

# No determinismo y la clase NP

Semana (6)<sub>2</sub> = 110

Lógica para Ciencia de la  
Computación - IIC2213

Prof. Sebastián Bugedo

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

Epílogo

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

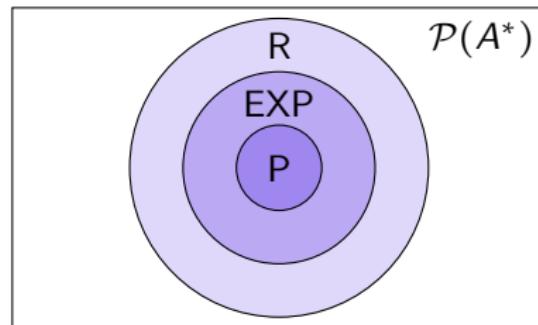
Epílogo



# P y EXP

## Teorema

Las clases P y EXP cumplen que  $P \subseteq EXP$



La contención es **estricta**

# ¿Dónde vive SAT?

Proposición

$SAT \in EXP$

¿Significa que SAT no se puede resolver en tiempo sub-exponencial?

## Dificultad relativa

Definición (hardness)

Dada una clase de complejidad  $\mathcal{C}$  que contiene a P, decimos que un lenguaje  $L$  es  $\mathcal{C}$ -hard si

$$\text{para todo } L' \in \mathcal{C}, \quad L' \leq_p L$$

Un lenguaje  $\mathcal{C}$ -hard es al menos tan difícil como todo problema de  $\mathcal{C}$

# Dificultad exacta

## Definición (completitud)

Dada una clase de complejidad  $\mathcal{C}$  que contiene a P, decimos que un lenguaje  $L$  es  $\mathcal{C}$ -completo si

- $L \in \mathcal{C}$
- $L$  es  $\mathcal{C}$ -hard



Los problemas completos son **representantes** de su clase

# Buscando la verdad sobre NP

No sabemos probar que  $SAT \in P\dots$

... en lugar de comparar  $SAT$  con  $P$ , busquemos su clase

*¿Qué clase representa la complejidad de SAT?*

*¿Para qué clase SAT es completo?*

*¿Esa supuesta clase es distinta de P?*

Hoy: no determinismo y la clase NP

# Playlist Unidad II y Orquesta



Playlist: LogiWawos #2

Además sigan en instagram:

@orquesta\_tamen

## Objetivos de la clase

- Comprender el modelo de máquina no determinista
- Diferenciar del modelo determinista
- Definir la clase de complejidad NP
- Comprender concepto de testigo y verificador en NP
- Relacionar NP con P y EXP
- Conocer un primer problema NP-completo

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

Epílogo

# Máquinas de Turing no deterministas

## Definición

Un **máquina de Turing no determinista** (NTM) es una estructura

$$\mathcal{M} = (Q, A, q_0, F, \Delta)$$

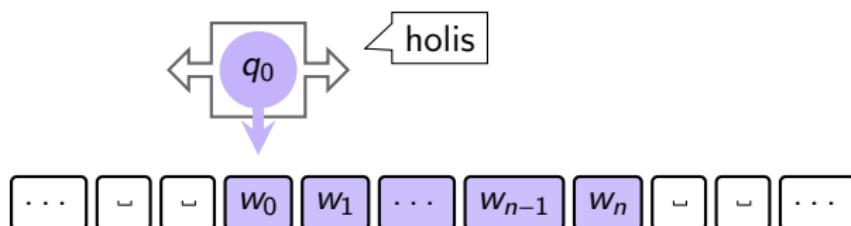
- $Q$  es un conjunto finito de **estados**
- $A$  es el alfabeto de **input**, tal que  $\sqcup \notin A$  está reservado
- $q_0$  es el estado **inicial**
- $F \subseteq Q$  es un conjunto finito de **estados finales** ( $F \neq \emptyset$ )
- $\Delta \subseteq Q \times (A \cup \{\sqcup\}) \times Q \times (A \cup \{\sqcup\}) \times \{\triangleleft, \triangleright\}$  es la **relación de transición** ( $\Delta \neq \emptyset$ )

