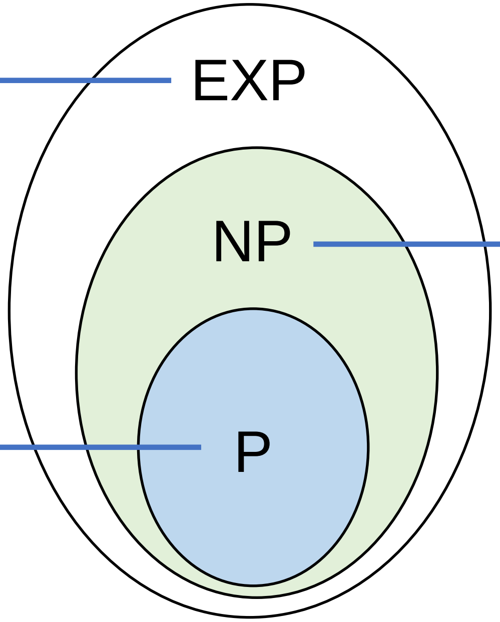
Clase teórica 6

NP-completitud

Versión de la jerarquía temporal a la que llegamos

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

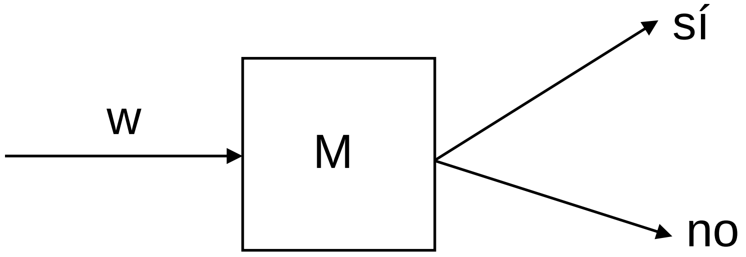


Lenguajes **verificables**
en tiempo poly(n)

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

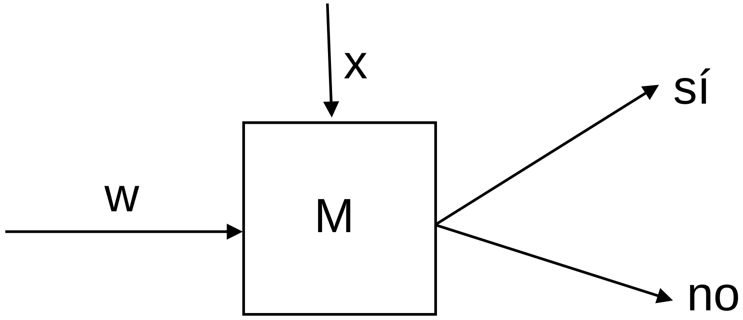
Asumiendo P ≠ NP
Asumiendo NP ≠ EXP
Se cumple P ⊂ EXP

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



<u>M tarda tiempo poly(|w|)</u>

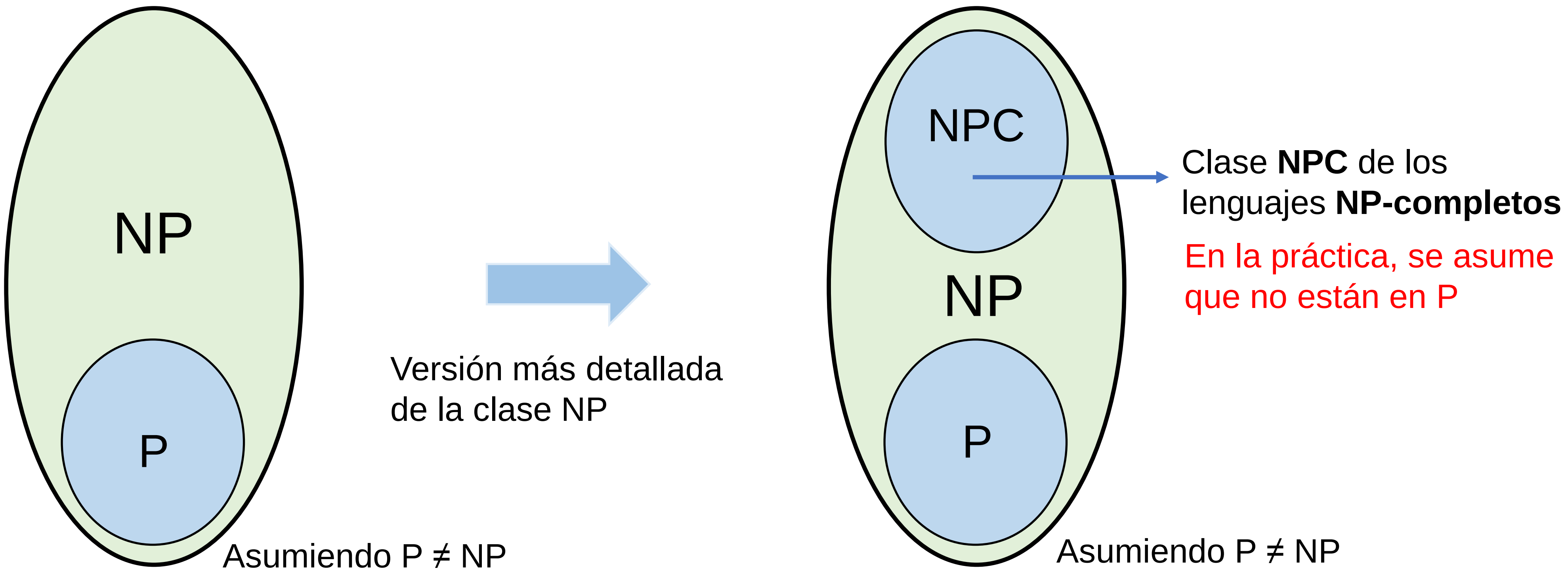
L está en NP sii existe una MT M que para todo w:
w ∈ L sii existe x tal que M acepta (w, x) en tiempo poly(n)



