Modelos de Computación y Complejidad

Grado en Ingeniería Informática. Tecnologías Informáticas

Tema 6: El problema P versus NP

Mario de J. Pérez Jiménez
David Orellana Martín

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática
Universidad de Sevilla

marper@us.es

http://www.cs.us.es/~marper/

Curso 2021-2022

Objetivos

* **Tratabilidad** e intratabilidad.
* Las clases de complejidad **P** y **NP**.
* El **problema P versus NP**.
* Problemas **NP-completos**.

Problemas tratables

Son los **más simples**, desde el punto de vista de la complejidad computacional.

- ⋆ **Resolubilidad algorítmica en términos prácticos**: **tratabilidad**.

Polinomial = **Razonable** **Exponencial** = **No razonable**

	10	50	100	300	1000
5n	50	250	500	1.500	5.000
n^2	100	2.500	10.000	90.000	10^6
n^3	1.000	125.000	10^6	$27 \cdot 10^6$	10^9
2^n	1.024	16 dígitos	31 dígitos	91 dígitos	302 dígitos
n!	7 dígitos	65 dígitos	161 dígitos	623 dígitos	inimaginable
n^n	10 dígitos	85 dígitos	201 dígitos	744 dígitos	inimaginable

Nota 1: El número de microsegundos desde el BIG BANG tiene 24 dígitos.

Nota 2: El número de protones en el universo consta de 71 dígitos.

La clase de complejidad P

Sea f una función total computable 1-aria:

Definición: TIME(f) es el conjunto de todos los **problemas de decisión resolubles** por MTDs que trabajan en tiempo acotado por f.

El conjunto de todos los **monomios** $\{n^k \mid k \in \mathbb{N}\}$ permite generar a la **familia de los polinomios** sobre $\mathbb{N}$.

La clase de complejidad P:

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

Definición: **P** es el conjunto de todos los problemas de decisión que son **resolubles por MTDs** que trabajan en **tiempo polinomial**.

P: clase de complejidad de los **problemas tratables** (en modo determinista).

El problema REACHABILITY

REACHABILITY es el siguiente problema de decisión:

* Dado un grafo $G = (V, E)$ (dirigido o no) y dos nodos distinguidos $u, w \in V$, determinar si existe un camino en G que va desde el nodo u hasta el nodo w.

Proposición 1: **REACHABILITY** $\in$ **P**.

Demostración: En primer lugar, recuérdese que un **recorrido en profundidad** de un grafo (al igual que un **recorrido en anchura**), con origen un nodo a, es un algoritmo que implementa la "visita" de cada nodo (una y sólo una vez).

La ejecución de ese algoritmo es descrita mediante un **bosque** que verifica la siguiente propiedad:

* Dos nodos del grafo están conectados por un camino **sii** los nodos pertenecen a un mismo árbol del bosque.

El tiempo de ejecución de un algoritmo de **recorrido en profundidad** es, en el caso peor, de orden cuadrático (en el tamaño del grafo).

Seguidamente, diseñamos un algoritmo determinista de coste en tiempo polinomial que resuelve el problema **REACHABILITY**.

Consideremos el siguiente algoritmo determinista $\mathcal{A}$:

`Entrada:` $G = (V, E)$ un grafo y $u, w \in V$.

Sea $a \in V$

Hallar un recorrido en profundidad de G, con origen $a \in V$

`si` u, w pertenecen a un mismo árbol del bosque `entonces`

`devolver` **sí**

`si no`

`devolver` **no**

Obviamente, el algoritmo determinista $\mathcal{A}$ resuelve **REACHABILITY** (debido a la propiedad de los recorridos en profundidad) y, además, es de coste en tiempo polinomial (específicamente, del orden $O(n^2)$).

En consecuencia, **REACHABILITY** $\in$ **P**.

■

La clase P es estable por reducibilidad en tiempo polinomial

Proposición 2: Sean X_1, X_2 problemas de decisión tales que $X_1 \leqslant^p X_2$ y $X_2 \in \mathbf{P}$. Entonces, $X_1 \in \mathbf{P}$.

Demostración:

- ⋆ Puesto que $X_2 \in \mathbf{P}$, existirá un algoritmo determinista $\mathcal{A}$ de coste en tiempo polinomial que resuelve el problema X_2.
- ⋆ Puesto que $X_1 \leqslant^p X_2$, existirá una reducibilidad en tiempo polinomial, g, de X_1 en X_2. Es decir, existe una función total g de E_{X_1} en E_{X_2} que es computable en tiempo polinomial y, además, para cada $u \in E_{X_1}$: $u \in L_{X_1} \iff g(u) \in L_{X_2}$.
- ⋆ Consideremos el algoritmo determinista $\mathcal{B}$ siguiente:

  `Entrada:` $u \in E_{X_1}$

  Calcular $g(u)$

  Ejecutar el algoritmo $\mathcal{A}$ con entrada $g(u)$

  Entonces, $\mathcal{B}$ es un algoritmo determinista de coste en tiempo polinomial que resuelve el problema X_1; es decir, $X_1 \in \mathbf{P}$.

■

Nota: Pueden existir problemas de decisión X_1, X_2 tales que $X_1 \leqslant^p X_2$ y $X_1 \in \mathbf{P}$ pero que, en cambio, $X_2 \notin \mathbf{P}$.

La clase de complejidad NP

Sea f una función total computable 1-aria:

Definición: NTIME(f) es el conjunto de todos los **problemas de decisión resolubles** por MTNDs que trabajan en tiempo acotado por f.

* Obviamente, TIME(f) $\subseteq$ NTIME(f).

La clase de complejidad NP:

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

Definición: **NP** es el conjunto de todos los problemas de decisión que son **resolubles por MTNDs** que trabajan en **tiempo polinomial**.

NP: conjunto de todos los **problemas de decisión resolubles** por **MTNDs** que trabajan en tiempo polinomial.

NP: clase de complejidad de los **problemas tratables** (en modo no determinista).

Se verifica que **P** $\subseteq$ **NP**.

El problema CLIQUE

CLIQUE es el siguiente problema de decisión:

- Dado un grafo no dirigido $G = (V, E)$ y un número natural $k \leq |V|$, determinar si existe en G un clique de tamaño k.

Proposición 3: **CLIQUE** $\in$ **NP**.

Demostración: Vamos a describir una metodología que permite establecer la pertenencia a la clase **NP** de un cierto problema de decisión X:

- Se considera un **algoritmo determinista** $\mathcal{A}$ de fuerza bruta (coste en tiempo, exponencial) que resuelve X.
- A partir de $\mathcal{A}$, se diseña un **algoritmo no determinista**, $\mathcal{B}$, de coste en tiempo, polinomial que resuelve X.

De esta manera se probará que $X \in$ **NP**.

Un algoritmo determinista, $\mathcal{A}$, de fuerza bruta que resuelve el problema **CLIQUE** es el siguiente:

Entrada: $G = (\{x_1, \ldots, x_n\}, E)$ un grafo no dirigido y $k \in \mathbb{N}, k \leq n$

para cada subconjunto V_1 de $\{x_1, \ldots, x_n\}$ hacer

si $|V_1| = k$ y V_1 es un clique entonces

devolver **sí**

devolver **no**

El coste en tiempo de este algoritmo es del orden $O(2^n)$.

A partir del algoritmo determinista $\mathcal{A}$ diseñamos el algoritmo **no** determinista $\mathcal{B}$;

Entrada: $G = (\{x_1, \ldots, x_n\}, E)$ un grafo no dirigido y $k \in \mathbb{N}, k \leq n$

$V_1 \longleftarrow \emptyset$

para $i = 1$ **hasta** n hacer

elección e_1, e_2

$e_1 : V_1 \longleftarrow V_1 \cup \{x_i\}$; $k \longleftarrow k - 1$

$e_2 : V_1 \longleftarrow V_1$

si $k = 0$ y V_1 es un clique de G entonces

devolver **sí**

Veamos que el algoritmo no determinista $\mathcal{B}$ es de coste en tiempo polinomial y, además, resuelve el problema **CLIQUE**.

El algoritmo no determinista $\mathcal{B}$ es de **coste en tiempo polinomial**:

- El tiempo de ejecución de la fase no determinista es del orden exacto de n.
- El tiempo de ejecución de la fase determinista es del orden $O(n^2)$.

Luego, el coste en tiempo del algoritmo no determinista $\mathcal{B}$ es del orden $O(n^2)$.

Veamos, ahora, que el algoritmo no determinista $\mathcal{B}$ **resuelve** el problema **CLIQUE**.

- Supongamos que V_1 es un clique de G de tamaño k. Entonces se considera la computación $\mathcal{C}$ del algoritmo $\mathcal{B}$ que, en la fase no determinista, selecciona el conjunto V_1. Entonces, la computación $\mathcal{C}$ es, obviamente, de aceptación.
- Supongamos ahora que la ejecución del algoritmo $\mathcal{B}$ con entrada (G, k) contiene alguna computación $\mathcal{C}'$ que es de aceptación.

  Sea V_1' el conjunto seleccionado en la fase no determinista de $\mathcal{C}'$. Puesto que $\mathcal{C}'$ es de aceptación, ha de verificarse que V_1' es un clique de G de tamaño k.

■

La clase NP es estable por reducibilidad en tiempo polinomial

Proposición 4: Sean X_1, X_2 problemas de decisión tales que $X_1 \leqslant^p X_2$ y $X_2 \in \mathbf{NP}$. Entonces, $X_1 \in \mathbf{NP}$.