# Funcionamiento de una máquina no determinista

## 1. Inicialización

- Se comienza con la palabra de input  $w = w_0 w_1 \dots w_n$
- El input se ubica en una posición **arbitraria** de la cinta
- La cabeza lectora se coloca en  $w_0$  con estado inicial  $q_0$

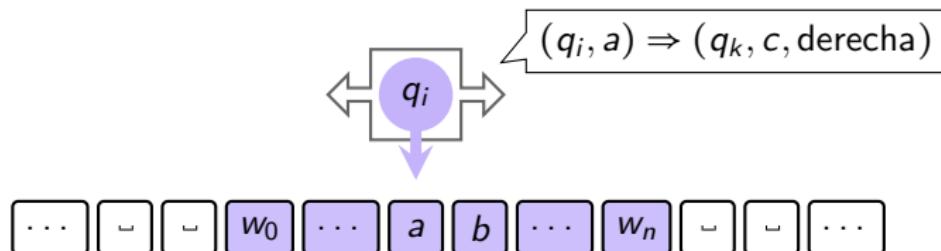
Idéntica a la inicialización en máquinas deterministas



# Funcionamiento de una máquina no determinista

## 2. Durante su funcionamiento

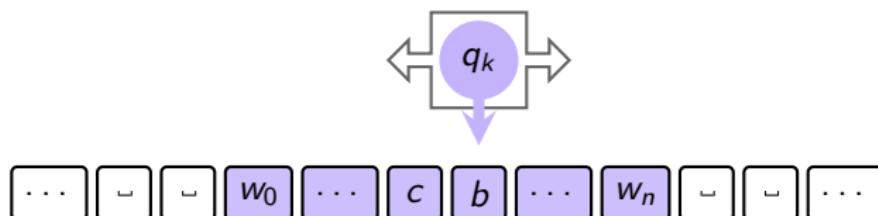
- La cabeza **lee** el símbolo  $a$  apuntado por ella
- Determina el conjunto de **instrucciones** para  $(q_i, a)$ .
  - ▶ Si tal conjunto es vacío: la máquina se **detiene**
  - ▶ Si no, se escoge una instrucción de  $\Delta$  y se **ejecuta**



# Funcionamiento de una máquina no determinista

## 2. Durante su funcionamiento

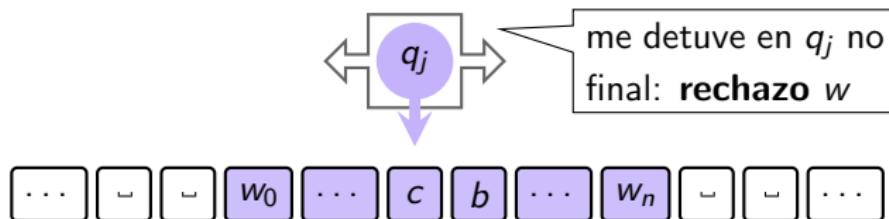
- La cabeza **lee** el símbolo  $a$  apuntado por ella
- Determina el conjunto de **instrucciones** para  $(q_i, a)$ .
  - ▶ Si tal conjunto es vacío: la máquina se **detiene**
  - ▶ Si no, se escoge una instrucción de  $\Delta$  y se **ejecuta**



# Funcionamiento de una máquina no determinista

## 3. Detención

- Si la máquina se detiene con estado  $q_j$ 
  - ▶ Si  $q_j$  es un **estado final**, la máquina **acepta** el input  $w$
  - ▶ Si no lo es, la máquina **rechaza**  $w$



Tal como en MT deterministas, la máquina puede no detenerse

# Configuraciones

Definición (configuraciones)

Decimos que la configuración

$$u \cdot q \cdot v \in A_{\sqcup}^* \cdot Q \cdot A_{\sqcup}^*$$

- es de **detención** si  $\delta(q, a)$  no está definido, con  $v = a \cdot v'$
- es de **aceptación** si es de detención y  $q \in F$
- es de **rechazo** si es de detención y  $q \notin F$