<u>M tarda tiempo poly(|w|)</u>

Lenguajes NP-completos

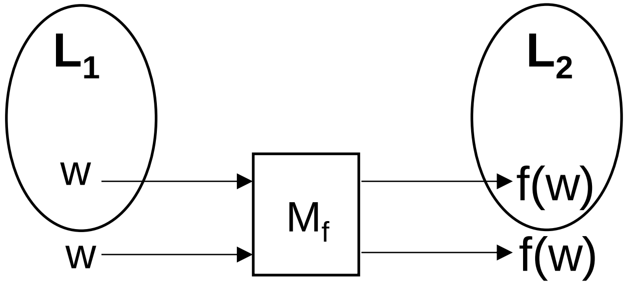
Una manera de **reforzar la sospecha** de que un lenguaje de NP no está en P es probando que es **NP-completo**.



- Se prueba que si un lenguaje NP-completo está en P, se cumple P = NP.
- Por lo tanto, **si se cumple P ≠ NP, un lenguaje NP-completo no está en P.**

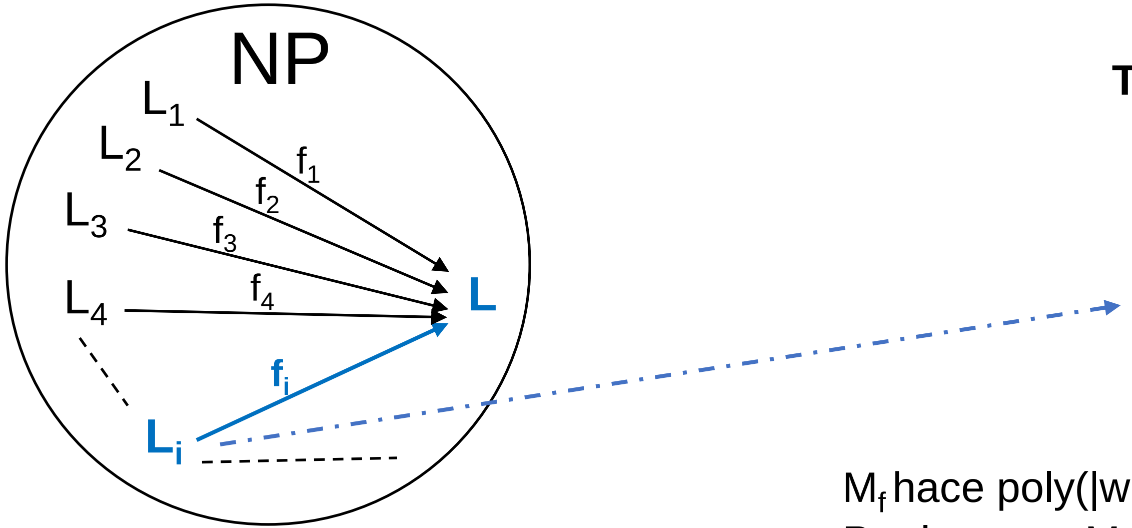
Definición y utilidad de la NP-completitud

- Una **reducción polinomial** de un lenguaje L_1 a un lenguaje L_2 es una reducción de L_1 a L_2 de tiempo polinomial ($\mathbf{L_1 \leq_p L_2}$)



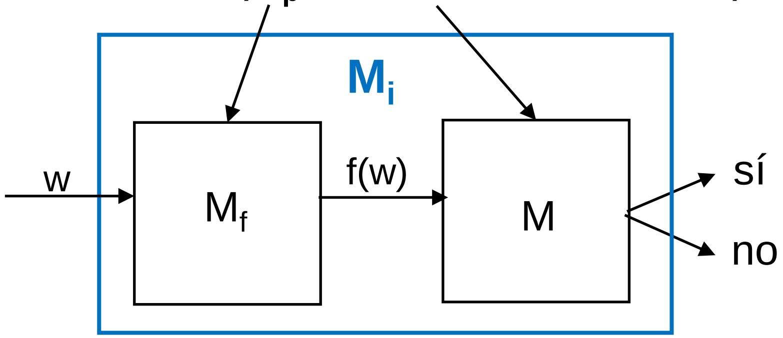
M_f hace poly(|w|) pasos

- Un lenguaje L es **NP-completo**, o **L ∈ NPC**, sii:
  1. **L ∈ NP**
  2. Todo $L_i \in$ NP cumple $L_i \leq_p L$ (se dice que **L es NP-difícil**)



