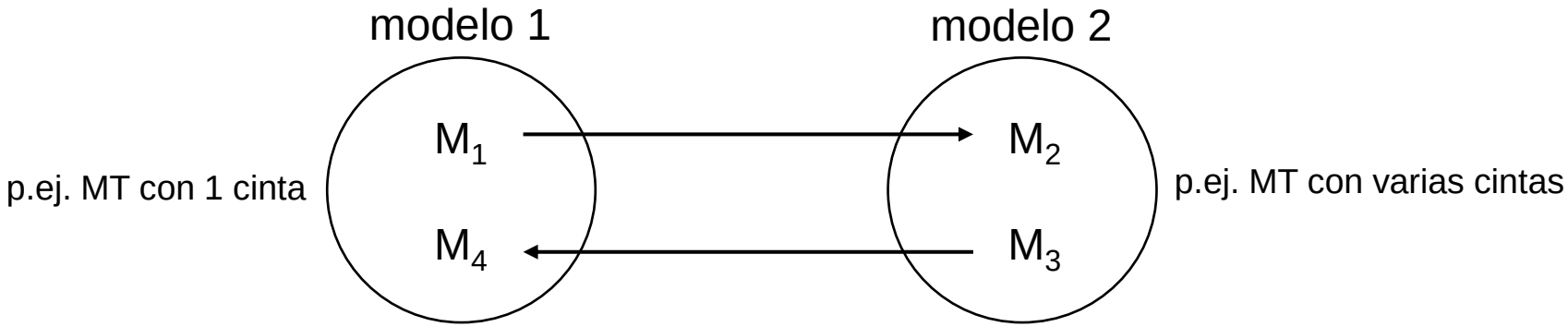
  Diferencia: $L_1 - L_2$

  Complemento: L^C, que con respecto al conjunto universal Σ*, equivale a $(\Sigma^* - L)$

  Producto (o Concatenación): $L_1 . L_2$

Modelos equivalentes de MT

- Dos MT son **equivalentes** si aceptan el mismo lenguaje.
- Dos **modelos de MT** son **equivalentes** si dada una MT de un modelo existe una MT equivalente del otro.



- Ejemplos de modelos de MT equivalentes al modelo de MT con **1 cinta**: MT con **varias cintas**, MT con **cintas semi-infinitas**, MT con **2 cintas y un solo estado**, MT **no determinísticas** (a partir de un estado y un símbolo pueden avanzar de distintas maneras), etc.

- Ejemplos de modelos computacionales equivalentes al modelo de las MT: **máquinas RAM**, **circuitos booleanos**, **lambda cálculo**, **funciones recursivas parciales**, **gramáticas**, **programas Java**, etc.

- Todo esto refuerza la **Tesis de Church-Turing**.

Clase práctica 1

Máquinas de Turing (MT)

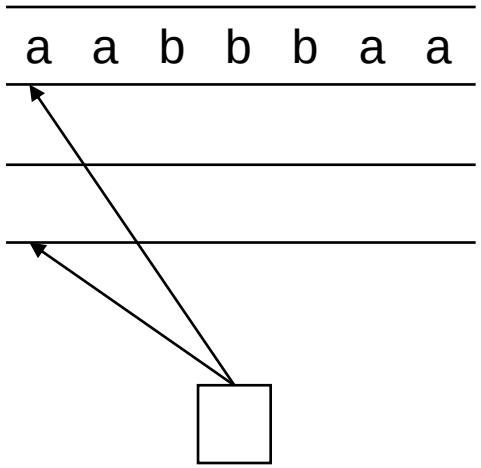
Ejemplo de construcción de una MT con varias cintas

L = {w | w es una cadena con cero o más símbolos *a* o b, y es un palíndromo o "capicúa"}
Comentario: w es un palíndromo si es igual a su reverso. Por ejemplo, w = aba es un palíndromo.

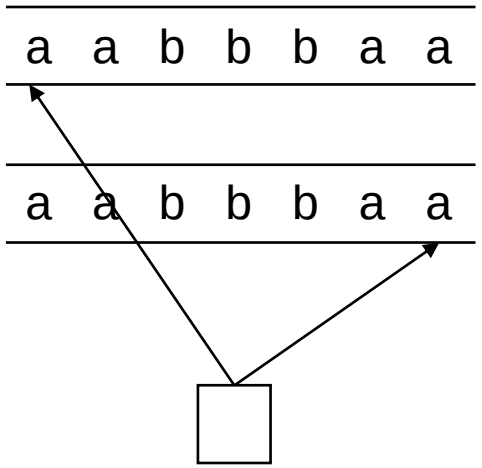
Queremos construir una MT que acepte L.