Igual que en MTD: resume el estado **actual** de la máquina

# Pasos

## Definición (pasos)

Sean  $a, b, c \in A$  y  $u, v \in A^*$ . Se define la relación de **siguiente paso**

$$\xrightarrow{\mathcal{M}} \subseteq (A^* \cdot Q \cdot A^*) \times (A^* \cdot Q \cdot A^*)$$

- Si  $(p, b, q, c, \triangleleft) \in \Delta$ , entonces

$$u \cdot a \cdot \mathbf{p} \cdot b \cdot v \xrightarrow{\mathcal{M}} u \cdot \mathbf{q} \cdot a \cdot c \cdot v$$

- Si  $(p, b, q, c, \triangleright) \in \Delta$ , entonces

$$u \cdot \mathbf{p} \cdot b \cdot v \xrightarrow{\mathcal{M}} u \cdot c \cdot \mathbf{q} \cdot v$$

Un paso es un cambio de configuración **válido** según  $\Delta$

# Ejecuciones

## Definición (ejecuciones)

Sea  $\mathcal{M} = (Q, A, q_0, F, \Delta)$  una NMT y  $w \in A^*$

- Una **ejecución**  $\rho$  de  $\mathcal{M}$  es una secuencia de configuraciones

$\rho : C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \dots$  (no necesariamente finita)

tal que  $C_i \xrightarrow{\mathcal{M}} C_{i+1}$  para todo  $C_i$  y  $C_{i+1}$  en  $\rho$ .

Idéntico hasta aquí al caso determinista

# Ejecuciones

## Definición (ejecuciones)

Sea  $\mathcal{M} = (Q, A, q_0, F, \Delta)$  una NMT y  $w \in A^*$

- Decimos que  $\rho : C_0 \rightarrow \dots$  es **una ejecución** de  $\mathcal{M}$  sobre  $w$  si:
  - $\rho$  es una ejecución de  $\mathcal{M}$
  - $C_0 = q_0 \cdot w$
  - Si  $\rho$  es **finita**, entonces su última conf.  $C_m$  es de **detención**
- Si  $C_m$  es de aceptación, entonces  $\rho$  es una **ejecución de aceptación** para  $w$ .

Ahora vienen los cambios respecto a la versión determinista!

# Aceptación

## Definición (aceptación)

Sea  $\mathcal{M} = (Q, A, q_0, F, \Delta)$  una NMT y  $w \in A^*$

- Decimos que  $\mathcal{M}$  **acepta**  $w$  si existe una ejecución de aceptación para  $w$
- Decimos que  $\mathcal{M}$  **rechaza**  $w$  si no existe ninguna ejecución de aceptación para  $w$

Basta con una ejecución de aceptación para aceptar...

**PUEDE HABER MÁS DE UNA**

# Árbol de ejecuciones

## Definición (árbol de ejecuciones)

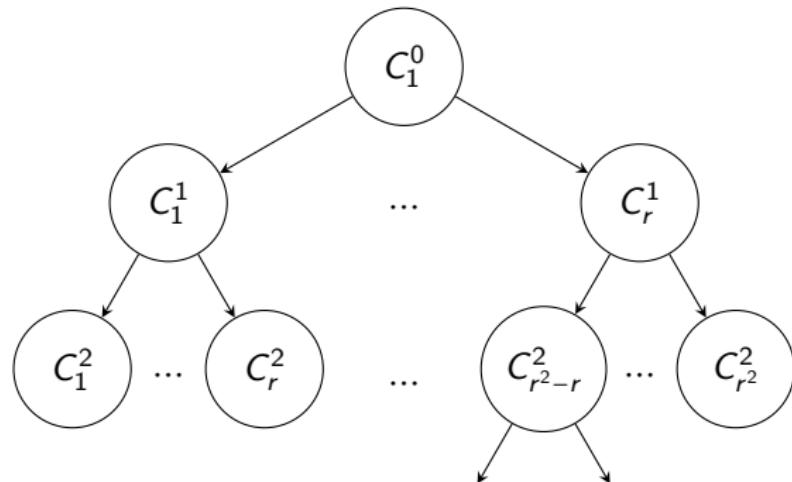
Sea  $\mathcal{M} = (Q, A, q_0, F, \Delta)$  una NMT y  $w \in A^*$ . El **árbol de ejecuciones** de  $\mathcal{M}$  sobre  $w$  es un árbol dirigido tal que

