FUNDAMENTOS DE TEORÍA DE LA COMPUTACIÓN 2025

Parte 1. Teoría de la Computación

PROF. RICARDO ROSENFELD

Computabilidad (4 clases)
Complejidad computacional (3 clases)

Parte 2. Lógica, Int. Artificial y Verificación de Programas

Lógica e inteligencia artificial (6 clases)
PROF. CLAUDIA PONS

Verificación de programas (1 clase)
PROF. RICARDO ROSENFELD

Datos generales de la materia

Plantel docente
Ricardo Rosenfeld (teoría clases 1 a 7 y 14), Claudia Pons (teoría clases 8 a 13).
Leandro Mendoza, Pedro Dal Bianco, Matías Manzín (práctica clases 1 a 14).

Horarios y aulas
Teoría: lunes de 14 a 16 hs. Aula 10B.
Práctica: martes de 19 a 21 hs. Aula 10A.
Consultas eventuales: viernes de 19 a 21 hs. Aula 10B.

Estructura de la materia y calendario
Parte 1. 7 clases de **computabilidad y complejidad computacional** (1er examen promocional **6 de mayo**).
Parte 2. 7 clases de **lógica, int. artificial y verificación de programas** (1er examen promocional **8 de julio**).
2dos exámenes promocionales de las partes 1 y 2 (los que no aprobaron): **4 y 5 de agosto,** respectivamente.

Trabajos prácticos semanales
Se recomienda trabajar en parejas. No se entregan, pero se recomienda hacerlos para llegar bien entrenados al examen (¡y para aprender!).

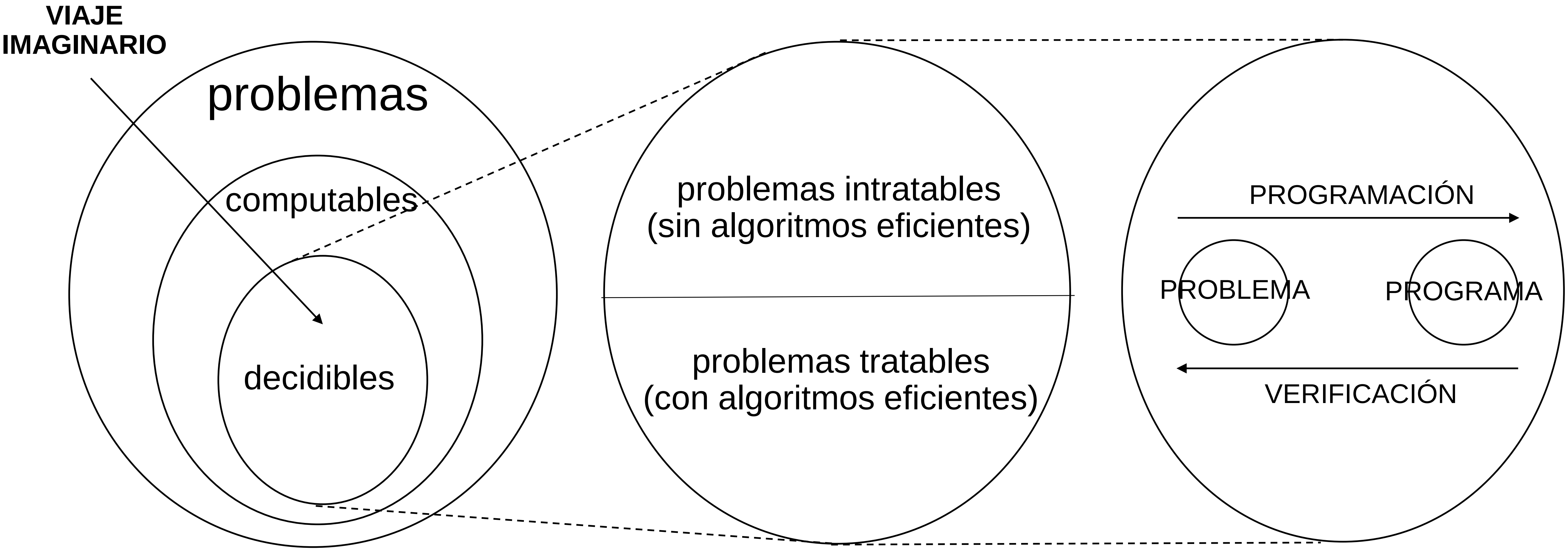
Consultas
Presencial en las clases prácticas, y virtual durante toda la semana (7x24) a través de la plataforma **IDEAS.**

Material

<u>**Plataforma IDEAS (inscribirse los que aún no lo han hecho)**</u>

- Carpeta Materiales y Actividades
  - Plan de la Materia
  - Libros y Apuntes
  - Clases y Trabajos Prácticos (clases y artículos de interés, distribuidos entre las partes 1 y 2)
- **Libros de cabecera para las clases 1 a 7 y 14 (en la Plataforma IDEAS)**
  Rosenfeld. 2025. Computabilidad y Complejidad Computacional (versión preliminar). EDULP.
  Rosenfeld. 2024. Verificación de Programas. Programas Secuenciales y Concurrentes. EDULP.
- **Algunos libros complementarios para las clases 1 a 7 y 14**
  En la Plataforma IDEAS:
  Rosenfeld & Irazábal. 2013. Computabilidad, Complejidad Computacional y Verificación de Programas. EDULP.
  Rosenfeld & Irazábal. 2010. Teoría de la Computación y Verificación de Programas. McGraw Hill y EDULP.
  Pons, Rosenfeld y Smith. 2017. Lógica para Informática. EDULP.
  En la Biblioteca:
  Hopcroft & Ullman. 1979. Introduction to Automata Theory, Language & Computation. Prentice-Hall.
  Sipser. 1997. Introduction to the Theory of Computation. PWS Publishing.
  Papadimitriou. 1995. Computational Complexity. Addison-Wesley.
  Arora & Barak. 2007. Computational Complexity: A Modern Approach. Princeton Univ.
  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.

Introducción

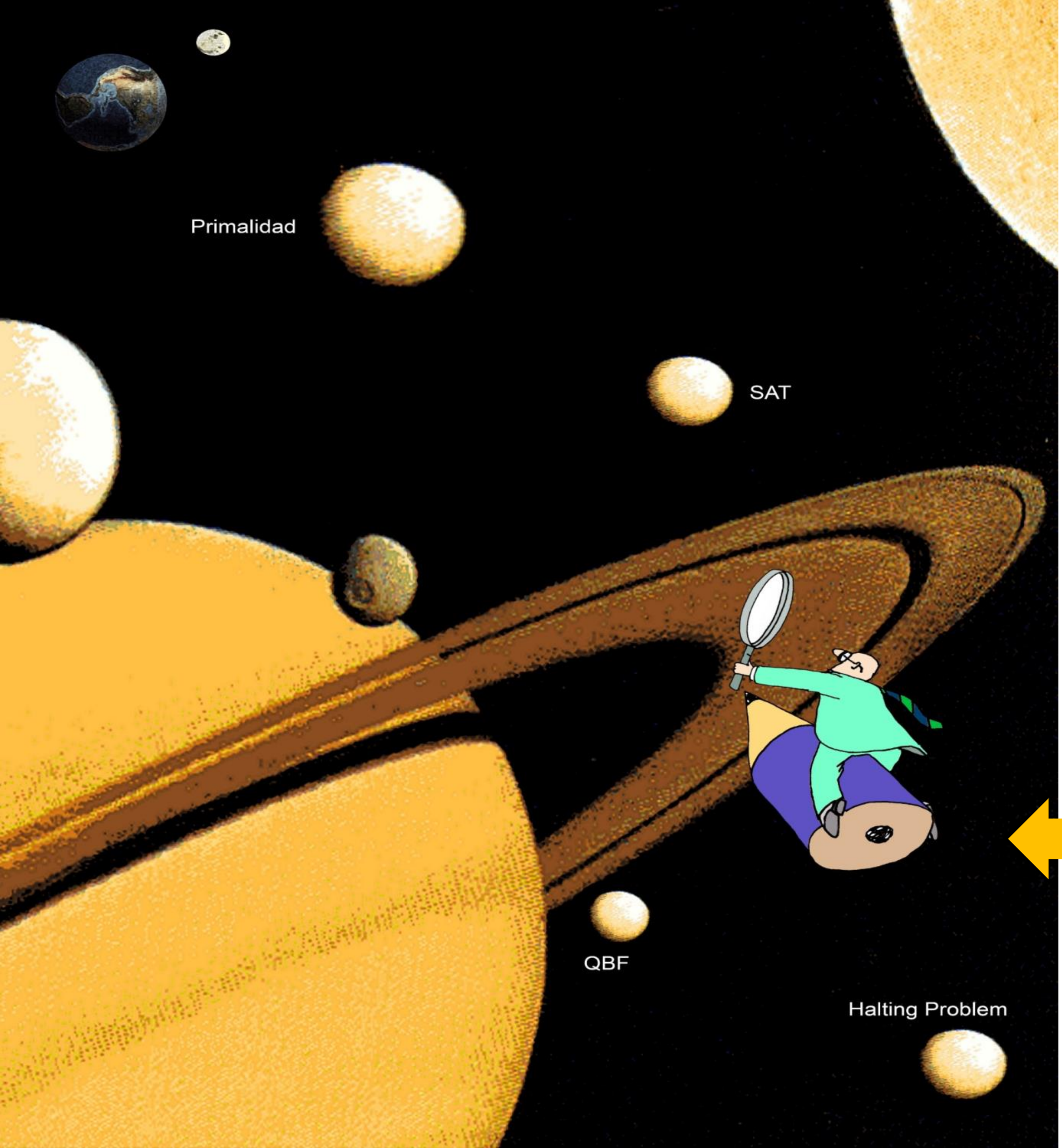


Computabilidad
(años 1930)

Complejidad Computacional
(años 1960)

Correctitud de Programas
(años 1970)

