

Teoría de la Computación

NP-completitud, demostrada

El teorema de Cook-Levin, de SAT a 3-SAT, coNP

Gerson Garrido

Semana 13

UNIVERSIDAD NACIONAL DE INGENIERÍA

Hoy

Sesión A

1. Reducciones y NP-completitud
2. El primer problema NP-completo
3. El tableau
4. Los cuatro grupos de cláusulas
5. Ventanas legales, y por qué bastan
6. La reducción, ejecutada

Sesión B

7. De la fórmula del tableau a CNF
8. De CNF a 3-SAT
9. De 3-SAT a camino hamiltoniano
10. 2-SAT está en P
11. coNP
12. Qué pasaría si $\mathrm{P} = \mathrm{NP}$

Lectura

Apuntes, semana 13. Apoyo: Sipser, sec. 7.4 y 7.5; Barak, caps. 14, 15 y 16. Código: `code/week-13`.

SECCIÓN 01

Reducciones y NP-completitud

01



Lo que ya saben de Algoritmos

Reducción en tiempo polinomial ***(polynomial-time reduction)***

$A \leq_p B$ si hay una función f, computable en tiempo polinomial, con $w \in A \iff f(w) \in B$.

- Si $A \leq_p B$ y $B \in \mathrm{P}$, entonces $A \in \mathrm{P}$.
- Al revés: si $A \notin \mathrm{P}$, entonces $B \notin \mathrm{P}$. B **es al menos tan difícil como** A.
- Es transitiva: $A \leq_p B$ y $B \leq_p C$ dan $A \leq_p C$.

NP-difícil y NP-completo

B es **NP-difícil** si todo $A \in \mathrm{NP}$ cumple $A \leq_p B$. Es **NP-completo** si además $B \in \mathrm{NP}$.

Dos consecuencias

1. Si B es NP-completo y $B \in \mathrm{P}$, entonces $\mathrm{P} = \mathrm{NP}$.
   - Todo $A \in \mathrm{NP}$ se reduce a B, y entonces está en P.
2. Si B es NP-completo, $C \in \mathrm{NP}$ y $B \leq_p C$, entonces C es NP-completo.
   - $A \leq_p B \leq_p C$, por transitividad.

La segunda es la receta: se reduce **desde** un problema NP-completo conocido **hacia** el nuevo.

Error común

Demostrar que un problema es NP-completo y olvidar que está en NP. La reducción solo da NP-difícil. El problema de la parada es NP-difícil y no está en NP.

El primer problema no sale de la receta

- La receta necesita un problema NP-completo para empezar.
- Para el primero hay que dar $A \leq_p B$ para **todo** $A \in \text{NP}$: también los que nadie ha nombrado.
- Una sola construcción para todos. Solo puede usar lo que todos tienen en común:

Lo único en común

Para cada $A \in \text{NP}$ existe una máquina de Turing no determinista N que decide A en tiempo n^k.

- El problema B tiene que poder **describir la ejecución de una máquina cualquiera**.
- Las fórmulas booleanas alcanzan.

SAT y el teorema

SAT

$\mathrm{SAT} = \{ \langle \varphi \rangle : \varphi$ es una fórmula booleana satisfacible $\}$

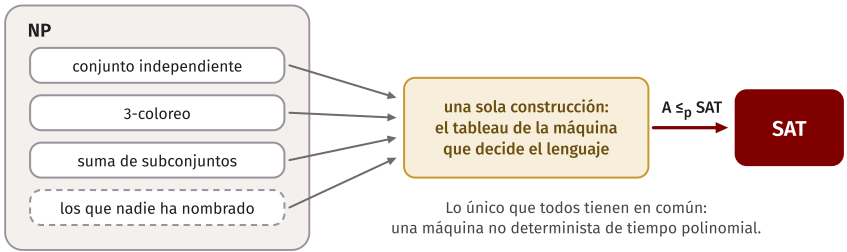
- $(x \vee y) \wedge (\neg x \vee \neg y)$ es satisfacible; $x \wedge \neg x$ no. Está en NP: el certificado es la asignación.

Teorema (Cook-Levin)

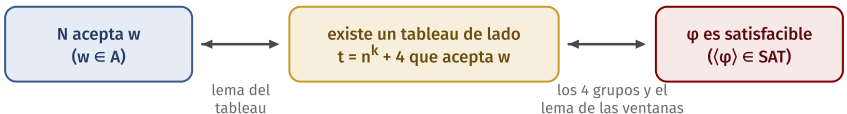
SAT es NP-completo.

Plan. Se fija $A \in$ NP y su máquina N. Dada w: (1) escribir la ejecución de N como una tabla, el **tableau**; (2) dar una fórmula φ, de tamaño polinomial en $|w|$, que dice "la tabla es una ejecución que acepta w".

