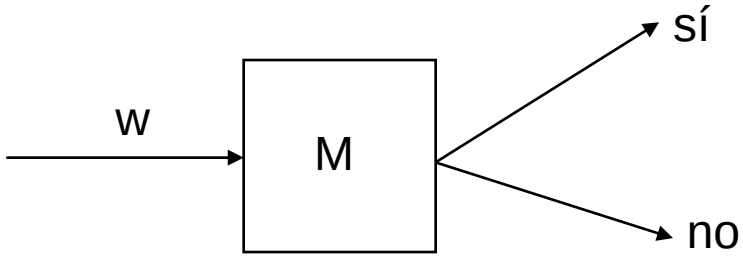
Clase teórica 6

Lenguajes NP-completos

El problema P vs NP

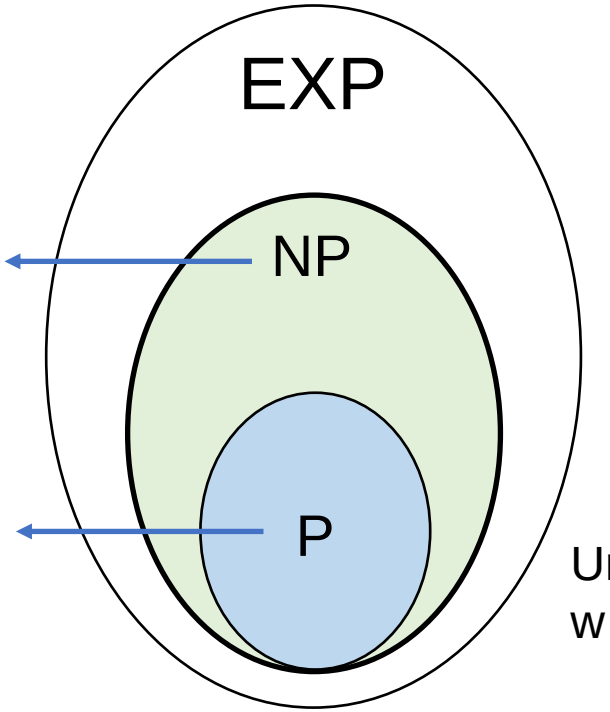
Un lenguaje L está en **P** sii existe una MT M tal que para todo w:
w ∈ L sii M acepta w en tiempo poly(n)



M tarda tiempo poly(|w|)

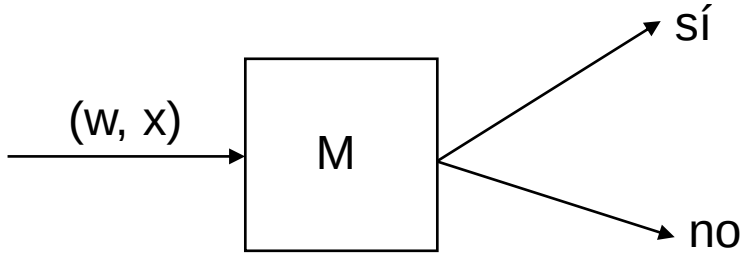
Lenguajes **verificables** en tiempo poly(n)

Lenguajes **decidibles** en tiempo poly(n)



Asumiendo P ≠ NP
Asumiendo NP ≠ EXP

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



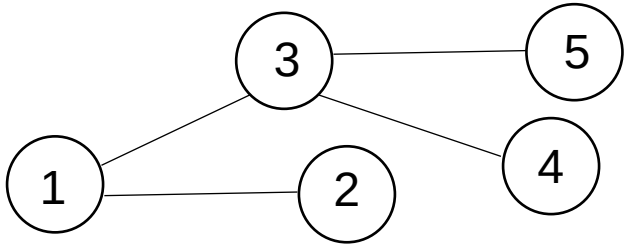
M tarda tiempo poly(|w|)
|x| ≤ poly(|w|)

- Por ejemplo, el lenguaje **ACCES** está en P, y el lenguaje **SAT** no estaría en P y está en NP.

- **Lenguaje ACCES**

ACCES = {G | G es un grafo con m vértices y tiene un camino del vértice 1 al vértice m}

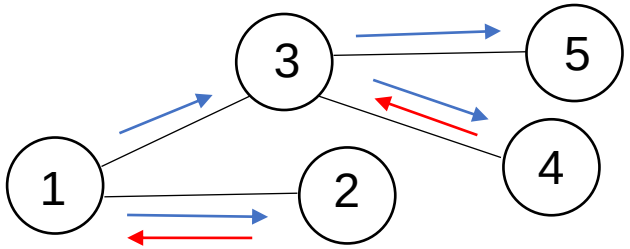
Representación de un grafo G



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

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

MT M que decide ACCES en tiempo poly(n)

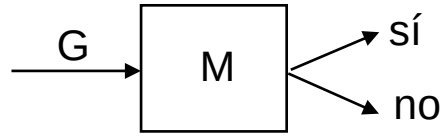


Recorrido DFS (en profundidad):

En el peor caso M recorre los arcos 2 veces

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

Por lo tanto, ACCES pertenece a P



M tarda tiempo O(|G|)

- **Lenguaje SAT**

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

P.ej.: $\varphi_2 = (x_1 \wedge x_2 \wedge x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

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

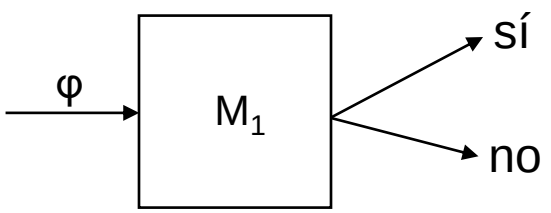
$(x_1 \wedge x_2 \wedge x_3)$			$\wedge$	$(\neg x_1 \vee \neg x_2 \vee \neg x_3)$		
V	V	V	**F**	V	V	V
V	V	F	**F**	V	V	F
V	F	V	**F**	V	F	V
V	F	F	**F**	V	F	F
F	V	V	**F**	F	V	V
F	V	F	**F**	F	V	F
F	F	V	**F**	F	F	V
F	F	F	**F**	F	F	F

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

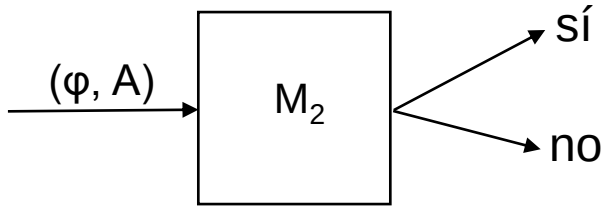
Cada evaluación se puede hacer en tiempo $O(|\varphi|^2)$, con el uso de una pila.

SAT no pertenecería a P (ver M_1)
M_1 ejecuta $O(2^m.|\varphi|^2) = O(2^n.n^2) = \exp(n)$ pasos.