Idea general: Una MT M con 2 cintas que hace:
1. Copia la entrada, de la cinta 1 a la cinta 2.
2. Vuelve el cabezal de la cinta 1 a la izquierda y deja el cabezal de la cinta 2 a la derecha.
3. Se desplaza a la derecha en la cinta 1 y a la izquierda en la cinta 2, comparando cada vez los símbolos apuntados. Si los pares de símbolos comparados son siempre iguales, acepta. Si no, rechaza.

Por ejemplo:



conjuntos de pasos 1 y 2



conjuntos de pasos 3

M compara el 1er *a* de arriba con el último *a* de abajo, luego el 2do *a* de arriba con el anteúltimo *a* de abajo, y así siguiendo hasta llegar a los blancos en las 2 cintas.

Función de transición de la MT M

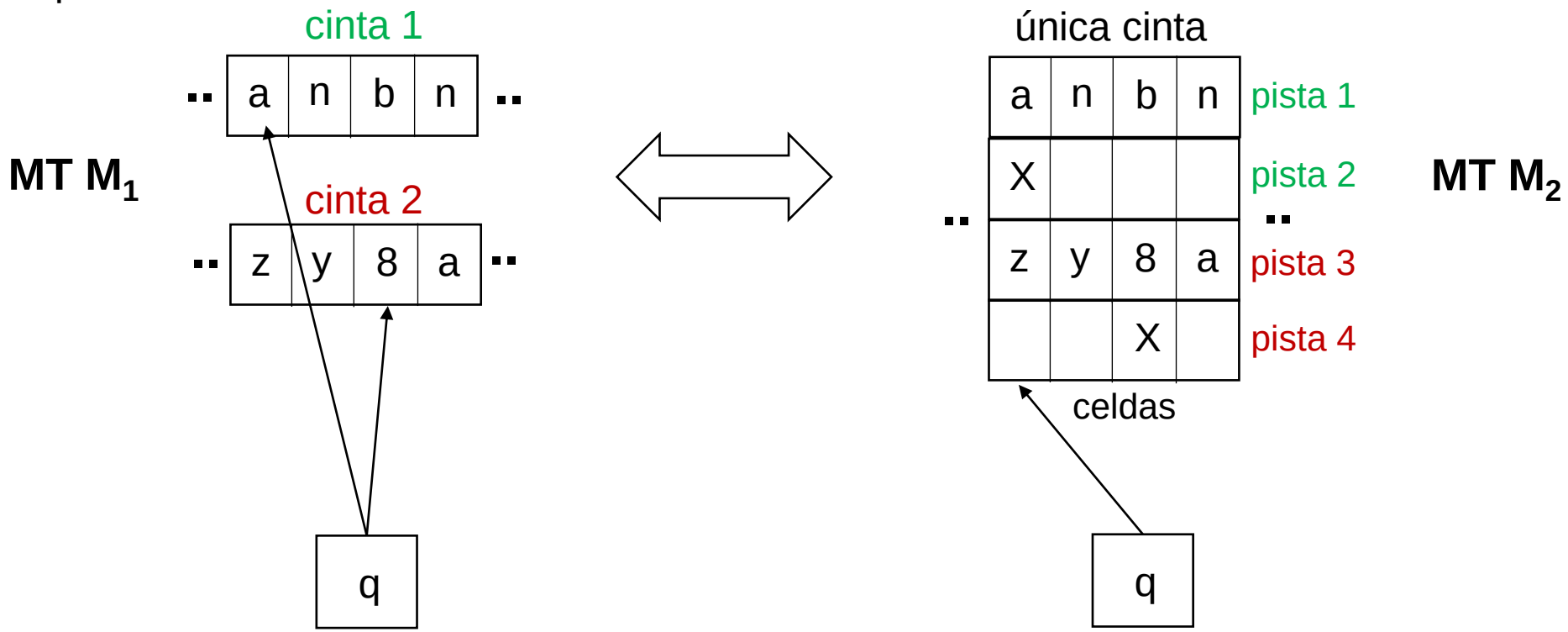
q_0: copia de la cinta 1 a la 2; q_1: reposicionamiento en la cinta 1; q_2: comparación de las 2 cintas

	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

Las celdas en blanco son casos de rechazo de la MT.