Demostración:

* Puesto que $X_2 \in \mathbf{NP}$, existirá un algoritmo **no determinista** $\mathcal{A}$ de coste en tiempo polinomial que resuelve el problema X_2.
* Puesto que $X_1 \leqslant^p X_2$, existirá una reducibilidad en tiempo polinomial, g, de X_1 en X_2; es decir, $g : E_{X_1} \to E_{X_2}$ es computable en tiempo polinomial y para cada $u \in E_{X_1}$ se tiene que $u \in L_{X_1} \iff g(u) \in L_{X_2}$.
* Consideremos el siguiente algoritmo **no determinista** $\mathcal{B}$:

  `Entrada:` $u \in E_{X_1}$

  Calcular $g(u)$

  Ejecutar el algoritmo **no determinista** $\mathcal{A}$ con entrada $g(u)$

  Entonces, $\mathcal{B}$ es un algoritmo **no determinista** de coste en tiempo polinomial.

  Además, $\mathcal{B}$ resuelve el problema X_1 ya que para cada $u \in E_{X_1}$ se tiene:

  $u \in L_{X_1} \overset{g \text{ r.t.p.}}{\iff} g(u) \in L_{X_2} \overset{\mathcal{A} \text{ resuelve } X_2}{\iff}$ Existe, al menos, una computación de aceptación de $\mathcal{A}(g(u))$

  $\overset{\text{definic. } \mathcal{B}}{\iff}$ Existe, al menos, una computación de aceptación de $\mathcal{B}(u)$

  Por tanto, $X_1 \in \mathbf{NP}$.

■

Nota: Pueden existir problemas de decisión X_1, X_2 tales que $X_1 \leqslant^p X_2$ y $X_1 \in \mathbf{NP}$ pero que, en cambio, $X_2 \notin \mathbf{NP}$.

El problema P versus NP

- Se tiene que **P** $\subseteq$ **NP**: ¿**Es estricta la inclusión P$\subseteq$NP**?

- Problema **P versus NP**: determinar si coinciden las clases **P** y **NP**.
- Sin lugar a dudas, es el problema más importante en *Computer Science*.
- **Problema central** de la Ciencia.

Se cree ...

Informalmente, el problema **P versus NP** consiste en decidir cuál de las dos cuestiones (asociadas a un problema abstracto) **es más difícil**:

* **Hallar** la solución exacta del mismo.
* **Comprobar** si una presunta solución de dicho problema es, realmente, una solución correcta.

Se cree que debería ser más difícil:

* **Encontrar** la solución que **chequear la corrección** de una presunta solución.
* **Resolver** un problema que **comprobar** la corrección de una solución.

Se cree que **resolver** un problema debería ser más difícil que **chequear** la corrección de una presunta solución del mismo.

Se cree ...

Esta creencia generalizada, se puede ilustrar con un ejemplo:

- Consideremos un problema abstracto que consiste en resolver un sistema lineal de tres ecuaciones lineales con tres incógnitas,
  - **Hallar la solución** (general) del sistema consiste en hallar el conjunto de todas las ternas ordenadas de números, que satisfagan las tres ecuaciones.
  - **Chequear** si tres números concretos es una solución del sistema consiste en **comprobar** si esa terna satisface las tres ecuaciones del sistema.
- **Todo parece indicar** que **hallar la solución** del sistema debería ser más difícil que **chequear** la adecuación de una presunta solución.

Lo que se expresa diciendo: **"todo parece indicar"** que $\mathbf{P} \neq \mathbf{NP}$.

La conjetura P $\neq$ NP

¿Cómo se abordaría la resolución de dicha conjetura?

- Definición formal del concepto: un problema abstracto es **más dificil** que otro (**reducibilidad en tiempo polinomial**).
- Si se quiere probar que **P** $\subsetneq$ **NP**, los candidatos idóneos serían los problemas **más difíciles** de **NP** (que se denominarán **problemas NP-completos**).
- En 1970, S. Cook demostró la **NP**-completitud de un problema de decisión relacionado con la **lógica** proposicional (el problema **SAT**).
- Una metodologia para probar que **P** $\neq$ **NP**:
  - Elegir **un** problema **NP**-completo y probar que **no** pertenece a **P**.
  - Ahora bien, si se prueba que **un** problema **NP**-completo **pertenece** a **P** entonces resultaría que **P**=**NP**.

Se cree que **P** $\neq$ **NP** y, por ello, se dice que los problemas **NP**-completos son **presuntamente intratables**.

La conjetura P $\neq$ NP

El **Clay Mathematics Institute** de Cambridge, Massachusetts (USA), ofrece un millón de dolares para aquél que lo resuelva (**The Millenium Prize Problems**).

¿Qué sucedería si ...?

- ¿Qué sucedería si se probara que **P** $\neq$ **NP**?
  - ⋆ Reforzamiento de la **seguridad** de los actuales **sistemas de encriptación**.
  - ⋆ Existencia de problemas de complejidad **intermedia** (R. Ladner, 1975).

- ¿Qué sucedería si se probara que **P** $=$ **NP**?
  - ⋆ Consecuencias funestas para los sistemas de encriptación actuales.
  - ⋆ Obtención de **pruebas mecánicas de teoremas** matemáticos de interés.
  - ⋆ Posibilidad de contruir un **decodificador universal**.

Criptosistemas

Sistema criptográfico: método para reescribir un texto/mensaje (**encriptación**) cuyo contenido no puede ser fácilmente interpretado (**desencriptación**), a menos que se disponga de un mecanismo "adecuado". Los procesos se implementan mediante **claves**.

- ⋆ Cripsosistema **simétrico**: Misma clave para encriptar y desencriptar (ejemplo **DES** - Data Encryption Standard) .
- ⋆ Cripsosistema **asimétrico**: Distintas claves para encriptar y desencriptar (una **pública** y otra privada; ejemplo **RSA**).

En 1976, W. Diffie y M. Hellman[1] formularon los principios teóricos que deberían satisfacer los criptosistemas de clave pública.

Primer criptosistema de clave pública verificando las condiciones de Diffie-Hellman: el algoritmo **RSA**, propuesto en 1978 por R. **R**ivest, A. **S**hamir y L. **A**dleman [2]

Asociado a cualquier criptosistema, existe un problema presuntamente intratable.

En el criptosistema **RSA**, se trata del problema de la **factorización de semiprimos**.

- ∗ Un número natural $n \geq 2$ se dice que es **semiprimo** si es el producto de dos números primos.
- ∗ Problema de la **factorización de semiprimos**: dado un número semprimo, hallar su factorización.

---

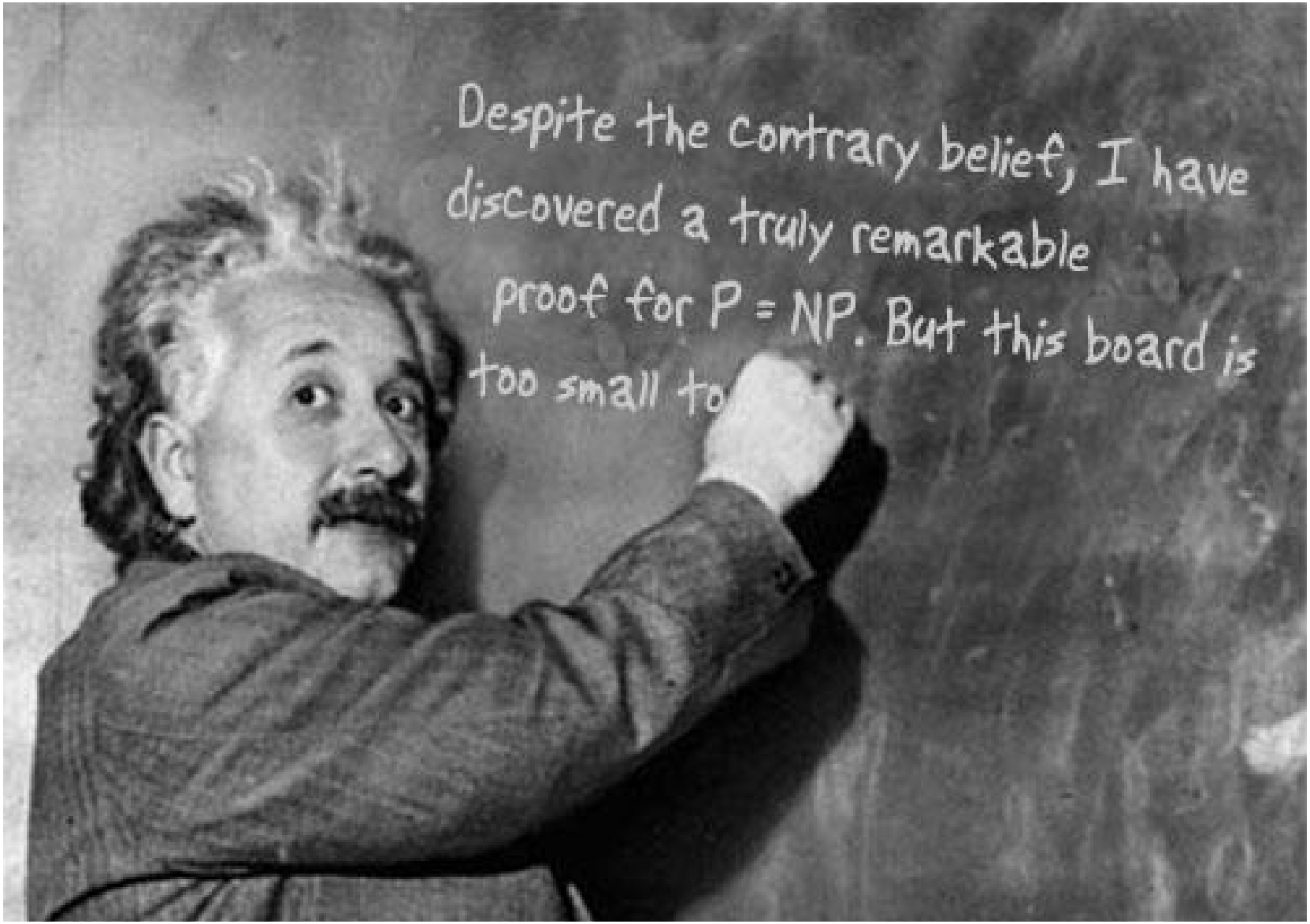
[1] W. Diffie, M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, **22**, 6 (1976), 644-654.