Una sola construcción para todo NP



Para un lenguaje A de NP, una máquina N que lo decide en tiempo n^k, y una entrada w de longitud n:



SECCIÓN 02

El tableau

02

Una máquina de ejemplo

machines/contains_11.tm

```
states: q0 q1 q_accept q_reject
alphabet: 0 1
tape: 0 1 _
start: q0
accept: q_accept
reject: q_reject
delta:
q0 0 -> q0 0 R
q0 1 -> q0 1 R
q0 1 -> q1 1 R
q1 1 -> q_accept 1 R
```

- Acepta las cadenas que contienen `11`.
- **Dos líneas** para $(q_0, 1)$: sigue de largo, o apuesta a que ahí empieza el `11`.
- Un par sin línea no tiene movimientos: esa rama rechaza.
- Toda rama se detiene en a lo sumo n pasos.

δ	`0`	`1`	$\sqcup$
q_0	$(q_0, 0, \mathrm{R})$	$(q_0, 1, \mathrm{R})$, $(q_1, 1, \mathrm{R})$	—
q_1	—	$(q_{\text{accept}}, 1, \mathrm{R})$	—

El árbol de ejecuciones

terminal

```
$ build/run_ntm machines/contains_11.tm 011
N = machines/contains_11.tm, no determinista; w = 011
paso 0:  q0 011
paso 1:  0 q0 11
paso 2:  01 q0 1  |  01 q1 1
paso 3:  011 q0 _  |  011 q1 _  |  011 q_accept _
acepta: hay una rama de 3 pasos
  q0 011
  0 q0 11
  01 q1 1
  011 q_accept _
```

- Una máquina no determinista **acepta** si alguna rama acepta.
- Con `010`: `rechaza: todas las ramas se detuvieron sin aceptar`.

El tableau

Tableau de lado t

Una tabla de $t \times t$ celdas. Cada celda contiene un elemento de $C = \Gamma \cup Q \cup \{\#\}$.

Cada fila es una configuración $u\,q\,v$, escrita `# u q v #`

- $t - 2$ celdas entre las #: una para el estado y $t - 3$ de cinta.
- El estado va justo antes de la celda que lee: **nunca queda pegado a la # derecha**.

Un tableau **acepta** w si:

1. la fila 1 es la configuración inicial con w;
2. cada fila sale de la anterior por **un paso** de N, o repite una fila que ya está en q_{accept};
3. alguna fila está en q_{accept}.

El tableau de `011`, de lado 7

fila 1	#	q0	0	1	1	–	#
fila 2	#	0	q0	1	1	–	#
fila 3	#	0	1	q1	1	–	#
fila 4	#	0	1	1	q_{acc}	–	#
fila 5	#	0	1	1	q_{acc}	–	#
fila 6	#	0	1	1	q_{acc}	–	#
fila 7	#	0	1	1	q_{acc}	–	#

las 4 configuraciones de la rama

la que acepta, repetida

- La rama acepta en 3 pasos, con el cabezal en la cuarta celda: hacen falta $t - 3 \geq 4$.
- Todas las decisiones de la máquina están escritas en la tabla. El no determinismo desapareció.

¿De qué lado?

- N da a lo sumo n^k pasos: a lo sumo $n^k + 1$ configuraciones.
- En cada paso el cabezal avanza una celda cuando más: nunca pasa de la celda $n^k + 1$.
- $n^k + 1$ celdas de cinta, la del estado y dos #:

$$t = n^k + 4$$

Lema del tableau

N acepta w si y solo si existe un tableau de lado $t = n^k + 4$ que acepta w.

- $\Rightarrow$: la rama que acepta cabe en la tabla; se rellena repitiendo la última fila. **Usa la cota de tiempo.**
- $\Leftarrow$: las filas hasta la primera que acepta son una rama de N.

SECCIÓN 03

La fórmula

03

Las variables y los cuatro grupos

Una variable por celda y por símbolo

$x_{i,j,s}$: "la celda de la fila i y la columna j contiene s". Son $t^2\,|C|$ variables.

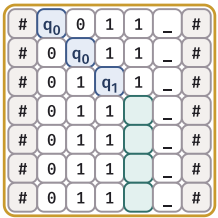
$$\varphi = \varphi_{\text{celda}} \wedge \varphi_{\text{inicio}} \wedge \varphi_{\text{paso}} \wedge \varphi_{\text{acepta}}$$

Grupo	Qué dice
φ_{celda}	Cada celda contiene exactamente un símbolo
φ_{inicio}	La fila 1 es la configuración inicial con w
φ_{paso}	Cada ventana de 2×3 es legal
φ_{acepta}	Alguna celda contiene q_{accept}

En DIMACS, $x_{i,j,s}$ es el entero $((i-1)\,t + (j-1))\,|C| + s + 1$. Con el ejemplo, $|C| = 8$ y $7^2 \cdot 8 = 392$ variables.

Los cuatro grupos, sobre el tableau

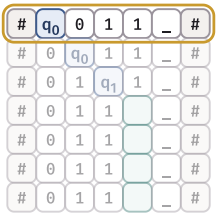
Grupo 1: φ_{celda}



Cada celda contiene exactamente un símbolo.

las $t^2 = 49$ celdas, una por una

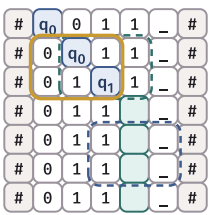
Grupo 2: φ_{inicio}



La fila 1 es la configuración inicial.

t = 7 cláusulas de un literal; solo aquí aparece w

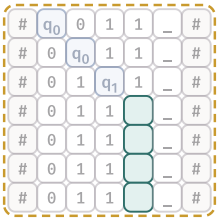
Grupo 3: φ_{paso}



Cada ventana de 2 × 3 es legal.