- Cada nodo es una configuración  $C_i^j$  ( $i$ -ésima configuración del nivel  $j$  del árbol)
- Su raíz es la configuración  $C_1^0 = q_0 \cdot w$
- El nodo con configuración  $C_i^j$  es padre de todos los nodos  $C_k^{j+1}$  tales que  $C_i^j \xrightarrow{\mathcal{M}} C_k^{j+1}$

¿Qué son las hojas del árbol? ¿Es un árbol finito?

# Árbol de ejecuciones

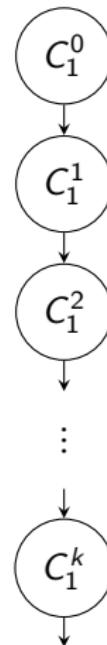
Si suponemos que la ramificación está acotada por  $r$



El árbol puede ser infinito y las hojas son configuraciones de detención

# Árbol de ejecuciones

Si una máquina tiene un árbol de la siguiente forma **para todo input**... ¿qué tipo de máquina es?



El árbol para MT deterministas es una línea (posiblemente infinita)

# Lenguaje aceptado

Definición (lenguaje aceptado)

Sea  $\mathcal{M} = (Q, A, q_0, F, \Delta)$  una NMT. Definimos el **lenguaje aceptado** por  $\mathcal{M}$  como

$$L(\mathcal{M}) = \{w \in A^* \mid \mathcal{M} \text{ acepta } w\}$$

$L(\mathcal{M})$  reúne las de palabras que tienen  
**al menos una** ejecución de aceptación

# Máquinas no deterministas

## Ejercicio

Construya una NMT que solo mueva su cabeza hacia la derecha y acepte el siguiente lenguaje

$$L = \{w \in \{0,1\}^* \mid \text{el largo de } w \text{ es par o solo tiene símbolos 1}\}$$



¿Se puede resolver con una máquina determinista con la misma restricción (ir solo a la derecha)?

## Poder expresivo

¿Son las NTM más poderosas que las TM deterministas?

Nos fijamos en los lenguajes que pueden aceptar

# Poder expresivo

Es claro que una máquina determinista  $\mathcal{M}$  se puede llevar a NTM  $\mathcal{N}$ :

- Alfabeto, estados y estados finales son los mismos
- La función de transición  $\delta$  se lleva a relación:

$$\delta(p, b) = (q, c, d) \Rightarrow (p, b, q, c, d) \in \Delta$$

- Cuando  $w$  es aceptada por  $\mathcal{M}$ , existe una ejecución de aceptación en  $\mathcal{M}$ : la misma ejecución permite aceptar  $w$  en  $\mathcal{N}$

## Teorema

Para toda MT  $\mathcal{M}$  existe una NTM  $\mathcal{N}$  tal que  $L(\mathcal{M}) = L(\mathcal{N})$

¿La otra dirección es posible?  
¿Hay una forma de determinizar máquinas?

# Poder expresivo

## Teorema

Para toda NMT  $\mathcal{N}$  existe una TM  $\mathcal{M}$  tal que  $L(\mathcal{N}) = L(\mathcal{M})$

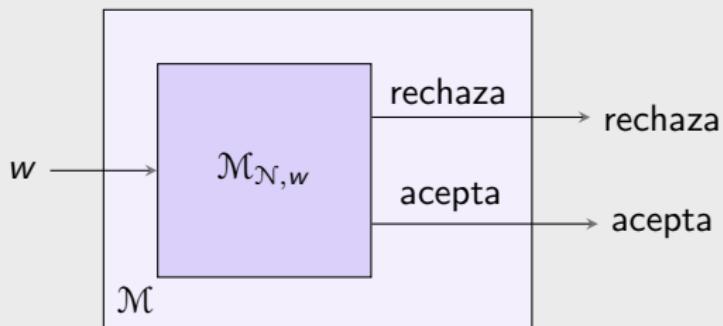
El no determinismo no entrega más poder a las máquinas

# Poder expresivo

## Demostración

Construiremos una máquina determinista  $\mathcal{M}$  tal que

1. Simula una máquina determinista  $\mathcal{M}_{\mathcal{N},w}$
2.  $\mathcal{M}_{\mathcal{N},w}$  representa las posibles ejecuciones de  $w$  en  $\mathcal{N}$



Definiremos ahora la máquina  $\mathcal{M}_{\mathcal{N},w}$  adecuada.