[2] R.L. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *CAMC*, **21**, 2 (1978), 120-126..

Desencriptación de textos codificados en el sistema **DES** (**Data Encryption Standard**).

Ataque convencional sobre un texto **DES** mediante búsqueda exhaustiva:

* Un ordenador capaz de realizar un millón de operaciones por segundo, tardaría unos **mil años**.
* Puede ser decodificada por un **ordenador molecular** en unos **cuatro meses** (D. Boneh, Ch. Dunworth y R.J. Lipton, 1996).


Despite the contrary belief, I have
discovered a truly remarkable
proof for P = NP. But this board is
too small to

Problemas NP–duros

Para demostrar que **P** $\neq$ **NP**, los candidatos idóneos (que pertenezcan a la clase **NP** y **no** a la clase **P**), serían los problemas "más difíciles" de la clase **NP**.

Definición: Un **problema de decisión** X es **NP–duro sii** para cada problema de decisión $X' \in$ **NP** se tiene que $X' \leqslant^p X$.

Proposición 5: Si X_1, X_2 son problemas de decisión tales que X_1 es **NP**–duro y $X_1 \leqslant^p X_2$, entonces X_2 es **NP**–duro.

Demostración:

Para probar que X_2 es **NP**–duro, sea Z un problema arbitrario de la clase **NP**.

Puesto que X_1 es **NP**–duro resultará que $Z \leqslant^p X_1$.

Finalmente, teniendo presente que $X_1 \leqslant^p X_2$, concluimos que $Z \leqslant^p X_2$.

Por tanto, el problema X_2 es **NP**–duro.

■

Problemas NP–completos

Definición: Un **problema de decisión** X es **NP–completo sii** $X \in \mathbf{NP}$ y X es **NP**–duro.

Los problemas **NP**–completos son los **más difíciles** de la clase **NP**:

- ⋆ Para probar que $\mathbf{P} = \mathbf{NP}$, los testigos ideales serían los problemas **NP**–completos.

Proposición 6: Si X_1, X_2 son problemas **NP**–completos, entonces $X_1 \equiv^p X_2$.

Demostración:

En primer lugar, veamos que $X_1 \leqslant^p X_2$.

- ⋆ En efecto: basta tener presente que X_2 es **NP**–duro y $X_1 \in \mathbf{NP}$ (ya que X_1 es **NP**-completo).

De manera similar, se prueba que $X_2 \leqslant^p X_1$.

- ⋆ En efecto: basta tener presente que X_1 es **NP**–duro y $X_2 \in \mathbf{NP}$ (ya que X_2 es **NP**-completo).

■

Generación de problemas NP–completos

Proposición 7: (**Generación de problemas NP–completos**)
Sea X un problema de decisión tal que:

* $X \in \mathbf{NP}$
* Existe Z (Z es **NP**–completo $\wedge\ Z \leqslant^p X$).

Entonces X es un problema **NP**–completo.

Demostración:

Puesto que $X \in \mathbf{NP}$, bastará probar que X es un problema **NP**–duro.

Para ello, téngase presente que si Z es un problema **NP**–completo tal que $Z \leqslant^p X$, entonces de la proposición 3 se deduce que X es **NP**–duro (al serlo Z). ■

Nota: De la proposición 5 resulta que, para establecer la **NP**-completitud de un problema $X \in \mathbf{NP}$, se necesita hallar/buscar/encontrar un problema **NP**-completo Z que sea **más fácil** que X (es decir, tal que $Z \leqslant^p X$).

Nacimiento de la teoría de la **Complejidad Computacional**:

- ⋆ **Teorema de Cook**: **SAT** es un problema **NP**–completo (1971).

Proposición 8: Son equivalentes:

(a) Existe **UN** problema **NP**–duro, X, tal que $X \in \mathbf{P}$.

(b) $\mathbf{P} = \mathbf{NP}$.

Demostración:

$\boxed{(a) \Longrightarrow (b)}$

Sea X un problema **NP**–duro, X, tal que $X \in \mathbf{P}$. Veamos que $\mathbf{NP} \subseteq \mathbf{P}$.

Para ello, si $Z \in \mathrm{NP}$ entonces $Z \leqslant^p X$ (ya que X es **NP**–duro).

Por tanto, de la proposición 1 se deduce que $Z \in \mathbf{P}$.

$\boxed{(b) \Longrightarrow (a)}$

Si $\mathbf{P} = \mathbf{NP}$ entonces el problema **SAT** es **NP**–duro y **SAT** $\in \mathbf{NP} = \mathbf{P}$

■

Proposición 9: Se verifica que $\mathbf{P} = \mathbf{NP} \iff$ **SAT** $\in \mathbf{P}$.

Demostración:

Supongamos que $\mathbf{P} = \mathbf{NP}$.

Puesto que el problema **SAT** es **NP**-completo resulta que **SAT** $\in \mathbf{NP} = \mathbf{P}$.

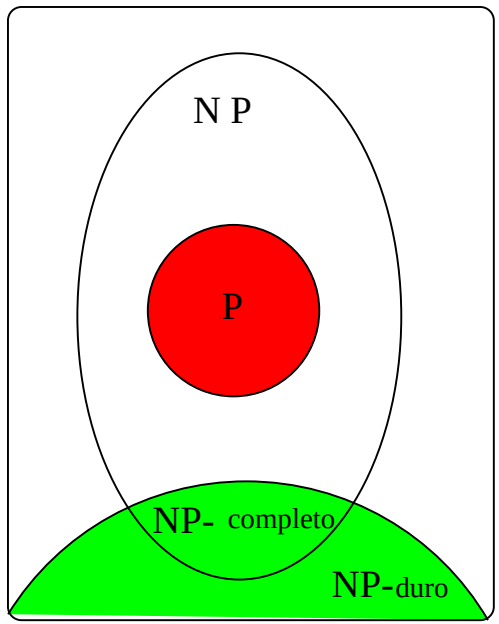
Recíprocamente, supongamos que **SAT** $\in \mathbf{P}$. Veamos que $\mathbf{P} = \mathbf{NP}$.

Para ello, si $Z \in \mathbf{NP}$ entonces $Z \leqslant^p$ **SAT** (ya que **SAT** es **NP**–duro).

Puesto que **SAT** $\in \mathbf{P}$, de la proposición 1 se deduce que $Z \in \mathbf{P}$.

■

La búsqueda de **un** adecuado problema **NP**–completo puede responder en sentido **afirmativo** o en sentido **negativo**, a la conjetura $\mathbf{P} \neq \mathbf{NP}$.



El problema SAT

Definición: El **lenguaje de la Lógica Proposicional** consta de:

- ⋆ Conjunto numerable, VP, de variables proposicionales.
- ⋆ Conectivas lógicas: $\neg$ (negación) y $\vee$ (disyunción).
- ⋆ Símbolos auxiliares: (y).

Definición: El conjunto $PForm$ de las **fórmulas proposicionales** es el menor conjunto, Γ, que contiene a VP y, además,

- ⋆ Si $P \in \Gamma$, entonces $\neg P \in \Gamma$.
- ⋆ Si $P, Q \in \Gamma$, entonces $(P \vee Q) \in \Gamma$.

A partir de $\neg$ y $\vee$ se definen las conectivas lógicas $\wedge$, $\rightarrow$ y $\leftrightarrow$.

Notaremos

- ⋆ $\neg P$ así $\overline{P}$.
- ⋆ $P \vee Q$ así $P + Q$.
- ⋆ $P \wedge Q$ así $P \cdot Q$.

Definición: Un **literal** es una variable proposicional o la negación de una variable proposicional.

Definición: Una **cláusula** es la disyunción de un número finito de literales.
Ejemplos: $x_1 \vee \overline{x_2} \vee \overline{x_3}$; $\overline{x_1} \vee x_2$; $x_1 \vee x_2 \vee \overline{x_3}$

Definición: Una fórmula proposicional está en **forma normal conjuntiva** (**FNC**) si es la conjunción de un número finito de cláusulas.
Ejemplo: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2)$

Toda fórmula proposicional en FNC es la conjunción de una disyunción de literales (las supondremos en **forma "simplificada"**: las cláusulas y las variables no aparecerán repetidas ni serán complementarias).

Definición: Una **valoración** (**asignación**) **de verdad** es una aplicación de VP en $\{0, 1\}$.

- Toda valoración de verdad se extiende de manera natural a una aplicación de $PForm$ en $\{0, 1\}$ (a través de las **tablas de verdad**).

Valoraciones de verdad **relevantes** para una fórmula.

- Una valoración relevante para una fórmula es una aplicación del conjunto de sus variables en $\{0, 1\}$.
- Si φ tiene n variables, entonces el número total de valoraciones relevantes para φ es 2^n.

Definición: Dos fórmulas proposicionales son **semánticamente equivalentes** si cualquier valoración le asigna a ambas el mismo valor.

- ⋆ Toda fórmula proposicional tiene una fórmula semánticamente equivalente en FNC (y en forma simplificada).

Definición: Una fórmula proposicional, φ, es **satisfactible sii** existe, al menos, una valoración, σ, tal que $\sigma(\varphi) = 1$.

Problema SAT: **Dada una fórmula proposicional en FNC** (y en forma simplificada), **determinar si es satisfactible**.

Proposición 10: El problema **SAT** pertenece a la clase **NP**.

Demostración: Se considera un algoritmo determinista $\mathcal{A}$ de fuerza bruta que resuelva el problema:

```
Entrada: Una fórmula φ ∈ E_SAT con Var(φ) = {x_1, ..., x_n}
    para cada valoración σ de {x_1, ..., x_n} hacer
        si σ(φ) = 1 entonces
            devolver sí
    devolver no
```

Este algoritmo determinista es de coste en tiempo **exponencial** (específicamente, del orden $O(2^n)$) y, obviamente, resuelve el problema **SAT**.