Matemática y Lógica

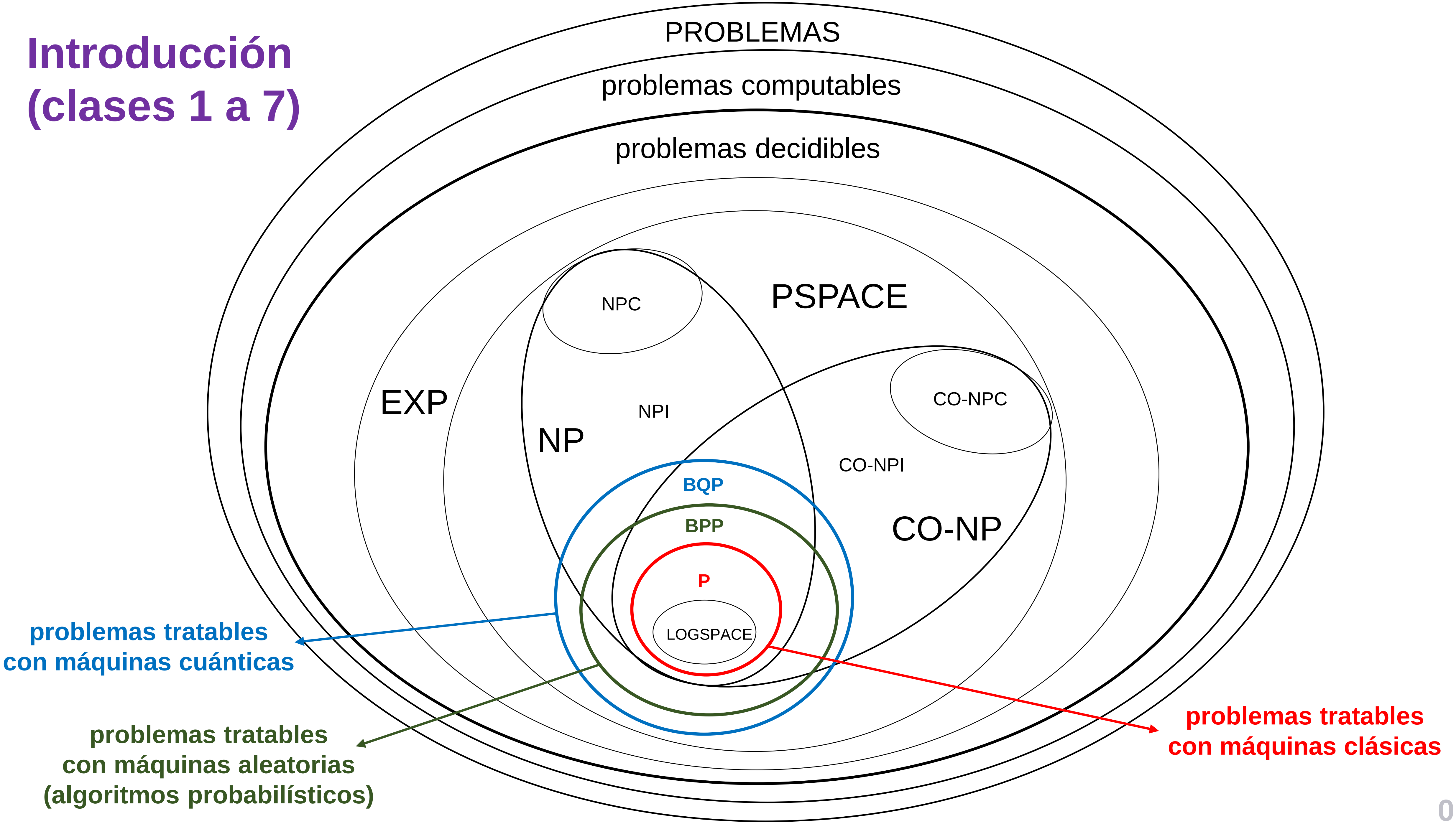


Introducción
otra mirada

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 en fundamentos de la computación
<u>Foco en algorítmica, eficiencia y verificación</u>
- Autómatas
- Lenguajes formales
- Lógica, combinatoria, conjuntos, grafos
- Diagonalización
- Reducciones
- Pruebas
- Especificaciones formales
- Etc.

Introducción (clases 1 a 7)



Introducción
(clase 14)

$\{x \geq 0 \wedge y > 0\}$

S_{div} :: **c := 0 ; r := x ;**
while r ≥ y
do r := r – y ;
c := c + 1
od

$\{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$

programa de división entre x e y

prueba del programa

1. $\{x = y \,.\, c + x \wedge x \geq 0\}$ r := x $\{x = y \,.\, c + r \wedge r \geq 0\}$
2. $\{x = y \,.\, 0 + x \wedge x \geq 0\}$ c := 0 $\{x = y \,.\, c + x \wedge x \geq 0\}$
3. $\{x = y \,.\, 0 + x \wedge x \geq 0\}$ c := 0 ; r := x $\{x = y \,.\, c + r \wedge r \geq 0\}$
4. $(x \geq 0 \wedge y > 0) \rightarrow (x = y \,.\, 0 + x \wedge x \geq 0)$
5. $\{x \geq 0 \wedge y > 0\}$ c := 0 ; r := x $\{x = y \,.\, c + r \wedge r \geq 0\}$
6. $\{x = y \,.\, (c + 1) + r \wedge r \geq 0\}$ c := c + 1 $\{x = y \,.\, c + r \wedge r \geq 0\}$
7. $\{x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0\}$ r := r – y $\{x = y \,.\, (c + 1) + r \wedge r \geq 0\}$
8. $\{x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0\}$ r := r – y ; c := c + 1 $\{x = y \,.\, c + r \wedge r \geq 0\}$
9. $(x = y \,.\, c + r \wedge r \geq 0 \wedge r \geq y) \rightarrow (x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0)$
10. $\{x = y \,.\, c + r \wedge r \geq 0 \wedge r \geq y\}$ r := r – y ; c := c + 1 $\{x = y \,.\, c + r \wedge r \geq 0\}$
11. $\{x = y \,.\, c + r \wedge r \geq 0\}$ while r ≥ y do r := r – y ; c := c + 1 od $\{x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)\}$
12. $\{x \geq 0 \wedge y > 0\}$ c := 0 ; r := x ; while r ≥ y do r := r – y ; c := c + 1 od $\{x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)\}$
13. $(x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)) \rightarrow (x = y \,.\, c + r \wedge r < y \wedge r \geq 0)$
14. **$\{x \geq 0 \wedge y > 0\}$ c := 0 ; r := x ; while r ≥ y do r := r – y ; c := c + 1 od $\{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$**

Introducción (algunos hitos históricos)

Comienzos del siglo XX. Búsqueda de un esquema mecánico para demostrar todas las verdades matemáticas (D. Hilbert).

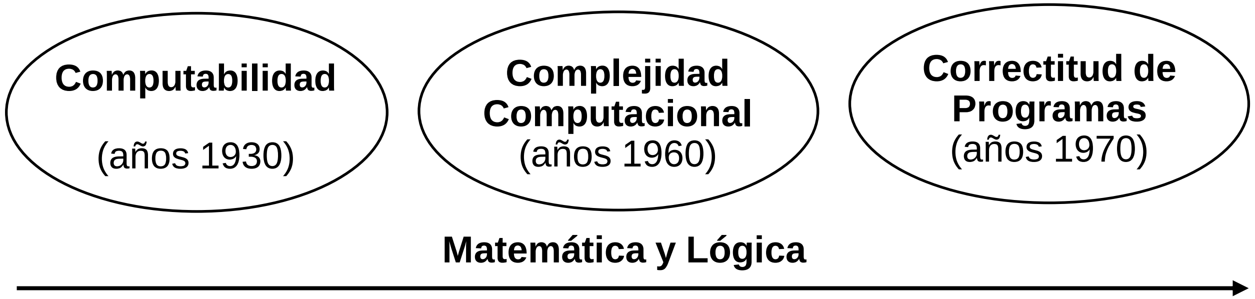
1931. Hay verdades matemáticas que no se pueden demostrar mecánicamente (K. Gödel).

1936. Hay enunciados matemáticos que no se puede decidir mecánicamente si son verdaderos o falsos (A. Church, A. Turing). **Origen de la computación**. Máquina de Turing como modelo de computación aceptado.

Años 1940. Primeras computadoras.

Años 1960. Origen de la complejidad computacional. Problemas fáciles y difíciles en términos de tiempo y espacio. Evolución: algoritmos probabilísticos, algoritmos cuánticos, pruebas interactivas, IA (inteligencia artificial), etc.

Años 1970. Origen de la correctitud de programas (verificación formal, semántica de lenguajes). Evolución: verificación semi-automática (con asistencia), verificación automática (*model checking*), etc.



Clase teórica 1

La máquina de Turing

Repaso de algunos conceptos matemáticos básicos

- Un ***conjunto*** es una colección de *elementos*.

  Por ejemplo, A = {a, b, c} es un conjunto de tres elementos: *a*, b y c. Se dice que *a pertenece* a A (**a ∈ A**).

  Dos conjuntos A y B son iguales (**A = B**) si tienen los mismos elementos.

  Por ejemplo, {casa, árbol, cielo} = {árbol, cielo, casa}.

  Un conjunto A es un *subconjunto* de un conjunto B o está *incluido* en B (**A $\subseteq$ B**) si los elementos de A pertenecen a B, y está incluido *estrictamente* en B (**A $\subset$ B**) si A $\subseteq$ B y A $\neq$ B.

  Por ejemplo, {1, 2} $\subseteq$ {1, 2, 3, 5}. Y también se cumple {1, 2} $\subset$ {1, 2, 3, 5}.

  Dados dos conjuntos A y B:
  su *intersección* (**A ∩ B**) es el conjunto de los elementos que están en A **y** en B,
  su *unión* (**A ∪ B**) es el conjunto de los elementos que están en A **o** en B,
  y su diferencia (**A – B**) es el conjunto de los elementos que están en A **y no** están en B.

  Por ejemplo, {1, 2, 3, 4, 5} ∩ {4, 5, 6} = {4, 5},
  {1, 2, 3, 4, 5} ∪ {4, 5, 6} = {1, 2, 3, 4, 5, 6},
  {1, 2, 3, 4, 5} – {4, 5, 6} = {1, 2, 3}.