(t − 1)(t − 2) = 30 ventanas, que se superponen

Grupo 4: φ_{acepta}



Alguna celda contiene q_{accept}.

una cláusula de $t^2 = 49$ literales

En verde, las celdas con q_{accept}. Solo el grupo 3 relaciona una fila con la siguiente; solo el grupo 2 depende de w.

Grupos 1, 2 y 4

Grupo 1. Al menos un símbolo, y no dos a la vez:

$$\varphi_{\text{celda}} = \bigwedge_{i,j}\left[\left(\bigvee_{s\in C} x_{i,j,s}\right) \wedge \bigwedge_{s\neq s'} \left(\neg x_{i,j,s} \vee \neg x_{i,j,s'}\right)\right]$$

Grupo 2. La fila 1, celda por celda. El único lugar donde aparece w.

$$\varphi_{\text{inicio}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \cdots \wedge x_{1,t-1,\sqcup} \wedge x_{1,t,\#}$$

Grupo 4. Una sola cláusula:

$$\varphi_{\text{acepta}} = \bigvee_{i,j} x_{i,j,q_{\text{accept}}}$$

Con estos tres, la fórmula todavía no dice nada de la máquina.

Un paso solo cambia tres columnas

Sea p la columna del estado. Con el movimiento $(r, b, D) \in \delta(q, a)$:

x	q	a
x	b	r

movimiento R

x	q	a
r	x	b

movimiento L

#	q	a
#	r	b

L en la primera celda

- Columnas $p-1$, p, $p+1$. Todas las demás se copian.
- Es la localidad de las historias de cómputo de la semana 10.
- Entonces no hace falta comparar filas enteras: basta mirar **trozos de tamaño fijo**.

Ventanas legales

Ventana

Un bloque de 2×3 celdas: las columnas j, $j+1$, $j+2$ de las filas i e $i+1$. Hay $(t-1)(t-2)$.

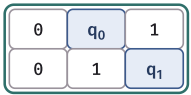
Ventana legal

La que aparece en algún paso de N entre dos filas que son configuraciones (o al repetir una que acepta).

- Que una ventana sea legal depende solo de δ: ni de w ni de t. Es una lista finita, una por máquina.

Legales e ilegales

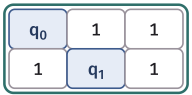
Legales: aparecen en algún paso de la máquina



estado en el centro

filas 2 y 3, columnas 2 a 4

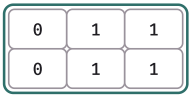
ve el paso completo:
el movimiento (q_1, 1, R)



estado a la izquierda

filas 2 y 3, columnas 3 a 5

ve el estado y lo que lee,
no lo que tiene a su izquierda

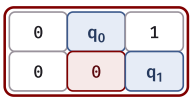


sin estado

filas 4 y 5, columnas 2 a 4

lejos del cabezal:
las dos filas son iguales

Ilegales: no aparecen en ningún paso



escribe un 0, y $\delta(q_0, 1)$
solo permite escribir 1

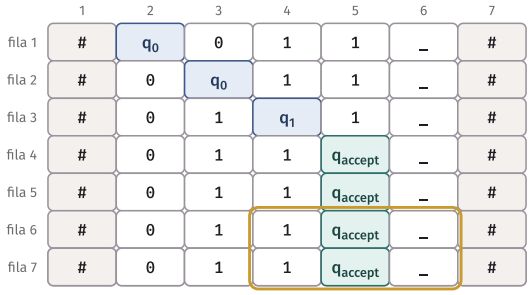


cambia una celda que no
está junto a ningún estado



q_1 queda pegado a la #:
abajo no hay configuración

El tableau, fila por fila



Fila 6 → fila 7: la fila 6 está en q_{accept}, que no tiene transiciones.

La tabla quedó llena y tiene q_{accept}: es un tableau que acepta `011`.

Las cuatro formas

Forma	Arriba	Abajo	Cuándo
1. Estado en el centro	$x\ q\ a$	$x\ b\ r$	$(r, b, \mathrm{R}) \in \delta(q, a)$
		$r\ x\ b$ o $\#\ r\ b$	$(r, b, \mathrm{L}) \in \delta(q, a)$; según $x \in \Gamma$ o $x = \#$
2. Estado a la izquierda	$q\ a\ y$	$b\ r\ y$	$(r, b, \mathrm{R}) \in \delta(q, a)$ y $y \neq \#$
		$x\ b\ y$ o $r\ b\ y$	$(r, b, \mathrm{L}) \in \delta(q, a)$; cualquier $x \in \Gamma$
3. Estado a la derecha	$z\ x\ q$	$z\ x\ b$	$(r, b, \mathrm{R}) \in \delta(q, a)$, algún a
		$z\ r\ x$	$(r, b, \mathrm{L}) \in \delta(q, a)$, algún a
4. Sin estado	$y_1\ y_2\ y_3$	$y_1\ y_2\ y_3$	siempre
		$r\ y_2\ y_3$ o $b\ y_2\ y_3$	el estado está a la izquierda y lee y_1
		$y_1\ y_2\ r$	el estado entra desde la derecha con L
Aceptación	con q_{accept}	igual	siempre

- La forma 1 ve todo el paso. Las otras, una parte: no ven x, o no ven a.
- En la forma 4, **el centro siempre se copia**.