Inspirándonos en él, vamos a diseñar un **algoritmo no determinista** $\mathcal{B}$, de coste en tiempo **polinomial**, que resuelve el problema **SAT**.

Consideremos el siguiente algoritmo $\mathcal{B}$ no determinista:

Entrada: Una fórmula $\varphi \in \mathbf{E}_{\mathbf{SAT}}$ con $Var(\varphi) = \{x_1, \ldots, x_n\}$

$\sigma \leftarrow \emptyset$

para cada $i = 1$ **hasta** n **hacer**

  Elegir e_1, e_2

    $e_1 : \sigma \leftarrow \sigma \cup \{(x_i, 0)\}$

    $e_2 : \sigma \leftarrow \sigma \cup \{(x_i, 1)\}$

} Fase no determinista

hallar $\sigma(\varphi)$

si $\sigma(\varphi) = 1$ **entonces**

  devolver sí

si no

  devolver no

} Fase determinista

Hallemos el coste en tiempo del algoritmo $\mathcal{B}$.

* Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
* Tiempo de la fase determinista: el número total de literales de la fórmula.

El coste en tiempo del algoritmo $\mathcal{B}$ es lineal en el tamaño de la fórmula de entrada (número total de literales de la fórmula).

Veamos que el algoritmo $\mathcal{B}$ resuelve el problema **SAT**. Para ello, sea φ una fórmula de entrada del problema.

* Supongamos que φ es satisfactible. Sea σ_0 una valoración tal que $\sigma_0(\varphi) = 1$. Sea $\mathcal{C}$ la computación que en la fase no determinista selecciona, precisamente, σ_0. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
* Supongamos que φ **no** es satisfactible. Sea $\mathcal{C}$ cualquier computación y notemos σ_1 la valoración seleccionada en la fase no determinista. Entonces $\sigma_1(\varphi) = 0$ y, por tanto, la computación $\mathcal{C}$ **no** será de aceptación.

En consecuencia, el problema **SAT** pertenece a la clase **NP**.

■

Problema SAT(k), para cada número natural $k \geq 1$

Para cada número natural $k \geq 1$ se considera el siguiente problema:

Problema SAT(k): Dada una fórmula proposicional en FNC en la que cada cláusula posee, a lo sumo, k literales, determinar si es satisfactible.

Proposición 11: $\forall k \geqslant 1$ **(SAT(k)** $\leqslant^p$ **SAT)**.

Demostración: Consideremos la aplicación $F : E_{\mathrm{SAT(k)}} \longrightarrow E_{\mathrm{SAT}}$ definida así: $F(\varphi) = \varphi$, para cada $\varphi \in E_{\mathrm{SAT(k)}}$. Obviamente, F es una reducibilidad en tiempo polinomial de **SAT(k)** en **SAT**.

■

Corolario: $\forall k \geqslant 1$ (SAT(k) $\in$ NP).

Demostración: Basta tener presente la estabilidad de la clase **NP** (proposición 4).

■

Proposición 12: $\forall k \geqslant 1$ **(SAT(k)** $\leqslant^p$ **SAT(k+1))**.

Demostración: Consideremos la aplicación $F : E_{\mathrm{SAT(k)}} \longrightarrow E_{\mathrm{SAT(k+1)}}$ definida así: $F(\varphi) = \varphi$, para cada $\varphi \in E_{\mathrm{SAT(k)}}$. Obviamente, F es una aplicación "**bien definida**" y, además, es una reducibilidad en tiempo polinomial de **SAT(k)** en **SAT(k+1)**.

■

Teorema 1: El problema **SAT(3)** es **NP**–completo.

Demostración: Por el teorema de generación de problemas **NP**-completos basta probar que **SAT** $\leqslant^p$ **SAT(3)**.

- Para ello, a cada cláusula c de $\varphi \in E_{\mathtt{SAT}}$ se le asocia una fórmula $D_c \in E_{\mathtt{SAT(3)}}$ definida como sigue:
  - Si c posee, a lo sumo, 3 literales, entonces $D_c = c$.
  - Si $c = l_1 + \cdots + l_n$ $(n \geqslant 4)$, entonces

$$D_c = (l_1 + \overline{z}_1) \cdot (l_2 + z_1 + \overline{z}_2) \cdots (l_n + z_{n-1} + \overline{z}_n) \cdot z_n$$

    en donde $z_1, \ldots, z_n$ son nuevas variables (distintas de las de c).

Se considera la aplicación $F : E_{\mathtt{SAT}} \to E_{\mathtt{SAT(3)}}$ definida así:

$$F(c_1 \cdots c_p) = D_{c_1} \cdots D_{c_p}$$

Veamos que F es computable en tiempo polinomial.

* El siguiente programa calcula F en tiempo polinomial:

```
Entrada: φ ≡ c_1 · ··· · c_p con c_i ≡ l_i^1 + ··· + l_i^{r_i}   (1 ⩽ i ⩽ p)
        para i ← 1 hasta p hacer
                si r_i ⩽ 3 entonces
                        D_{c_i} ← c_i
                si no
                        Considerar nuevas variables z_i^1, ..., z_i^{r_i}
                        D_{c_i} ← (l_i^1 + z̄_i^1) · z_i^{r_i}
                        para j ← 2 hasta r_i hacer
                                D_{c_i} ← D_{c_i} · (l_i^j + z_i^{j-1} · z̄_i^j)
        devolver D_{c_1} ... D_{c_p}
```

Coste en tiempo: $\theta(r_1 + \cdots + r_p)$.

Veamos que $\forall \varphi \in E_{SAT}\ (\varphi \in L_{SAT} \iff F(\varphi) \in L_{SAT(3)})$; es decir, para cada $\varphi \in E_{SAT}$ se tiene que φ es satisfactible **sii** $F(\varphi)$ es satisfactible.

Lema 1: Para cada cláusula c se tiene que c es satisfactible **sii** D_c es satisfactible.

Demostración: Si c tiene, a lo sumo, tres literales, el resultado es obvio (ya que $c = D_c$). Sea c una cláusula tal que $c = l_1 + \ldots + l_n$ (con $n \geqslant 4$). Entonces, $D_c = (l_1 + \overline{z}_1) \cdot (l_2 + z_1 + \overline{z}_2) \cdots (l_n + z_{n-1} + \overline{z}_n) \cdot z_n$.

* Supongamos que c es satisfactible. Sea σ una valoración tal que $\sigma(c) = 1$ y consideremos $j_0 = \text{mín}\{j :\ \sigma(l_j) = 1\}$. Sea τ la valoración:

$$\begin{cases} \tau(l_j) = \sigma(l_j) & \text{si } 1 \leqslant j \leqslant n \\ \tau(z_j) = 0 & \text{si } 1 \leqslant j < j_0 \\ \tau(z_j) = 1 & \text{si } j_0 \leqslant j \leqslant n \\ \tau(x) = 1 & \text{para cualquier otra variable} \end{cases}$$

  Se tiene que $\tau(D_c) = 1$.

* Supongamos que D_c es satisfactible. Sea σ una valoración tal que $\sigma(D_c) = 1$, entonces se verifica que $\sigma(c) = 1$. En efecto, de lo contrario, $\sigma(l_1) = \cdots = \sigma(l_n) = 0$. En tal situación, se prueba (por inducción acotada) que $\sigma(z_1) = \cdots = \sigma(z_n) = 0$.

  Lo que es una contradicción (pues z_n es una cláusula de la fórmula D_c).

■

Lema 2: Para cada $\varphi \in E_{\text{SAT}}$ se tiene que $\varphi \in L_{\text{SAT}} \Longleftrightarrow F(\varphi) \in L_{\text{SAT(3)}}$

Demostración: Sea $\varphi \equiv c_1 \cdot \cdots \cdot c_p$ con $c_i \equiv l_i^1 + \ldots + l_i^{r_i}$.

* Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Entonces, para cada i, $1 \leqslant i \leqslant p$, $\sigma(c_i) = 1$. Luego, del lema 1 resulta que D_{c_i} es satisfactible por una valoración que "extiende" σ.

  Considerando, en su caso, las nuevas variables en D_{c_i} distintas todas ellas entre sí, para $1 \leq i \leq p$, existirá una extensión de σ que hace verdadera las fórmulas $D_{c_1}, \ldots, D_{c_p}$. Luego $F(\varphi)$ será satisfactible.

* Supongamos que $F(\varphi)$ es satisfactible. Sea σ una valoración tal que $\sigma(F(\varphi)) = 1$. Entonces $\sigma(D_{c_1}) = \ldots = \sigma(D_{c_p}) = 1$. De la prueba del lema 1 resulta que $\sigma(c_1) = \ldots = \sigma(c_p) = 1$. Luego, $\sigma(\varphi) = 1$.

■

En consecuencia: **SAT** $\leqslant^p$ **SAT(3)**.

* Como **SAT** es **NP**–completo y **SAT(3)** $\in$ NP, del teorema de generación de problemas **NP**–completos se concluye que **SAT(3)** es **NP**-completo.

■

Corolario: $\forall k \geqslant 3$ (**SAT(k)** es **NP**-completo).

Demostración: Por inducción sobre k (para $k \geq 3$).

* Caso base: $k = 3$. Se deduce del teorema anterior.
* Paso inductivo: Sea $k \geq 3$ tal que **SAT(k)** es **NP**-completo. Veamos que **SAT(k+1)** también es **NP**-completo. En efecto:
  * Por una parte, del corolario de la proposición 11 resulta que **SAT(k+1)** $\in$ NP.
  * Por otra parte, de la proposición 12 resulta que **SAT(k)** $\leqslant^p$ **SAT(k+1)**.

  De estas relaciones y del teorema de generación de problemas **NP**-completos, se deduce que **SAT(k+1)** es, también, **NP**-completo.