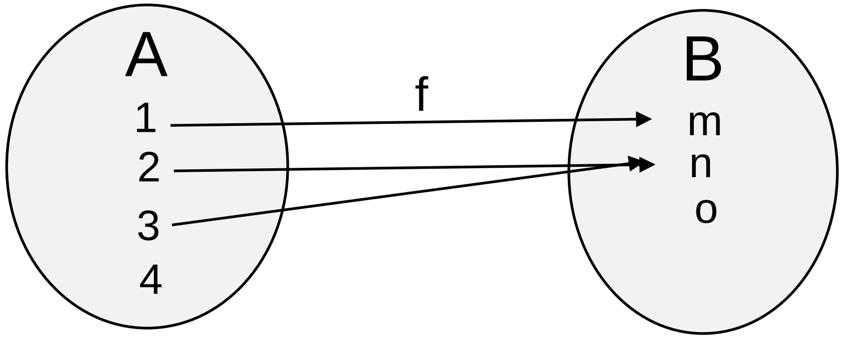
Repaso de algunos conceptos matemáticos básicos (continuación)

- Una ***función*** de un conjunto A a un conjunto B (**f : A → B**) es un conjunto de pares (a, b), con a ∈ A y b ∈ B, tales que a lo sumo existe un par (a, b) para todo a ∈ A.

  f(a) = b expresa que el par (a, b) pertenece a la función f.

  Por ejemplo, si A = {1, 2, 3, 4} y B = {m, n, o},
  entonces {(1, m), (2, n), (3, n)} es una función f de A a B,
  que también se puede definir así: f(1) = m, f(2) = n, f(3) = n.

  Gráficamente:



El conjunto A es el *dominio* y el conjunto B es el *codominio* de la función f.

Repaso de algunos conceptos matemáticos básicos (continuación)

- Un ***alfabeto*** es un conjunto finito de símbolos.

  Por ejemplo, $\Sigma = \{a, b, c\}$ es un alfabeto de tres símbolos.

- Un ***lenguaje*** con alfabeto Σ es un conjunto de cadenas finitas de símbolos de Σ.

  Por ejemplo, si $\Sigma = \{a, b, c\}$, un lenguaje L con alfabeto Σ es $L = \{aaa, b, ababab, ccb\}$.

- **Operaciones típicas** entre lenguajes (las mismas que entre conjuntos):

  Por ejemplo, si $L_1 = \{aa, ab, ba, bb\}$ y $L_2 = \{ab, aabb, aaabbb, aaaabbbb\}$,

  $L_1 \cap L_2 = \{ab\}$.
  $L_1 \cup L_2 = \{aa, ab, ba, bb, aabb, aaabbb, aaaabbbb\}$.
  $L_1 - L_2 = \{aa, ba, bb\}$.

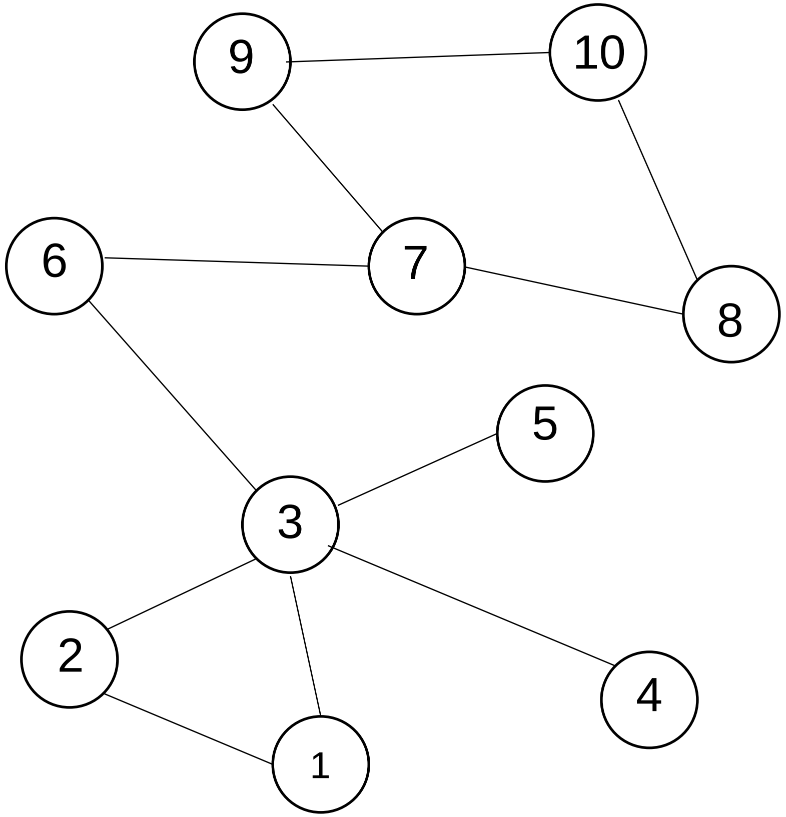
- Otra **operación típica** es el *complemento*. El complemento de un lenguaje L_1 con respecto a otro lenguaje L_2 que lo incluye, es el lenguaje formado por las cadenas de $L_2 - L_1$.

  Por ejemplo, si $L_1 = \{a, aa\}$ y $L_2 = \{a, aa, aaa, aaaa, aaaaa, \ldots\}$,
  entonces el complemento de L_1 con respecto a L_2 es $L_1^C = \{aaa, aaaa, aaaaa, \ldots\}$.

Repaso de algunos conceptos matemáticos básicos (continuación)

- Un ***grafo*** es un par (V, E), tal que V es un conjunto de **vértices** y E es un conjunto de **arcos**.

  Por ejemplo, el siguiente grafo tiene 10 vértices y 11 arcos.



- Vértices como 1 y 2 son **adyacentes.**
- Arcos como (1, 2) y (2, 3) son **adyacentes.**
- La secuencia de vértices (1, 2, 3, 6, 7, 9, 10) representa un **camino** del vértice 1 al vértice 10.

Repaso de algunos conceptos matemáticos básicos (continuación)

- Una ***fórmula booleana*** es una fórmula lógica que se obtiene combinando ***variables booleanas***, de valor de verdad *verdadero* (V) o *falso* (F), con los operadores lógicos ***and*** **($\wedge$),** ***or*** **($\vee$) y** ***not*** **($\neg$).**

  Por ejemplo, $\boldsymbol{\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)}$ es una fórmula booleana.

- El significado de los operadores lógicos es el siguiente:

  $\boldsymbol{\varphi = x_1 \wedge x_2}$ es verdadera sii x_1 **y** x_2 son verdaderas
  $\boldsymbol{\varphi = x_1 \vee x_2}$ es verdadera sii x_1 **o** x_2 son verdaderas
  $\boldsymbol{\varphi = \neg x}$ es verdadera sii x es falsa

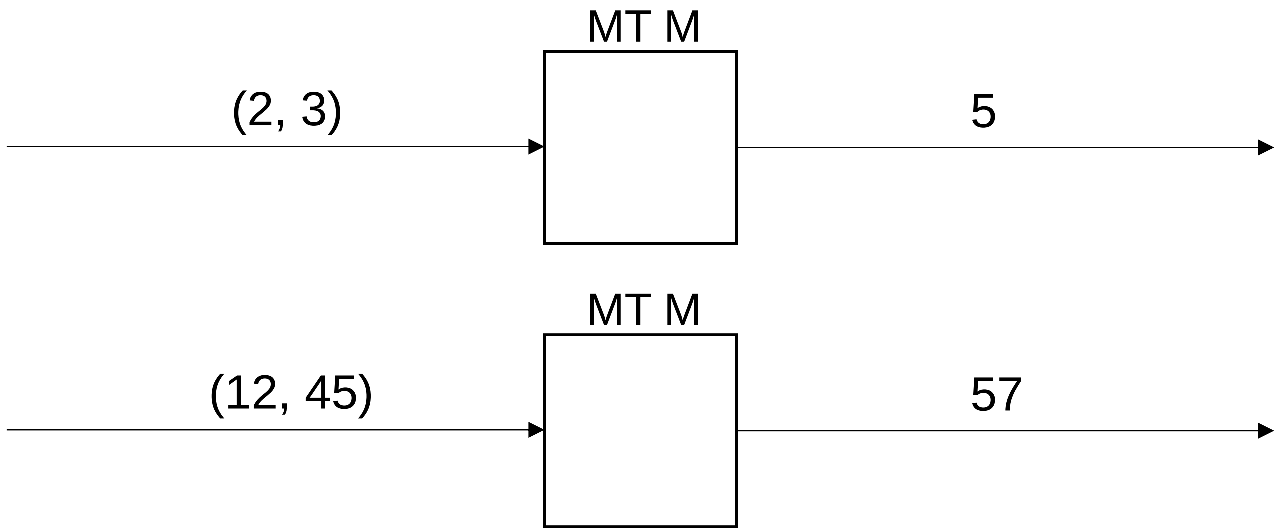
  Por ejemplo, dada la fórmula booleana $\boldsymbol{\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)}$**,**
  si x_1 = V y x_2 = F, entonces φ resulta **verdadera**: (V $\vee$ F) $\wedge$ ($\neg$V $\vee$ $\neg$F) = (V $\vee$ F) $\wedge$ (F $\vee$ V) = V $\wedge$ V = **V**
  y si x_1 = V y x_2 = V, entonces φ resulta **falsa**: (V $\vee$ V) $\wedge$ ($\neg$V $\vee$ $\neg$V) = (V $\vee$ V) $\wedge$ (F $\vee$ F) = V $\wedge$ F = **F**

- Una fórmula booleana φ es *satisfactible* si existe al menos una asignación de valores de verdad que la evalúa verdadera. Por ejemplo, la fórmula anterior $\boldsymbol{\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)}$ **es satisfactible**.

Máquina de Turing (MT). Generalidades.

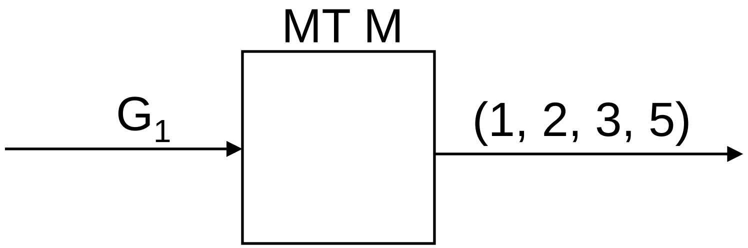
- **Modelización muy simple de una computadora**, aceptada universalmente para estudiar la computabilidad y la complejidad computacional de los problemas.

