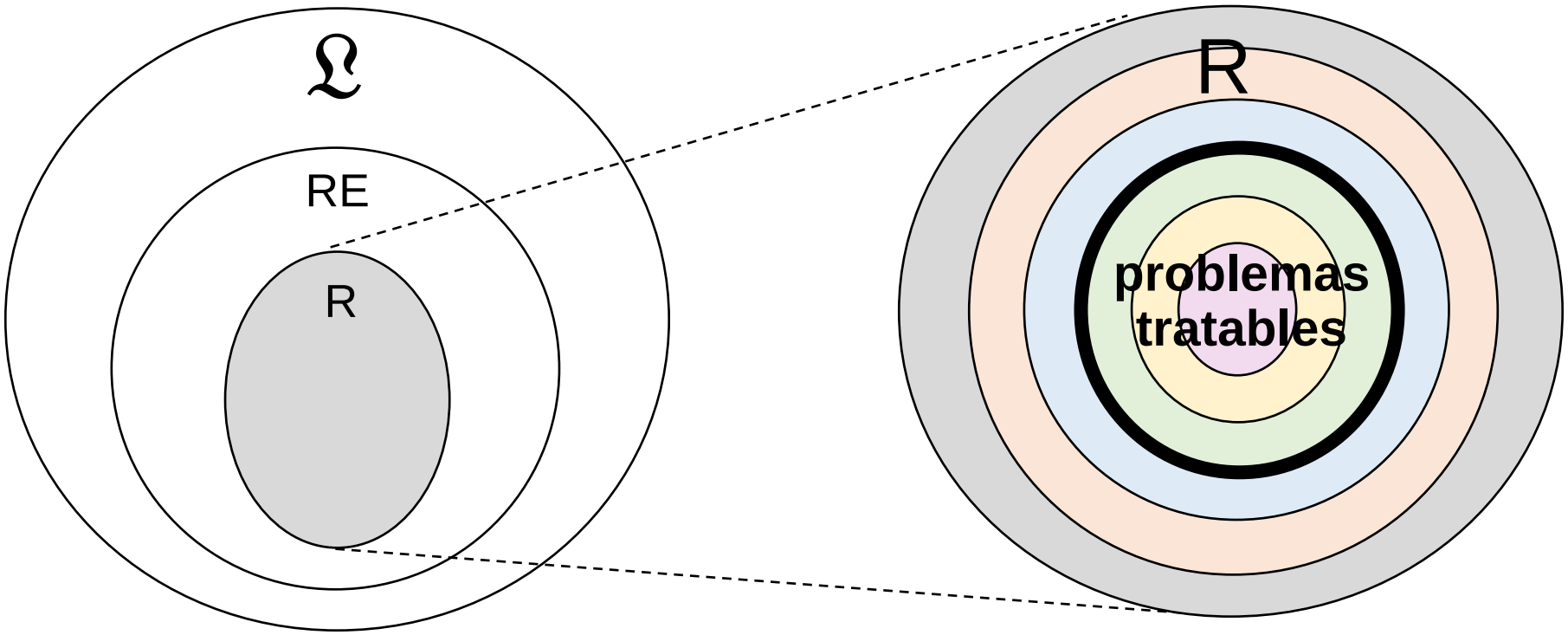
Clase teórica 5

Introducción a la Complejidad Computacional

Introducción

- Continuamos el viaje hacia el interior de los problemas.
- Ahora atravesaremos las fronteras de la **complejidad computacional**, ya dentro de la clase R.



- Se repiten muchos conceptos y técnicas de la computabilidad. Contraste: muchos problemas abiertos.

- Seguimos en el marco de los **problemas de decisión o problemas sí/no** (problemas “=“ lenguajes).

- Seguimos utilizando como modelo computacional las **máquinas de Turing** (estándar: varias cintas).

- **Métricas** de complejidad computacional que vamos a considerar:
  - ✓ **Tiempo** = cantidad de pasos ejecutados por una MT.
  - ✓ **Espacio** = cantidad de celdas ocupadas por una MT.

- Otras métricas (dinámicas):
  - ✓ **Cantidad de cambios de dirección** del cabezal de una MT (Hennie).
  - ✓ **Consumo de recursos abstractos** (Blum).
  - ✓ etc.

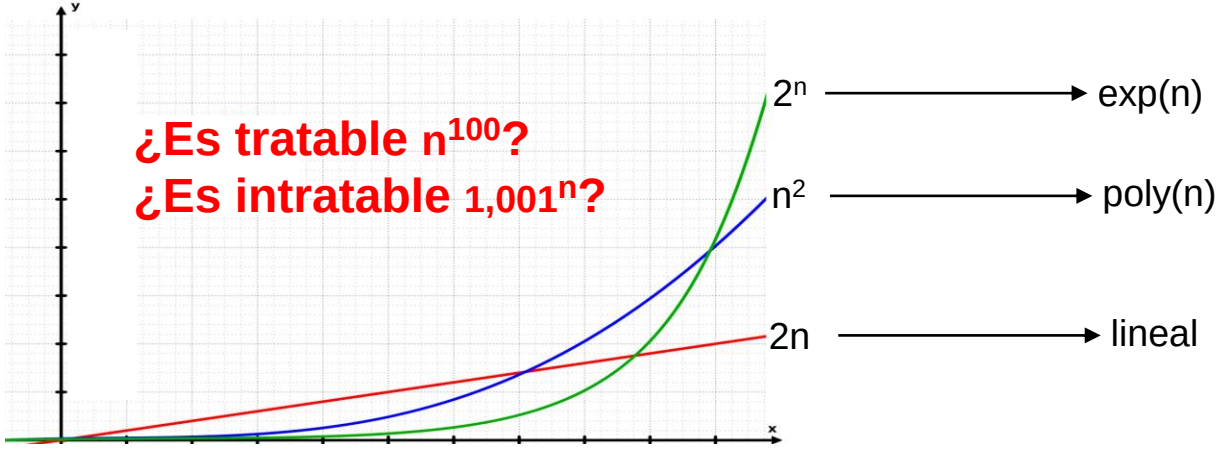
- Otras métricas (estáticas):
  - ✓ **Mínimo valor |Q|.|Σ|** de una MT (Shannon).
  - ✓ **Complejidad estructural** de una MT (Chomsky): según las cintas, movimientos, espacio recorrido, etc. (autómatas finitos, autómatas con pila, otros autómatas).
  - ✓ etc.