Cuántas son

terminal

```
$ build/windows machines/contains_11.tm
|Γ| = 3, |Q| = 4, |C| = 8
ventanas posibles: 8^6 = 262144
ventanas legales:  193
  con el estado en el centro:    28
  con el estado a la izquierda:  24
  con el estado a la derecha:    48
  sin estado arriba:             93
```

- La lista depende solo de δ. No de w ni de t: es una **constante** de la máquina.
- `test_tableau` compara la tabla de las cuatro formas con las ventanas que aparecen en pasos de verdad.

Grupo 3: cada ventana es legal

Sea $\mathcal{L}$ el conjunto de las ventanas legales.

$$\varphi_{\text{paso}} = \bigwedge_{\substack{1 \le i \le t-1 \\ 1 \le j \le t-2}} \bigvee_{(a_1 a_2 a_3 / b_1 b_2 b_3) \in \mathcal{L}} \left(\begin{array}{l} X_{i,j,a_1} \wedge X_{i,j+1,a_2} \wedge X_{i,j+2,a_3} \wedge \\ X_{i+1,j,b_1} \wedge X_{i+1,j+1,b_2} \wedge X_{i+1,j+2,b_3} \end{array} \right)$$

- Para cada posición: “en estas seis celdas está alguna ventana legal”.
- No está en CNF. Para el teorema no hace falta; en la sesión B se arregla.
- **Falta lo importante:** ¿por qué ventanas legales, revisadas una por una, dan un paso de verdad? Con una máquina no determinista, cada ventana podría seguir un movimiento distinto.

Cuatro propiedades de las ventanas legales

P1 **Sin estado arriba, el centro se copia:** $b_2 = a_2$.
Un paso solo cambia las columnas $p-1$, p, $p+1$.

P2 **Con el estado en el centro, la ventana ve el paso completo:** $b_1 b_2 b_3$ es el resultado de **un** movimiento de $\delta(q, a_3)$, o la copia si $q = q_{\text{accept}}$.

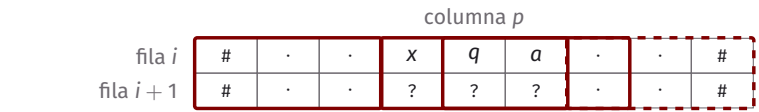
P3 **Las # no se mueven:** debajo de una # hay una #, y solo ahí.

P4 **Ningún estado queda pegado a la # derecha.**

Lema de las ventanas

Si la fila i es una configuración $\#\, C\, \#$ y las $t-2$ ventanas entre las filas i e $i+1$ son legales, la fila $i+1$ es una configuración $\#\, C'\, \#$, y C' sale de C por un paso de N (o C acepta y $C' = C$).

Demostración del lema

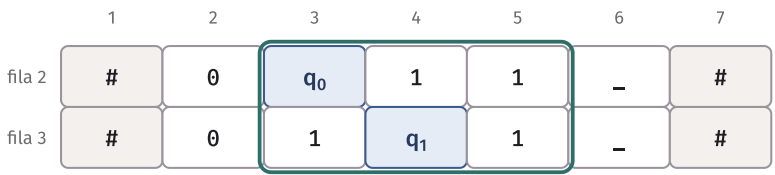


1. **Las tres columnas del cabezal.** La ventana $p-1$, p, $p+1$ tiene el estado en el centro. Por P2, fija las tres celdas de abajo con **un solo** movimiento. Aquí se elige, una vez.
2. **Las demás.** La ventana centrada en otra columna no tiene estado arriba: por P1, se copia. Las dos #, por P3.
3. **El estado tiene qué leer.** Si se movió a la derecha, la ventana p, $p+1$, $p+2$ lo tiene abajo en el centro: por P4, a su derecha no está la #.

Las ventanas vecinas admiten varios movimientos, pero comparten celdas con la del centro. **La superposición las obliga a ponerse de acuerdo.**

El lema, en las filas 2 y 3 del ejemplo

paso 5 de 5



(P4) En la ventana 3–5 el estado q_1 queda abajo en el centro, y a su derecha no puede haber una #.

La fila 3 es una configuración: la que deja el movimiento $(q_1, 1, R)$ a partir de la fila 2.

El tamaño de la fórmula

$t = n^k + 4$. Para una máquina fija, $|C|$ y $L = |\mathcal{L}| \leq |C|^6$ son **constantes**.