- **Ejemplo.** MT M que resuelve el problema de la **suma** de dos números naturales (enteros ≥ 0):

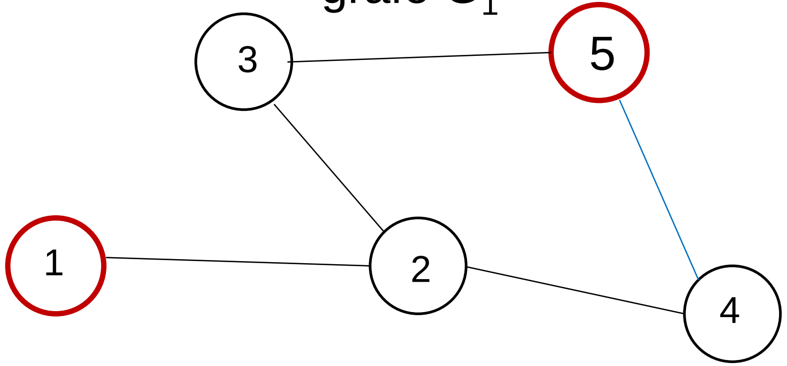


M se comporta de esta manera a partir de **todo par de números naturales**.

- **Otro ejemplo.** MT M que resuelve el problema de encontrar en un grafo no dirigido un **camino** del vértice inicial al vértice final (**problema de búsqueda,** como el del ejemplo anterior):

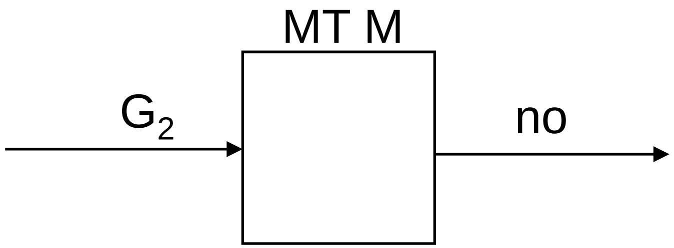


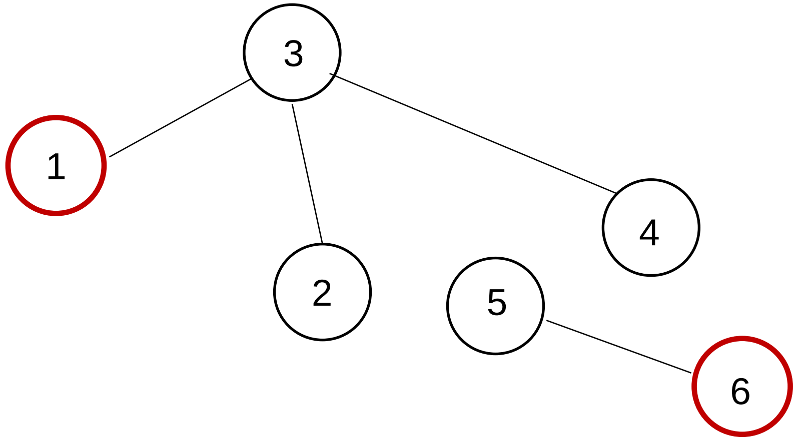
grafo G_1



Cuando hay solución, la MT M devuelve **alguna**.

grafo G_2

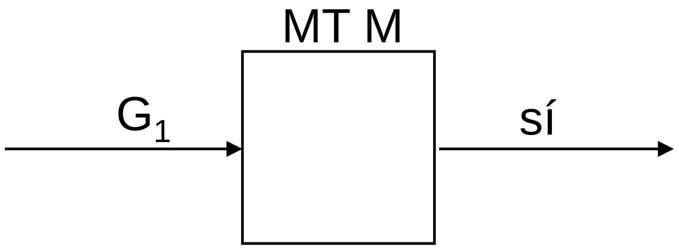


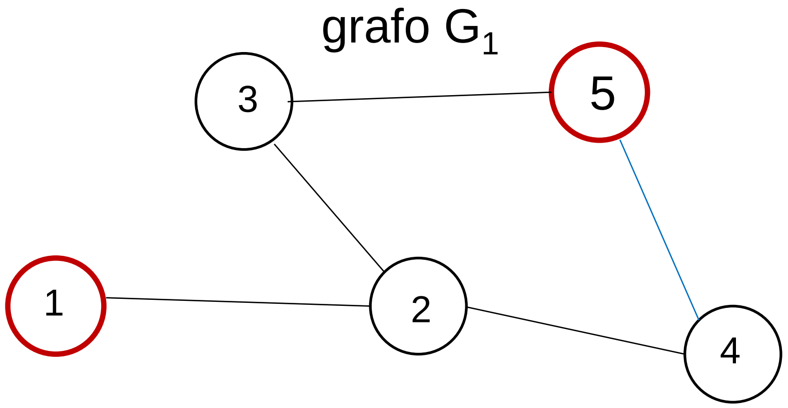


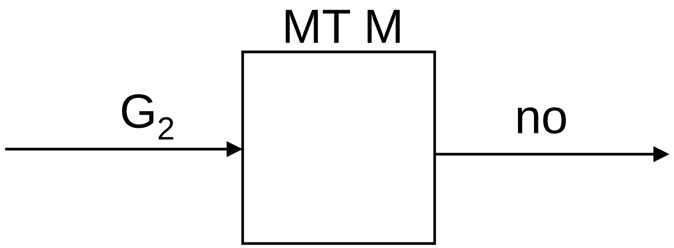
Cuando no hay solución, la MT M **responde no**.

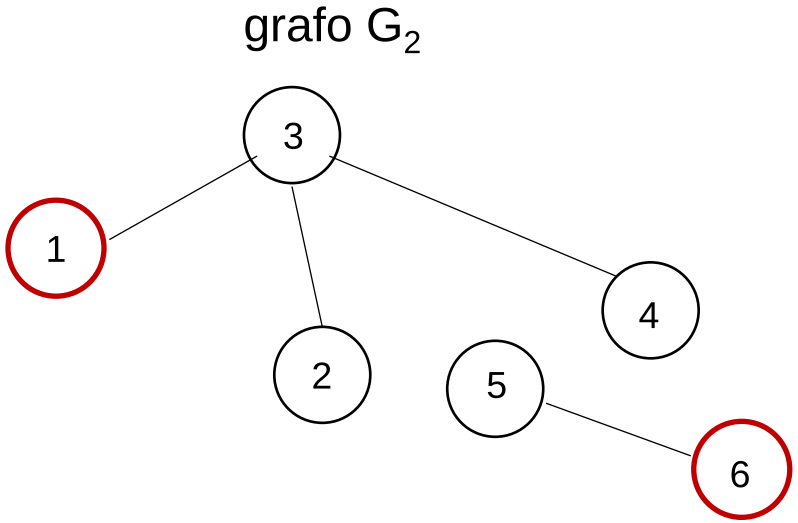
En general consideraremos grafos no dirigidos.

- **Otro ejemplo.** Problema similar al anterior, pero **más simple**: MT M que **decide** si un grafo tiene un camino del vértice inicial al vértice final (**problema de decisión**):



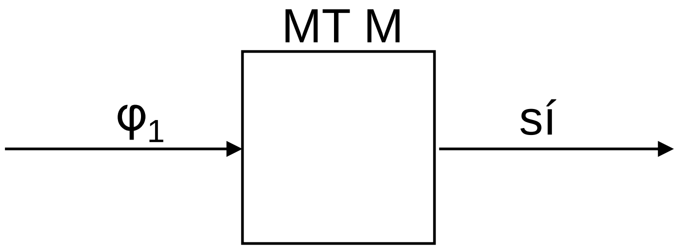






En este tipo de problemas, la MT sólo responde **sí o no** (**acepta** o **rechaza** su entrada).

- **Otro ejemplo.** MT M que **decide** si una fórmula booleana es satisfactible (otro ejemplo de **problema de decisión**):

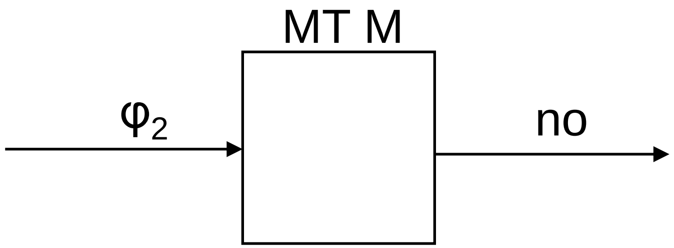


$\boldsymbol{\varphi_1 = (x_1 \vee x_2) \wedge (x_3 \wedge x_4)}$

Por ejemplo, la asignación $\mathcal{A}$ = (V, F, V, V) satisface φ_1:

$$\varphi_1 = (x_1 \vee x_2) \wedge (x_3 \wedge x_4)$$

V **V** F **V** V **V** V



$\boldsymbol{\varphi_2 = x_1 \wedge \neg x_1}$

No existe asignación que satisfaga φ_2.

$$\varphi_2 = x_1 \wedge \neg x_1$$

V **F** **F** V

F **F** **V** F

- Vamos a considerar en general **problemas de decisión**, no **problemas de búsqueda**.
- Por lo tanto, las máquinas de Turing sólo van a responder **sí o no** (van a **aceptar** o **rechazar**).
- Así se simplifican las descripciones pero **sin perder generalidad**.
- Con esta visión, una MT M **acepta o reconoce un lenguaje:** el lenguaje formado por **las cadenas que M acepta.**
- Vamos a denotar con **L(M)** al lenguaje que acepta la MT M.
- Por ejemplo:

En el problema de los grafos,

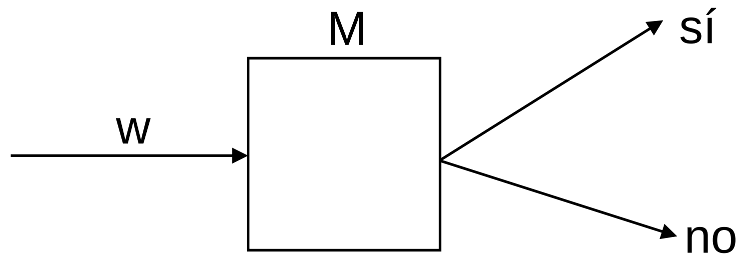
$$L(M) = \{G_1, G_2, G_3, \ldots\},$$

donde G_i es un grafo con un camino del primero al último vértice.

En el problema de las fórmulas booleanas,

$$L(M) = \{\varphi_1, \varphi_2, \varphi_3, \ldots\},$$

donde φ_i es una fórmula booleana satisfactible.