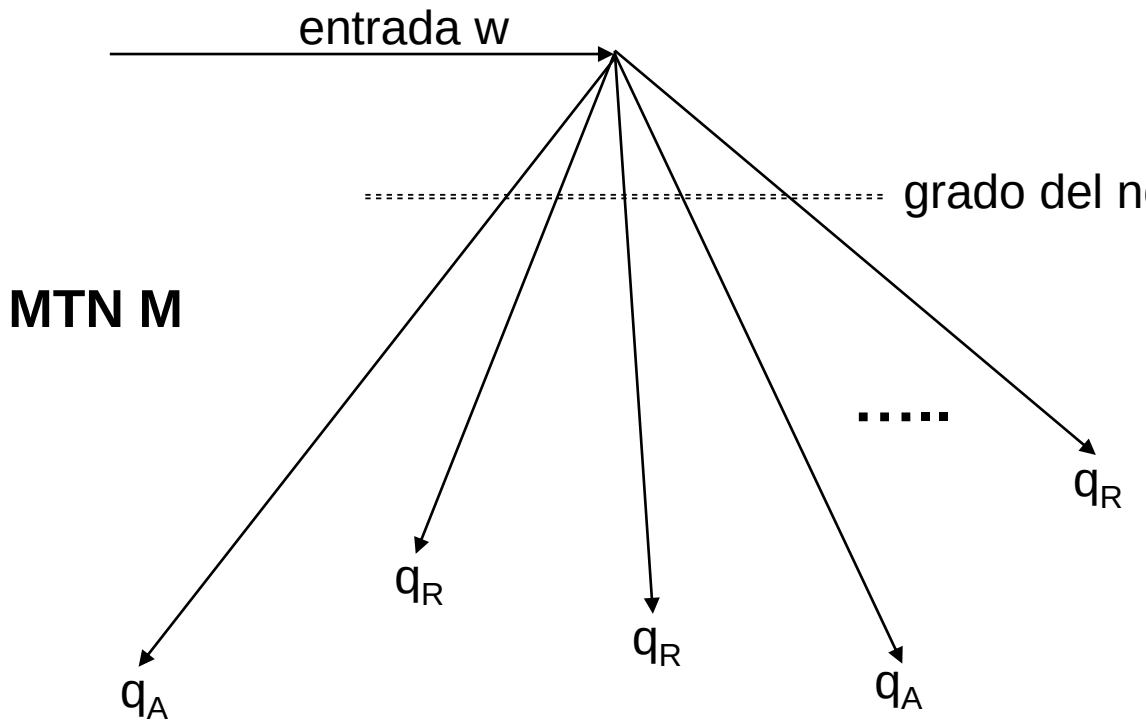
Simulación de una MT con varias cintas mediante una MT con 1 cinta

- Aunque no es intuitivo, una MT con varias cintas **no tiene más potencia computacional** que una MT con 1 cinta (lo que hace una MT con K cintas lo puede hacer una MT con 1 cinta).
- Idea de prueba con 2 cintas:



- Idea general: uso de **cintas con pistas**. En el ejemplo, las 2 cintas de M_1 se simulan con 1 cinta con 4 pistas de M_2 (sus celdas almacenan 4-tuplas de símbolos, para los contenidos y los cabezales).
- Un paso de M_1 se simula con varios pasos de M_2. Se pueba que el retardo es **cuadrático**.

Modelo de máquina de Turing no determinística (MTN)



Las MTN son máquinas "ficticias", no modelizan una computadora, sirven para abreviar descripciones de MTD.

- Para un mismo par (q, w), la máquina puede responder de más de una manera.
- Una MTN acepta sii **al menos** una computación acepta.
- Las MTN sirven para expresar abreviadamente el comportamiento de las MT determinísticas (MTD).
- Para simular una MTN con una MTD hay que recorrer en el peor caso todas sus computaciones. Asumiendo que toda computación hace a lo sumo h pasos, ejecutar secuencialmente todas las computaciones requiere unos c^h pasos, siendo c el grado del no determinismo (retardo **exponencial**).