SAT pertenece a NP (ver M_2)
M_2 ejecuta $O(|\varphi|^2) = O(n^2) = \text{poly}(n)$ pasos.



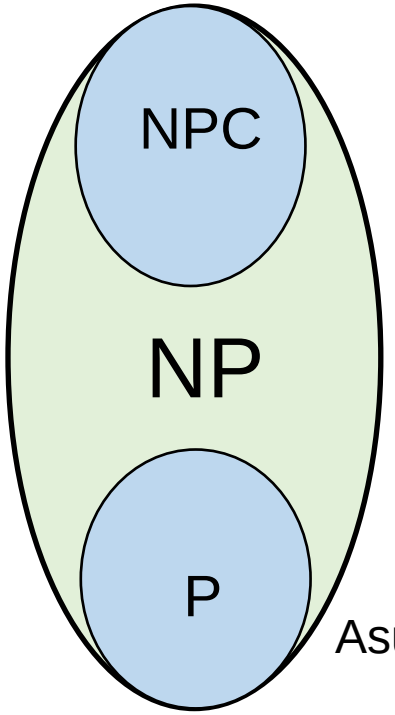
M tarda tiempo exp(|φ|)



M tarda tiempo poly(|φ|)

Lenguajes NP-completos

- Una visión más detallada de NP:



Asumiendo P ≠ NP

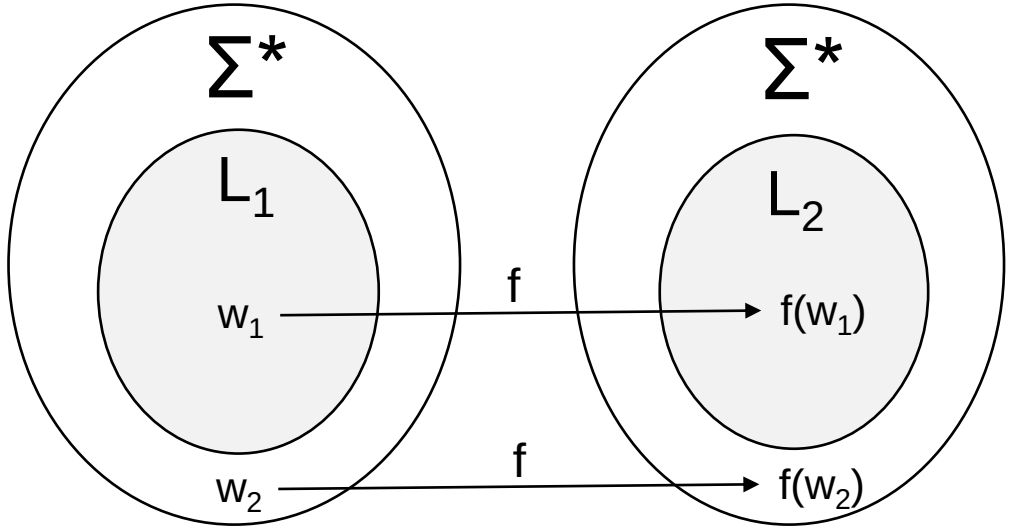
- **Buena noticia**: asumiendo la conjetura P ≠ NP, hay una manera de establecer que un lenguaje L de NP no está en P.

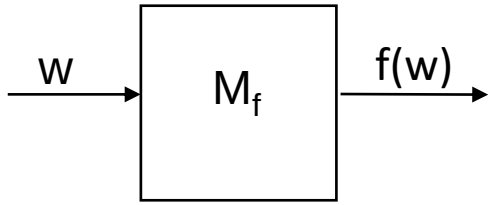
- Esto ocurre cuando se prueba que L es **NP-completo,** o que está en la clase **NPC.**

- Los lenguajes NP-completos son **los más difíciles** de la clase NP.

Reducciones polinomiales

- Para definir a los problemas NP-completos, tenemos que volver a utilizar **reducciones**, ahora **polinomiales**, es decir computables en tiempo poly(n):





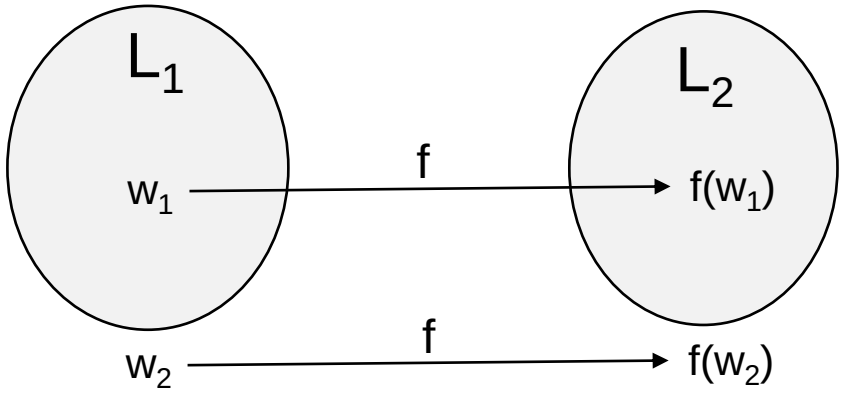
tiempo poly(|w|)

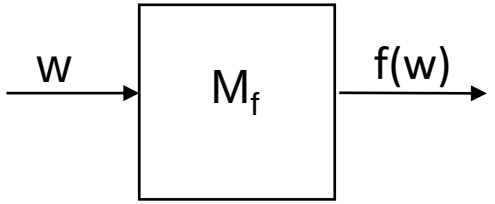
- La expresión $\mathbf{L_1 \leq_p L_2}$ establece que existe una reducción polinomial de L_1 a L_2.
- Se cumple, como en el caso general, que las reducciones polinomiales son **reflexivas, transitivas** y **no simétricas** (ejercicio).
- También como en el caso general, las reducciones polinomiales permiten **relacionar lenguajes.**

Reducciones polinomiales (continuación)

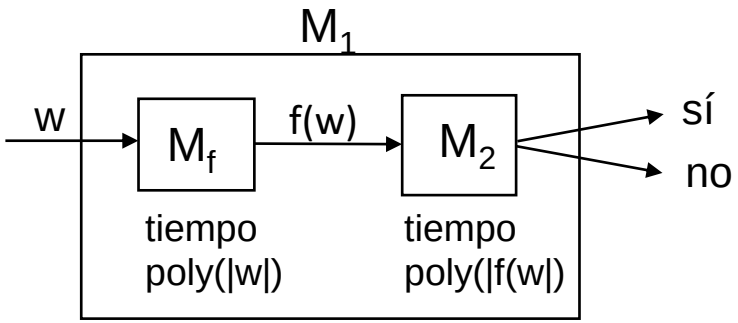
Teorema

(a) $L_1 \leq_p L_2$ y $L_2 \in P$: $L_1 \in P$
(b) $L_1 \leq_p L_2$ y $L_2 \in NP$: $L_1 \in NP$