- Comenzamos con la **complejidad temporal.** Después trataremos (brevemente) la **complejidad espacial.**

Complejidad temporal

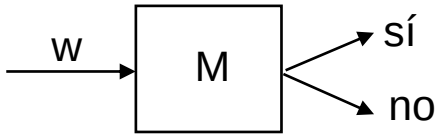
- Una MT tarda más a medida que sus cadenas de entrada w son más grandes.
- Por eso se usan **funciones temporales T(n)** definidas en términos de |w| = n.
- Funciones temporales T(n) típicas: **$5n$, $3n^3$, 2^n, $n^{\log_2 n}$, $7^{\sqrt{n}}$, $n!$, 6^{n^5}**
- Dos grandes grupos de funciones (de acuerdo al nivel de abstracción pretendido):
  Polinomiales o poly(n): $T(n) = c.n^k$
  Exponenciales o exp(n): el resto (cuasipolinomiales, subexponenciales, exponenciales, etc).
- Convención, respaldada por las matemáticas y la experiencia: **tiempo tratable = tiempo poly(n)**

n	2n	n^2	2^n
0	0	0	1
1	2	1	2
2	4	4	4
3	6	9	8
4	8	16	16
5	10	25	32
6	12	36	64
7	14	49	128
8	16	64	256
9	18	81	512
10	20	100	1024



Primeras definiciones

- Una MT M **tarda o se ejecuta en tiempo T(n),** sii a partir de toda entrada w, con |w| = n, M hace **a lo sumo T(n) pasos**.

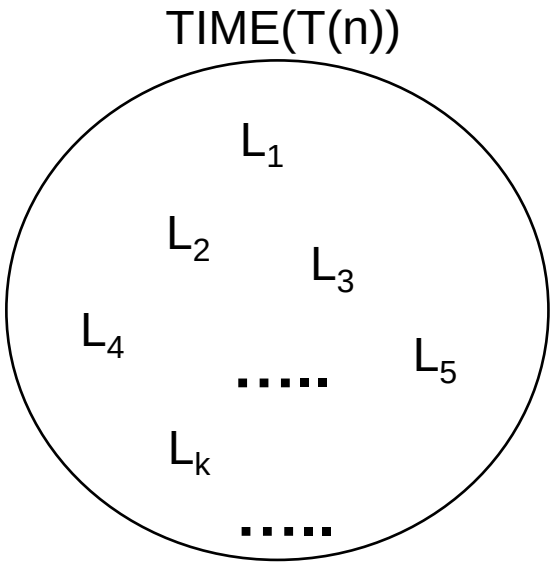


M hace a lo sumo T(|w|) pasos

- Una función $T_1(n)$ es del **orden** de una función $T_2(n)$, es decir **$T_1(n) = O(T_2(n))$,** sii para todo n se cumple **$T_1(n) \leq c.T_2(n)$,** con c > 0.

  Por ejemplo:
  $5n^3 + 8n + 25 = O(n^3)$ (**ejercicio**)
  $n^2 = O(n^3)$ (**ejercicio**)
  $n^3 = O(2^n)$ (**ejercicio**)

- Un lenguaje **L ∈ TIME(T(n))** sii existe una MT M que lo decide en tiempo **O(T(n)).**