■

Problema k–SAT, para cada número natural $k \geq 1$

Problema k–SAT: Dada una fórmula proposicional en FNC en la que cada cláusula posee exactamente k literales, determinar si es satisfactible.

El problema **k–SAT** también se suele denotar así: **SAT*(k)**.

Proposición 13: $\forall k \geqslant 1$ (**k–SAT** $\leqslant^p$ **SAT**).

Demostración: Consideremos la aplicación $F : E_{\mathrm{k-SAT}} \longrightarrow E_{\mathrm{SAT}}$ definida como sigue: $F(\varphi) = \varphi$, para cada $\varphi \in E_{\mathrm{k-SAT}}$. Obviamente, F es una reducibilidad en tiempo polinomial de **k–SAT** en **SAT**.

■

Corolario: $\forall k \geqslant 1$ (**k–SAT** $\in$ **NP**).

Demostración: Basta tener presente la estabilidad de la clase **NP** (proposición 3).

■

Proposición 14: $\forall k \geqslant 1$ (**k–SAT** $\leqslant^p$ **(k+1)–SAT**).

Demostración: Consideremos la aplicación $F : E_{\mathrm{k-SAT}} \longrightarrow E_{\mathrm{(k+1)-SAT}}$ definida como sigue: si $\varphi = c_1 \cdot \cdots \cdot c_p \in E_{k-\mathtt{SAT}}$ entonces

$$F(\varphi) = (c_1 + z_1) \cdot (c_1 + \overline{z}_1) \cdot \cdots \cdot (c_p + z_p) \cdot (c_p + \overline{z}_p)$$

siendo $z_1, \ldots, z_p$ nuevas variables (que no aparecen en las cláusulas $c_1, \ldots, c_p$).

Veamos que F es una reducibilidad en tiempo polinomial de **k–SAT** en **(k+1)–SAT**. En efecto:

- $\star$ $F(\varphi) \in E_{\mathrm{(k+1)-SAT}}$, para cada $\varphi \in E_{\mathrm{k-SAT}}$.
- $\star$ F es computable en tiempo polinomial.
- $\star$ Si φ es satisfactible, existe una valoración σ tal que $\sigma(\varphi) = 1$. Luego $\sigma(c_1) = \ldots = \sigma(c_p) = 1$. Por tanto, $\sigma(F(\varphi)) = 1$
- $\star$ Si $F(\varphi)$ es satisfactible, existe una valoración τ tal que $\tau(F(\varphi)) = 1$. Luego $\tau(c_1 + z_1) = \tau(c_1 + \overline{z}_1) = \ldots = \tau(c_p + z_p) = \tau(c_p + \overline{z}_p) = 1$ y, por tanto, $\tau(c_1) = \ldots = \tau(c_p) = 1$; es decir $\tau(\varphi) = 1$

En consecuencia, **k–SAT** $\leqslant^p$ $(k+1)$–**SAT**.

■

Teorema 4: El problema **3–SAT** es **NP**–completo.

Demostración: Basta ver que **SAT(3)** $\leqslant^p$ **3–SAT**.

* Para cada cláusula c de $\varphi \in E_{\text{SAT(3)}}$ se define la fórmula D_c como sigue:
  - Si $c = l_1$, entonces
    $D_c = (l_1 + z_1 + z_2) \cdot (l_1 + \overline{z}_1 + z_2) \cdot (l_1 + z_1 + \overline{z}_2) \cdot (l_1 + \overline{z}_1 + \overline{z}_2)$
    en donde z_1, z_2 son nuevas variables.
  - Si $c = l_1 + l_2$, entonces

    $$D_c = (l_1 + l_2 + t) \cdot (l_1 + l_2 + \overline{t})$$

    en donde t es una nueva variable.
  - Si c posee exactamente 3 literales, $D_c = c$.

Sea $F : E_{\text{SAT(3)}} \to E_{3-\text{SAT}}$ definida así:

$$F(c_1 \cdots c_p) = D_{c_1} \cdots D_{c_p}$$

Veamos que F es computable en tiempo polinomial.

* El siguiente programa calcula F en tiempo polinomial:

Entrada: $\varphi \equiv c_1 \cdots c_p$ con $c_i \equiv l_i^1 + \cdots + l_i^{r_i}$ $(1 \leqslant i \leqslant p,\ 1 \leqslant r_i \leqslant 3)$

para $i \leftarrow 1$ **hasta** p **hacer**

si $r_i = 1$ **entonces**

Considerar dos nuevas variables z_i^1, z_i^2

$D_{c_i} \leftarrow (l_i^1 + z_i^1 + z_i^2) \cdot (l_i^1 + \overline{z}_i^1 + z_i^2) \cdot (l_i^1 + z_i^1 + \overline{z}_i^2) \cdot (l_i^1 + \overline{z}_i^1 + \overline{z}_i^2)$

si $r_i = 2$ **entonces**

Considerar una nueva variable z_i

$D_{c_i} \leftarrow (l_i^1 + l_i^2 + z_i) \cdot (l_i^1 + l_i^2 + \overline{z}_i)$

si $r_i = 3$ **entonces**

$D_{c_i} \leftarrow c_i$

devolver $D_{c_1} \ldots D_{c_p}$

Coste en tiempo: $\theta(r_1 + \cdots + r_p)$.

Veamos que $\forall \varphi \in E_{\text{SAT(3)}}$ $(\varphi \in L_{\text{SAT(3)}} \Longleftrightarrow F(\varphi) \in L_{3-\text{SAT}})$; es decir, para cada $\varphi \in E_{\text{SAT(3)}}$ se tiene que φ es satisfactible **sii** $F(\varphi)$ es satisfactible. .

Lema 1: *Una cláusula c con, a lo sumo, tres literales es satisfactible* **sii** *D_c es satisfactible.*

Demostración: Si c posee exactamente tres literales entonces $c = D_c$. Luego el resultado es trivial. Supongamos, pues, que c posee, a los sumo, dos literales. Sea σ una valoración tal que $\sigma(c) = 1$ y sea τ una valoración que extiende a σ.

* Caso 1: Sea $c = l_1$. Entonces $\tau(l_1) = 1$. Luego

  $\tau(D_c) = \tau(l_1 + z_1 + z_2) \cdot \tau(l_1 + \overline{z}_1 + z_2) \cdot \tau(l_1 + z_1 + \overline{z}_2) \cdot \tau(l_1 + \overline{z}_1 + \overline{z}_2) = 1$

* Caso 2: Sea $c = l_1 + l_2$. Como $\tau(l_1 + l_2) = \sigma(l_1 + l_2) = 1$, se tiene que $\tau(D_c) = \tau(l_1 + l_2 + t) \cdot \tau(l_1 + l_2 + \overline{t}) = 1$.

Sea σ una valoración tal que $\sigma(D_c) = 1$.

* Caso 1: Sea $c = l_1$. Entonces $\sigma(l_1) = 1$, pues de lo contrario $\sigma(z_1) = \sigma(\overline{z}_1) = 1$.
* Caso 2: Sea $c = l_1 + l_2$. Si $\sigma(l_1) = \sigma(l_2) = 0$, entonces $\sigma(t) = \sigma(\overline{t}) = 1$.

■

Lema 2: Para cada $\varphi \in E_{SAT(3)}$ se tiene que $\varphi \in L_{SAT(3)} \iff F(\varphi) \in L_{3-SAT}$.

Demostración: Sea $\varphi \equiv c_1 \cdots c_p \in E_{SAT(3)}$.

- Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Entonces $\sigma(c_1) = \cdots = \sigma(c_p) = 1$. Si τ es una valoración que extiende a σ, entonces $\tau(D_{c_1}) = \cdots = \tau(D_{c_p}) = 1$.
- Supongamos que $F(\varphi)$ es satisfactible. Sea σ una valoración tal que $\sigma(F(\varphi)) = 1$. Entonces $\sigma(D_{c_1}) = \cdots = \sigma(D_{c_p}) = 1$. Razonando como en el lema 1, se tiene $\sigma(c_1) = \cdots = \sigma(c_p) = 1$. Por tanto, $\sigma(\varphi) = 1$.

■

En consecuencia, se tiene que **SAT(3)** $\leqslant^p$ **3–SAT**.

Como **SAT(3)** es **NP**–completo y **3–SAT** $\in$ NP, del teorema de generación de problemas **NP**–completos se concluye que **3–SAT** es **NP**-completo.

■

Corolario: $\forall k \geqslant 3$ (**k–SAT** es **NP**-completo).

Demostración: Se prueba por inducción sobre k.

* Caso base: $k = 3$. Se deduce del teorema anterior.
* Paso inductivo: Sea $k \geq 3$ tal que **k–SAT**. es **NP**-completo. Veamos que **(k+1)–SAT**. también es **NP**-completo. En efecto:
  * Por una parte, del corolario de la proposición 13 resulta que **(k+1)–SAT** $\in$ **NP**.
  * Por otra parte, de la proposición 14 resulta que **k–SAT** $\leqslant^p$ **(k+1)–SAT**.

  De estas relaciones y del teorema de generación de problemas **NP**-completos, se deduce que **(k+1)–SAT** es, también, **NP**-completo.

■

Proposición 15: **2–SAT** $\in$ P.

Demostración: Se establece una reducibilidad en tiempo polinomial del problema **2–SAT** a una versión del problema de la **REACHABILITY**.