# Poder expresivo

## Demostración

La máquina  $\mathcal{M}_{\mathcal{N},w}$  inicia con la cinta con palabra

...  $\sqcup \sqcup \bullet \bullet q_0 W \bullet \bullet \sqcup \sqcup \dots$

donde  $\bullet$  es un carácter reservado y  $q_0$  es el estado inicial de  $\mathcal{N}$ .

El funcionamiento de la máquina es el siguiente:

- Se borra la configuración escrita en el extremo izquierdo
- Se escribe en el extremo derecho cada configuración alcanzable desde la que se borró
- Se repite hasta que se borre una configuración sin transiciones y que sea **de aceptación**

Cada fase de borrar y escribir las configuraciones alcanzables se llama **simulación de un paso no determinista**.

# Poder expresivo

## Demostración

Si en un momento dado, la cinta de  $\mathcal{M}_{\mathcal{N}, w}$  tiene el siguiente contenido

$$\dots \sqcup \sqcup \bullet \bullet y_1 q_1 z_1 \bullet \dots \bullet y_r q_r z_r \bullet \bullet \sqcup \sqcup \dots$$

en la siguiente simulación de un paso de  $\mathcal{N}$  se obtiene

$$\dots \sqcup \sqcup \bullet \bullet y_2 q_2 z_2 \bullet \dots \bullet y_r q_r z_r \bullet \tilde{y}_1 \tilde{q}_1 \tilde{z}_1 \bullet \dots \bullet \tilde{y}_r \tilde{q}_r \tilde{z}_r \bullet \bullet \sqcup \sqcup \dots$$

donde las últimas  $r$  configuraciones se obtuvieron desde  $y_1 q_1 z_1$ .

Este proceso termina cuando se encuentra la ejecución de aceptación más corta de  $\mathcal{N}$  sobre  $w$ , aceptando. Por construcción, solo se aceptarán palabras tales que  $w \in L(\mathcal{N})$ , de donde se concluye que  $L(\mathcal{M}) = L(\mathcal{N})$ . □

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

Epílogo

# El concepto de paso

## Notación

Dada una NMT determinista  $\mathcal{M}$  con alfabeto  $A$ , llamamos

- **Paso de  $\mathcal{M}$  en  $w$**  a un par  $(C_i, C_{i+1})$  en una ejecución de  $\mathcal{M}$  sobre el input  $w$
- **Tiempo de  $\mathcal{M}$  en  $w$**  al número de pasos de la ejecución más corta de  $\mathcal{M}$  que acepta  $w$ . Lo denotamos por

$$\text{tiempo}_{\mathcal{M}}^*(w) = |\{(C_i, C_{i+1}) \mid (C_i, C_{i+1}) \text{ es un paso de } \mathcal{M} \text{ en } w\}|$$

Solo definimos  $\text{tiempo}_{\mathcal{M}}(w)$  para palabras **aceptadas** por  $\mathcal{M}$

# Tiempo de ejecución de una máquina

Definición (tiempo de ejecución)

Sea  $\mathcal{M}$  una NMT con alfabeto  $A$  que se detiene en todo input. Dado un natural  $n$ , definimos el **tiempo de ejecución de  $\mathcal{M}$  en el peor caso** como

$$t_{\mathcal{M}}(n) = \max\{\{n\} \cup \{\text{tiempo}_{\mathcal{M}}^*(w) \mid w \in A^* \text{ y } |w| = n\}\}$$

¿Por qué se agrega  $\{n\}$ ?