Idea general de la prueba



(a) M_2 **decide** si $f(w) \in L_2$ y así M_1 **decide** si $w \in L_1$
(b) M_2 **verifica** si $f(w) \in L_2$ y así M_1 **verifica** si $w \in L_1$

Tiempo de M_1 = poly(|w|) + poly(|f(w)|)
Como |f(w)| = poly(|w|) ¿por qué?
entonces **tiempo de M_1 = poly(|w|) + poly(poly(|w|)) = poly(|w|)**

Corolario

(a´) $L_1 \leq_p L_2$ y $L_1 \notin P$: $L_2 \notin P$
(b´) $L_1 \leq_p L_2$ y $L_1 \notin NP$: $L_2 \notin NP$

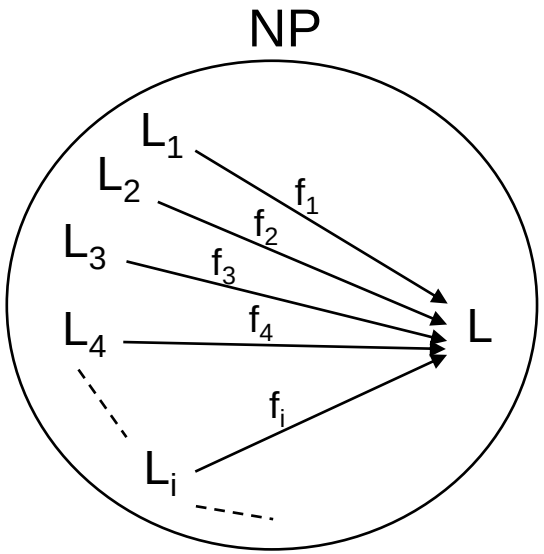
Si $L_1 \leq_p L_2$, L_2 es tan o más difícil que L_1

No puede ser que $L_1 \notin P$ y $L_2 \in P$
No puede ser que $L_1 \notin NP$ y $L_2 \in NP$

Definición de los lenguajes NP-completos

Un lenguaje L es **NP-completo**, o L ∈ NPC, sii:

a) **L ∈ NP**
b) Para todo L´ ∈ NP se cumple $L' \leq_p L$ (se dice que L es **NP-difícil**)



Todos los lenguajes de NP
se reducen polinomialmente a L

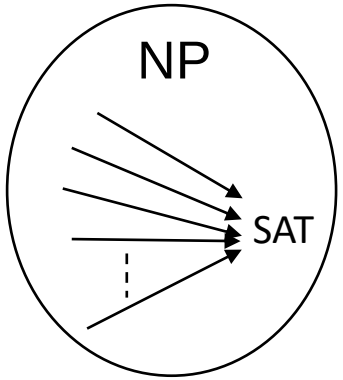
- Si L estuviera en P, entonces todos los lenguajes de NP estarían en P (¿por qué?)

  y de esta manera, la relación entre P y NP sería: **P = NP**

- Resumiendo: **los lenguajes NP-completos no están en P a menos que P = NP**

- Existen miles de lenguajes en la clase NPC.

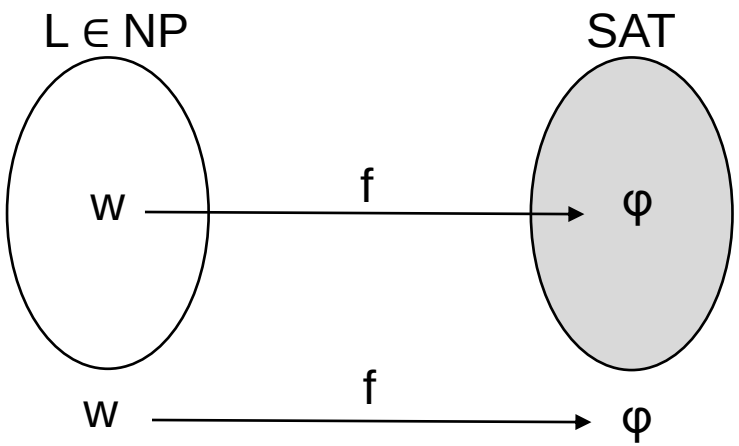
- Históricamente, Cook (EEUU) y Levin (Rusia), en 1971, encontraron casi en simultáneo un primer lenguaje NP-completo, **el lenguaje SAT.**



reducciones polinomiales

SAT = {φ | φ es una fórmula booleana sin cuantificadores y es satisfactible}

- La prueba es muy ingeniosa, similar a la que utilizó Turing en 1936 para probar que la lógica de predicados no es decidible:



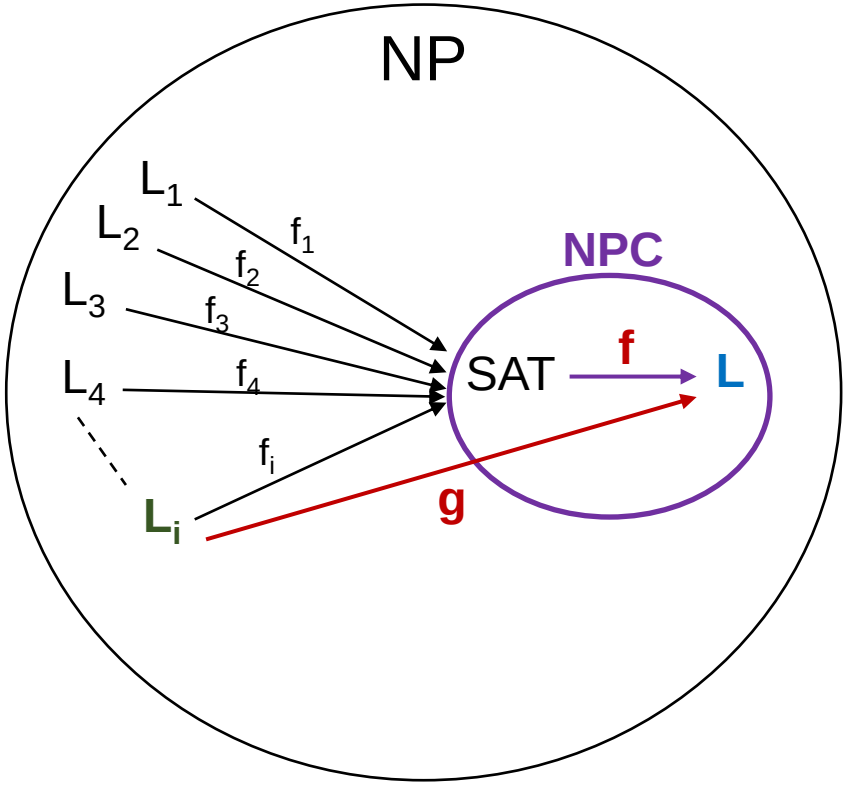
Dado cualquier L de NP:

f(w) obtiene en tiempo poly(|w|) una fórmula booleana φ que:

- Es satisfactible si w ∈ L
- No es satisfactible si w ∉ L

- Utilizando reducciones polinomiales a partir de SAT podemos **poblar la clase NPC:**

1. Sea **L** un lenguaje de NP
2. Sea **f** una reducción polinomial de SAT a L
3. Sea **g** la reducción polinomial obtenida componiendo una de las reducciones polinomiales f_i con **f**
3. Como lo anterior vale para todo L_i se cumple que **L es NP-completo**



Resumiendo:

$L_1 \in NPC$

$L_2 \in NP$

$L_1 \leq_p L_2$

$L_2 \in NPC$

Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos

SAT = {φ | φ es una fórmula booleana sin cuantificadores **satisfactible**}

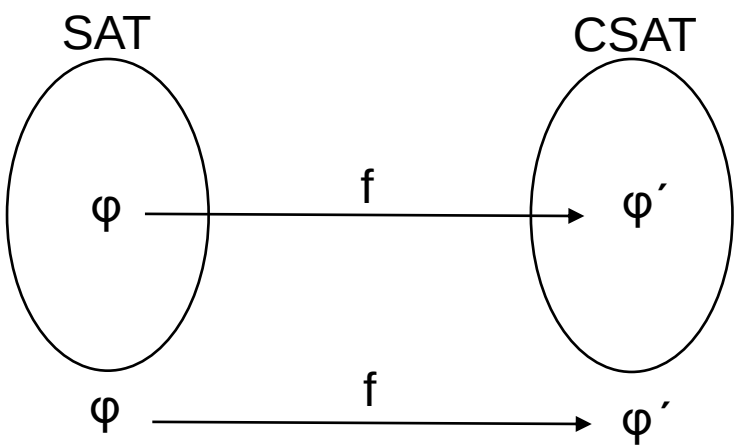
$$x_1 \vee (x_2 \wedge \neg x_3 \vee x_1) \wedge (x_3 \vee \neg x_1 \wedge x_3) \vee (x_5 \vee \neg x_1) \wedge \neg x_4$$

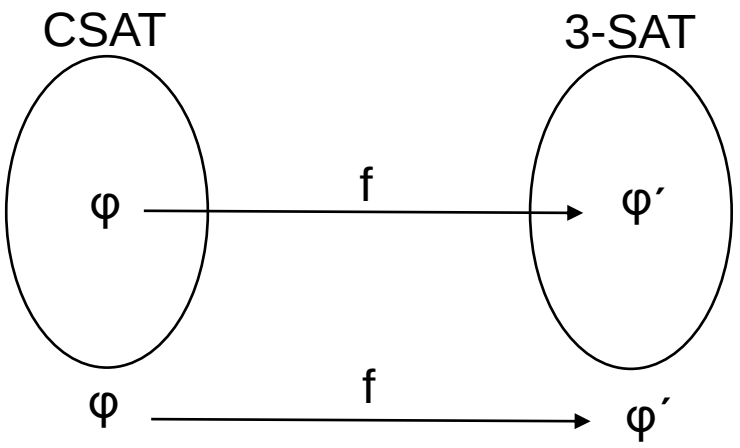
CSAT = {φ | φ es una fórmula booleana sin cuantificadores en la **forma normal conjuntiva (FNC) satisfactible**}

$$x_1 \wedge (x_1 \vee x_2 \vee x_5 \vee \neg x_3) \wedge (\neg x_5 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4 \vee \neg x_3 \vee x_3)$$

3-SAT = {φ | φ es una fórmula booleana sin cuantificadores en FNC **con tres literales por cláusula satisfactible**}

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_1)$$



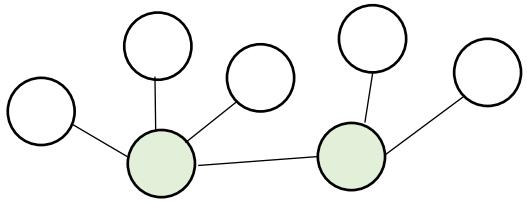


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

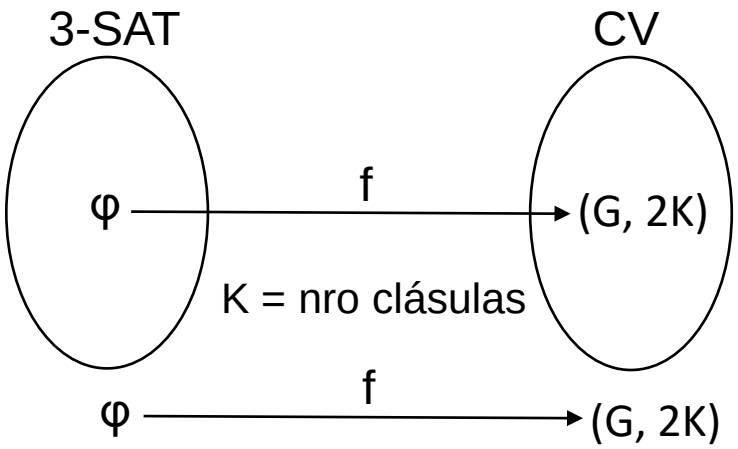
3-SAT = {φ | φ es una fórmula booleana sin cuantificadores en FNC **con tres literales por cláusula satisfactible**}

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_1)$$

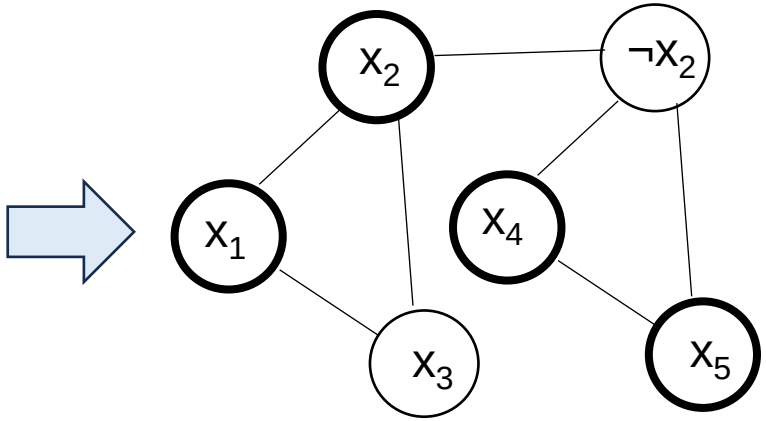
CV = {(G, K) | G es un grafo y tiene un **cubrimiento de vértices de tamaño K**}



Ejemplo de cubrimiento de vértices de tamaño 2 (con 2 vértices toca todos los arcos)