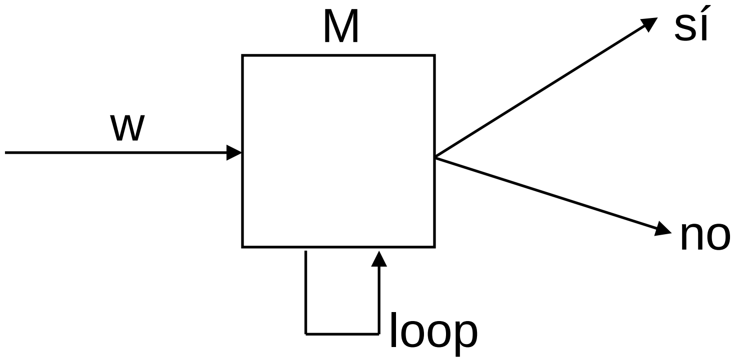
Genéricamente, dada una cadena w:

si $w \in L(M)$, M responde sí (acepta w);
si $w \notin L(M)$, M responde no (rechaza w).

Por lo tanto, hasta nuevo aviso, vamos a utilizar las palabras **problema** y **lenguaje** indistintamente.

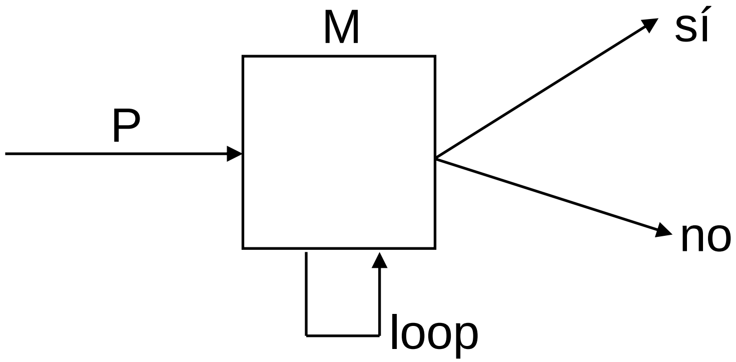
¡MALA NOTICIA!

- El caso más general de MT corresponde al siguiente esquema:



- Es decir: si w pertenece a L(M), M responde sí, y si w no pertenece a L(M), en el mejor de los casos M responde no, pero en el peor de los casos M no responde nada porque **no para, “loopea”**.
- Lamentablemente, existen problemas que **sólo cuentan con MT de este tipo.**
- **Son problemas computables no decidibles** (los que vimos antes son **problemas computables decidibles**).

- Ejemplo clásico: **el problema de decidir si un programa P para (*halting problem*)**



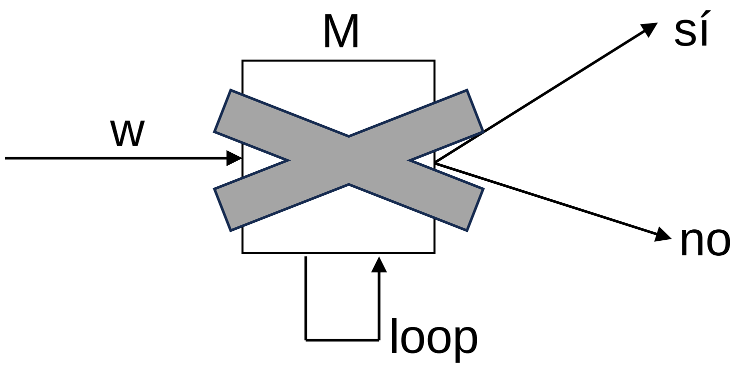
La mejor MT M posible:

Si P para, responde sí.
Si P no para, responde no o loopea.

Es decir, en este caso **no existe una MT M que pare siempre.**

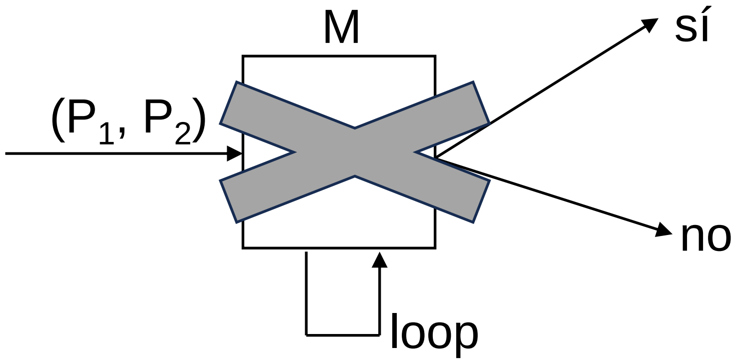
¡PEOR NOTICIA!

- Existen problemas que directamente **no tienen MT que los resuelvan**:



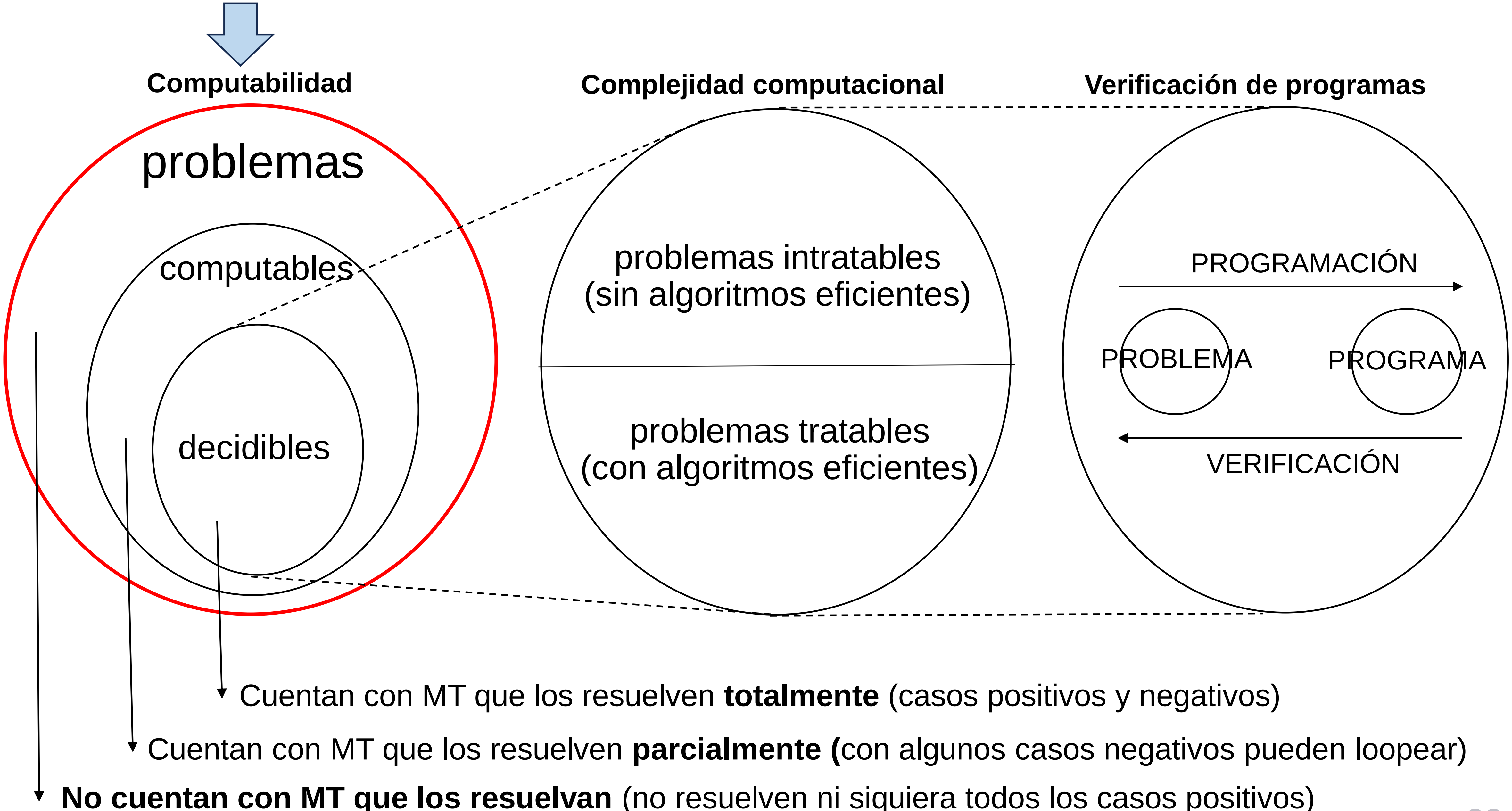
- Es decir: dichos problemas **ni siquiera cuentan con MT M que respondan sí en todos los casos positivos** (que es lo mínimo que se puede esperar de una MT).
- **Son problemas no computables.**

Ejemplo clásico: **el problema de decidir la equivalencia de dos programas P_1 y P_2**



No existe ni siquiera una MT que responda sí
para todos los casos en que P_1 sea equivalente a P_2
(lo mínimo que se puede pedir),
y por eso **este problema no es computable**.