■

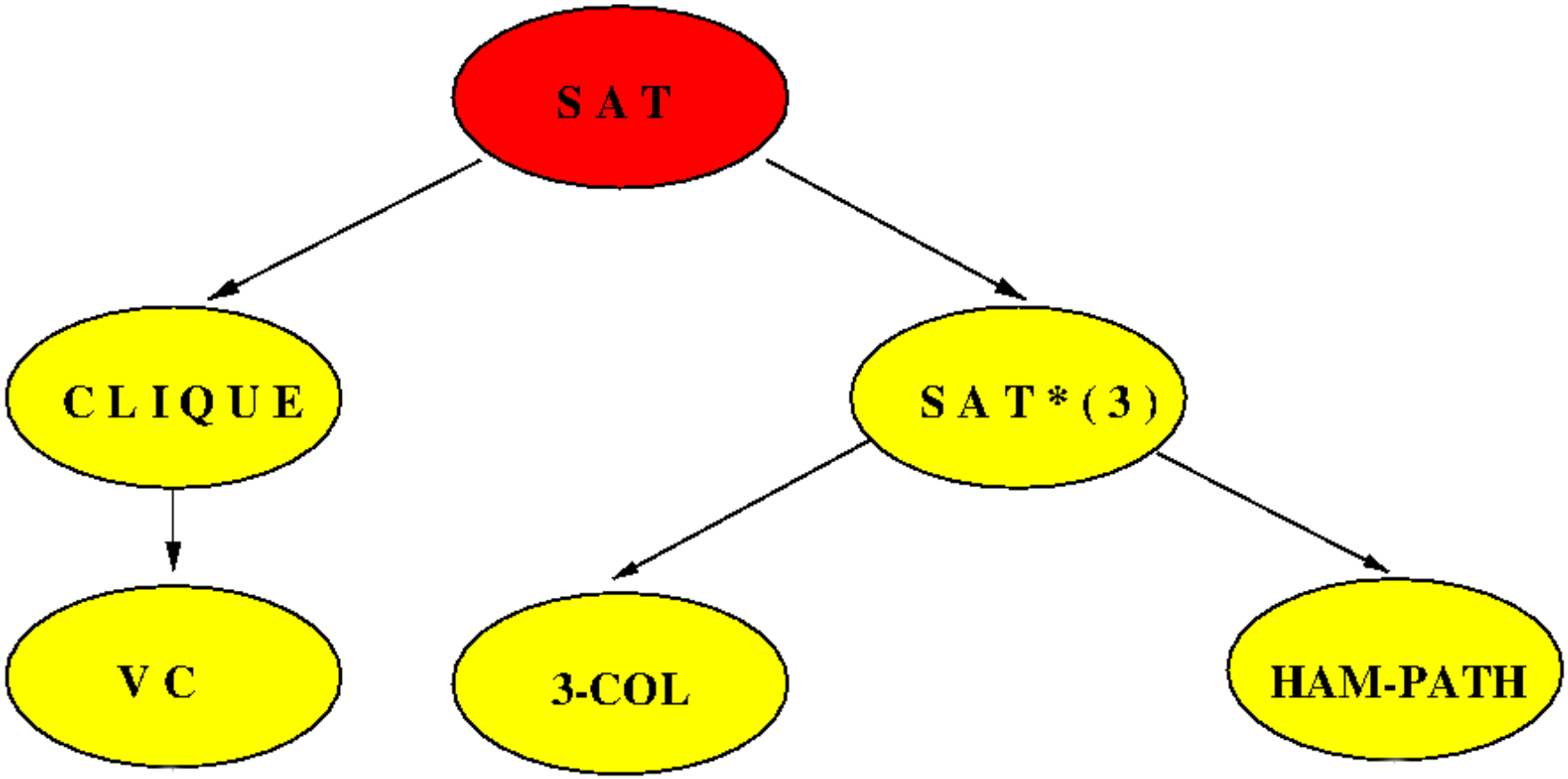
* **Corolario:** **1–SAT** $\in$ P.

Proposición 16: **SAT(2)** $\leqslant^p$ **2–SAT**.

Demostración: Es similar a la prueba de que **SAT(3)** $\leqslant^p$ **3–SAT**.

* **Corolario:** **SAT(2)** $\in$ P y **SAT(1)** $\in$ P.

Problemas relevantes NP-completos



El problema CLIQUE

- ⋆ **Dado un grafo no dirigido, G, y un número natural, k, determinar si existe en G un clique de tamaño k.**

En la **Proposición 3**, se ha probado que **CLIQUE** $\in$ NP.

Teorema 5: El problema **CLIQUE** es **NP**–completo.

Demostración: Veamos que **SAT** $\leqslant^p$ magentaCLIQUE.

Sea $\varphi \in E_{\mathtt{SAT}}$, con $\varphi \equiv c_1 \cdots c_p$ y $c_i = l_i^1 + \ldots + l_i^{r_i}$.

Construimos el grafo $G_\varphi = (V_\varphi, E_\varphi)$ como sigue:

$$V_\varphi = \{(i,1), \ldots, (i,r_i) : \ 1 \leqslant i \leqslant p\}$$

$$\{(i,j),(k,q)\} \in E_\varphi \iff i \neq k \wedge l_i^j \neq \bar{l}_k^q \wedge 1 \leqslant i,k \leqslant p \wedge 1 \leqslant j \leqslant r_i \wedge 1 \leqslant q \leqslant r_k$$

Consideramos la aplicación $F : E_{\mathtt{SAT}} \to E_{\mathtt{CLIQUE}}$ definida como sigue:
$F(\varphi) = (G_\varphi, p)$.

Vamos a probar que F es una reducibilidad en tiempo polinomial del problema SAT en el problema CLIQUE (con lo cual, del teorema de generación de problemas **NP**-completos, se deducirá la **NP**-completitud del problema CLIQUE).

* F es total computable en tiempo polinomial.

```
Entrada:  φ = (l_1^1 + ··· + l_1^{r_1}) ··· (l_p^1 + ··· l_p^{r_p})
    para α ← 1 hasta p hacer
        para β ← 1 hasta p hacer
            si α ≠ β entonces
                para γ ← 1 hasta r_α hacer
                    para δ ← 1 hasta r_β hacer
                        si l_α^γ ≠ l̄_β^δ entonces
                            A[i, j] ← 1
```

En donde:

- A es la matriz de adyacencia del grafo G_φ.
- $i = r_1 + \cdots + r_{\alpha-1} + \gamma$ es la posición que ocupa l_α^γ en φ.
- $j = r_1 + \cdots + r_{\beta-1} + \delta$ es la posición que ocupa l_β^δ en φ.

Obviamente, el algoritmo es de coste en tiempo polinomial.

* Veamos que la fórmula $\varphi \equiv c_1 \cdot \cdots \cdot c_p$, con $c_i \equiv l_i^1 + \cdots + l_i^{r_i}$ $(1 \leqslant i \leqslant p)$, es satisfactible si y solo si el grafo G_φ posee un clique de tamaño p.

  * Supongamos que la fórmula φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$.
    Para cada $i, 1 \leq i \leq p$, sea $j_i = \text{mín}\,\{k \in N \mid \sigma(l_i^k) = 1\}$. Consideremos el conjunto $V_1 = \{l_1^{j_1}, \ldots, l_p^{j_p}\}$. Entonces $\sigma(l_1^{j_1}) = \ldots = \sigma(l_p^{j_p}) = 1$ y, por tanto, de acuerdo con la definición de G_φ resulta que V_1 es un clique de dicho grafo de tamaño p.

  * Supongamos que el grafo G_φ posee un clique, V_1', de tamaño p.
    Entonces, el conjunto V_1' ha de ser del tipo $V_1' = \{l_1', \ldots, l_p'\}$, en donde l_i' será un literal de la cláusula c_i de φ.

    Consideremos la valoración τ definida como sigue: $\tau(l_i') = 1$ y, para las restantes variables. el valor de τ es 0. Entonces, τ está bien definida ya que en V_1' no puede aparecer un literal y su negación.

    De acuerdo con la definición anterior $\tau(c_i) = 1$, para cada cláusula c_i ya que, al menos, uno de los literales de c_i es verdadero por τ (concretamente, el literal l_i').

    Por tanto, $\tau(\varphi) = 1$; es decir, la fórmula φ es satisfactible.

■

El problema VC del recubrimiento de vértices

- ⋆ **Dado un grafo no dirigido, G, y un número natural, k, determinar si existe un recubrimiento de vértices de G de tamaño k.**

Proposición 17: VC $\in$ NP.

Demostración: Un algoritmo determinista, $\mathcal{A}$, de fuerza bruta que resuelve VC es:

```
Entrada: G = ({x_1, ..., x_n}, E) un grafo no dirigido y k ∈ ℕ, k ≤ n
    para cada subconjunto V_1 de {x_1, ..., x_n} hacer
      si |V_1| = k y V_1 es un r.v. entonces
        devolver sí
    devolver no
```

El coste en tiempo de este algoritmo es del orden $O(2^n)$.

A partir del algoritmo determinista $\mathcal{A}$ diseñamos el siguiente algoritmo **no** determinista $\mathcal{B}$ de coste en tiempo polinomial que, además, resuelve el problema VC.

Entrada: $G = (\{x_1, \ldots, x_n\}, E)$ un grafo no dirigido y $k \in \mathbb{N}, k \leq n$

$V_1 \leftarrow \emptyset$

para cada $i = 1$ hasta n hacer
  Elegir e_1, e_2
    $e_1 : V_1 \leftarrow V_1$
    $e_2 : V_1 \leftarrow V_1 \cup \{x_i\}; k \leftarrow k - 1$

} Fase no determinista

si $k = 0$ y V_1 es un **r.v.** de G entonces
  devolver sí
si no
  devolver no

} Fase determinista

Veamos que este algoritmo **no** determinista es de coste en tiempo polinomial y, además, resuelve el problema **VC**.

El algoritmo no determinista $\mathcal{B}$ es de **coste en tiempo polinomial**:

- El tiempo de ejecución de la fase no determinista es del orden exacto de n.
- El tiempo de ejecución de la fase determinista es del orden $O(n^2)$.

Luego, el coste en tiempo del algoritmo no determinista $\mathcal{B}$ es del orden $O(n^2)$.

Veamos, ahora, que el algoritmo no determinista $\mathcal{B}$ **resuelve** el problema **VC**.