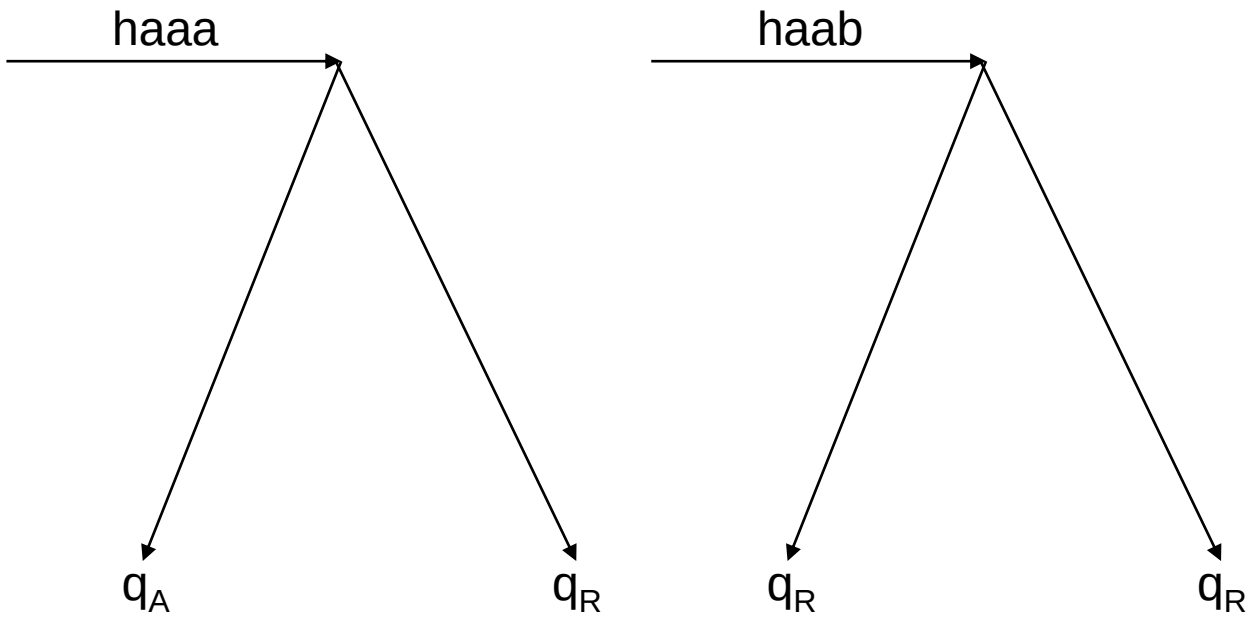
Ejemplo sencillo de uso de MTN

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:*

Solución propuesta:

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. $\Delta(q_b, B) = \{(q_A, B, S)\}$

Por ejemplo:

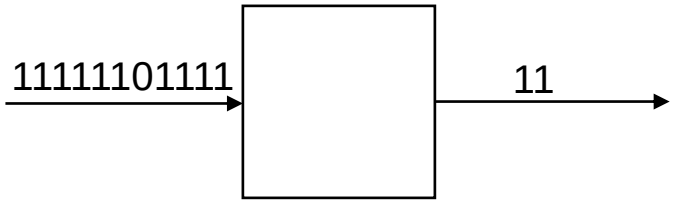


Otras visiones de MT

Visión de MT calculadora (el caso que vimos al comienzo, el más general, para problemas de búsqueda)

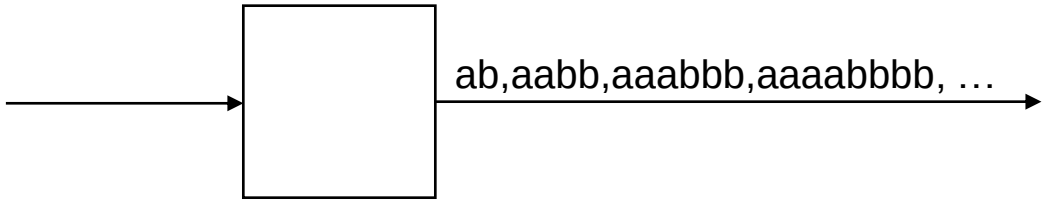
Ejemplo: construir una MT que reste 2 números representados en notación unaria y separados por un cero.
Por ejemplo, dada la entrada 11111101111, obtener la salida 11 (6 – 4 = 2).
Idea general: tachar el primer 1 antes del 0, luego el primer 1 después del 0, luego el segundo 1 antes del 0, y así hasta tachar al final el 0.

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



Visión de MT generadora

Ejemplo: construir una MT que genere todas las cadenas de la forma $a^n b^n$, con n ≥ 1. Es decir, que en una cinta especial de salida **genere las cadenas ab, aabb, aaabbb, aaaabbbb, etc.**
Idea general:
(1) i := 1
(2) imprimir i veces *a*, imprimir i veces b, e imprimir una coma
(3) i := i + 1 y volver a (2)



Teorema: Existe una MT M que acepta un lenguaje L sii existe una MT M´ que genera el lenguaje L.