CLASES 1 A 4: LO COMPUTABLE Y LO DECIDIBLE

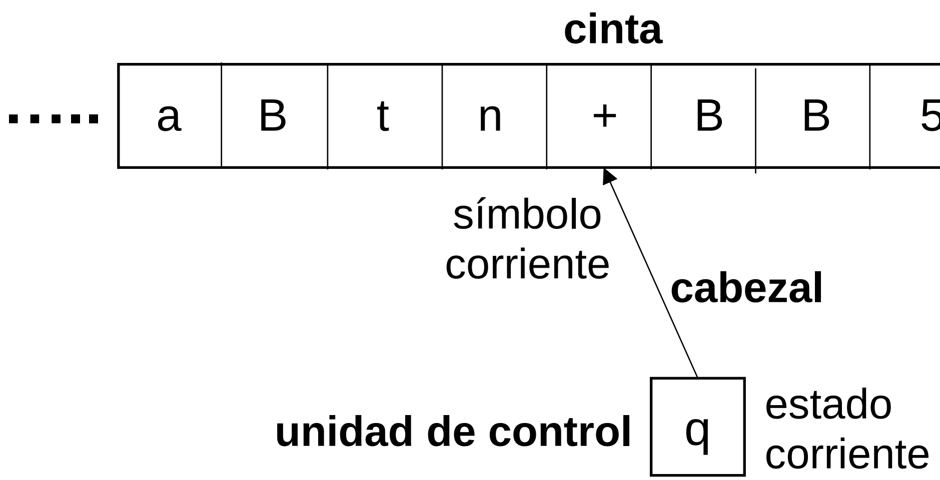


Cuentan con MT que los resuelven **totalmente** (casos positivos y negativos)

Cuentan con MT que los resuelven **parcialmente (**con algunos casos negativos pueden loopear)

No cuentan con MT que los resuelvan (no resuelven ni siquiera todos los casos positivos)

Componentes y funcionamiento de una MT

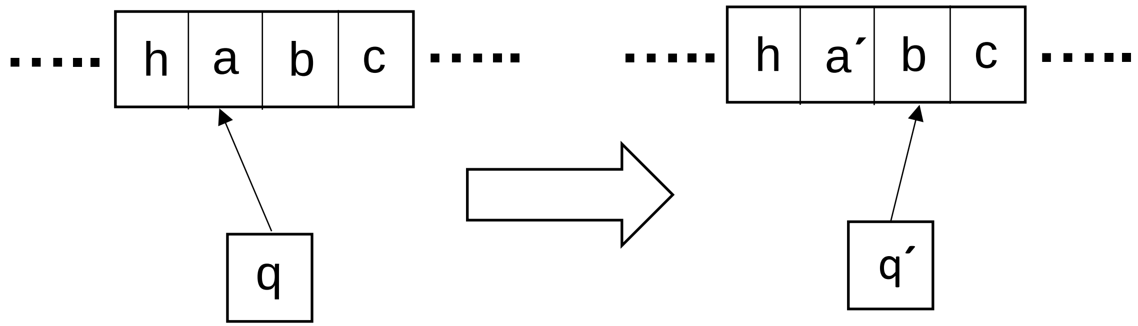


La cinta almacena símbolos de un alfabeto Γ, y la unidad de control tiene siempre un estado de un conjunto de estados Q.

Tesis de Church-Turing

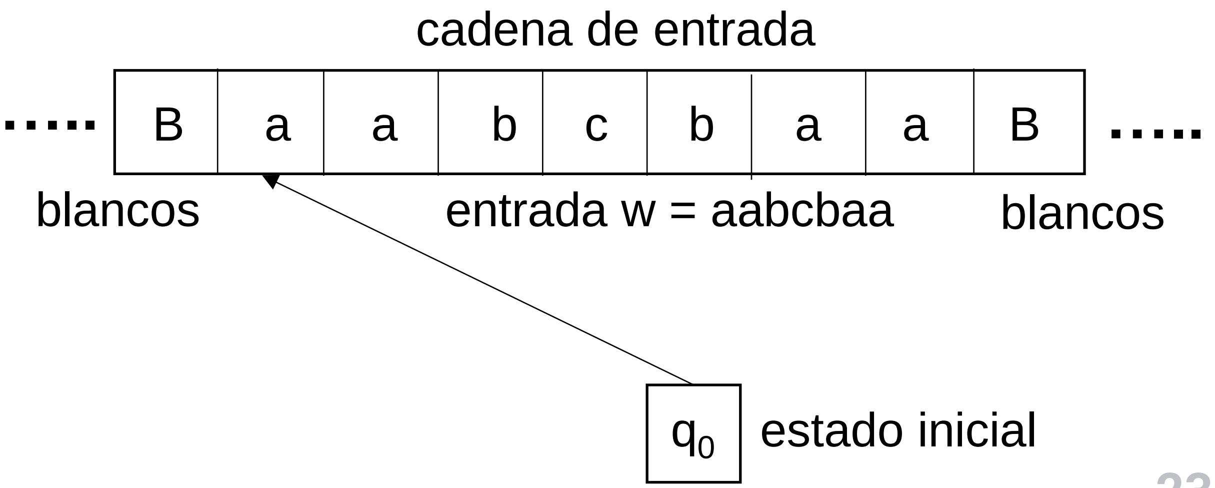
Todo dispositivo computacional físicamente realizable puede ser simulado por una MT.

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



un paso de la MT

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



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

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

- Γ es el **alfabeto** (conjunto de símbolos) de M.

  Γ 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 M (de aceptación y de rechazo).

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

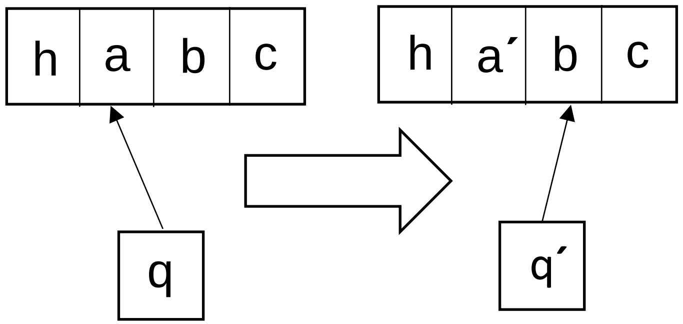
  Dado un estado corriente de Q, y un símbolo corriente de Γ, la MT M:

  - pasa eventualmente a un nuevo estado de Q,
  - modifica eventualmente el símbolo corriente de Γ,
  - y se mueve un lugar a la derecha (R), a la izquierda (L), o no se mueve (S).

  La máquina para si en algún momento alcanza el estado q_A o q_R.

Ejemplo de transición de M

$\delta(q, a) = (q´, a', R)$



Ejemplo de construcción de una MT

Problema: decidir si una cadena de símbolos *a* y b tiene la forma ab, o bien aabb, o bien aaabbb, o bien aaaabbbb, …

Vamos a construir una MT M que resuelva el problema, es decir, tal que L(M) = $\{a^n b^n \mid n \geq 1\}$.

1. **Idea General.** Un posible algoritmo podría ser, dada una cadena de entrada w, marcar el primer símbolo *a*, luego el primer símbolo b, luego el segundo símbolo *a*, luego 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, si w = **aaaaabbbbb,** M haría:

   aaaaabbbbb

   αaaaabbbbb

   αaaaaβbbbb

   ααaaaβbbbb

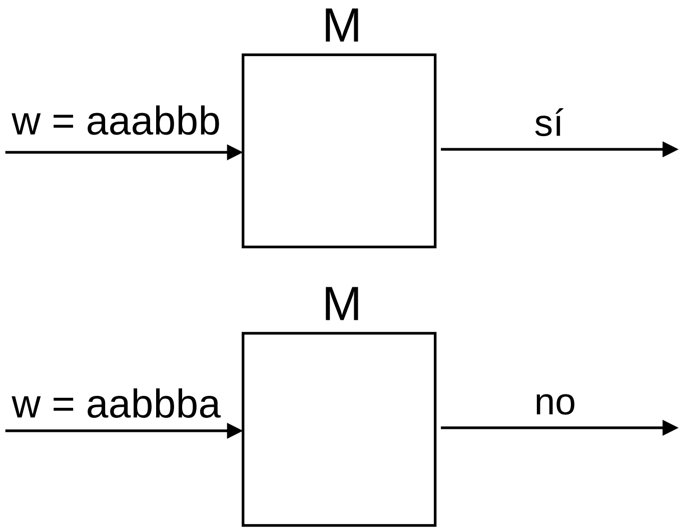
   ααaaaββbbb

   ……………..

   αααααβββββ

   En este caso, la MT M acepta w.

Otros ejemplos:



2. Construcción de la MT

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

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

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

Ayuda memoria del algoritmo con 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. $\delta(q_H, B) = (q_A, B, S)$

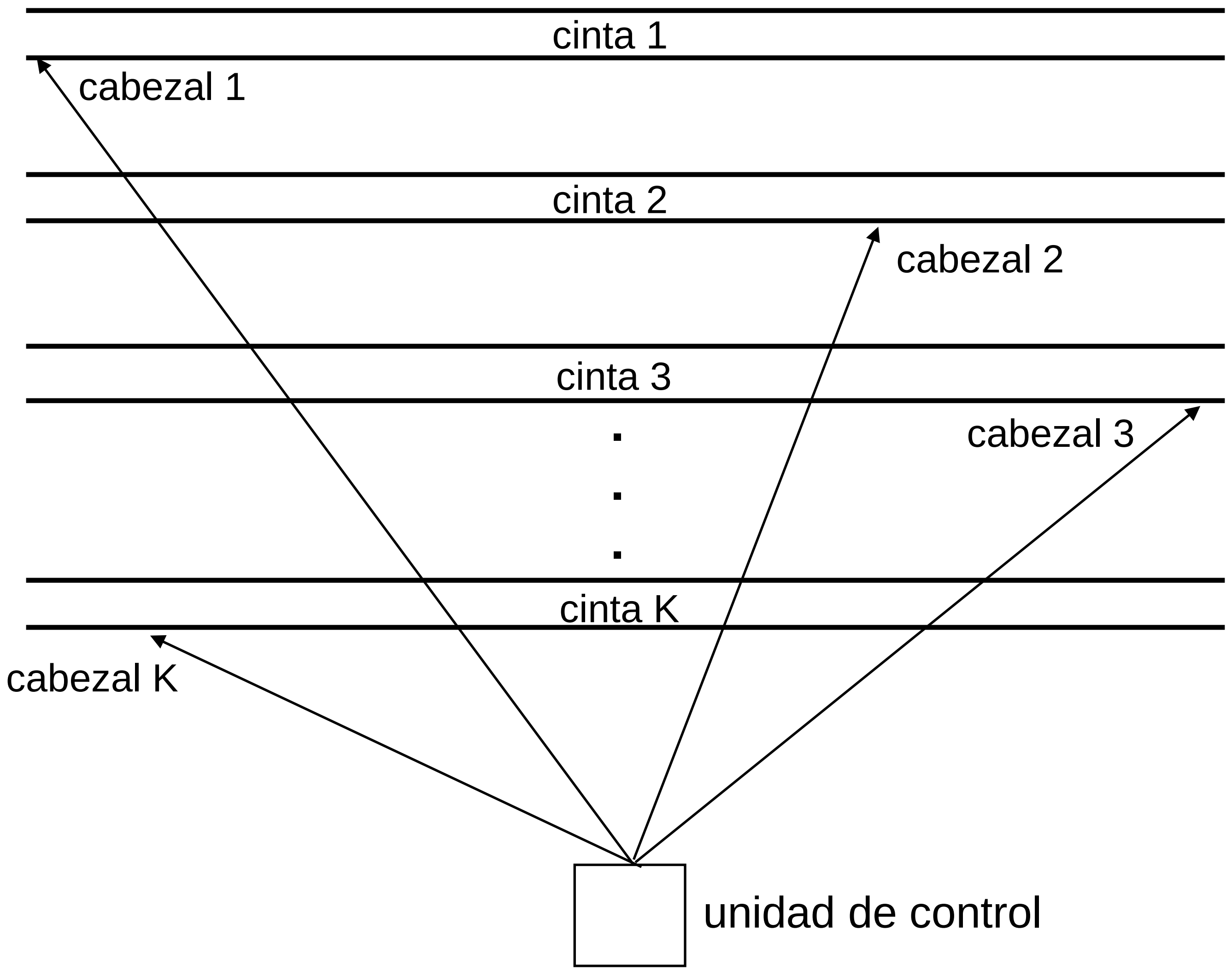
Todos los pares omitidos corresponden a rechazos (estado q_R).