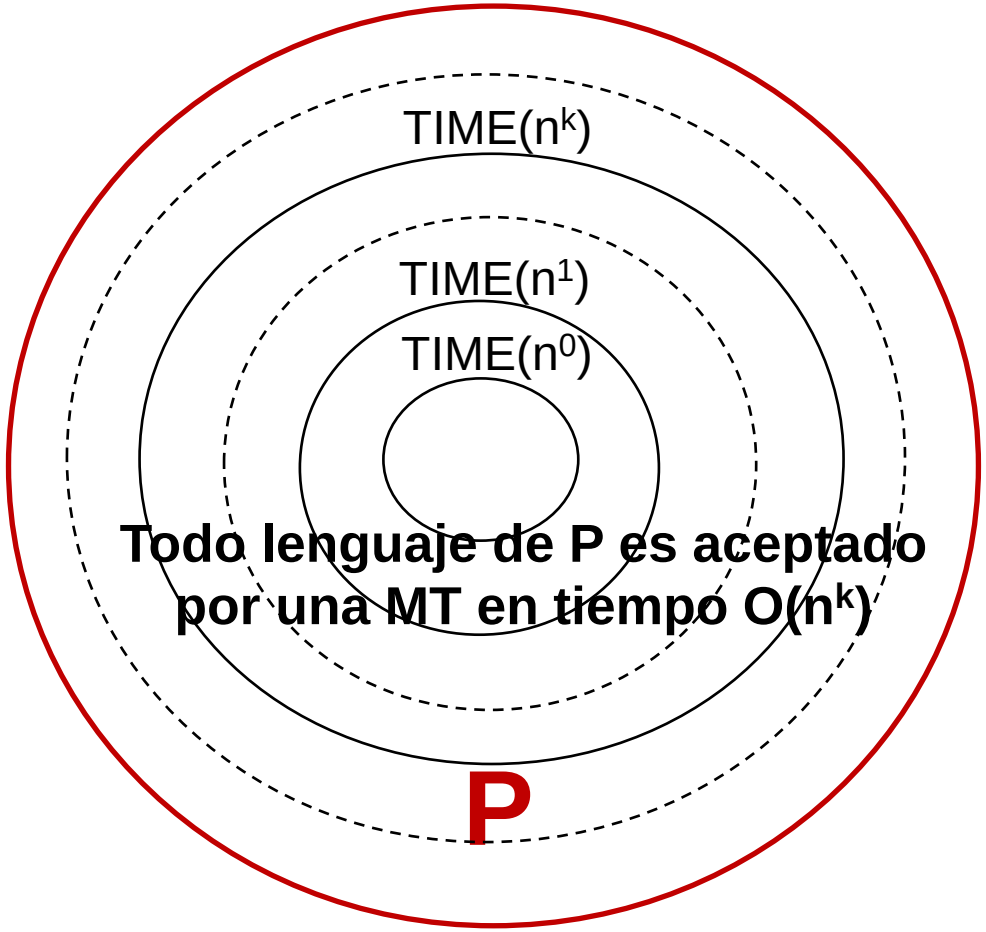


Para todo L_i existe una MT M_i que lo decide en tiempo O(T(n))

Se considera el **peor caso (cota superior).** Por ejemplo, si a partir de la mayoría de las cadenas de un lenguaje el tiempo de ejecución es O(n), y sólo en algunos casos es $O(n^3)$, el tiempo de ejecución queda *castigado* en $O(n^3)$.

Naturalmente. sería mejor considerar el **tiempo promedio** o **mínimo (cota inferior)**, pero son mucho más difíciles de calcular. Hoy día hay pocos valores conocidos de este tipo.

TIME(n^k) = la clase P



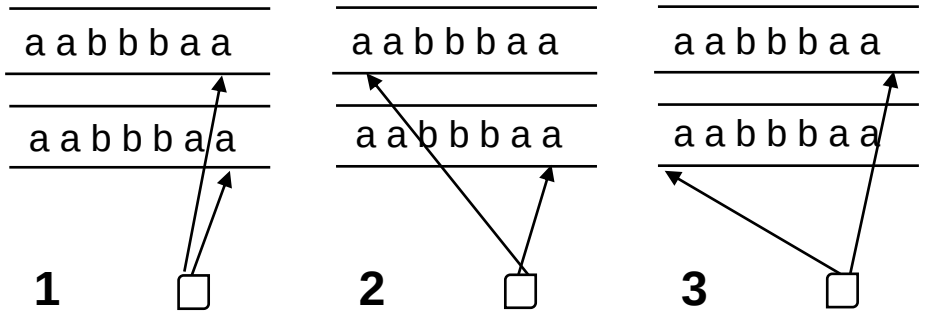
Ejemplo sencillo de lenguaje en P

PALÍNDROMOS = {w | w es un palíndromo con símbolos *a* y b}

Una MT muy simple que decide el lenguaje, con 2 cintas, hace:

1. Copia w de la cinta 1 a la cinta 2
   Tiempo O(n)
2. Posiciona el cabezal de la cinta 1 a la izquierda
   Tiempo O(n)
3. Compara en direcciones contrarias uno a uno los símbolos de las 2 cintas hasta llegar a un blanco en ámbas
   Tiempo O(n)

   Tiempo total: T(n) = O(n) + O(n) + O(n) = **O(n)**

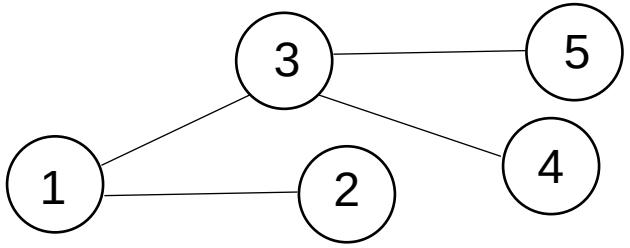


Recordar: si una MT M_1 con K_1 cintas tarda tiempo poly(n), una MT M_2 equivalente con K_2 cintas tarda tiempo poly(n). El retardo es a lo sumo **cuadrático.**

- **Otro ejemplo de lenguaje en la clase P**.