- Supongamos que V_1 es un **r.v.** de G de tamaño k. Entonces se considera la computación $\mathcal{C}$ del algoritmo $\mathcal{B}$ que, en la fase no determinista, selecciona el conjunto V_1. Entonces, la computación $\mathcal{C}$ es, obviamente, de aceptación.
- Supongamos ahora que la ejecución del algoritmo $\mathcal{B}$ con entrada (G, k) contiene alguna computación $\mathcal{C}'$ que es de aceptación.

  Sea V_1' el conjunto seleccionado en la fase no determinista de $\mathcal{C}'$. Puesto que $\mathcal{C}'$ es de aceptación, ha de verificarse que V_1' es un **r.v.** de G de tamaño k.

■

Teorema 6: El problema **VC** es **NP**–completo.

Demostración: Basta tener presente que **CLIQUE** $\leqslant^p$ **VC**.

■

El problema 3–COL

Sea $G = (V, E)$ un un grafo no dirigido y $k \geq 1$.

- ⋆ Una **coloración** de G **con** k **colores** es una aplicación, f, de V en $\{1, \ldots, k\}$. La imagen $f(v)$ de $v \in V$ es el **color asignado** al nodo v por la coloración f.

- ⋆ Diremos que una **coloración**, f, de G con k colores es **válida** si para cada $\{u, v\} \in E$, se tiene que $f(u) \neq f(v)$.

- ⋆ Diremos que G es **coloreable con k colores** si existe, al menos, una **coloración válida** de G con k colores.

Problema 3–COL: **Dado un grafo no dirigido, determinar si es coloreable con 3 colores**.

A continuación, vamos a demostrar que el problema **3–COL** es **NP**-completo.

Para ello, probaremos: (a) **3–COL** $\in$ NP; y (b) **3–SAT** $\leq^p$ **3–COL**.

NP-completitud del problema 3–COL

Proposición 18: 3–COL $\in$ NP.

Demostración: Consideremos el siguiente algoritmo $\mathcal{A}$ no determinista:

Entrada: $G = (V, E)$ un grafo no dirigido con $V = \{x_1, \ldots, x_n\}$.

$f \leftarrow \emptyset$

para $i = 1$ hasta n hacer
  Elegir e_1, e_2, e_3
    $e_1 : f \leftarrow f \cup \{(x_i, 1)\}$
    $e_2 : f \leftarrow f \cup \{(x_i, 2)\}$
    $e_3 : f \leftarrow f \cup \{(x_i, 3)\}$

} Fase no determinista

para cada $\{u, w\} \in E$ hacer
  si $f(u) = f(w)$ entonces
    devolver **no**

devolver **sí**

} Fase determinista

Hallemos el coste en tiempo del algoritmo $\mathcal{A}$.

- Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
- Tiempo de la fase determinista: cuadrático en el número de nodos del grafo.

El coste en tiempo del algoritmo $\mathcal{A}$ es cuadrático en el número de nodos del grafo de entrada; se decir, lineal en el tamaño de dicho grafo.

Veamos que $\mathcal{A}$ resuelve el problema **3-COL**. Para ello, sea G un grafo no dirigido:

- Supongamos que G es coloreable con 3 colores. Sea f una coloración válida de G con 3 colores y consideremos la computación $\mathcal{C}$ que en la fase no determinista selecciona, precisamente, la coloración f. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
- Supongamos que G **no** es coloreable con 3 colores. Sea $\mathcal{C}'$ cualquier computación y notemos f' la coloración de G con 3 colores seleccionada en la fase no determinista. Entonces f' **no** será una coloración válida y, por tanto, la computación $\mathcal{C}'$ **no** será de aceptación.

En consecuencia, el problema **3–COL** pertenece a la clase **NP**.

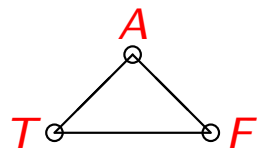
■

Teorema 7: El problema **3–COL** es **NP**–completo.

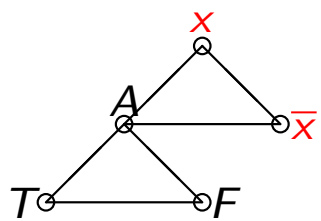
Demostración: Veamose que **3–SAT** $\leq^p$ **3–COL**.

Para cada $\varphi \in E_{3-\mathrm{SAT}}$ se construye un grafo no dirigido G_φ como sigue:

* **Triángulo base** (vértices T, F y A):

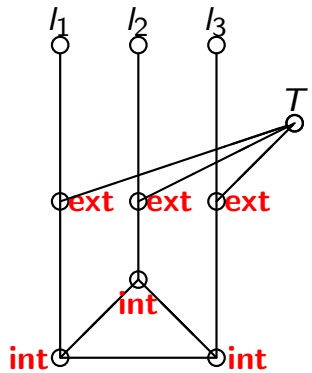


* **Cada variable** x de φ proporciona dos nuevos vértices (x y $\overline{x}$):



■

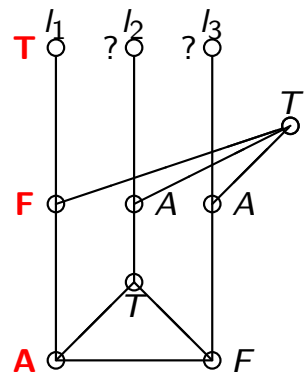
* **Cada cláusula** $c = l_1 + l_2 + l_3$ de φ proporciona seis nuevos vértices.



Vértices añadidos: **externos** e **internos**.

Se considera la aplicación $F : E_{3-\mathrm{SAT}} \to E_{3-\mathrm{COL}}$ definida por $F(\varphi) = G_\varphi$.

* La aplicación F es total computable en tiempo polinomial.
  - Si φ consta de n variables y p cláusulas, entonces el grafo G_φ posee $3 + 2n + 6p$ nodos y $3 + 3n + 12p$ aristas.
* Veamos que φ es satisfacible si y sólo si G_φ es coloreable con 3 colores.
  - Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Consideremos en G_φ la siguiente coloración:
    * Triángulo básico: *colores* T, F y A.
    * Vértice asociado a x: color T si $\sigma(x) = 1$ y color F si $\sigma(x) = 0$.
    * Si $c = l_1 + l_2 + l_3$ es cláusula de φ y $\sigma(l_1) = 1$, entonces, el vértice externo adyacente a l_1 lo coloreamos con F, los restantes externos con A y los internos como "*correspondan*".



La coloración así definida es un coloreado válido de G.

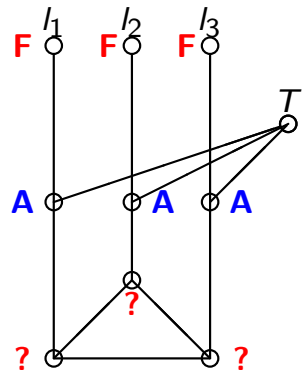
- Supongamos ahora que G_φ es coloreable con 3 colores (T, F y A).

  ⋆ En tal situación, las variables de φ sólo pueden estar coloreadas con T o con F (y de forma *compatible*).

  ⋆ Consideramos la valoración σ definida como sigue:
  - $\sigma(x) = 1$ si x es una variable coloreada con T.
  - $\sigma(x) = 0$ si x es una variable coloreada con F.

  Veamos que esa valoración σ verifica que $\sigma(\varphi) = 1$.

  En efecto: si $\sigma(\varphi) = 0$ entonces existiría una cláusula $c = l_1 + l_2 + l_3$ de φ tal que $\sigma(l_1) = \sigma(l_2) = \sigma(l_3) = 0$. Pues bien, en ese caso no sería posible colorear válidamente los nodos asociados a esa cláusula ya que los nodos externos tendrían que ser coloreados con A y, entonces, sería imposible colorear los tres nodos internos (ya que forman un triángulo).



■

El problema HAM–PATH

Dado un grafo G (dirigido o no dirigido) y dos nodos x, y de G, un **camino hamiltoniano** de x a y en G es un camino **simple** de x a y que pasa por todos los nodos de G.

Un **ciclo hamiltoniano** de G es un ciclo **simple** de G que pasa por todos los nodos de G.

Versión dirigida del camino hamiltoniano con dos nodos distinguidos:

> **Dado un grafo dirigido, G, y dos nodos distinguidos x, y, determinar si existe un camino hamiltoniano desde x hasta y.**

Proposición 17: **HAM–PATH** $\in$ NP.

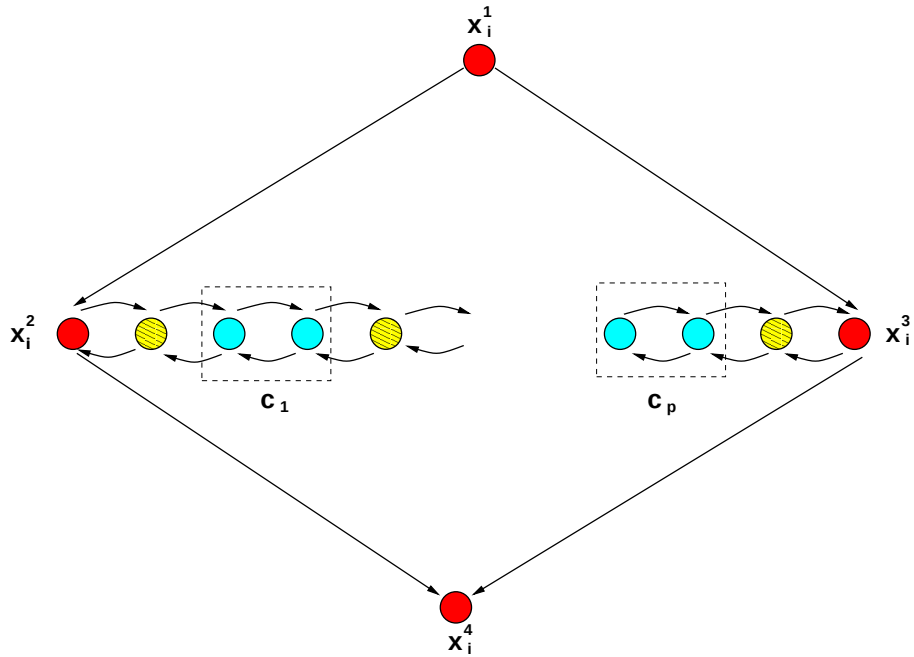
Se propone como **ejercicio voluntario**.