Forma alternativa (tabular) para 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 casos de rechazo (estado q_R).

Otro modelo de MT: MT con varias cintas



Características generales de una MT con varias cintas:

Teorema

Para toda MT M_1 con varias cintas, existe una MT M_2 equivalente (acepta el mismo lenguaje) con una cinta (ver idea de prueba en la hoja siguiente).

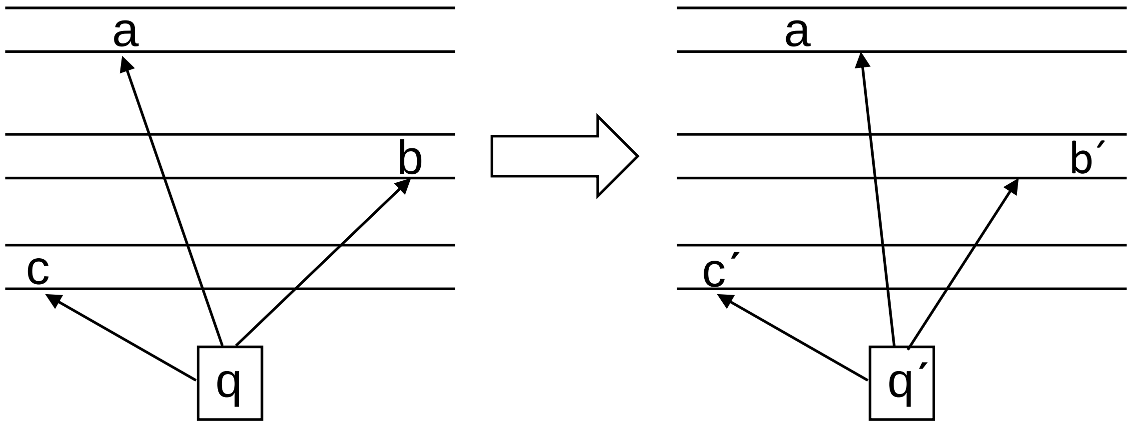
- La primera cinta contiene la entrada.

- 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, en un paso:

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

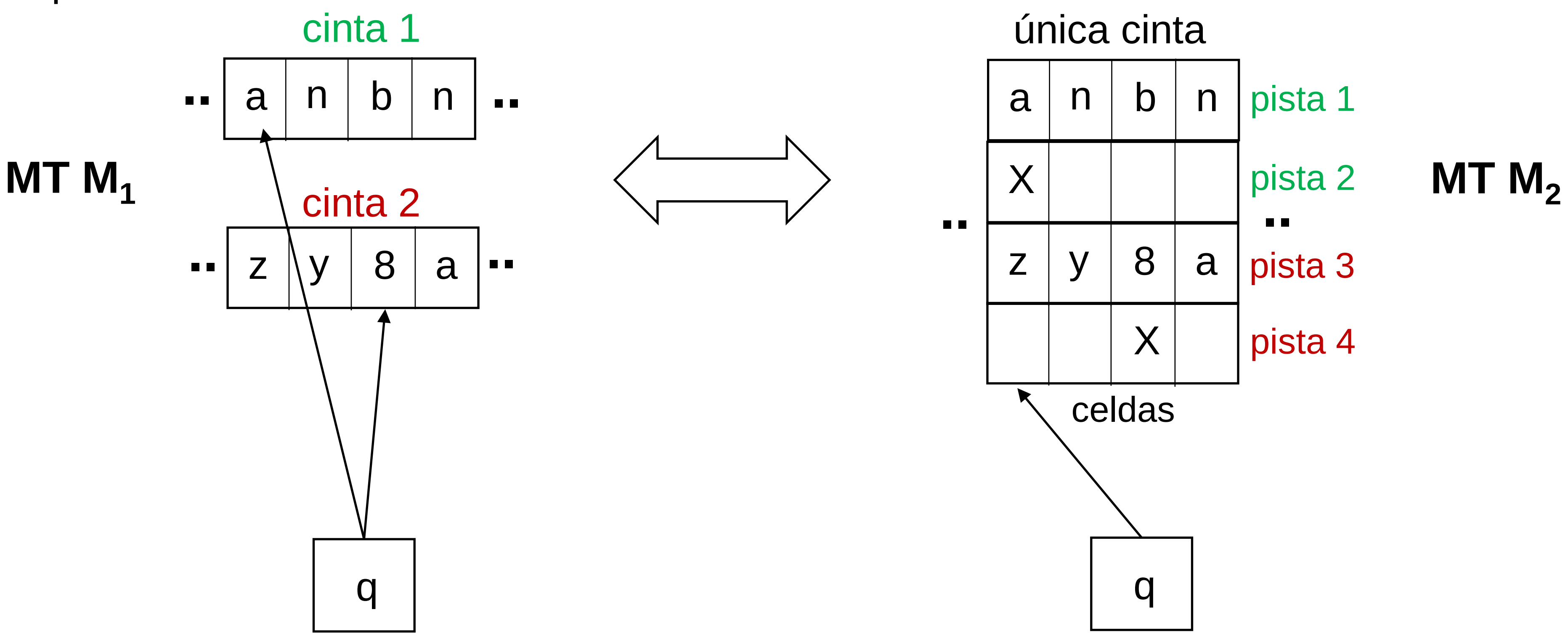
- Por ejemplo, para el caso de 3 cintas, una transición podría ser:

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



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

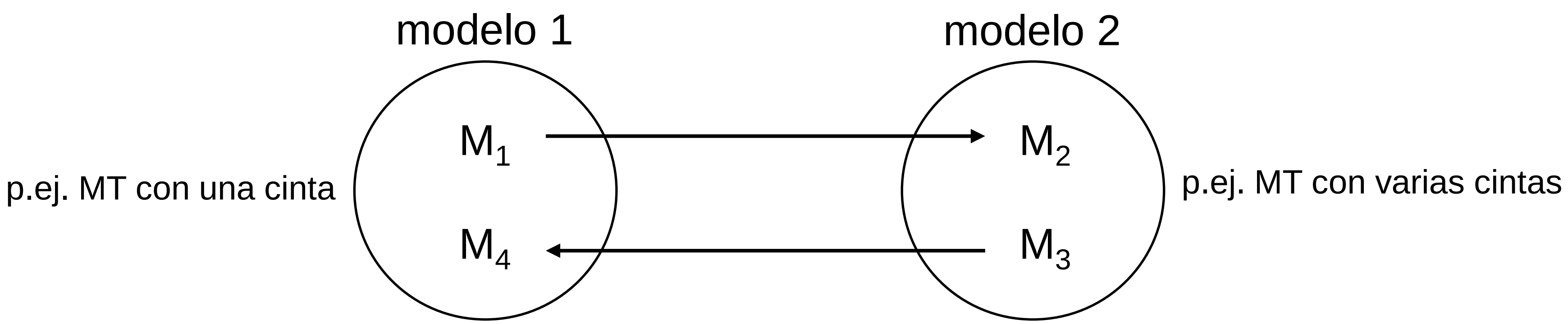
- Aunque no es intuitivo, una MT con varias cintas **no tiene más potencia computacional** que una MT con 1 cinta (lo que hace una MT con K cintas lo puede hacer una MT con 1 cinta).
- Idea de prueba con 2 cintas:



- 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).
- Se prueba que h pasos de M_1 se pueden simular con unos h^2 pasos de M_2 (**retardo cuadrático**).

Modelos equivalentes de MT

- **Dos MT son equivalentes** si aceptan el mismo lenguaje (es decir, si resuelven el mismo problema).
- **Dos modelos de MT son equivalentes** si dada una MT de un modelo existe una MT equivalente del otro.



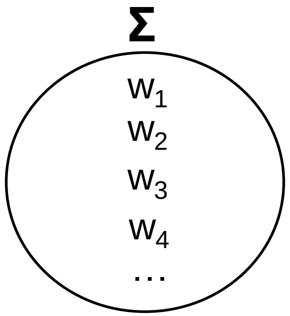
- Ejemplos de modelos de MT equivalentes al modelo de MT con **una cinta**: MT con **varias cintas**, MT con **cintas semi-infinitas** (sólo infinitas a la derecha), MT con **dos cintas y un solo estado**, etc.

- Ejemplos de modelos computacionales equivalentes al modelo de las MT: **máquinas RAM (acceso directo)**, **circuitos booleanos, lambda cálculo**, **funciones recursivas, gramáticas**, **programas C,** etc.

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

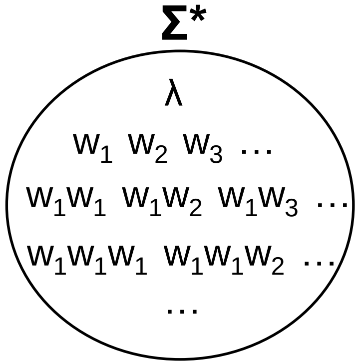
Anexo

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.

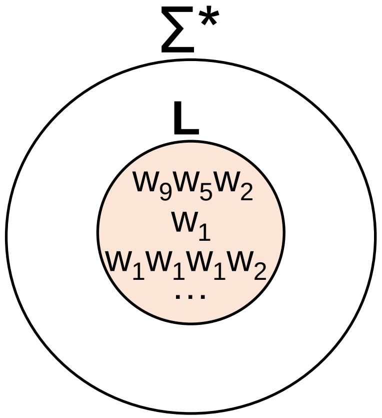
  $\Sigma^* = \{\lambda, w_1, w_2, w_3, \ldots, w_1w_1, w_1w_2, w_1w_3, \ldots, w_1w_1w_1, w_1w_1w_2, \ldots\}$

  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.**

  Es decir: $L \subseteq \Sigma^*$.