  ACCES = {G | G es un grafo no dirigido con m vértices y tiene un camino del vértice 1 al vértice m}

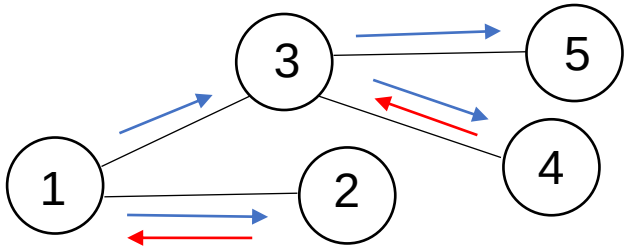
Representación de un grafo G



G = (V, E), con V el conjunto de vértices y E el conjunto de arcos

G = ({1, 2, 3, 4, 5}, {(1,2), (1,3), (3,4), (3,5)})

MT M que decide ACCES en tiempo poly(n)

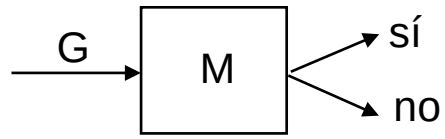


Recorrido DFS (en profundidad):

En el peor caso M recorre los arcos 2 veces

T(n) = O(|E|) = O(|G|) = O(n)

Por lo tanto, ACCES ∈ P



M tarda tiempo O(|G|)

- **Un ejemplo de lenguaje que <u>no estaría</u> en P**.

SAT = {φ | φ es una fórmula booleana sin cuantificadores, con m variables, y es satisfactible}

P.ej.: $\varphi_1 = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ es satisfactible con la asignación A = (V, V, V)

$\varphi_2 = (x_1 \wedge x_2 \wedge x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ no es satisfactible con ninguna asignación

Una MT M que decide SAT emplea una tabla de verdad (**no se conoce otro algoritmo**). P.ej., para φ_2:

$(x_1 \wedge x_2 \wedge x_3)$			$\wedge$	$(\neg x_1 \vee \neg x_2 \vee \neg x_3)$		
V	V	V	**F**	V	V	V
V	V	F	**F**	V	V	F
V	F	V	**F**	V	F	V
V	F	F	**F**	V	F	F
F	V	V	**F**	F	V	V
F	V	F	**F**	F	V	F
F	F	V	**F**	F	F	V
F	F	F	**F**	F	F	F

2^m posibilidades
(por cada variable, los valores V y F)

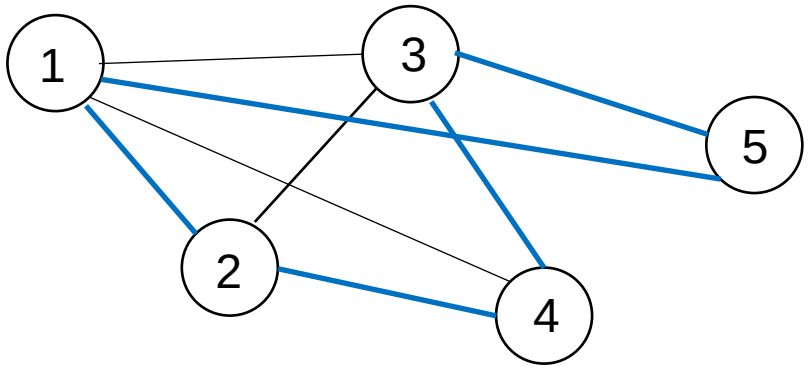
Cada evaluación se puede hacer en tiempo **$O(|\varphi|^2)$,** con el uso de una pila (ejercicio).

Por lo tanto, M puede llegar a ejecutar $O(2^m.|\varphi|^2) = O(2^n.n^2)$ = **exp(n) pasos.**

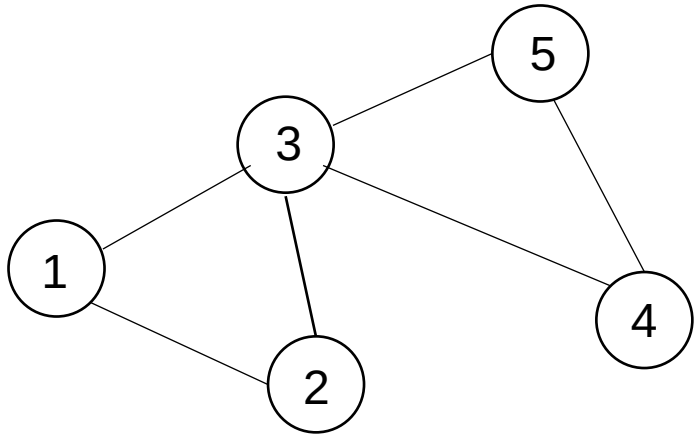
- **Otro ejemplo de lenguaje que <u>no estaría</u> en P.**

CH = {G | G es un grafo no dirigido con m vértices y tiene un Circuito de Hamilton}

<u>Circuito de Hamilton (CdeH)</u>: recorre todos los vértices del grafo sin repetirlos salvo el primero al final.



Grafo con un CdeH, p.ej. (1, 2, 4, 3, 5)



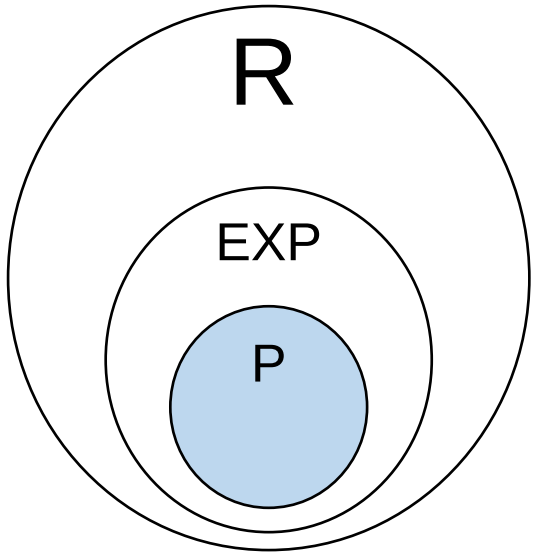
Grafo sin CdeH

Una MT M que decide CH prueba con todas las permutaciones de vértices (**no se conoce otro algoritmo**).