Parte	**Cuánto mide**
Variables	$t^2\,\lvert C\rvert$
φ_{celda}	t^2 celdas, con $\lvert C\rvert + \lvert C\rvert(\lvert C\rvert - 1)$ literales cada una
φ_{inicio}	t literales
φ_{paso}	$(t-1)(t-2)$ posiciones, con $6L$ literales cada una
φ_{acepta}	t^2 literales

- En total $O(t^2) = O(n^{2k})$ variables y literales: $O(n^{2k} \log n)$ símbolos.
- Se escribe con cuatro ciclos sobre las celdas. **La reducción no ejecuta la máquina.**
- Pregunta de repaso 3: $(n^k + 4)^2\,(|\Gamma| + |Q| + 1)$ variables.

Las dos direcciones

Afirmación. φ es satisfacible si y solo si existe un tableau de lado t que acepta w.

- **Si existe el tableau:** $x_{i,j,s}$ es verdadera cuando la celda (i, j) contiene s.
  - Un símbolo por celda; la fila 1 es la inicial; alguna fila acepta.
  - Cada ventana aparece en un paso: es legal **por definición**.
- **Si φ es satisfacible:**
  - φ_{celda}: la asignación describe una tabla.
  - φ_{inicio}: la fila 1 es la configuración inicial.
  - φ_{paso} y el lema de las ventanas, fila por fila: cada fila sale de la anterior.
  - φ_{acepta}: alguna fila está en q_{accept}.

Cook-Levin

$w \in A \iff N$ acepta $w \iff \varphi$ es satisfacible. Para todo $A \in$ NP: $A \leq_p \mathrm{SAT}$.

La reducción, ejecutada

terminal

```
$ build/tableau_cnf machines/contains_11.tm 011 7 > build/f.cnf
$ minisat -verb=0 build/f.cnf build/f.res
SATISFIABLE
$ build/decode_tableau machines/contains_11.tm 011 7 build/f.res
#       q0      0       1       1       _       #
#       0       q0      1       1       _       #
#       0       1       q1      1       _       #
#       0       1       1       q_accept _      #
#       0       1       1       q_accept _      #
#       0       1       1       q_accept _      #
#       0       1       1       q_accept _      #
es una ejecución de la máquina: acepta en 3 pasos
```

- En la fila 2 había dos movimientos. **Nadie programó cuál elegir**: lo encontró `minisat`.

Cuánto mide

terminal

```
$ build/tableau_cnf machines/contains_11.tm 011 7 --cuenta
t = 7, |C| = 8, ventanas legales = 193
variables de celdas:    392
variables de ventanas:  5790
variables en total:     6182
cláusulas del grupo 1:  1421
cláusulas del grupo 2:  7
cláusulas del grupo 3:  34770
cláusulas del grupo 4:  1
cláusulas en total:     36199
```

t	Variables	Cláusulas
5	2516	14 639
10	14 696	86 359
20	69 206	407 999
40	298 826	1 764 079

- Al duplicar t, las cláusulas se multiplican por algo cercano a 4: es cuadrática.
- La constante es grande: la reducción existe para demostrar el teorema.
- Con $t = 6$ y `011`: `UNSATISFIABLE`. La ejecución no cabe.

La fórmula no simula

Error común

Creer que la fórmula “simula” a la máquina.

- La fórmula no calcula nada: **describe cómo se vería** una ejecución que acepta.
- Encontrar esa ejecución es trabajo de quien resuelve SAT. Ahí queda toda la dificultad.
- La reducción tampoco simula: mira δ y $|w|$, y no da ni un paso de la máquina.

En la práctica

Verificación acotada de modelos *(bounded model checking)*: una copia de las variables de un circuito por cada paso, y una cláusula que dice “alguna copia es el error”.

SECCIÓN 04

De SAT a 3-SAT

04



Dos escalones

- SAT es incómodo para diseñar reducciones: paréntesis anidados, operaciones mezcladas.
- En Algoritmos todo partía de 3-SAT. Hay que bajar hasta ahí.



1. Escribir la fórmula del tableau en CNF **sin que crezca exponencialmente**.
2. Partir las cláusulas largas en cláusulas de tres literales.

En los dos aparece la misma idea: **variables nuevas**, y conservar que la fórmula sea satisfacible, no que sea equivalente.

Qué forma tiene cada grupo

Grupo	Forma	¿CNF?
φ_{celda}	cláusulas de $\lvert C\rvert$ literales y de 2 literales	sí
φ_{inicio}	t cláusulas de 1 literal	sí
φ_{acepta}	1 cláusula de t^2 literales	sí
φ_{paso}	conjunción de disyunciones de conjunciones	no

Para cada posición de ventana, φ_{paso} tiene

$$W_1 \vee W_2 \vee \cdots \vee W_L \qquad \text{con cada } W_k \text{ una conjunción de 6 variables.}$$

Tres maneras de pasarla a CNF. Las tres sirven para el teorema; una sirve para programar.

Tres maneras

1. **Distribuir.** Una cláusula por cada manera de elegir un literal de cada término: 6^L cláusulas.
   - L no depende de n: es una constante, y el teorema vale. Con el ejemplo, 6^{193} tiene 151 dígitos.
2. **Prohibir las ilegales.** Con un símbolo por celda, "alguna legal" es "ninguna ilegal".
   - Una cláusula de 6 literales negados por ventana ilegal: $|C|^6 - L = 261\,951$ por posición.
3. **Variables nuevas.** z_k: "en esta posición está la ventana k". Cláusulas: $(z_1 \vee \cdots \vee z_L)$ y, para cada una de las 6 variables x de W_k, $(\neg z_k \vee x)$.
   - $6L + 1 = 1159$ cláusulas por posición. Es lo que hace `tableau_cnf`.