Teorema 2: El problema **HAM–PATH** es **NP**–completo.

Demostración: Veamos que **3–SAT** $\leq^p$ **HAM–PATH**.

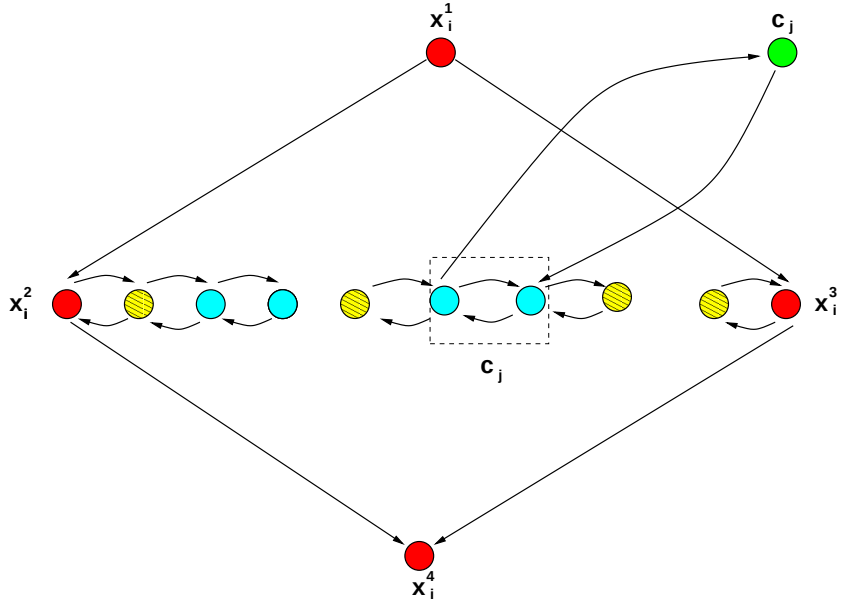
Para cada $\varphi = c_1 \cdot \ldots \cdot c_p \in E_{3-\mathrm{SAT}}$, con $\mathrm{Var}(\varphi) = \{x_1, \ldots, x_n\}$, se construye un grafo dirigido $G_\varphi = (V_\varphi, E_\varphi)$ como sigue:

- **Cada variable**, x_i, de φ proporciona el siguiente subgrafo de G_φ con estructura de *diamante*:

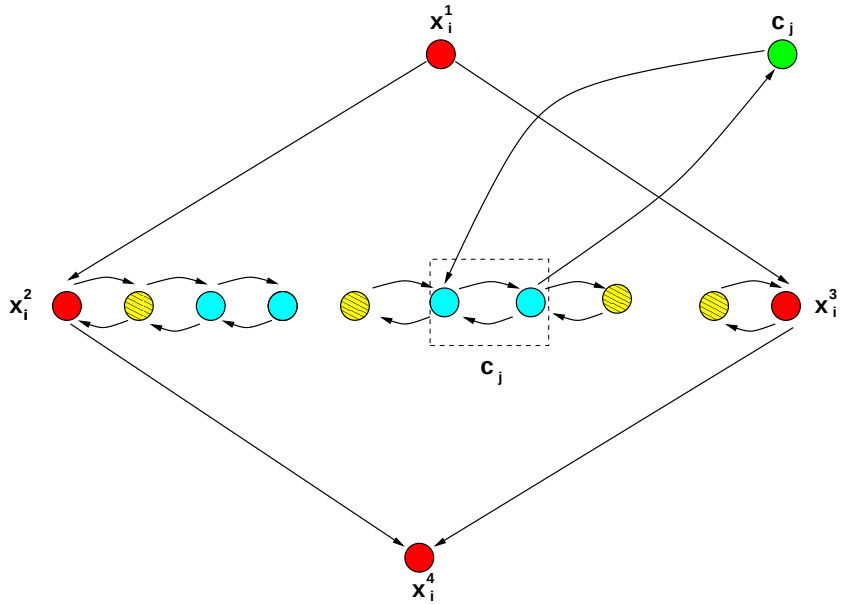


- Para cada $i, 1 \leqslant i < n$, $(x_i^4, x_{i+1}^1) \in E_\varphi$,

- **Cada cláusula**, c_j, proporciona un nodo conectado en el grafo como sigue:
  - Si x_i aparece en c_j, entonces



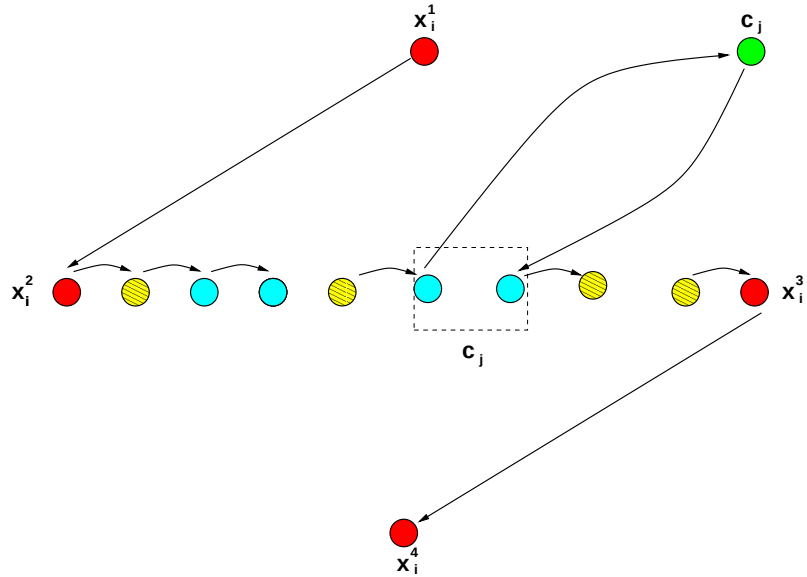
  - Si $\overline{x}_i$ aparece en c_j, entonces



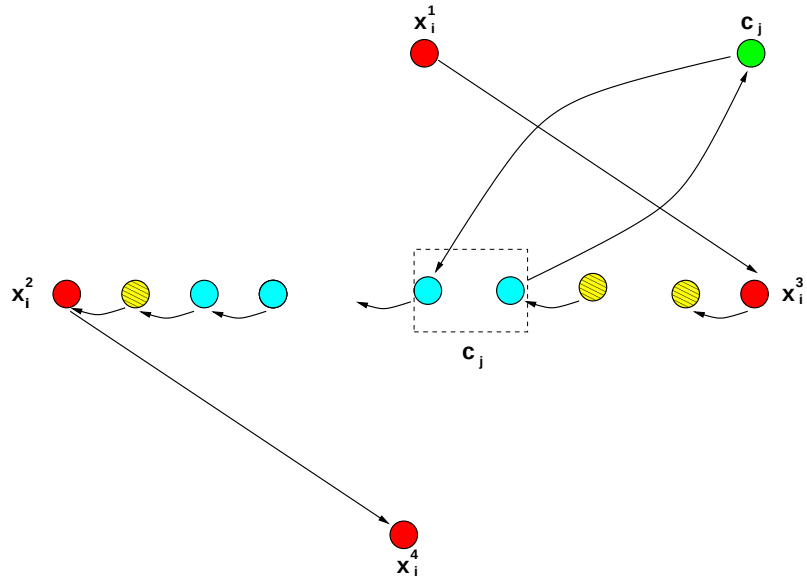
Sea $F : E_{3-SAT} \to E_{HAM-PATH}$ definida por $F(\varphi) = (G_\varphi, x_1^1, x_n^4)$.

* F es total computable en tiempo polinomial.
* **Aserto:** Para cada $\varphi \in E_{3-SAT}$, φ es satisfactible **sii** el grafo G_φ posee un camino hamiltoniano de x_1^1 a x_n^4.
  - Sean $\varphi = c_1 \ldots c_p$, con $\text{Var}(\varphi) = \{x_1, \ldots, x_n\}$ y σ una valoración tal que $\sigma(\varphi) = 1$. Se construye un camino hamiltoniano de x_1^1 a x_n^4 como sigue:
    - Si $\sigma(x_i) = 1$, entonces se recorre el diamante asociado **de izquierda a derecha**.
    - Si $\sigma(x_i) = 0$, entonces se recorre el diamante asociado **de derecha a izquierda**.

- Sea c_j una cláusula y $r_0 = \text{mín}\{r : \ \sigma(l_j^r) = 1\}$.
  Si $l_j^{r_0} = x_i$, entonces se hace el siguiente desvío:



  Si $l_j^{r_0} = \overline{x}_i$, entonces se hace el siguiente desvío:



- Recíprocamente, sea γ un camino hamiltoniano de x_1^1 a x_n^4. Vamos a construir una valoración σ tal que $\sigma(\varphi) = 1$.
  - ⋆ El camino γ debe ser **normal**: recorre cada diamante bien de izquierda a derecha, o de derecha a izquierda, con la posible salvedad del pequeño atajo para visitar una cláusula.
  - ⋆ Consideramos la valoración σ definida así:

$$\sigma(x_i) = \begin{cases} 1, & \text{si el diamante asociado a } x_i \text{ se recorre de izquierda a derecha} \\ 0, & \text{si el diamante asociado a } x_i \text{ se recorre de derecha a izquierda} \end{cases}$$

Se tiene que $\sigma(\varphi) = 1$.

■