Dados m vértices existen **m!** permutaciones. Por ej., si m = 5: (1,2,3,4,5), (1,2,3,5,4), (1,2,4,3,5), etc.
Por otro lado, chequear si una permutación es un CdeH lleva tiempo $O(|V|.|E|) = O(|G|^2)$ = **$O(n^2)$** ¿por qué?

Como $m! = m.(m-1).(m-2).(m-3)\ldots2.1 = O(m^m)$ = **$O(n^n)$,** M puede alcanzar los $O(n^n.n^2)$ = **exp(n) pasos.**

Primera versión de la jerarquía temporal



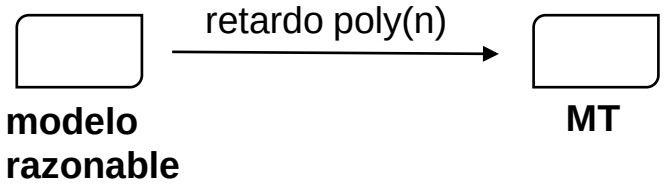
P es la clase de los lenguajes decidibles en tiempo poly(n) (**lenguajes tratables**)

EXP es la clase de los lenguajes decidibles en tiempo $O(c^{poly(n)})$

Se prueba que existen lenguajes fuera de P y fuera de EXP.

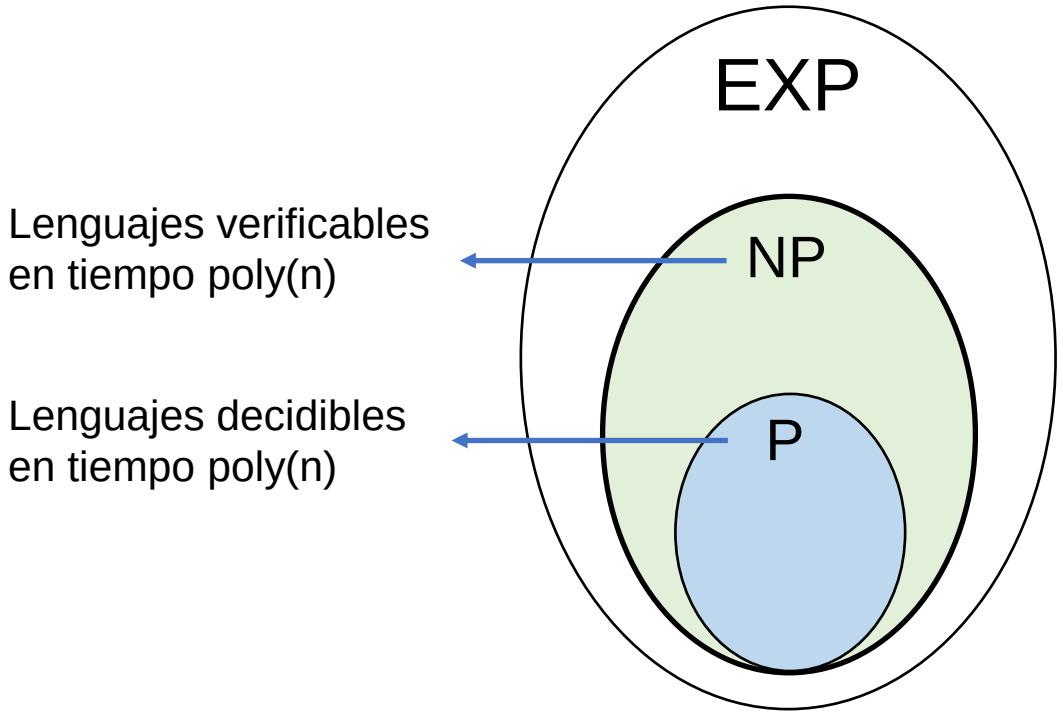
Tesis fuerte de Church-Turing (robustez de la clase P)
Si un lenguaje es decidible en tiempo poly(n) por un modelo computacional **razonable (realizable),** también es decidible en tiempo poly(n) por una MT **(¿hasta que las máquinas cuánticas sean una realidad?).**

Modelos computacionales razonables:
Máquinas RAM, programas Java, circuitos booleanos, etc.



Codificación razonable (realizable) de números
Cualquiera menos la de base unaria **(pensar p.ej. en 1.000.000.000 codificado como 1111111111111111…).**

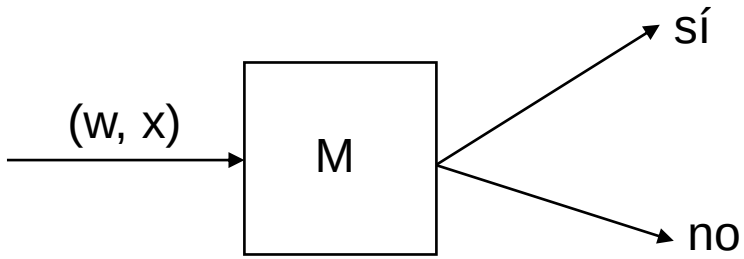
Segunda versión de la jerarquía temporal. La clase NP.



Asumiendo P ≠ NP
Asumiendo NP ≠ EXP

Un lenguaje L está en **NP** si existe una MT M tal que:
w ∈ L sii existe x tal que M acepta (w, x) en tiempo poly(n).