No es equivalente: es equisatisfacible

Afirmación. Para cada asignación de las x: se cumple $W_1 \vee \cdots \vee W_L$ si y solo si existen valores de las z que cumplen las cláusulas nuevas.

- $\Rightarrow$: algún W_k es verdadero. Se pone z_k verdadera y las demás falsas.
- $\Leftarrow$: alguna z_k es verdadera, y sus seis cláusulas obligan a W_k.

Idea clave

Una reducción no tiene que dar una fórmula **equivalente**. Tiene que conservar la respuesta: satisfacible va a satisfacible. Dar nombre a partes de la fórmula con variables nuevas evita el crecimiento exponencial.

Con una variable por subfórmula sirve para cualquier fórmula (transformación de Tseitin): $\mathrm{SAT} \leq_p \mathrm{SAT}$ en CNF.

De CNF a 3-SAT: partir las cláusulas

3-SAT

CNF con exactamente tres literales por cláusula, de tres variables distintas.

k	$(\ell_1 \vee \cdots \vee \ell_k)$ **se reemplaza por**
1	$(\ell_1 \vee y_1 \vee y_2) \wedge (\ell_1 \vee y_1 \vee \neg y_2) \wedge (\ell_1 \vee \neg y_1 \vee y_2) \wedge (\ell_1 \vee \neg y_1 \vee \neg y_2)$
2	$(\ell_1 \vee \ell_2 \vee y_1) \wedge (\ell_1 \vee \ell_2 \vee \neg y_1)$
3	la misma cláusula
≥ 4	la cadena, con $k-3$ variables nuevas y $k-2$ cláusulas

$$(\ell_1 \vee \ell_2 \vee y_1) \wedge (\neg y_1 \vee \ell_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-4} \vee \ell_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)$$

y_m dice: “el literal verdadero está más adelante”. La última cláusula ya no puede pasar el encargo.

Por qué conserva la respuesta

terminal

```
$ build/three_sat --ver < formulas/five.cnf
entrada, con 5 variables y 1 cláusulas:
  (x1 ∨ x2 ∨ x3 ∨ x4 ∨ x5)
salida, con 7 variables y 3 cláusulas:
  (x1 ∨ x2 ∨ x6)
  (¬x6 ∨ x3 ∨ x7)
  (¬x7 ∨ x4 ∨ x5)
```

- **Si la cláusula se cumple**, con ℓ_m el primer literal verdadero: las y anteriores a ℓ_m verdaderas, las demás falsas. Cada eslabón queda cumplido.
- **Si no se cumple:** la primera obliga a y_1, la segunda a y_2, … y la última queda con tres literales falsos.
- No es equivalente. Con solo x_5 verdadera, la salida se cumple si x_6 y x_7 son verdaderas, y no si x_6 es falsa.

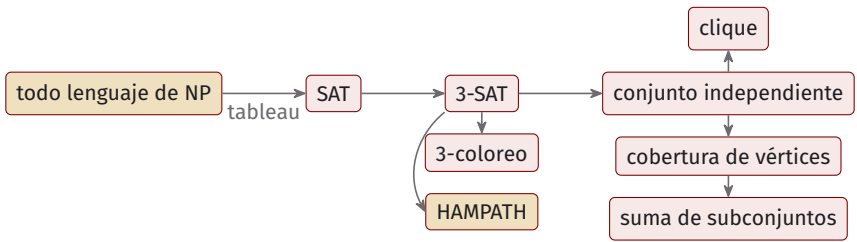
La fórmula del tableau, en 3-SAT

terminal

```
$ build/three_sat < build/f.cnf > build/f3.cnf
entrada: 6182 variables, 36199 cláusulas, la más larga de 193 literales
salida:  48299 variables, 78323 cláusulas, todas de 3 literales
$ minisat -verb=0 build/f3.cnf build/f3.res
SATISFIABLE
```

- Las variables de las celdas conservan sus números: `decode_tableau` imprime el mismo tableau.
- De una máquina no determinista y una entrada sale, en tiempo polinomial, una fórmula de 3-SAT que es satisfacible exactamente cuando la máquina acepta. **3-SAT es NP-completo.**

La red de problemas NP-completos



- Las flechas de la derecha son las de Algoritmos. Ahora el dibujo empieza en **todo NP**.
- Todos están en NP: se reducen entre sí. **Basta que uno esté en P para que todos lo estén.**

SECCIÓN 05

De 3-SAT a camino hamiltoniano

05

Camino hamiltoniano

HAMPATH

$\text{HAMPATH} = \{ \langle G, s, t \rangle :$
G es un grafo dirigido con un camino hamiltoniano de s a $t \}$

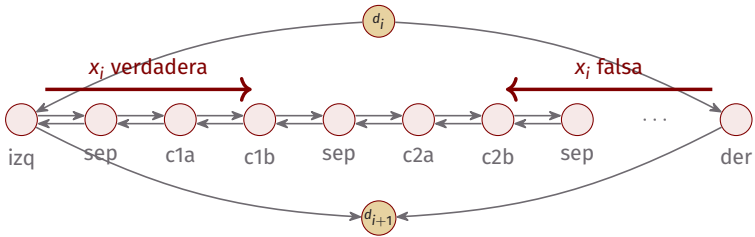
- Hamiltoniano: pasa por cada nodo exactamente una vez.
- Está en NP: el certificado es la lista de nodos en orden. (Semana 12.)