# Clases de complejidad no deterministas

Definición (tiempo de aceptación)

Decimos que un lenguaje  $L$  **puede ser aceptado en tiempo  $g$  por una máquina no determinista** si existe una NTM  $\mathcal{M}$  tal que

- $L = L(\mathcal{M})$
- $t_{\mathcal{M}} \in \mathcal{O}(g)$

Notemos que no exigimos que la máquina decida  $L$ , solo que lo acepte

## Clases de complejidad no deterministas

## Definición (clases de tiempo)

Dado un alfabeto  $A$ , se define

$$\text{NTIME}(g) = \{L \subseteq A^* \mid L \text{ puede ser aceptado en tiempo } g \\ \text{por una máquina no determinista}\}$$

### Una clase fundamental

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

¿Conocemos algún problema en NP?  
JIJIJIJIJIIIIJJJJJJJJJJJJJJJJJji ×7000

# Programa

Obertura

Primer acto

No determinismo

Complejidad

**Intermedio**

Segundo acto

La clase NP

NP vs P

Epílogo

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

Epílogo

# La definición de NP

La definición que tenemos de NP es

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

pero no nos dice mucho de *qué* es *NP* realmente

Algunas ideas para guiar nuestra búsqueda

- Así como P es una clase de problemas con solución eficiente...
- ¿Cómo podemos caracterizar los problemas en NP?
- ¿Qué hace la máquina  $\mathcal{N}$  que acepta  $L \in \text{NP}$ ?

Daremos una definición mucho más intuitiva

# La definición de NP

Teorema (def. alternativa de NP)

Sea  $L$  un lenguaje.  $L \in \text{NP}$  si, y solo si, existe una máquina determinista  $\mathcal{M}$  que funciona en tiempo polinomial y un polinomio  $p(\cdot)$  tales que

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$

¿Cómo describimos NP *en chileno* según el teorema?

# La definición de NP

Teorema (def. alternativa de NP)

Sea  $L$  un lenguaje.  $L \in \text{NP}$  si, y solo si, existe una máquina determinista  $\mathcal{M}$  que funciona en tiempo polinomial y un polinomio  $p(\cdot)$  tales que

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$

Observaciones

- La máquina  $\mathcal{M}$  se conoce como **verificador**
- La palabra  $y$  se conoce como **testigo** o **certificado** de  $x$
- El testigo sirve para que el verificador compruebe que  $x \in L$

El testigo solo sirve para el caso afirmativo en que  $x \in L$

# La definición de NP

Teorema (def. alternativa de NP)

Sea  $L$  un lenguaje.  $L \in \text{NP}$  si, y solo si, existe una máquina determinista  $\mathcal{M}$  que funciona en tiempo polinomial y un polinomio  $p(\cdot)$  tales que

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$

Traducción a chileno:

NP es el conjunto de problemas cuya solución  
se puede verificar fácilmente

¿Por qué? ¿Qué es el testigo en la práctica?

# La definición de NP

## Demostración

( $\Leftarrow$ ) Sea  $L$  tal que existen  $\mathcal{M}$  MT polinomial y polinomio  $p(\cdot)$  que cumplen

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$

Debemos probar que  $L$  se puede aceptar en tiempo polinomial por una máquina **no determinista**  $N$ . Dicha máquina es esencialmente  $\mathcal{M}$ .

# La definición de NP

## Demostración

Para  $w$  cualquiera

- Si  $w \in L$

- existe un testigo  $y$  de largo polinomial respecto a  $|w|$
- la máquina  $\mathcal{M}$  acepta  $(w, y)$  en tiempo polinomial respecto a  $|(w, y)|$
- $|(w, y)|$  es polinomial respecto a  $|w|$

Como la máquina  $\mathcal{M}$  es determinista, en particular sirve como la máquina no determinista que acepta  $L$  en tiempo polinomial respecto a  $|w|$ . Concluimos que  $L \in \text{NP}$ .

Al paso de invocar el testigo que existe para  $w$   
se le llama informalmente **adivinar el testigo**

# La definición de NP

## Demostración

( $\Rightarrow$ ) Sea  $L \in \text{NP}$ . Es decir, existe una máquina no determinista  $\mathcal{N}$  tal que acepta  $L$  en tiempo polinomial. Más precisamente, para  $w \in L$

- existe una ejecución de aceptación  $\rho$  en  $\mathcal{N}$
- $\rho$  tiene una cantidad de pasos polinomial en  $|w|$

Luego, el testigo  $y$  para  $w$  es simplemente la ejecución. La máquina determinista polinomial  $\mathcal{M}$  se define según:

- toma input  $(w, \rho)$
- verifica si  $\rho$  es una ejecución de aceptación de  $\mathcal{N}$  en  $w$

Como el largo del testigo es polinomial y  $\mathcal{M}$  funciona en tiempo polinomial respecto al largo del testigo,  $L$  cumple que

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$



# La definición de NP

Teorema (def. alternativa de NP)

Sea  $L$  un lenguaje.  $L \in \text{NP}$  si, y solo si, existe una máquina determinista  $\mathcal{M}$  que funciona en tiempo polinomial y un polinomio  $p(\cdot)$  tales que

$$L = \{x \mid \text{existe } y \text{ tal que } |y| = p(|x|) \text{ y } \mathcal{M} \text{ acepta } (x, y)\}$$

El teorema nos da una nueva forma de demostrar que  $L \in \text{NP}$ :  
buscar testigo y verificador polinomiales

## Testigos polinomiales

Dada una palabra  $w \in L$ , con  $L \in \text{NP}$ , ya sabemos que una ejecución de aceptación sirve como testigo

¿Es el más conveniente en general?

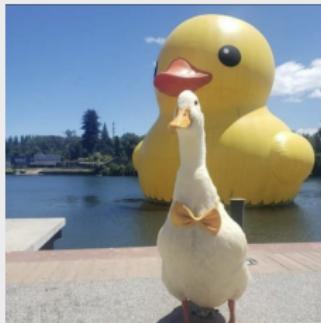
¿Podemos usar otra palabra como testigo?

# Testigos polinomiales

## Ejercicio

Demuestre que los siguientes lenguajes están en NP

1. SAT =  $\{\varphi \mid \varphi \text{ es satisfacible}\}$
2. 3SAT =  $\{\varphi \mid \varphi \text{ está en 3-CNF y es satisfacible}\}$
3. 3COL =  $\{G \mid G \text{ grafo no dirigido 3-coloreable}\}$



# Testigos polinomiales

Entregaremos el testigo y verificador polinomiales

1.  $SAT = \{\varphi \mid \varphi \text{ es satisfacible}\}$

Dada una fórmula  $\varphi \in SAT$ ,

- testigo  $\sigma$  valuación de largo polinomial respecto a  $|\varphi|$
- verificador  $\mathcal{M}$  que evalúa  $\sigma(\varphi)$  (vimos que era polinomial evaluar!!)

2.  $3SAT = \{\varphi \mid \varphi \text{ está en 3-CNF y es satisfacible}\}$

Dada una fórmula  $\varphi \in 3SAT$ , idem que SAT

3.  $3COL = \{G \mid G \text{ grafo no dirigido 3-coloreable}\}$

Dado un grafo  $G \in 3COL$ ,

- testigo 3-coloración para los vértices
- verificador  $\mathcal{M}$  que revisa cada arista (cantidad de aristas polinomial respecto a  $|G|$ )

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

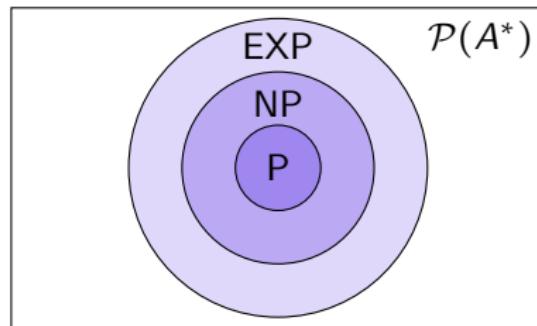
NP vs P

Epílogo

# ¿Cómo se relaciona NP con P y EXP?

Teorema

Las clases P, NP y EXP cumplen que  $P \subseteq NP \subseteq EXP$



Al menos una de las inclusiones es estricta. ¿Por qué?

# ¿Cómo se relaciona NP?

## Demostración

$$(P \subseteq NP)$$

Sea  $L \in P$ , i.e. existe  $\mathcal{M}$  que acepta  $L$  en tiempo polinomial. Sabemos que la misma máquina sirve como máquina no determinista que acepta  $L$ . Por lo tanto,  $L \in NP$ .

# ¿Cómo se relaciona NP?

## Demostración ( $\text{NP} \subseteq \text{EXP}$ )

Sea  $L \in \text{NP}$ , i.e. existe máquina no determinista  $\mathcal{N}$  que acepta  $L$  en tiempo polinomial. Demostramos que existe una máquina determinista  $\mathcal{M}$  tal que

$$L(\mathcal{M}) = L(\mathcal{N}) = L$$

Dicha construcción cumple que luego de simular  $k$  pasos no deterministas de  $\mathcal{N}$ , tiene un contenido de largo  $\mathcal{O}(r^k)$  en la cinta, cuando la ramificación del árbol es  $\mathcal{O}(r)$  (peor caso).

# ¿Cómo se relaciona NP?

## Demostración

Como el procesamiento de ese contenido de tamaño  $\mathcal{O}(r^k)$  toma tiempo polinomial (lectura y escritura de nuevas configuraciones a partir de un paso de  $\Delta$ ), simular  $k$  pasos de  $\mathcal{N}$  toma tiempo exponencial.

Es claro que para simular una ejecución de aceptación, existe un  $k$  tal que hay una ejecución de  $k$  pasos en  $\mathcal{N}$ , y se simula en tiempo  $\mathcal{O}(r^k)$  en  $\mathcal{M}$ . Concluimos que  $\mathcal{M}$  acepta  $L$  en tiempo exponencial. □

Notemos que solo nos interesa aceptar, pues esa es la definición de

$$L(\mathcal{N}) = L$$

# Problemas NP-completos

En nuestro objetivo de ubicar SAT, es útil saber si es completo para alguna clase

Teorema (Cook-Levin)

El problema SAT es NP-completo

SAT es el más difícil de los problemas en la clase NP

# Problemas NP-completos

Teorema (Cook-Levin)

El problema SAT es NP-completo

Demostración

Próxima clase jj



# Problemas NP-completos

## Teorema (Cook-Levin)

El problema SAT es NP-completo

El teorema de Cook-Levin es clave por varias razones

- Nos entrega un primer problema NP-completo
- Cualquier  $L \in \text{NP}$  tal que  $\text{SAT} \leq_p L$  es también NP-completo
- Si probamos que  $\text{SAT} \in \text{P}$ , por teorema de reducciones tenemos que  $\text{P} = \text{NP}$

SAT es clave para relacionar problemas con solución eficiente y solución verificable de forma eficiente

# Conjetura de Cook

Conjetura de Cook

$P \neq NP$

¿Por qué es tan importante esta conjetura?

Si es cierta

- No hay algoritmo polinomial para SAT
- Tampoco lo hay para los problemas NP-completos

Uno de los Problemas del Milenio

**¿P=NP?**

**NADIE SABE CÓMO HACER  
ESTO AYUDAA**



# Hacia dónde vamos

Estudiaremos en detalle otros problemas NP-completos y sus potencialidades

*¿Hay versiones de problemas NP-completos que sí se puedan resolver eficientemente?*

*¿Todo problema en NP es NP-completo?*

*¿Qué implicancias tendría si  $P=NP$ ?  
¿Se acaba el mundo?*

Próxima clase: problemas NP-completos y consecuencias de  $P=NP$

# Programa

Obertura

Primer acto

No determinismo

Complejidad

Intermedio

Segundo acto

La clase NP

NP vs P

Epílogo

## Objetivos de la clase

- Comprender el modelo de máquina no determinista
- Diferenciar del modelo determinista
- Definir la clase de complejidad NP
- Comprender concepto de testigo y verificador en NP
- Relacionar NP con P y EXP
- Conocer un primer problema NP-completo

¿Qué aprendí hoy? ¿Comentarios?

Vea

[www.menti.com](https://www.menti.com)

Introduce el código

**78 95 48 5**



O usa el código QR