En otras palabras: toda cadena w de L cuenta con un **certificado** o **prueba** x que permite verificar **eficientemente** que pertenece a L.



M tarda tiempo poly(|w|)
|x| ≤ poly(|w|) **¿por qué?**

La MT M es un **verificador eficiente** de L.
x es un **certificado sucinto (o prueba sucinta)** de w.

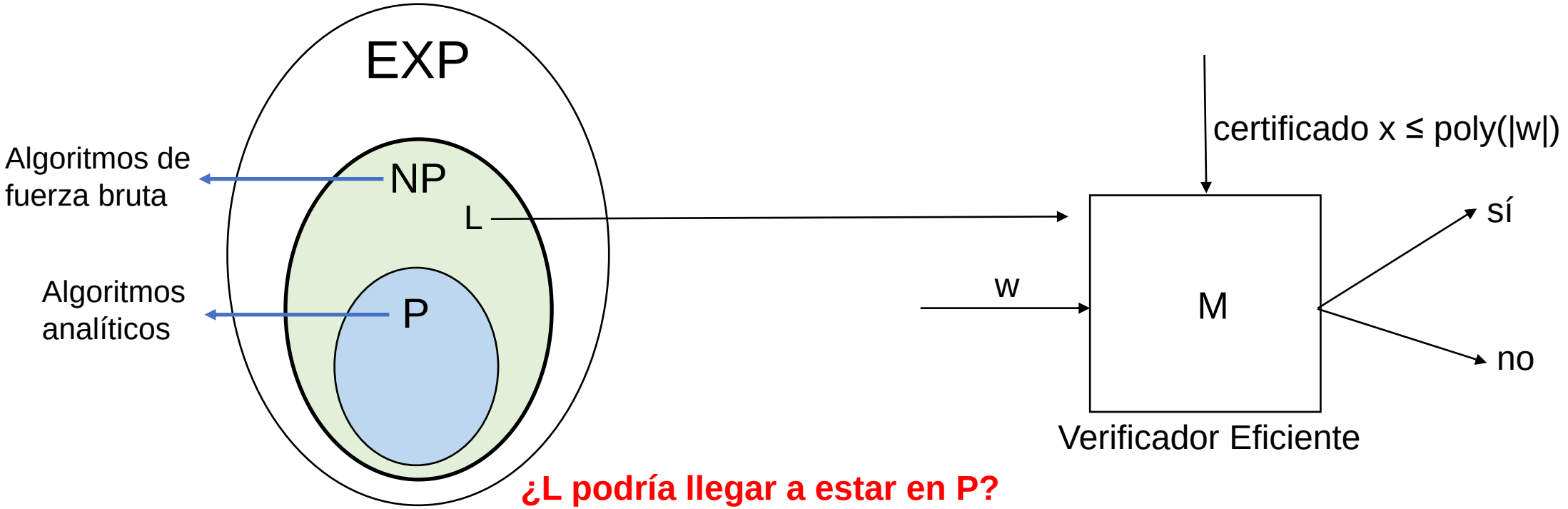
Aunque intuitivo, hasta hoy día no se ha encontrado una prueba de P ≠ NP.

Ejercicio: ¿Por qué P ⊆ NP?

- El problema P vs NP es uno de los **siete problemas del milenio**. Quien lo resuelva recibirá un premio de USD 1 MM por el Instituto Clay de los EEUU.

- Las resoluciones de los problemas de (NP – P) parecieran poder realizarse sólo empleando la **fuerza bruta** (**búsqueda exhaustiva en el espacio de todos los posibles certificados**).

- SAT y CH son dos ejemplos de lenguajes de NP (lo justificamos en el slide siguiente).

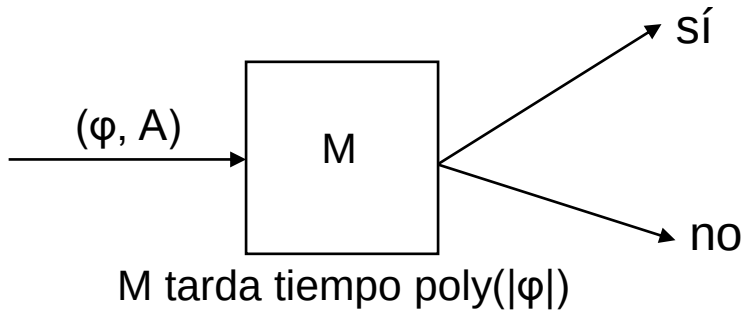


¿L podría llegar a estar en P?

- **SAT = {φ | φ es una fórmula booleana sin cuantificadores, con m variables, y es satisfactible}**

SAT no estaría en P:
Si φ tiene m variables, hay 2^m asignaciones A para chequear.

SAT está en NP:
Dada una fórmula booleana φ y una asignación A,
se puede **verificar en tiempo poly(n)** si A satisface φ.

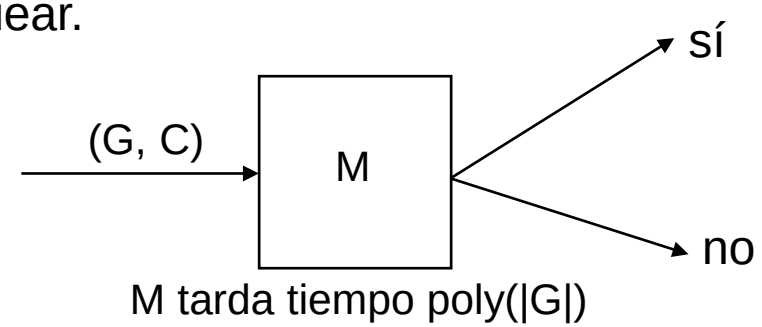


Ejercicio: ¿está SAT^C en NP?