Teorema

3-SAT $\leq_p$ HAMPATH. Entonces HAMPATH es NP-completo.

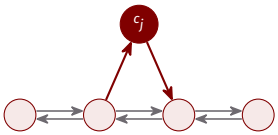
- En Algoritmos los gadgets **prohibían** combinaciones con aristas.
- Aquí obligan a un camino a **recorrer** el grafo de cierta manera.

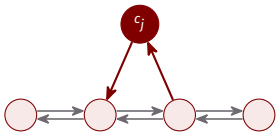
Un rombo por variable



- Una fila con un **par de nodos por cláusula** y separadores entre los pares: $3k + 1$ nodos.
- Para pasar por toda la fila hay dos opciones: de izquierda a derecha, o de derecha a izquierda.
- **Esa elección es el valor de x_i.** Los rombos van en cadena: $s = d_1$ y $t = d_{l+1}$.

Un nodo por cláusula, con desvíos





- Si x_i está en c_j: $\texttt{cja} \to c_j \to \texttt{cjb}$. Solo se puede tomar de izquierda a derecha.
- Si $\neg x_i$ está en c_j: $\texttt{cjb} \to c_j \to \texttt{cja}$. Solo de derecha a izquierda.
- $(l+1) + l\,(3k+3) + k$ nodos: polinomial.

El grafo de una fórmula

terminal

```
$ build/hampath < formulas/two_clauses.cnf
fórmula: 3 variables, 2 cláusulas
  (x1 ∨ x2 ∨ ¬x3)
  (¬x1 ∨ x2 ∨ x3)
grafo: 33 nodos, 72 aristas
camino hamiltoniano de s a t:
  s x1.izq x1.sep0 x1.c1a x1.c1b x1.sep1 x1.c2a x1.c2b x1.sep2 x1.der d2 x2.izq x2.sep0 x2.c1a
  x2.c1b x2.sep1 x2.c2a c2 x2.c2b x2.sep2 x2.der d3 x3.der x3.sep2 x3.c2b x3.c2a x3.sep1 x3.c1b c1
  x3.c1a x3.sep0 x3.izq t
asignación leída del camino: x1 = V x2 = V x3 = F
cumple la fórmula
```

- `c2` se visita desde el rombo de x_2, hacia la derecha; `c1`, desde el de x_3, hacia la izquierda.
- Con las 8 cláusulas sobre 3 variables: 93 nodos, y `no hay camino hamiltoniano de s a t`.

Una dirección, completa

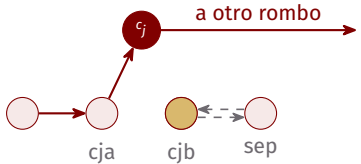
Afirmación. Si φ es satisfacible, G tiene un camino hamiltoniano de s a t.

1. **Sin desvíos.** Para cada i: si x_i es verdadera, de d_i a `izq`, toda la fila hacia la derecha, y a d_{i+1}. Si es falsa, al revés. Pasa por todos los nodos de los rombos.
2. **Los desvíos.** Para cada cláusula c_j se elige **un** literal verdadero.
   - Si es x_i: la fila va hacia la derecha y pasa de `cja` a `cjb`. Se cambia por `cja` $\to c_j \to$ `cjb`.
   - Si es $\neg x_i$: la fila va hacia la izquierda. Se cambia `cjb` $\to$ `cja` por `cjb` $\to c_j \to$ `cja`.
3. **Es hamiltoniano.** Cada c_j entra una sola vez, y cláusulas distintas usan pares distintos. □

La otra dirección, en una idea

Afirmación. Si G tiene un camino hamiltoniano de s a t, φ es satisfacible.

- Lo único que puede romper el plan: entrar a c_j desde un rombo y **salir hacia otro**.
- Entonces `cjb` queda sin visitar. Solo se puede llegar a él desde el separador…
- … y solo se puede salir de él hacia ese mismo separador. El camino muere ahí.



- El camino es entonces de la forma normal, y de él se lee la asignación: x_i es verdadera si su fila va de izquierda a derecha.

SECCIÓN 06

El borde: 2-SAT y coNP

06



2-SAT está en P

- ¿Partir hasta dos literales? $(\ell_1 \vee \ell_2 \vee y) \wedge (\neg y \vee \ell_3)$: la primera sigue teniendo tres.
- $(a \vee b)$ son dos implicaciones: $\neg a \to b$ y $\neg b \to a$. El **grafo de implicaciones**.

Criterio

Una fórmula de 2-SAT es satisfacible si y solo si ninguna variable tiene a x y a $\neg x$ en la misma componente fuertemente conexa.