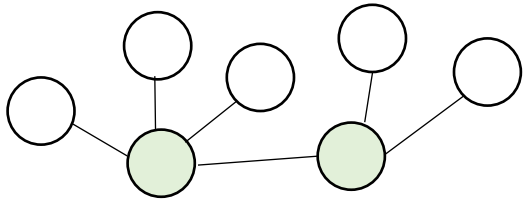


$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee \neg x_2)$$

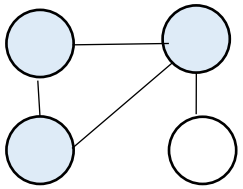


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

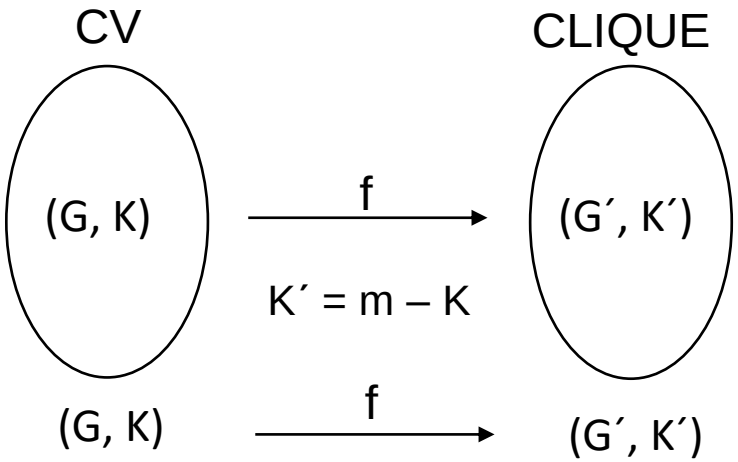
CV = {(G, K) | G es un grafo y tiene un **cubrimiento de vértices de tamaño K**}

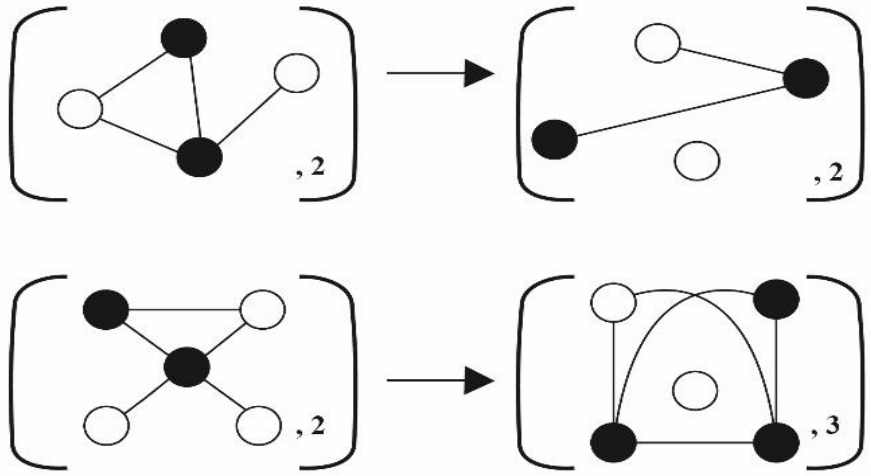


CLIQUE = {(G, K) | G es un grafo y tiene un **clique de tamaño K**}



Ejemplo de clique de tamaño 3



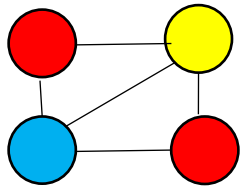


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

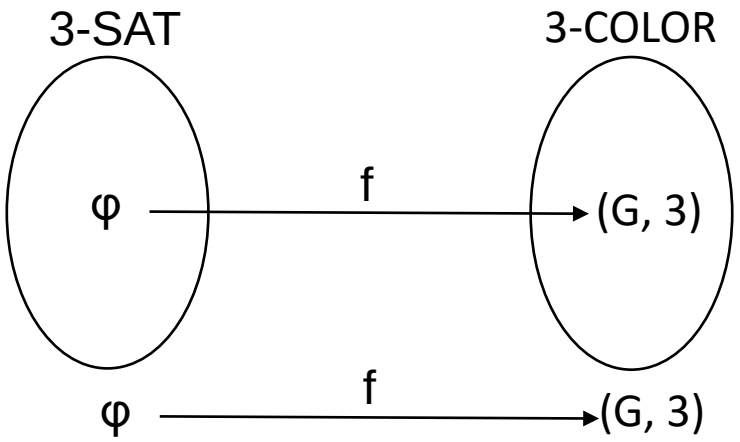
3-SAT = {φ | φ es una fórmula booleana sin cuantificadores en FNC **con tres literales por cláusula satisfactible**}

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_1)$$

K-COLOR = {(G, K) | G es un grafo y es coloreable con K colores sin producir vértices adyacentes con igual color}



Ejemplo de grafo coloreable con 3 colores



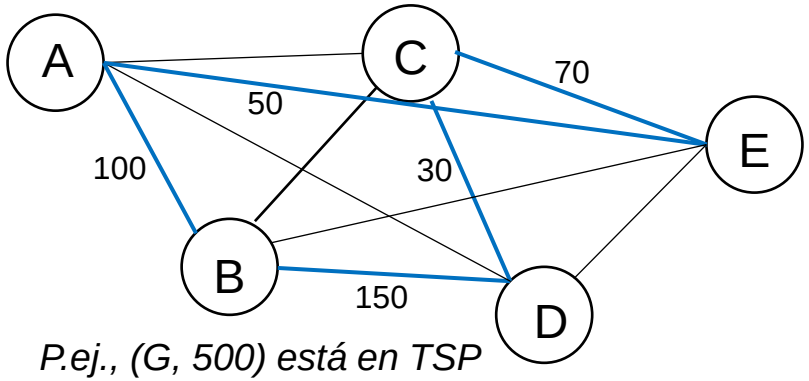
Teorema de los Cuatro Colores

Todo grafo planar se puede colorear con cuatro colores sin producir vértices adyacentes con el mismo color.

Reducción polinomial clásica

TSP = {(G, K) | G es un grafo completo con arcos con números y tiene un Circuito de Hamilton que mide ≤ K}

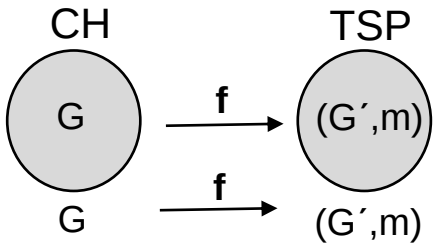
TSP (por *Travelling Salesman Problem*) representa el problema del viajante de comercio: un vendedor debe recorrer varias ciudades y volver a la inicial, de modo tal que en su recorrido no haga más que una distancia de K kilómetros.