- **CH = {G : G es un grafo no dirigido con m vértices y tiene un circuito de Hamilton}**

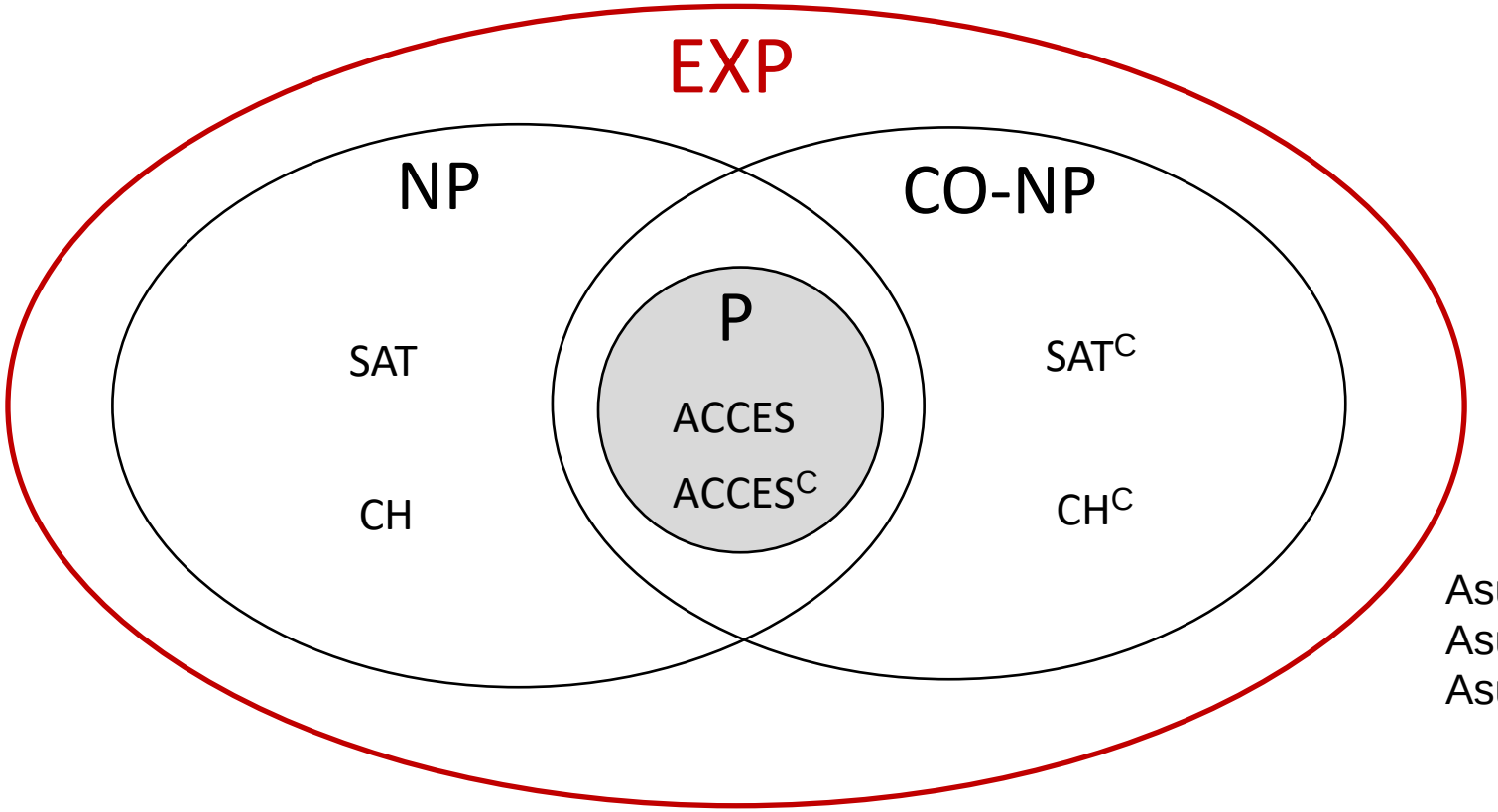
CH no estaría en P:
Si G tiene m vértices, hay m! secuencias C de m vértices para chequear.

CH está en NP:
Dado un grafo G y una secuencia C de m vértices,
se puede **verificar en tiempo poly(n)** si C es un CdeH de G.



Ejercicio: ¿está CH^C en NP?

Tercera versión de la jerarquía temporal. La clase CO-NP.



Asumiendo P ⊂ NP
Asumiendo P ⊂ NP ∩ CO-NP
Asumiendo NP ≠ CO-NP

CO-NP tiene los complementos de los lenguajes de NP. La conjetura aceptada es **NP ≠ CO-NP**.

Es decir, NP no sería cerrada con respecto al complemento. P sí lo es, lo que refuerza P ≠ NP.

Ejercicio: ¿Por qué P es cerrada con respecto al complemento?

Anexo de la clase teórica 5

Introducción a la Complejidad Computacional

Complejidad temporal y codificación de números

- **Ejemplo. Sea DIV-3 = {N | N es un número natural que tiene un divisor que termina en 3}**

  Una MT M que decide DIV-3 divide N por 3, 13, 23, etc., hasta encontrar eventualmente un divisor de N (**no se conoce otro algoritmo**).