- Una asignación que cumple la fórmula no tiene caminos de verdadero a falso.
- Al revés: en orden topológico de las componentes, es verdadero el literal que queda más tarde.
- Tiempo lineal. **El borde pasa entre 2 y 3:** con dos literales, fijar un valor *obliga* a otro; con tres, solo deja una alternativa entre dos.

coNP

$\text{UNSAT} = \{ \langle\varphi\rangle : \varphi \text{ no es satisfacible} \}$ $\text{TAUTOLOGY} = \{ \langle\varphi\rangle : \text{toda asignación cumple } \varphi \}$

- Para convencer de que una fórmula **es** satisfacible: una asignación.
- ¿Y de que **no** lo es? Nadie conoce un certificado corto.

coNP

$\text{coNP} = \{ A : \overline{A} \in \text{NP} \}$: las respuestas **no** tienen certificado corto.

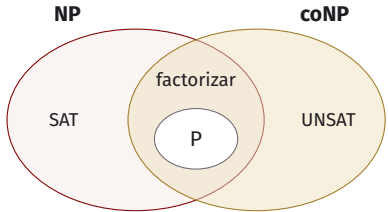
- No es el complemento de NP: $\text{P} \subseteq \text{NP} \cap \text{coNP}$.
- En $\text{NP} \cap \text{coNP}$: "¿tiene n un divisor d con $1 < d \leq m$?". Para el sí, el divisor. Para el no, la factorización en primos, todos mayores que m.

¿NP = coNP?

Teorema

Si algún lenguaje NP-completo está en coNP, entonces NP = coNP.

- La misma f que reduce A a B reduce $\overline{A}$ a $\overline{B}$.
- Pregunta de repaso 5: si 3-SAT estuviera en coNP, toda fórmula insatisfacible tendría una demostración corta.
- Se cree que NP $\neq$ coNP. Y eso implica P $\neq$ NP.
- Con más alternancias de $\exists$ y $\forall$: la jerarquía polinomial. Sin tope: PSPACE, la próxima semana.



Lo que se cree. Solo se sabe que P está en la intersección.

Qué pasaría si $\mathrm{P} = \mathrm{NP}$

Si $\mathrm{P} = \mathrm{NP}$

- Algoritmo polinomial para todo NP, y también para **encontrar** la solución.
- $\mathrm{NP} = \mathrm{coNP}$.
- Encontrar demostraciones: tan fácil como revisarlas.
- La criptografía de clave pública de hoy caería.

Lo que no diría

- Que el algoritmo sea práctico: n^{100} es polinomial.
- Nada sobre lo indecidible.

Si se demuestra $\mathrm{P} \neq \mathrm{NP}$

- Ningún problema NP-completo está en P. Es lo que ya se supone.
- **No** basta para la criptografía: habla del peor caso, no del caso típico.
- Los solucionadores SAT siguen sirviendo.
- Hay problemas intermedios (Ladner).

Abierta desde 1971. En la semana 15: por qué la diagonalización no alcanza.

Los solucionadores SAT en la práctica

tableau de lado 20

```
conflicts             : 1
decisions             : 41
propagations          : 71470
SATISFIABLE
```

palomar con 9 casillas

```
conflicts             : 551941
decisions             : 669045
UNSATISFIABLE
```

- NP-completo habla del **peor caso**, no de cada fórmula.
- Tableau: 69 206 variables y 407 999 cláusulas. Casi todo es propagación.
- Palomar: 10 palomas en 9 casillas. 90 variables y 415 cláusulas.
- El tamaño dice poco de la dificultad.

Corridas del `minisat` del entorno del curso, sin las velocidades que imprime al lado de cada cuenta.

SECCIÓN 07

Laboratorio

07

Laboratorio: Cook-Levin ejecutado

1. Leer una máquina no determinista, una entrada w y el lado t.
2. Escribir en DIMACS la fórmula del tableau. El esqueleto trae la numeración y el grupo 1; ustedes, los otros tres.
3. Resolverla con `minisat` y decodificar el tableau.
4. Compararlo con su simulador de la semana 7.
5. Contar variables y cláusulas para $t = 5$, 10 y 20.

Dónde: `code/week-13/lab/cook_levin`. La fórmula no es única: un verificador revisa qué tableaux admite y cuáles rechaza.

el final de casos/01.in

```
q0 0 -> q0 0 R
q0 1 -> q0 1 R
q0 1 -> q1 1 R
q1 1 -> q_accept 1 R
strings: 1
011
tableau: 7
```

la salida empieza así

```
p cnf 6182 36199
1 2 3 4 5 6 7 8 0
-1 -2 0
```

Para la próxima clase

- Leer los apuntes de la semana 13 y terminar el laboratorio.
- Preguntas de repaso de la guía: una ventana legal y una ilegal para una transición; cuántas variables tiene la fórmula; partir una cláusula de cinco literales; qué se sigue si 3-SAT está en coNP.
- Próxima semana: el espacio. El teorema de Savitch, PSPACE y las fórmulas booleanas cuantificadas.

Pregunta

El tableau de una máquina de tiempo n^k tiene n^{2k} celdas. ¿Cuánta memoria hace falta para *recorrer* una ejecución, si no hay que guardarla entera?