P. ej., L puede ser el lenguaje de las cadenas que representan los grafos con un camino del vértice inicial al vértice final.

- **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* es: $\Sigma^* - L$

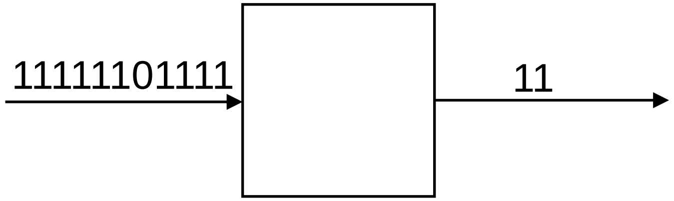
  Producto (o Concatenación): $L_1 \, . \, L_2$ (se concatenan las cadenas de uno y otro).

Otras visiones de MT

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

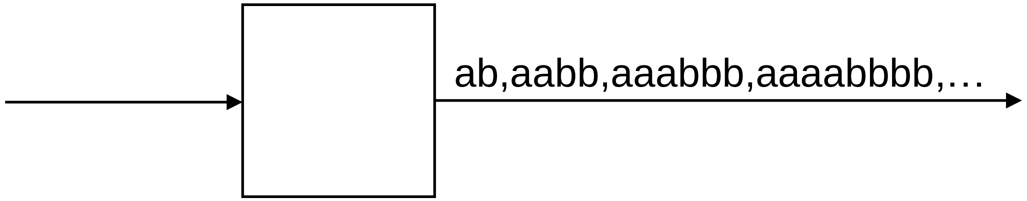
Problema: construir una MT que reste 2 números codificados en unario (todos 1) y separados por un cero.
Por ejemplo, dada la entrada 11111101111, hay que obtener la salida 11 (6 – 4 = 2).
Idea general: tachar el primer 1 antes del 0, luego el primer 1 después del 0, luego el segundo 1 antes del 0, luego el segundo 1 después 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 (genera en una cinta de salida todas las cadenas del lenguaje que acepta)

Problema: construir una MT que genere todas las cadenas de la forma a^nb^n, con $n \geq 1$. Es decir, en una cinta especial de salida debe **generar 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.

Clase práctica 1

Ejemplo de construcción de una MT con varias cintas

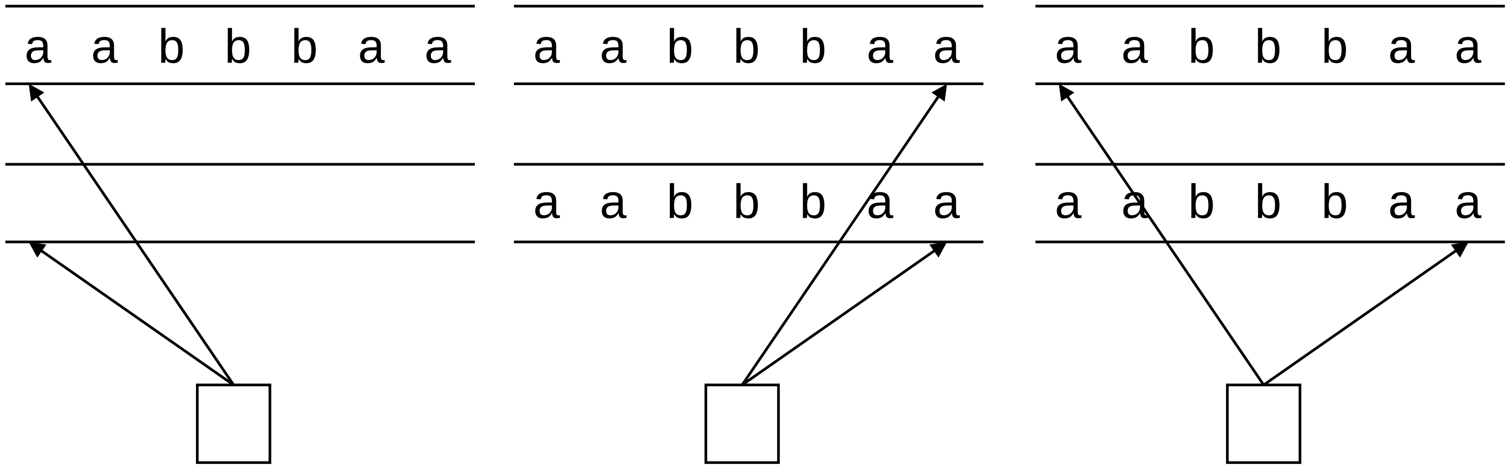
Problema: decidir si una cadena w de cero o más símbolos a y b es un *palíndromo* o *capicúa* (w es igual a w^R, siendo w^R la cadena inversa de w). Por ejemplo, la cadena w = aabbbaa es un palíndromo.

Vamos a construir una MT M con varias cintas que acepte el lenguaje de los palíndromos con símbolos a y b (incluyendo la cadena vacía λ).

Idea general: Una MT M con 2 cintas que hace:

1. Copia la entrada, de la cinta 1 a la cinta 2. Si algún símbolo no es *a* ni b, rechaza.
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, y si no, rechaza.

Por ejemplo: etapa 1 etapa 2



etapa 3

1. Comparar el símbolo apuntado de arriba con el símbolo apuntado de abajo. Si son distintos, rechazar.
2. Moverse un lugar a la derecha arriba y un lugar a la izquierda abajo. Si se llegó a los B, aceptar.
3. Volver a 1.

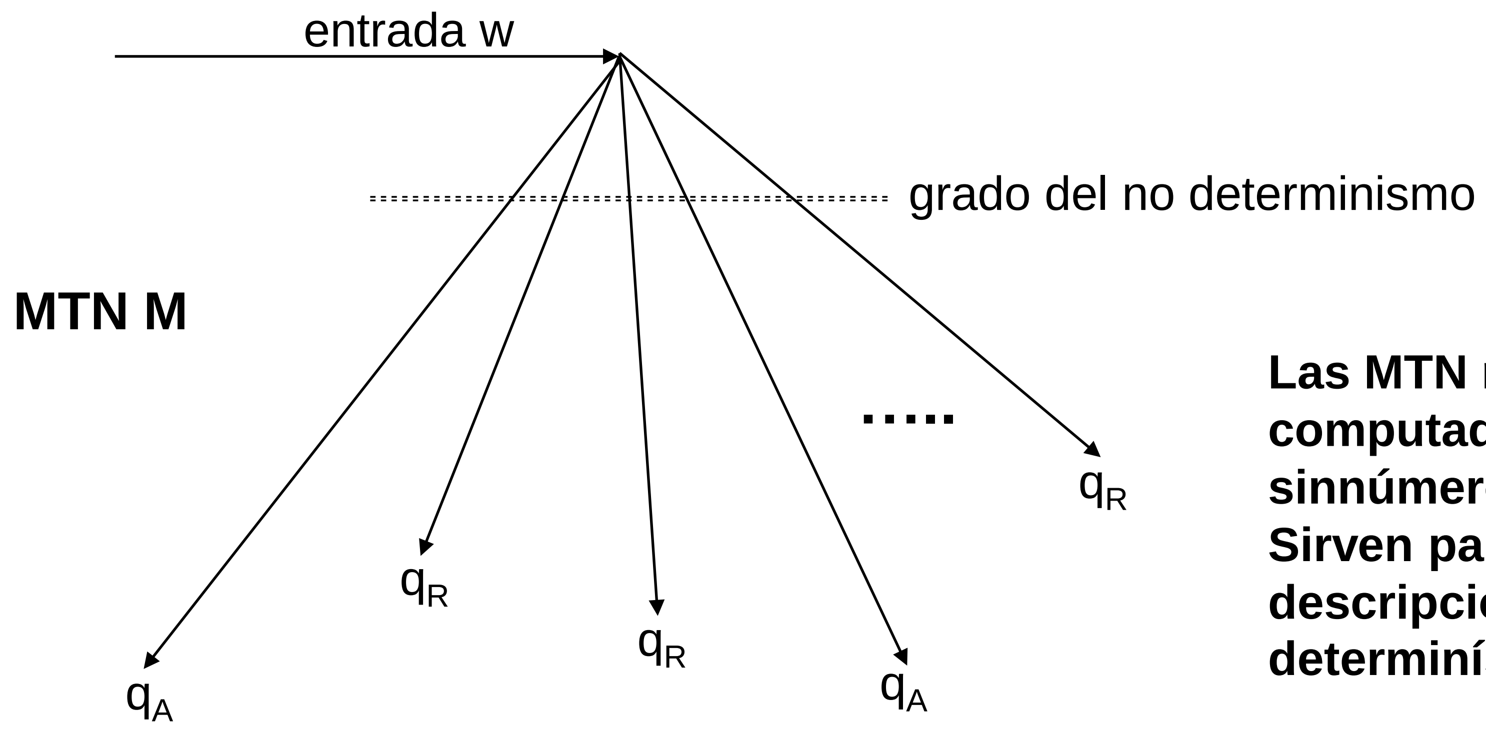
Función de transición de la MT M

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 representan casos de rechazo (estado q_R).

Modelo de máquina de Turing no determinística (MTN)



Las MTN no modelizan una computadora (haría falta un sinnúmero de procesadores). Sirven para simplificar las descripciones de las MT determinísticas (MTD).

- Para un mismo par (q, w), la máquina puede responder **de más de una manera**.
- Una MTN M acepta una cadena w sii **al menos una computación de M acepta w.**
- 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, entonces ejecutar secuencialmente todas las computaciones requiere unos c^h pasos, siendo c el grado del no determinismo (**retardo exponencial**).

Ejemplo sencillo de uso de una MTN

Problema: construir una MTN que acepte todas las cadenas que empiezan con un símbolo h y continúan con cero o más símbolos *a* o con 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. $\Delta(q_b, B) = \{(q_A, B, S)\}$

Notar que se utiliza el símbolo Δ en lugar del símbolo δ.

Por ejemplo: