Complejidad Computacional

Santiago Figueira

Departamento de Computación - FCEN - UBA

clase 4

Clase 4

- Máquinas no-determinísticas
- Reducibilidad polinomial
- Lógica proposicional

Máquinas no-deterministas

Clase 4

Máquinas no-deterministas

Reducibilidad polinomial

Lógica proposicional

Máquinas determinísticas

Las máquinas que vimos hasta ahora son *determinísticas*: cada configuración evoluciona de una única forma a partir de la configuración inicial.

Un cómputo de $M = (\Sigma, Q, \delta)$ a partir de x es

$$C_0 \xrightarrow{\delta} C_1 \xrightarrow{\delta} C_2 \xrightarrow{\delta} \ldots \xrightarrow{\delta} C_\ell$$

tal que C_ℓ es final.
A partir de ahora, las llamamos también **máquinas determinísticas**.

Notación: Máquina determinística

Máquina = máquina determinística

Máquinas no-deterministas

Las máquinas **no-deterministas** son triplas (Σ, Q, δ), como antes, pero con dos diferencias respecto a las deterministas:

1. la función de transición es

$$\delta : Q \times \Sigma^{k-1} \to \left(\underbrace{\{L, R.S\}}_{\text{entrada}} \times \underbrace{\Sigma^{k-2} \times \{L, R, S\}^{k-2}}_{\text{trabajo}} \times \underbrace{(\Sigma \cup \{S\})}_{\text{salida}} \times Q \right)^2$$

   Ahora δ especifica una o dos posibles **evoluciones en un paso** a partir de una configuración dada.

2. Q tiene 3 estados distinguidos:

$$q_0 \quad , \quad q_{\text{sí}} \quad , \quad q_{\text{no}}$$

Cómputo de una máquina no-determinística

Definición

Una **configuración final** de una máquina no-determinística $N = (\Sigma, Q, \delta)$ es una en la que el estado es $q_{\text{sí}}$ o q_{no}. Un **cómputo** de N a partir de $x \in \{0,1\}^*$ es una secuencia $C_0, \ldots, C_\ell$ de configuraciones tal que

1. C_0 es inicial a partir de x
2. C_{i+1} es la evolución de C_i en un paso dado por alguna de las 2 tuplas de δ
3. C_ℓ está en estado $q_{\text{sí}}$ o q_{no}

Un **cómputo aceptador** es uno en el que C_ℓ está en estado $q_{\text{sí}}$, y en ese caso a C_ℓ lo llamamos configuración **aceptadora** Llamamos **longitud** del cómputo a ℓ.

A diferencia de las máquinas determinísticas, hay posiblemente muchos cómputos a partir de una configuración dada.

Lenguaje aceptado por una máquina no-determinística

Definición

Decimos que la máquina no-determinística N **acepta** x si existe un cómputo aceptador $C_0, \ldots, C_\ell$ de N a partir de x. En caso contrario decimos que N **rechaza** x.
Decimos que N **decide** el lenguaje $\mathcal{L}$ si para todo $x \in \{0,1\}^*$,

$$x \in \mathcal{L} \quad \text{sii} \quad N \text{ acepta } x$$

Observación

- Las máquinas no-determinísticas no computan funciones; solo aceptan lenguajes (que es una forma de computar funciones valuadas en $\{0,1\}$).
- En las máquinas no-determinísticas la cinta de salida es irrelevante (porque no participa de ninguna definición).

Lenguaje aceptado por una máquina no-determinística

Notación: $N(x), \mathcal{L}(N)$

Sea N una máquina no-determinística. $N(x) = 1$ si N acepta x; $N(x) = 0$ si N rechaza x.
El lenguaje decidido por N es $\mathcal{L}(N)$.

Tiempo de cómputo de una máquina no-determinística

Definición

La máquina no-determinística N **corre en tiempo $T(n)$** si para todo $x \in \{0,1\}^*$, *todo* cómputo de N a partir de x tiene longitud $\leq T(|x|)$.

Definición

La máquina no-determinística N **corre en tiempo $O(T(n))$** si existe una constante c tal que para todo $x \in \{0,1\}^*$, salvo finitos, *todo* cómputo de N a partir de x tiene longitud $\leq c \cdot T(|x|)$.

La clase $\mathbf{NDTime}(T(n))$

Clase de complejidad: $\mathbf{NDTime}(T(n))$

$\mathbf{NDTime}(T(n))$ es la clase de lenguajes $\mathcal{L}$ tal que existe una máquina no-determinística N tal que

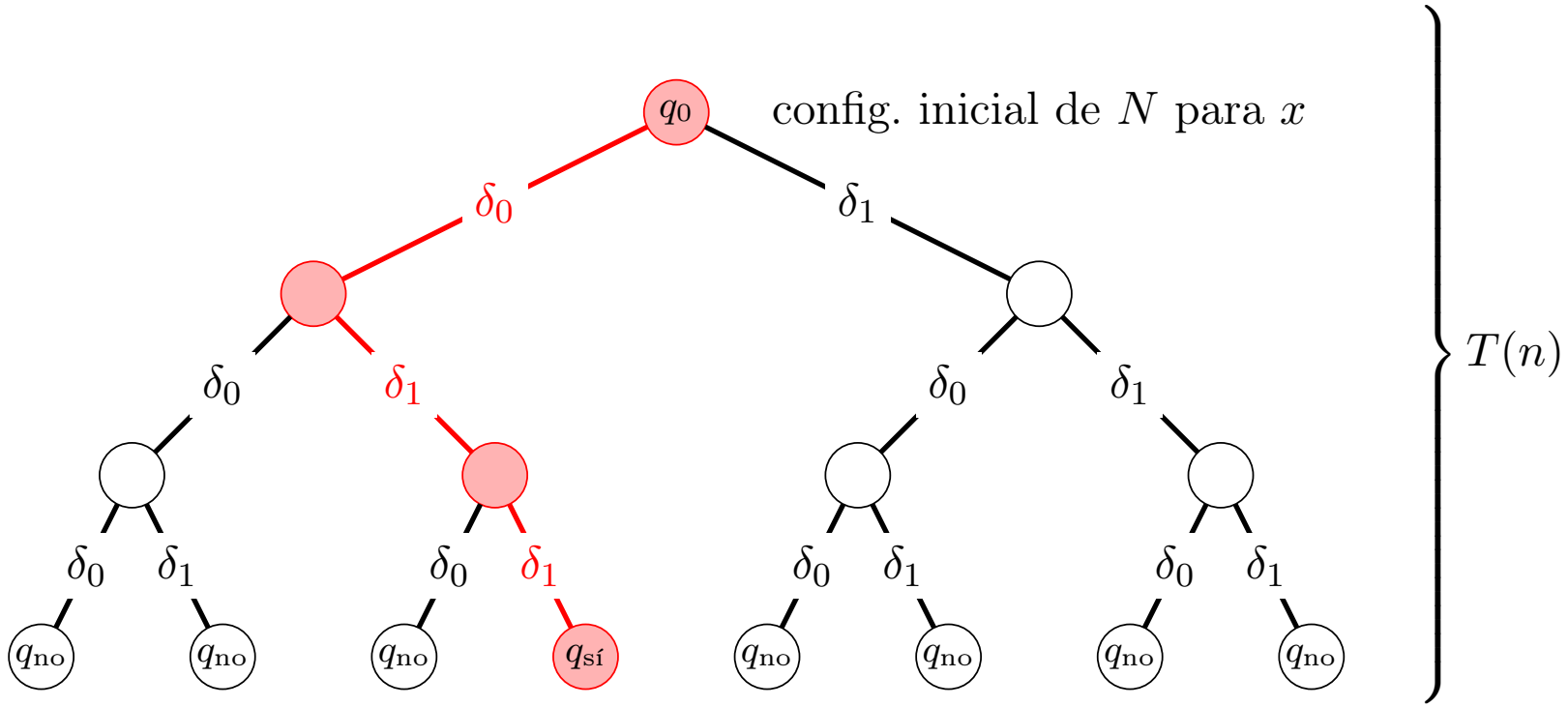
- N decide $\mathcal{L}$
- N corre en tiempo $O(T(n))$

Cómputos en máquinas no-deterministas

Supongamos la máquina no-determinística $N = (\Sigma, Q, \delta)$ que corre en tiempo $T(n)$. Dada $\delta(q, s) = (a, b)$ definamos

$$\delta_0(q, s) = a \quad \text{y} \quad \delta_1(q, s) = b.$$

Sea $x \in \{0,1\}^*$ y $n = |x|$



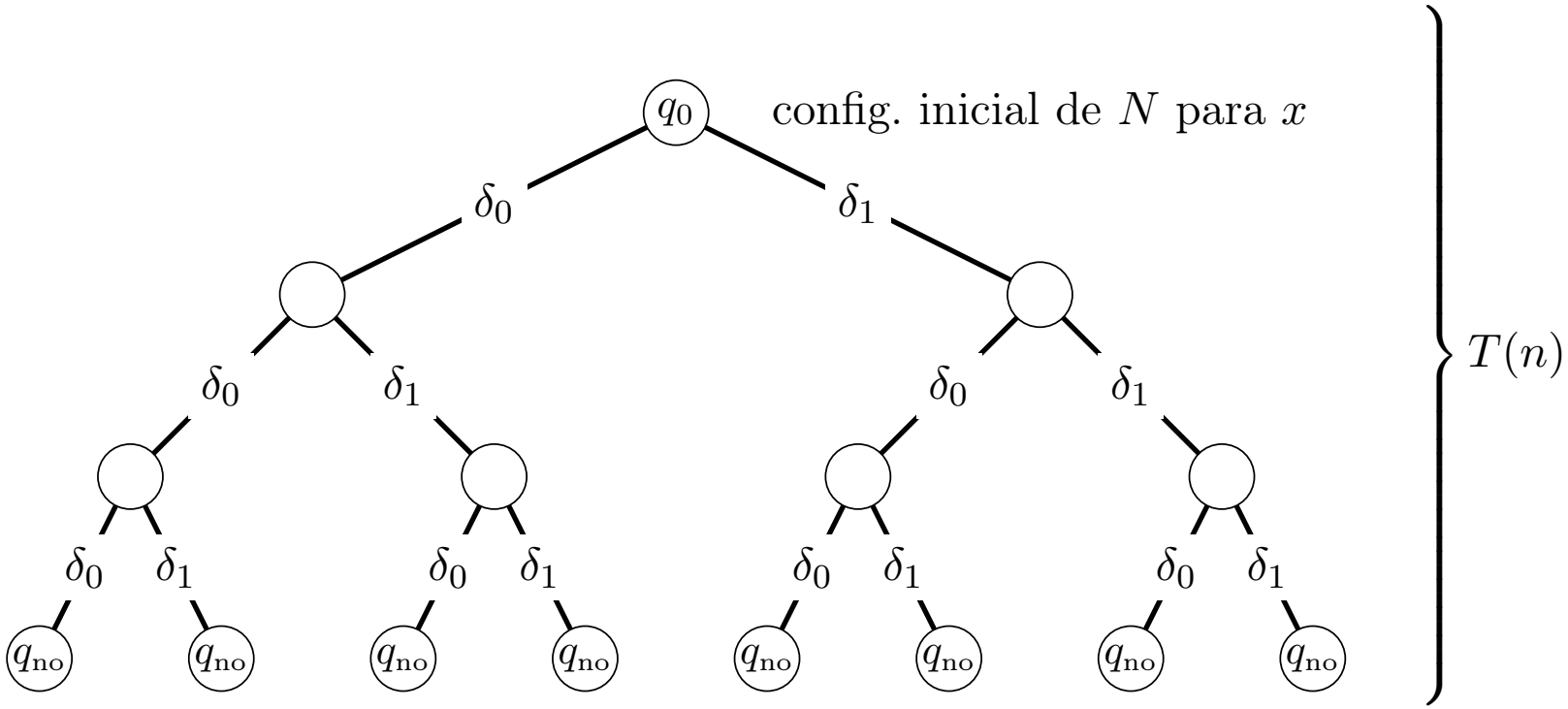
N acepta x porque existe un cómputo aceptador
Codificamos cómputos con secuencias $\{0,1\}^{T(n)}$.
011 es un cómputo aceptador de $N(x)$.

Cómputos en máquinas no-deterministas

Supongamos la máquina no-determinística $N = (\Sigma, Q, \delta)$ que corre en tiempo $T(n)$. Dada $\delta(q, s) = (a, b)$ definamos

$$\delta_0(q, s) = a \quad \text{y} \quad \delta_1(q, s) = b.$$

Sea $x \in \{0,1\}^*$ y $n = |x|$



N rechaza x porque no existe un cómputo aceptador
Codificamos cómputos con secuencias $\{0,1\}^{T(n)}$.
011 no es un cómputo aceptador de $N(x)$.

Simulación de una máquina no-determinística

- Sea $N = (\Sigma, Q, \delta)$ una máquina no-determinística
- Podemos simular determinísticamente *todos* los cómputos de longitud a lo sumo t de N a partir de cualquier x y determinar cuáles de esos son aceptadores.
- A cada cómputo lo codificamos con una palabra de $u \in \{0,1\}^t$
- Hay 2^t tales cómputos
- Para para u, simulamos N como si fuera determinística siguiendo el cómputo u (toma tiempo $O(t \cdot 2^t)$)
- Si N corre en tiempo $T(n)$, podemos decidir si N acepta o rechaza x con una máquina determinística que corre en tiempo $O(2^{T(n)^2})$

Definición alternativa de **NP**

Teorema

$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NDTime}(n^c)$.

Existe un polinomio $p : \mathbb{N} \to \mathbb{N}$ y una máquina determinística M que corre en tiempo polinomial tal que para todo x,

$$x \in \mathcal{L}$$
sii
existe $u \in \{0,1\}^{p(|x|)}$
tal que $M(\langle x, u \rangle) = 1$

$\iff$

Existe una máquina no-determinística N que corre en tiempo polinomial tal que

$$x \in \mathcal{L}$$
sii
existe un
cómputo aceptador
de N a partir de x

Prueba de $\mathbf{NP} \supseteq \bigcup_{c \in \mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina no-determinística $N = (\Sigma, Q, \delta)$ y un polinomio p tal que N corre en tiempo p y N decide $\mathcal{L}$.

$$x \in \mathcal{L} \quad \text{sii} \quad \text{existe un cómputo aceptador } C_0, \ldots, C_{p(|x|)} \text{ de } N \text{ a partir de } x.$$

Prueba de $\mathbf{NP} \supseteq \bigcup_{c\in\mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina no-determinística $N = (\Sigma, Q, \delta)$ y un polinomio p tal que N corre en tiempo p y N decide $\mathcal{L}$.

$$x \in \mathcal{L} \quad \text{sii} \quad \begin{array}{l}\text{existe un cómputo aceptador } C_0, \ldots, C_{p(|x|)} \\ \text{de } N \text{ a partir de } x.\end{array}$$

Ya vimos que podemos codificar cómputos con cadenas binarias. Consideremos $u \in \{0,1\}^{p(|x|)}$ tal que $u(i) = 0$ si se usa la primera componente de δ para pasar de C_i a C_{i+1} y $u(i) = 1$ en caso contrario.

Prueba de $\mathbf{NP} \supseteq \bigcup_{c\in\mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina no-determinística $N = (\Sigma, Q, \delta)$ y un polinomio p tal que N corre en tiempo p y N decide $\mathcal{L}$.

$$x \in \mathcal{L} \quad \text{sii} \quad \begin{array}{l}\text{existe un cómputo aceptador } C_0, \ldots, C_{p(|x|)} \\ \text{de } N \text{ a partir de } x.\end{array}$$

Ya vimos que podemos codificar cómputos con cadenas binarias. Consideremos $u \in \{0,1\}^{p(|x|)}$ tal que $u(i) = 0$ si se usa la primera componente de δ para pasar de C_i a C_{i+1} y $u(i) = 1$ en caso contrario.

Existe una máquina determinística M (verificador) tal que M con entrada $\langle x, u\rangle$ verifica que u (certificado) sea la codificación de un cómputo aceptador de N a partir de x. Si lo es, escribe 1 en la salida y si no escribe 0.

Prueba de $\mathbf{NP} \supseteq \bigcup_{c\in\mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina no-determinística $N = (\Sigma, Q, \delta)$ y un polinomio p tal que N corre en tiempo p y N decide $\mathcal{L}$.

$$x \in \mathcal{L} \quad \text{sii} \quad \begin{array}{l}\text{existe un cómputo aceptador } C_0, \ldots, C_{p(|x|)} \\ \text{de } N \text{ a partir de } x.\end{array}$$

Ya vimos que podemos codificar cómputos con cadenas binarias. Consideremos $u \in \{0,1\}^{p(|x|)}$ tal que $u(i) = 0$ si se usa la primera componente de δ para pasar de C_i a C_{i+1} y $u(i) = 1$ en caso contrario.

Existe una máquina determinística M (verificador) tal que M con entrada $\langle x, u\rangle$ verifica que u (certificado) sea la codificación de un cómputo aceptador de N a partir de x. Si lo es, escribe 1 en la salida y si no escribe 0.

$$x \in \mathcal{L} \quad \text{sii} \quad \text{existe } u \in \{0,1\}^{p(|x|)} \text{ tal que } M(\langle x, u\rangle) = 1.$$

M corre en tiempo polinomial, por lo tanto $\mathcal{L} \in \mathbf{NP}$. □

Prueba de $\mathbf{NP} \subseteq \bigcup_{c \in \mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina determinística M que corre en tiempo polinomial y un polinomio $p : \mathbb{N} \to \mathbb{N}$ tal que para todo x,

$$x \in \mathcal{L} \quad \text{sii} \quad \text{existe } u \in \{0,1\}^{p(|x|)} \text{ tal que } M(\langle x, u \rangle) = 1$$

Prueba de $\mathbf{NP} \subseteq \bigcup_{c \in \mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina determinística M que corre en tiempo polinomial y un polinomio $p : \mathbb{N} \to \mathbb{N}$ tal que para todo x,

$$x \in \mathcal{L} \quad \text{sii} \quad \text{existe } u \in \{0,1\}^{p(|x|)} \text{ tal que } M(\langle x, u \rangle) = 1$$

Definimos una máquina no-determinística N que con entrada x hace esto:

> *"inventa" una palabra $u \in \{0,1\}^{p(|x|)}$ en su cinta de trabajo; lo logra con el no-determinismo de δ, que escribe 0 o 1. Esto le toma tiempo $p(|x|)$.*
>
> *simula M con entrada $\langle x, u \rangle$. Esto le toma tiempo polinomial.*
>
> *si $M(\langle x, u \rangle) = 1$ entonces entra al estado $q_{sí}$; si no entra al estado q_{no}*

Prueba de $\mathbf{NP} \subseteq \bigcup_{c\in\mathbb{N}} \mathbf{NDTime}(n^c)$.

Supongamos una máquina determinística M que corre en tiempo polinomial y un polinomio $p : \mathbb{N} \to \mathbb{N}$ tal que para todo x,

$$x \in \mathcal{L} \quad \text{sii} \quad \text{existe } u \in \{0,1\}^{p(|x|)} \text{ tal que } M(\langle x, u\rangle) = 1$$

Definimos una máquina no-determinística N que con entrada x hace esto:

> *"inventa" una palabra $u \in \{0,1\}^{p(|x|)}$ en su cinta de trabajo; lo logra con el no-determinismo de δ, que escribe 0 o 1. Esto le toma tiempo $p(|x|)$.*
>
> *simula M con entrada $\langle x, u\rangle$. Esto le toma tiempo polinomial.*
>
> *si $M(\langle x, u\rangle) = 1$ entonces entra al estado $q_{sí}$; si no entra al estado q_{no}*

Tenemos

$$x \in \mathcal{L} \quad \text{sii} \quad \begin{array}{l}\text{existe un cómputo aceptador de } N \\ \text{a partir de } x.\end{array}$$

N corre en tiempo polinomial, entonces $\mathcal{L} \in \bigcup_{c\in\mathbb{N}} \mathbf{NDTime}(n^c)$. □

Máquinas no-deterministas $\longleftrightarrow$ palabras en binario

- análogamente a lo que pasa para las máquinas deterministas, hay una codificación de máquinas no-deterministas con palabras en $\{0,1\}^*$
- toda palabra $x \in \{0,1\}^*$ representa alguna máquina no-determinística
- identificamos máquinas no-deterministas con palabras $x \in \{0,1\}^*$; hablamos de 'la máquina no-determinística x' o 'la x-ésima máquina no-determinística' para referirnos a la única máquina no-determinística N tal que $\langle N \rangle = x$
- hay una cantidad numerable de máquinas no-deterministas

La máquina no-determinística universal

Llamemos N_i a la máquina no-determinística tal que $\langle N \rangle = i$.

Teorema

Existe una máquina no-determinística NU que tal que NU acepta $(\langle i, x \rangle)$ sii N_i acepta x y si N_i corre en tiempo $T(n)$ entonces $NU(\langle i, x \rangle)$ decide si N_i acepta o rechaza x en $c \cdot T(|x|)$ pasos, donde c depende solo de i.

Reducibilidad polinomial

Clase 4

- Máquinas no-determinísticas
- Reducibilidad polinomial
- Lógica proposicional

Reducción polinomial

Definición

$\mathcal{L} \subseteq \{0,1\}^*$ es **Karp reducible polinomialmente** o simplemente **reducible polinomialmente** a $\mathcal{L}' \subseteq \{0,1\}^*$, notado $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$, si existe una función computable en tiempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$

$$x \in \mathcal{L} \qquad \text{sii} \qquad f(x) \in \mathcal{L}'.$$

f se llama **reducción polinomial** de $\mathcal{L}$ a $\mathcal{L}'$. En este caso decimos que $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$ **vía** f.

Reducción polinomial

Definición

$\mathcal{L} \subseteq \{0,1\}^*$ es **Karp reducible polinomialmente** o simplemente **reducible polinomialmente** a $\mathcal{L}' \subseteq \{0,1\}^*$, notado $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$, si existe una función computable en tiempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$

$$x \in \mathcal{L} \qquad \text{sii} \qquad f(x) \in \mathcal{L}'.$$

f se llama **reducción polinomial** de $\mathcal{L}$ a $\mathcal{L}'$. En este caso decimos que $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$ **vía** f.

Clase de complejidad: NP-hard, NP-completo

$\mathcal{L}$ es **NP-hard** si $\mathcal{L}' \leq_{\mathrm{p}} \mathcal{L}$ para todo $\mathcal{L}' \in \mathbf{NP}$
$\mathcal{L}$ es **NP-completo** si $\mathcal{L} \in \mathbf{NP}$ y $\mathcal{L}$ es **NP-hard**.

Reducción polinomial

Definición

$\mathcal{L} \subseteq \{0,1\}^*$ es **Karp reducible polinomialmente** o simplemente **reducible polinomialmente** a $\mathcal{L}' \subseteq \{0,1\}^*$, notado $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$, si existe una función computable en tiempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$

$$x \in \mathcal{L} \qquad \text{sii} \qquad f(x) \in \mathcal{L}'.$$

f se llama **reducción polinomial** de $\mathcal{L}$ a $\mathcal{L}'$. En este caso decimos que $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$ **vía** f.

Clase de complejidad: NP-hard, NP-completo

$\mathcal{L}$ es **NP-hard** si $\mathcal{L}' \leq_{\mathrm{p}} \mathcal{L}$ para todo $\mathcal{L}' \in \mathbf{NP}$
$\mathcal{L}$ es **NP-completo** si $\mathcal{L} \in \mathbf{NP}$ y $\mathcal{L}$ es **NP-hard**.

Ejercicio

Si $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$ y $\mathcal{L}' \in \mathbf{P}$, entonces $\mathcal{L} \in \mathbf{P}$.

Teorema

La relación $\leq_{\mathrm{p}}$ es transitiva.

Demostración.

Supongamos que $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}'$ via f y $\mathcal{L}' \leq_{\mathrm{p}} \mathcal{L}''$ via g.

$$x \in \mathcal{L} \qquad \text{sii} \qquad f(x) \in \mathcal{L}' \qquad \text{sii} \qquad g(f(x)) \in \mathcal{L}''$$

Veamos que $g \circ f$ es computable en tiempo polinomial.
Supongamos M_f (det.) que computa f en tiempo $O(n^c)$
Supongamos M_g (det.) que computa g en tiempo $O(n^d)$
Definimos M (det.) así: simula M_f con entrada x; si obtiene $M_f(x) = f(x) = y$, entonces simula $M_g(y)$ y escribe el resultado en la cinta de salida.

- $M(x) = g(f(x))$, o sea M computa $g \circ f$
- $|y| = O(|x|^c)$
- M corre en tiempo $O(n^{cd})$

Entonces $\mathcal{L} \leq_{\mathrm{p}} \mathcal{L}''$ vía $g \circ f$. □

Teorema

Si $\mathbf{NP\text{-}hard} \cap \mathbf{P} \neq \emptyset$, entonces $\mathbf{P} = \mathbf{NP}$.

Demostración.

Sea $\mathcal{L} \in \mathbf{NP\text{-}hard} \cap \mathbf{P}$.
Como $\mathcal{L} \in \mathbf{NP\text{-}hard}$, tenemos $\mathcal{L}' \leq_{\mathrm{p}} \mathcal{L}$ para todo $\mathcal{L}' \in \mathbf{NP}$.
Fijemos $\mathcal{L}' \in \mathbf{NP}$ cualquiera y veamos que $\mathcal{L}' \in \mathbf{P}$.
Existe una función f computable en tiempo polinomial tal que para todo x

$$x \in \mathcal{L}' \qquad \text{sii} \qquad f(x) \in \mathcal{L}.$$

Como $\mathcal{L} \in \mathbf{P}$, $\chi_{\mathcal{L}}$ es computable en tiempo polinomial.
Entonces

$$x \in \mathcal{L}' \qquad \text{sii} \qquad \chi_{\mathcal{L}}(f(x)) = 1$$

y $\chi_{\mathcal{L}} \circ f$ es computable en tiempo polinomial.
Luego $\chi_{\mathcal{L}'}$ es computable en tiempo polinomial, o sea $\mathcal{L}' \in \mathbf{P}$. □

Teorema

Si $\mathcal{L} \in$ **NP-completo**, entonces $\mathcal{L} \in \mathbf{P}$ sii $\mathbf{P} = \mathbf{NP}$.

Demostración.

Supongamos $\mathcal{L} \in$ **NP-completo**

$\Rightarrow$ Supongamos $\mathcal{L} \in \mathbf{P}$.
Por hipótesis tenemos $\mathcal{L} \in$ **NP-completo**.
Luego $\mathcal{L} \in$ **NP-completo** $\cap \mathbf{P}$.
Entonces $\mathcal{L} \in$ **NP-hard** $\cap \mathbf{P}$.
Por teorema anterior, $\mathbf{P} = \mathbf{NP}$.

$\Leftarrow$ Supongamos $\mathbf{P} = \mathbf{NP}$
Por hipótesis $\mathcal{L} \in \mathbf{NP}$, de modo que $\mathcal{L} \in \mathbf{P}$.

□

Un ejemplo de problema **NP-completo**

Llamemos M_y a la máquina determinística tal que $\langle M \rangle = y$.

Problema: TMSAT

$$\mathsf{TMSAT} = \{\langle y, x, 1^n, 1^t \rangle : \exists u \in \{0,1\}^n \ M_y(xu) = 1 \text{ en} \leq t \text{ pasos}\}$$

Proposición

TMSAT $\in$ **NP-completo**.

Un ejemplo de problema **NP-completo**

Llamemos M_y a la máquina determinística tal que $\langle M \rangle = y$.

Problema: TMSAT

$$\mathsf{TMSAT} = \{\langle y, x, 1^n, 1^t \rangle : \exists u \in \{0,1\}^n \ M_y(xu) = 1 \text{ en} \leq t \text{ pasos}\}$$

Proposición

TMSAT $\in$ **NP-completo**.

Demostración.

Es claro que TMSAT $\in \mathbf{NP}$. Tomemos $\mathcal{L} \in \mathbf{NP}$ y veamos que $\mathcal{L} \leq_{\mathrm{p}} \mathsf{TMSAT}$. Existen polinomios p, q y una máquina determinística M tal que $x \in \mathcal{L}$ sii $\exists u \in \{0,1\}^{\leq p(|x|)}$ $M(xu) = 1$ y M corre en tiempo $q(n)$.

Un ejemplo de problema **NP-completo**

Llamemos M_y a la máquina determinística tal que $\langle M \rangle = y$.

Problema: TMSAT

$$\mathsf{TMSAT} = \{\langle y, x, 1^n, 1^t \rangle : \exists u \in \{0,1\}^n \ M_y(xu) = 1 \text{ en} \leq t \text{ pasos}\}$$

Proposición

TMSAT $\in$ **NP-completo**.

Demostración.

Es claro que TMSAT $\in \mathbf{NP}$. Tomemos $\mathcal{L} \in \mathbf{NP}$ y veamos que $\mathcal{L} \leq_{\mathrm{p}} \mathsf{TMSAT}$. Existen polinomios p, q y una máquina determinística M tal que $x \in \mathcal{L}$ sii $\exists u \in \{0,1\}^{\leq p(|x|)}$ $M(xu) = 1$ y M corre en tiempo $q(n)$. Definimos $f(x) = \langle \langle M \rangle, x, 1^{p(|x|)}, 1^{q(|x|+p(|x|))} \rangle$. f es computable en tiempo polinomial.

$$\begin{aligned} f(x) \in \mathsf{TMSAT} \quad &\text{sii} \quad \langle \langle M \rangle, x, 1^{p(|x|)}, 1^{q(|x|+p(|x|))} \rangle \in \mathsf{TMSAT} \\ &\text{sii} \quad \exists u \in \{0,1\}^{p(|x|)} \ M(xu) = 1 \\ &\qquad \text{en} \leq q(|xu|) = q(|x| + p(|x|)) \text{ pasos} \\ &\text{sii} \quad x \in \mathcal{L} \end{aligned}$$

□

Lógica proposicional

Clase 4

- Máquinas no-determinísticas
- Reducibilidad polinomial
- Lógica proposicional

Lógica proposicional

Una **fórmula booleana** se forma con la gramática

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

donde $p \in \mathsf{PROP}$. El **tamaño** de φ es la cantidad de símbolos que contiene.

Una **valuación** es una función $v : \mathsf{PROP} \to \{0, 1\}$. Definimos $v \models \varphi$ por inducción en φ:

- si $p \in \mathsf{PROP}$, $v \models p$ si $v(p) = 1$
- $v \models \neg\varphi$ si no $v \models \varphi$
- $v \models \varphi \wedge \psi$ si $v \models \varphi$ y $v \models \psi$
- $v \models \varphi \vee \psi$ si $v \models \varphi$ o $v \models \psi$

Decimos que

- φ **es verdadera para** v o que v **satisface** φ si $v \models \varphi$.
- φ **es falsa para** v o que v **no satisface** φ si $v \models \varphi$ es falso, notado $v \not\models \varphi$.
- φ es **satisfacible** si existe una valuación v tal que $v \models \varphi$.
- φ es una **tautología** si $v \models \varphi$ para toda valuación v.

Valuaciones

Notación: $\varphi(p_1, \ldots, p_n)$

Notamos $\varphi(p_1, \ldots, p_n)$ para marcar que las variables proposicionales que aparecen en φ están entre $p_1, \ldots, p_n$.

Proposición

Sea $\varphi(p_1, \ldots, p_n)$ una fórmula. Si v y v' son dos valuaciones tales que $v(p_i) = v'(p_i)$ para $i \in \{1, \ldots, n\}$, entonces $v \models \varphi(p_1, \ldots, p_n)$ sii $v' \models \varphi(p_1, \ldots, p_n)$.

Para saber si $v \models \varphi(p_1, \ldots, p_n)$ alcanza con conocer $v \restriction \{p_1, \ldots, p_n\}$. Usamos $v \in \{0,1\}^n$ para referirnos a $\{p_i \mapsto v(i)\}$.

Valuaciones

Notación: $\varphi(p_1, \ldots, p_n)$

Notamos $\varphi(p_1, \ldots, p_n)$ para marcar que las variables proposicionales que aparecen en φ están entre $p_1, \ldots, p_n$.

Proposición

Sea $\varphi(p_1, \ldots, p_n)$ una fórmula. Si v y v' son dos valuaciones tales que $v(p_i) = v'(p_i)$ para $i \in \{1, \ldots, n\}$, entonces $v \models \varphi(p_1, \ldots, p_n)$ sii $v' \models \varphi(p_1, \ldots, p_n)$.

Para saber si $v \models \varphi(p_1, \ldots, p_n)$ alcanza con conocer $v \restriction \{p_1, \ldots, p_n\}$. Usamos $v \in \{0,1\}^n$ para referirnos a $\{p_i \mapsto v(i)\}$.

Ejemplo

La valuación parcial $\{p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 1\}$ se representa como $(0,0,1)$ o como 001.

- $(0,0,1) \models \neg p_1 \wedge \neg p_2 \wedge p_3$ o $001 \models \neg p_1 \wedge \neg p_2 \wedge p_3$
- $(0,0,1) \not\models p_1 \vee p_2$ o $001 \not\models p_1 \vee p_2$

Forma normal conjuntiva

Una fórmula está en **forma normal conjuntiva (CNF)** si es de la forma

$$\underbrace{(a_{11} \vee \cdots \vee a_{1n_1})}_{\text{cláusula}} \wedge \underbrace{(a_{21} \vee \cdots \vee a_{2n_2})}_{\text{cláusula}} \wedge \cdots \wedge \underbrace{(a_{m1} \vee \cdots \vee a_{mn_m})}_{\text{cláusula}}$$

donde a_{ij} es un **literal**, es decir, de la forma p o $\neg p$ para algún $p \in \mathsf{PROP}$.

Una fórmula está en 3CNF si está en CNF y cada cláusula tiene a lo sumo 3 literales.

Representamos una fórmula $\varphi \in$ CNF con una palabra $\langle \varphi \rangle$ en $\{0,1\}^*$

Forma normal conjuntiva

Una fórmula está en **forma normal conjuntiva (CNF)** si es de la forma

$$\underbrace{(a_{11} \vee \cdots \vee a_{1n_1})}_{\text{cláusula}} \wedge \underbrace{(a_{21} \vee \cdots \vee a_{2n_2})}_{\text{cláusula}} \wedge \cdots \wedge \underbrace{(a_{m1} \vee \cdots \vee a_{mn_m})}_{\text{cláusula}}$$

donde a_{ij} es un **literal**, es decir, de la forma p o $\neg p$ para algún $p \in \mathsf{PROP}$.
Una fórmula está en 3CNF si está en CNF y cada cláusula tiene a lo sumo 3 literales.
Representamos una fórmula $\varphi \in$ CNF con una palabra $\langle \varphi \rangle$ en $\{0,1\}^*$

Problema: SAT (satisfacción booleana en CNF)

$$\mathsf{SAT} = \{\langle \varphi \rangle : \varphi \in \text{CNF es satisfacible}\}$$

Problema: 3SAT (satisfacción booleana en 3CNF)

$$\mathsf{3SAT} = \{\langle \varphi \rangle : \varphi \in \text{3CNF es satisfacible}\}$$

SAT y 3SAT son **NP**

Teorema

$\mathsf{SAT}, \mathsf{3SAT} \in \mathbf{NP}$.

SAT y 3SAT son **NP**

Teorema

$\mathsf{SAT}, \mathsf{3SAT} \in \mathbf{NP}$.

Demostración.

Alcanza con verlo para SAT. Definimos una máquina determinística M que recibe como entrada $\langle x, u\rangle$ y hace esto:

> *si x no representa una fórmula, rechazar.*
> *si no, supongamos que $x = \langle\varphi\rangle$ y supongamos que φ tiene m variables*
> *si $|u| < m$ rechazar*
> *si no, aceptar si $u \restriction m \models \varphi$ y rechazar en caso contrario.*

Tenemos

$$\langle\varphi\rangle \in \mathsf{SAT} \qquad \text{sii} \qquad \exists u \in \{0,1\}^{|\langle\varphi\rangle|}\ M(\langle x, u\rangle) = 1$$

Como M corre en tiempo polinomial, $\mathsf{SAT} \in \mathbf{NP}$.
El certificado de M es la valuación que hace verdadera a φ: es una palabra en $\{0,1\}^{p(|\langle\varphi\rangle|)}$, donde $p(n) = n$. □

SAT y 3SAT son **NP**

Ejemplo

Sea

$$\varphi = (\neg p_1 \vee p_2 \vee \neg p_3) \wedge (p_1 \vee p_2 \vee p_3).$$

$u = 001$ es un certificado para φ porque

$$001 \models \varphi$$

Representación de funciones booleanas

Proposición

Para toda $F : \{0,1\}^\ell \to \{0,1\}$ existe una fórmula $\varphi_F(p_1, \ldots, p_\ell)$ en CNF tal que $v \models \varphi_F$ sii $F(v) = 1$ para todo $v \in \{0,1\}^\ell$. Más aún, φ_F se computa en tiempo polinomial a partir de $\langle F \rangle$ y tiene tamaño $O(\ell 2^\ell)$.

Representación de funciones booleanas

Proposición

Para toda $F : \{0,1\}^\ell \to \{0,1\}$ existe una fórmula $\varphi_F(p_1, \ldots, p_\ell)$ en CNF tal que $v \models \varphi_F$ sii $F(v) = 1$ para todo $v \in \{0,1\}^\ell$. Más aún, φ_F se computa en tiempo polinomial a partir de $\langle F \rangle$ y tiene tamaño $O(\ell 2^\ell)$.

Demostración.

$$\varphi = \bigwedge_{v:F(v)=0} \neg \left(\bigwedge_{i:v(i)=1} p_i \wedge \bigwedge_{i:v(i)=0} \neg p_i \right)$$

$$= \underbrace{\bigwedge_{v:F(v)=0} \underbrace{\left(\bigvee_{i:v(i)=1} \neg p_i \vee \bigvee_{i:v(i)=0} p_i \right)}_{\text{tamaño } O(\ell)}}_{\text{tamaño } O(\ell 2^\ell)}$$

Representación de funciones booleanas

Ejemplo

$v(0)$	$v(1)$	$v(2)$	$F(v)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$\begin{aligned}\varphi =&(p_1 \vee p_2 \vee p_3) \wedge \\ &(p_1 \vee p_2 \vee \neg p_3) \wedge \\ &(\neg p_1 \vee p_2 \vee p_3) \wedge \\ &(\neg p_1 \vee p_2 \vee \neg p_3)\end{aligned}$$

Representación de funciones booleanas

Corolario

Para toda $F : \{0,1\}^\ell \to \{0,1\}^k$ existe una fórmula $\varphi_F(p_1, \ldots, p_{\ell+k})$ en CNF tal que $uv \models \varphi_F$ sii $F(u) = v$ para todo $u \in \{0,1\}^\ell$, $v \in \{0,1\}^k$. Más aún, φ_F se computa en tiempo polinomial a partir de $\langle F \rangle$ y tiene tamaño $O((\ell + k)2^{\ell+k})$.

Representación de funciones booleanas

Corolario

Para toda $F : \{0,1\}^\ell \to \{0,1\}^k$ existe una fórmula $\varphi_F(p_1, \ldots, p_{\ell+k})$ en CNF tal que $uv \models \varphi_F$ sii $F(u) = v$ para todo $u \in \{0,1\}^\ell$, $v \in \{0,1\}^k$. Más aún, φ_F se computa en tiempo polinomial a partir de $\langle F \rangle$ y tiene tamaño $O((\ell + k)2^{\ell+k})$.

Demostración.

Considerar $G : \{0,1\}^{\ell+k} \to \{0,1\}$ definida como

$$G(uv) = 1 \qquad \text{sii} \qquad F(u) = v$$

y aplicar el resultado anterior. □