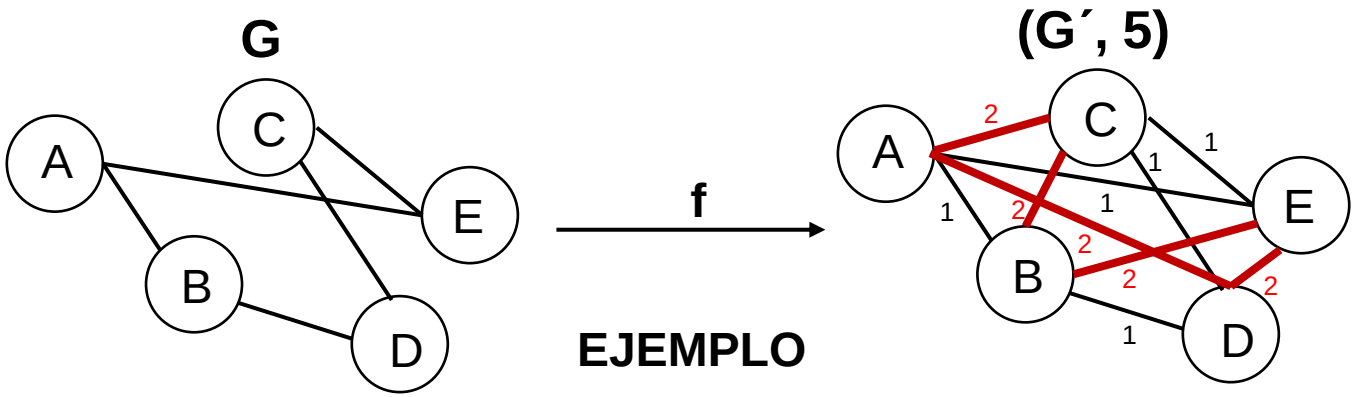
P.ej., (G, 500) está en TSP

Existe una reducción polinomial de CH a TSP

CH = {G | G tiene un Circuito de Hamilton}.



G tiene m vértices
G´ tiene m vértices y es completo (tiene todos los vértices conectados)
Veamos qué forma tiene G':



f se computa en tiempo poly(|G|) (ejercicio)

G ∈ HC sii (G´, m) ∈ TSP:
Si G tiene un CH, G' tiene un CH que mide: **m**
Si G no tiene un CH, todo CH mide al menos: **m + 1**

- Hay toda una heurística para poblar la clase NPC con reducciones polinomiales (un compendio insuperable es el libro de **Garey y Johnson de 1979**).

- Luego de que Cook y Levin demostraron que SAT es NP-completo, a partir de dicho problema **Karp en 1972 introdujo 21 problemas NP-completos**, que impulsó sobremanera esta área de la complejidad computacional.

- Levin llamó a los problemas NP-completos **problemas universales**. La idea subyacente es que todos los problemas NP-completos son un **único problema**, codificado en términos de grafos, o de la lógica, o de la aritmética, etc.

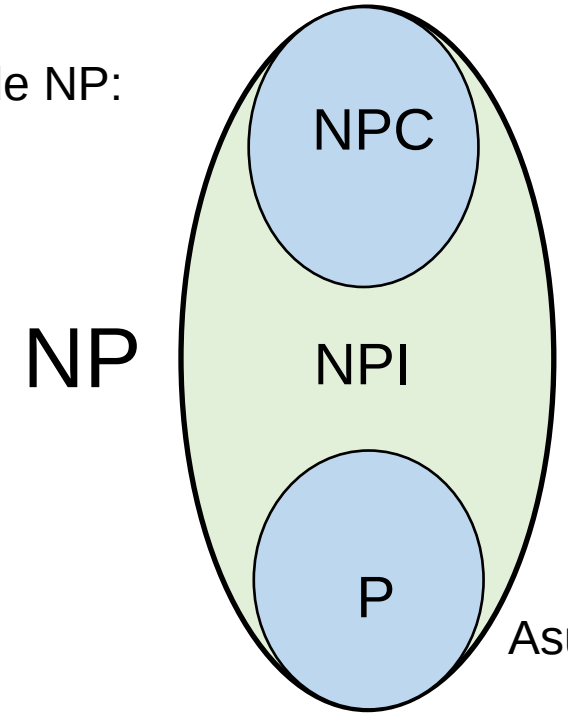
- **Hay un fenómeno curioso con el número 3**. Por ejemplo, problemas como 3-SAT y 3-COLOR son NP-completos. Sin embargo 2-SAT y 2-COLOR están en P. Esto ocurre con numerosos problemas.

- Curiosidades como la anterior ocurren con problemas a priori muy similares pero con comportamientos en cuanto a la complejidad temporal muy distintos. Pej., la **programación lineal** con soluciones reales es **tratable** mientras que con soluciones enteras **no**.

- El concepto de completitud se extiende a toda la jerarquía temporal. De hecho, se considera que una clase sin problemas completos identificados **no tiene mucha razón de existir**.

La clase NPI

- Una visión aún más detallada de NP:

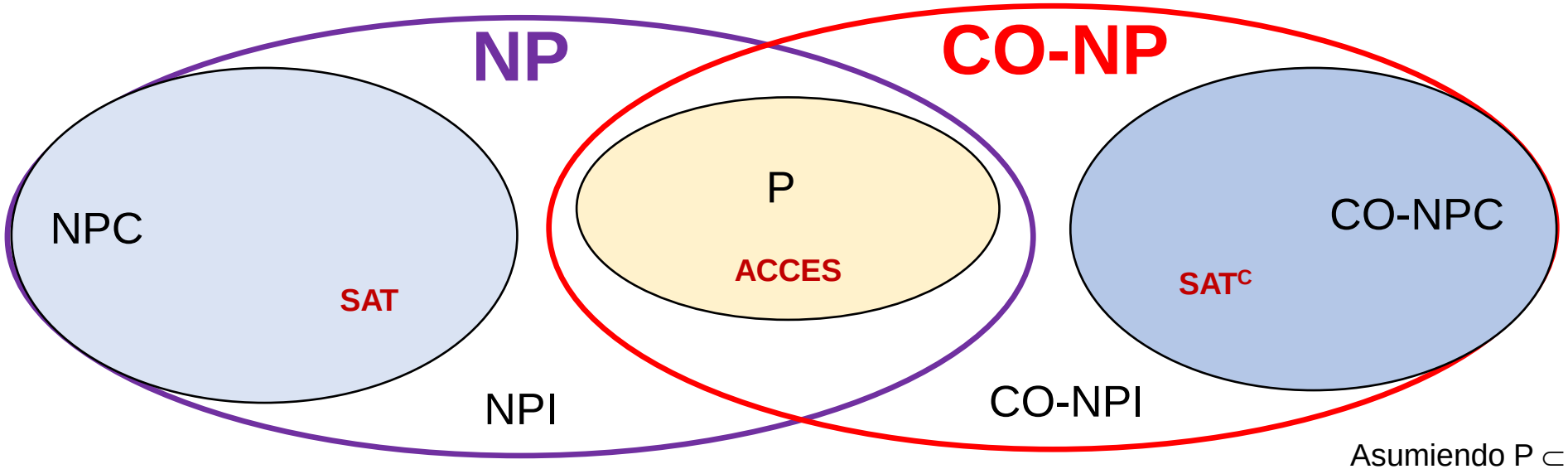


Asumiendo P ≠ NP

- Asumiendo P ≠ NP, se prueba que además de P y NPC, NP incluye una tercera clase de lenguajes: **NPI.**

- Los lenguajes de NPI **no son ni tan fáciles como los de P ni tan difíciles como los de NPC.**

P, NPC y NPI en la jerarquía temporal



Asumiendo $P \subset NP$
Asumiendo $P \subset NP \cap CO\text{-}NP$
Asumiendo $NP \neq CO\text{-}NP$

- Los lenguajes de CO-NP que son CO-NP-completos se definen como los lenguajes de NP que son NP-completos: todos los lenguajes de CO-NP se reducen polinomialmente a ellos.
- Los complementos de los lenguajes NP-completos son CO-NP-completos (ejercicio).
- La clase NPI también reúne lenguajes de mucho interés computacional. La veremos en la próxima clase.