  P. ej., si N = 100, M divide N por 3, 13, 23, 33, 43, 53, 63, 73, 83, 93 (10 divisiones).

  M ejecuta unas N/10 iteraciones, es decir O(N).
  Cada división se puede hacer en tiempo O(n^2)**.**
  Por lo tanto, **M ejecuta O(N.n^2) pasos.** Resta expresar N en términos de n.

  1) Si N se codifica en **unario**, n = N. Así, M tarda tiempo O(n.n^2) = O(n^3) = **poly(n).**

  2) Si N se codifica en **binario**, n = O($\log_2$N), y por lo tanto N = O(2^n). Así, M tarda tiempo O(2^n.n^2) = **exp(n).**

  3) Si N se codifica en **cualquier otra base**, M tarda tiempo **exp(n).**

  La única codificación de los números no utilizable es la unaria.

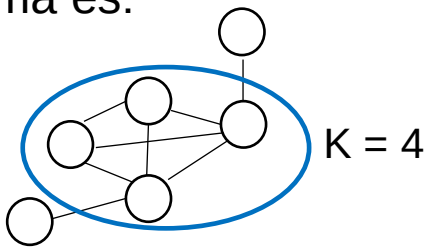
Clase práctica 5

Introducción a la Complejidad Computacional

Ejemplo 1. El problema del clique no estaría en P y está en NP.

El problema consiste en determinar si un grafo G tiene un clique de tamaño K. Un clique de tamaño K en un grafo G es un subgrafo completo de G con K vértices. El lenguaje que representa el problema es:

CLIQUE = {(G, K) | G es un grafo que tiene un clique de tamaño K}



- La mejor MT conocida para **decidir** si un grafo G tiene un clique de tamaño K, consiste en recorrer una a una todas las secuencias C de K vértices de V y chequear en cada caso si C determina un clique.

  Existen $\binom{m}{K}$ secuencias C de K vértices en V, siendo m la cantidad de vértices de V.

  Así, las iteraciones de la MT suman $O(m.(m-1).(m-2)\ldots(m-K+1)/K!) = O(m^m) = O(n^n) =$ **exp(n).**

  CLIQUE no estaría en P.

- Por otro lado, se puede **verificar** en tiempo poly(n) si una secuencia C de K vértices define un clique de tamaño K de un grafo G: se chequea que todos los pares de vértices de C son arcos de G, lo que tarda $O(|K|^2.|E|) = O(|V|^2.|E|) = O(|G|^3) = O(n^3) =$ **poly(n).**

  CLIQUE está en NP. M es un verificador eficiente de CLIQUE, con certificados C de tamaño O(n).

Ejemplo 2. Variante del problema del clique, que está en la clase P.

El enunciado del problema es el mismo que antes, pero con la diferencia de que ahora el tamaño K del clique no forma parte de las instancias del problema, es una constante. El lenguaje que representa el problema es:

K-CLIQUE = {G | G es un grafo que tiene un clique de tamaño K}

- Como K no forma parte de las entradas del algoritmo descripto previamente, el problema ahora tiene resolución polinomial, **está en P**:

  a) Hay $O(m.(m – 1).(m – 2)…(m – K + 1) / K!) = O(m^K) = O(n^K)$ secuencias de K vértices en V para chequear.

  b) Cada chequeo se puede hacer en tiempo $O(n^3)$.

  Total: $O(n^K.n^3)$ = poly(n) pasos.

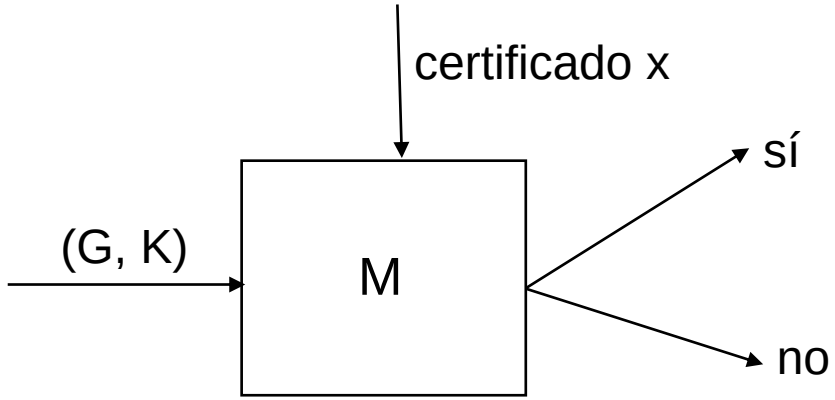
  Nota: podría discutirse de todos modos si para un K muy grande, por ejemplo 1000, el tiempo de la MT construida puede considerarse aceptable.

Ejemplo 3. Otra variante del problema del clique, que estaría fuera de NP.

Sea el problema complemento del problema del clique. El lenguaje que lo representa es:

NO-CLIQUE = {(G, K) | G es un grafo que no tiene un clique de tamaño K}

- En este caso, no alcanza con certificados de K vértices. Un certificado debe incluir a **todas** las secuencias de K vértices de V, porque se tiene que chequear que ninguna determina un clique de tamaño K de G.



x tiene que incluir las exp(n) secuencias de K vértices de G. De esta manera, NO-CLIQUE **no estaría en NP.**

- Se conjetura que **la clase NP no es cerrada con respecto al complemento.**