Anexo de la clase teórica 6

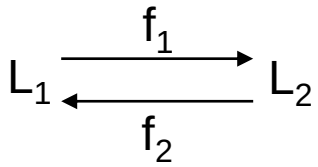
Lenguajes NP-completos

Los 21 problemas NP-completos de Karp

- CSAT (Problema de satisfactibilidad booleana con la forma normal conjuntiva)
  - INTEGER PROGRAMMING (Problema de la programación lineal entera)
  - CLIQUE (Problema del clique)
    - SET PACKING (Problema del empaquetamiento de conjuntos)
    - VERTEX COVER (Problema de la cobertura de vértices)
      - SET COVERING (Problema del conjunto de cobertura)
      - FEEDBACK NODE SET
      - FEEDBACK ARC SET
      - DIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano dirigido)
        - UNDIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano no dirigido)
  - 3-SAT (Problema de satisfactibilidad booleana con la forma normal conjuntiva con 3 literales por cláusula)
    - CHROMATIC NUMBER (Problema de la coloración de grafos)
      - CLIQUE COVER (Problema de la cobertura de cliques)
      - EXACT COVER (Problema de la cobertura exacta)
        - HITTING SET
        - STEINER TREE
        - 3-DIMENSIONAL MATCHING (Problema del matching tridimensional)
        - KNAPSACK (Problema de la mochila)
          - JOB SEQUENCING (Problema de las secuencias de trabajo)
          - PARTITION (Problema de la partición)
            - MAX-CUT (Problema del corte máximo)

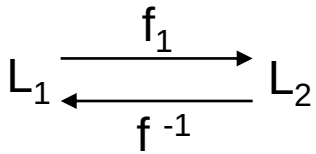
Dos características de los problemas NP-completos

- Por definición, para todo par de lenguajes L_1 y L_2 de NPC se cumple $\mathbf{L_1 \leq_p L_2}$ **y** $\mathbf{L_2 \leq_p L_1}$ (¿por qué?)



No necesariamente f_2 es la función inversa de f_1.

Sin embargo, **todos los lenguajes NP-completos conocidos cumplen dicha propiedad**



La **Conjetura de Berman-Hartmanis** plantea que todos los lenguajes NP-completos cumplen la propiedad.
Se prueba que si se cumple la conjetura, entonces **P ≠ NP.**

- **Todos los lenguajes NP-completos conocidos son densos**.

Un lenguaje es denso si para todo n tiene exp(n) cadenas de longitud a lo sumo n.
Un lenguaje es disperso en caso contrario (para todo n tiene poly(n) cadenas de longitud a lo sumo n).

Se prueba que si existe un lenguaje NP-completo disperso, entonces **P = NP.**

Clase práctica 6

Lenguajes NP-completos

Ejemplo 1

DSAT = {φ | φ es una fórmula booleana sin cuantificadores, en la forma normal disyuntiva o FND, satisfactible}

La forma FND consiste en disyunciones de cláusulas formadas por conjunciones de literales (variables o variables negadas), como por ejemplo:

$$(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_4 \wedge \neg x_4 \wedge x_5) \vee x_6 \vee (x_5 \wedge x_6)$$

Se cumple que **DSAT ∈ P**. Existe una MT M que acepta DSAT en tiempo polinomial:

Dada una fórmula φ, M hace:

1) Verifica la sintaxis de φ. Si la sintaxis es errónea, rechaza.
   Tiempo poly(n). Ejercicio.

2) Chequea si existe una cláusula de la disyunción que no tenga al mismo tiempo variables y variables negadas x_i y $\neg x_i$. Si existe una cláusula así, significa que φ es satisfactible, y acepta; en caso contrario, rechaza.
   Tiempo $O(n^2)$. Ejercicio.

Ejemplo 2

NO-DSAT = {φ | φ es una fórmula booleana sin cuantificadores, en la forma FND, y tiene una asignación que no la satisface}

No pareciera que NO-DSAT ∈ P:

Si φ tiene m variables, en el peor caso deben probarse 2^m asignaciones de valores de verdad, por lo tanto $O(2^n)$ asignaciones, con n = |φ|.
Tiempo exp(n).

Más aún, se prueba que **NO-DSAT ∈ NPC**:

1) Se prueba fácilmente que NO-DSAT ∈ NP (ejercicio).

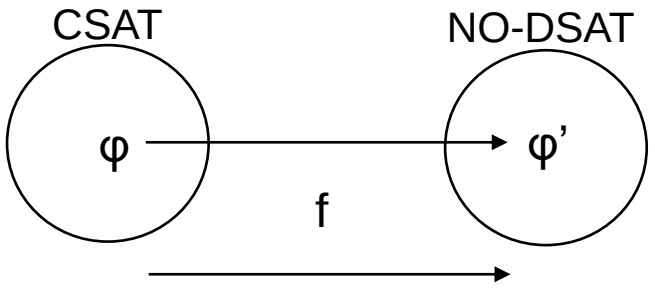
2) Y se cumple que todos los lenguajes de NP se reducen polinomialmente a NO-DSAT.

Esto lo vamos a probar a continuación, **encontrando una reducción polinomial de CSAT, que es un lenguaje NP-completo, a NO-DSAT:**

Sea la siguiente función de reducción f(φ) = φ´, tal que f niega la fórmula φ en base a las leyes de De Morgan para obtener la fórmula φ'.

Por ejemplo:

Si $\varphi = (x_1 \vee x_2) \wedge (x_4 \vee \neg x_4) \wedge (\neg x_3 \vee x_5 \vee x_6)$,
$\varphi' = (\neg x_1 \wedge \neg x_2) \vee (\neg x_4 \wedge x_4) \vee (x_3 \wedge \neg x_5 \wedge \neg x_6)$



Se cumple que:

f es una función computable en tiempo polinomial

M_f transforma φ en φ' negando φ de acuerdo a las leyes de De Morgan.
M_f trabaja en tiempo polinomial, transformar φ en φ' según lo especificado es lineal. Ejercicio.
(Si φ es una fórmula incorrecta sintácticamente, también lo es φ').

$\varphi \in$ CSAT $\leftrightarrow$ f(φ) = φ' $\in$ NODSAT

$\varphi \in$ CSAT sii
φ está en la forma FNC y existe una asignación A que la satisface sii
φ' está en la forma FND y existe una asignación A que no la satisface sii
$\varphi' \in$ NO-DSAT

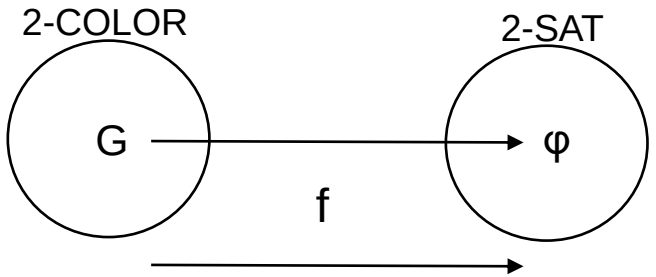
Ejemplo 3

Existe una reducción polinomial de 2-COLOR a 2-SAT, siendo:

2-COLOR = {G | G es un grafo tal que sus vértices se pueden colorear con 2 colores de manera tal que dos vértices vecinos no tengan el mismo color}

2-SAT = {φ | φ es una fórmula booleana sin cuantificadores satisfactible, en la forma FNC y con dos literales (variables o variables negadas) por cláusula}

Función de reducción f de 2-COLOR a 2-SAT:
A todo grafo válido G, la función f le asigna una fórmula booleana φ en la FNC con dos literales por cláusula, de modo tal que por cada arco (i, j) de G, f construye una subfórmula $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$.
A los grafos inválidos f le asigna una cadena inválida, por ejemplo “1”.



La función f se computa en tiempo polinomial:
La validación de la sintaxis de un grafo es polinomial (ejercicio). Escribir un “1” tarda tiempo constante. Y la generación de la fórmula booleana descripta tarda tiempo lineal (ejercicio).

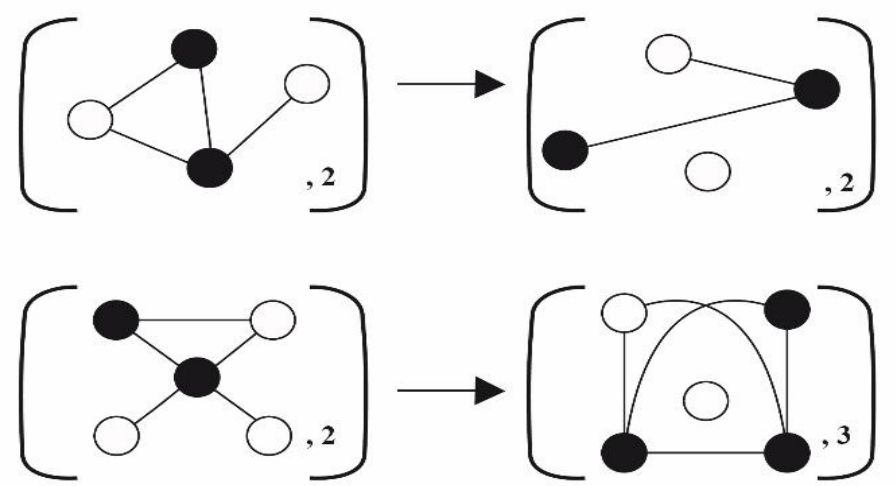
G ∈ 2-COLOR sii φ ∈ 2-SAT:
Asociando dos colores c_1 y c_2 con los valores de verdad verdadero y falso, respectivamente, claramente los vértices de todo arco de G tienen colores distintos si y sólo si la conjunción de las dos cláusulas que se construyen a partir de ellos es satisfactible.

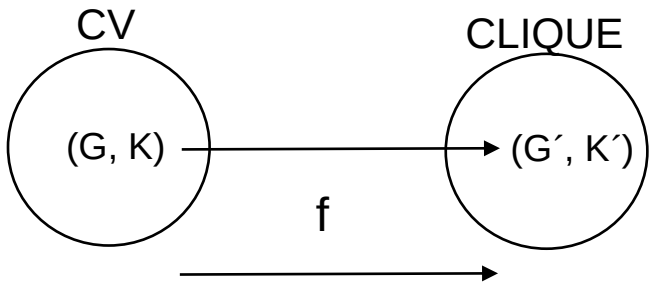
Ejemplo 4

Sea CLIQUE = {(G, K) | G es un grafo y tiene un clique - subgrafo completo - de tamaño K}

Se prueba que **CLIQUE es NP-completo.** CLIQUE ∈ NP (ejercicio). Falta probar que todos los lenguajes de NP se reducen a él. Encontraremos una reducción polinomial de CV a CLIQUE, siendo CV el lenguaje que representa el problema del cubrimiento de vértices:
CV = {(G, K) | G tiene un cubrimiento de K vértices, es decir que K vértices tocan a todos los arcos de G}. **Como CV es NP-completo, entonces también lo será CLIQUE.**

Función de reducción f de CV a CLIQUE. Dado un grafo válido G (si es inválido se asigna un "1"), con m vértices y un número natural K ≤ m, la función es: f((G, K)) = (G^C, m – K), siendo G^C el grafo "complemento" de G (tiene los mismos vértices que G y sólo los arcos que G no tiene). Abajo hay dos casos de aplicación de f:





f es una función computable en tiempo polinomial (ejercicio).

$(G, K) \in CV$ sii $(G^C, m - K) \in CLIQUE$.

Para mayor claridad, descomponemos la prueba en los dos sentidos:

Primero veremos que si $(G, K) \in CV$, entonces $(G^C, m - K) \in CLIQUE$.

Sea $(G, K) \in CV$, y V' un cubrimiento de vértices de G de tamaño K. Veamos que V – V' es un clique de G^C de tamaño m – K, y así que $(G^C, m - K) \in CLIQUE$. Por un lado, el conjunto de vértices V – V' mide m – K. Por otro lado, supongamos que G^C no es un clique, por ejemplo que no incluye un arco (i , h), siendo i y h vértices de V – V'. Entonces (i, h) es un arco de G, siendo i y h vértices que no están en V', por lo que V' no es un cubrimiento de vértices de G (absurdo).

Ahora veremos que si $(G^C, m - K) \in CLIQUE$, entonces $(G, K) \in CV$.

Sea $(G^C, m - K) \in CLIQUE$, y V' un clique de G^C de tamaño m – K. Veamos que V – V' es un cubrimiento de G de tamaño K, y así que $(G, K) \in CV$. Por un lado, el conjunto de vértices V – V' mide $m - (m - K) = K$. Por otro lado, supongamos que V – V' no es un cubrimiento de vértices de G, por ejemplo que existe un arco (i, h), con i y h vértices no pertenecientes a V – V' (y así pertenecientes a V´). Pero entonces V' no es un clique de G (absurdo).