toda f_i es una reducción polinomial

Teorema: si $L_i \leq_p L$ y $L \in P$, entonces $L_i \in P$



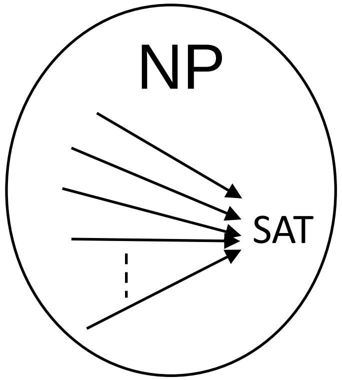
M_f hace poly(|w|) pasos y M hace poly(|f(w)| pasos, con |f(w)| ≤ poly(|w|)
Por lo tanto, M_i decide L_i en poly(|w|) + poly(poly(|w|)) = poly(|w|) pasos

Si L está en P, todo L_i de NP está en P, es decir: **NP = P**. Así, **si P ≠ NP, entonces L no está en P.**

Primer lenguaje NP-completo encontrado: SAT

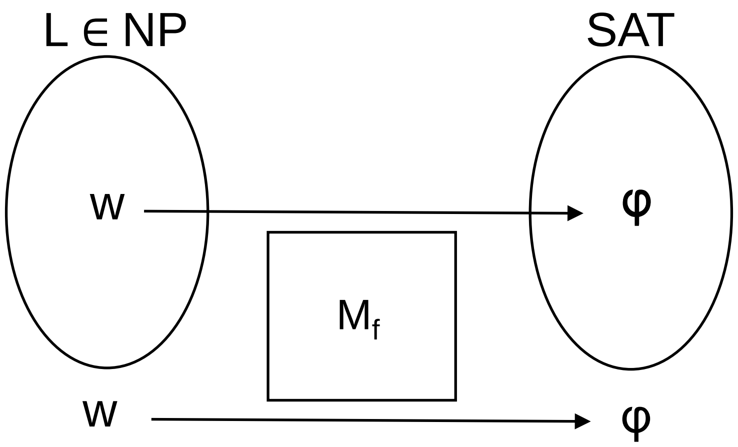
- S. Cook (EEUU) y L. Levin (URSS), en 1971, encontraron casi en simultáneo un primer lenguaje NP-completo, **el lenguaje SAT.**

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



Todos los lenguajes de NP se reducen polinomialmente al lenguaje SAT

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



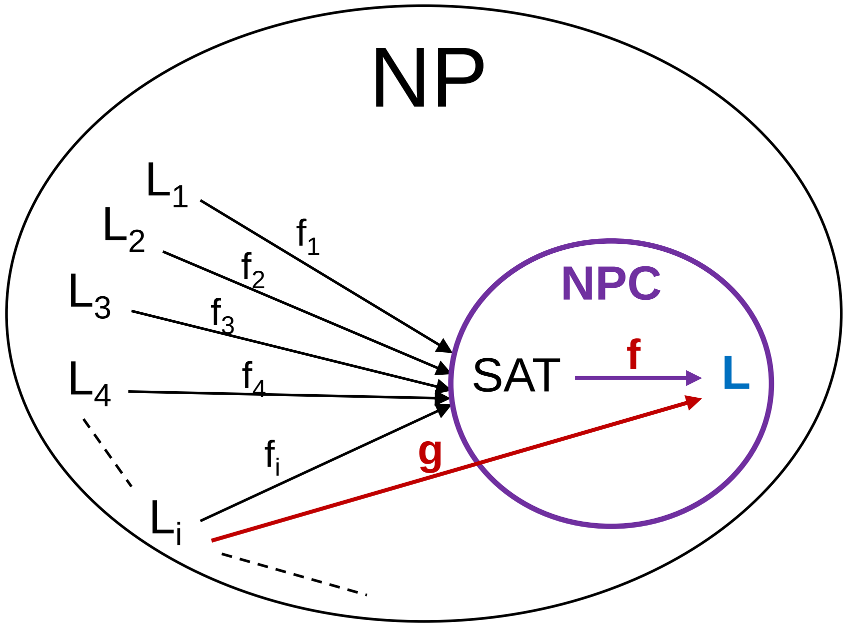
Dado cualquier L de NP:

La MT M_f asigna a toda cadena w, en tiempo poly(|w|), una fórmula booleana φ tal que:

- Si w ∈ L, entonces φ es satisfactible.
- Si w ∉ L, entonces φ no es satisfactible.

- Idea general: si M es una MT asociada a L, φ expresa una computación de M a partir de w.

¿Cómo encontrar otro lenguaje L que sea NP-completo?



1. Probar que **L** pertenece a NP
2. Construir una reducción polinomial f de SAT a **L**

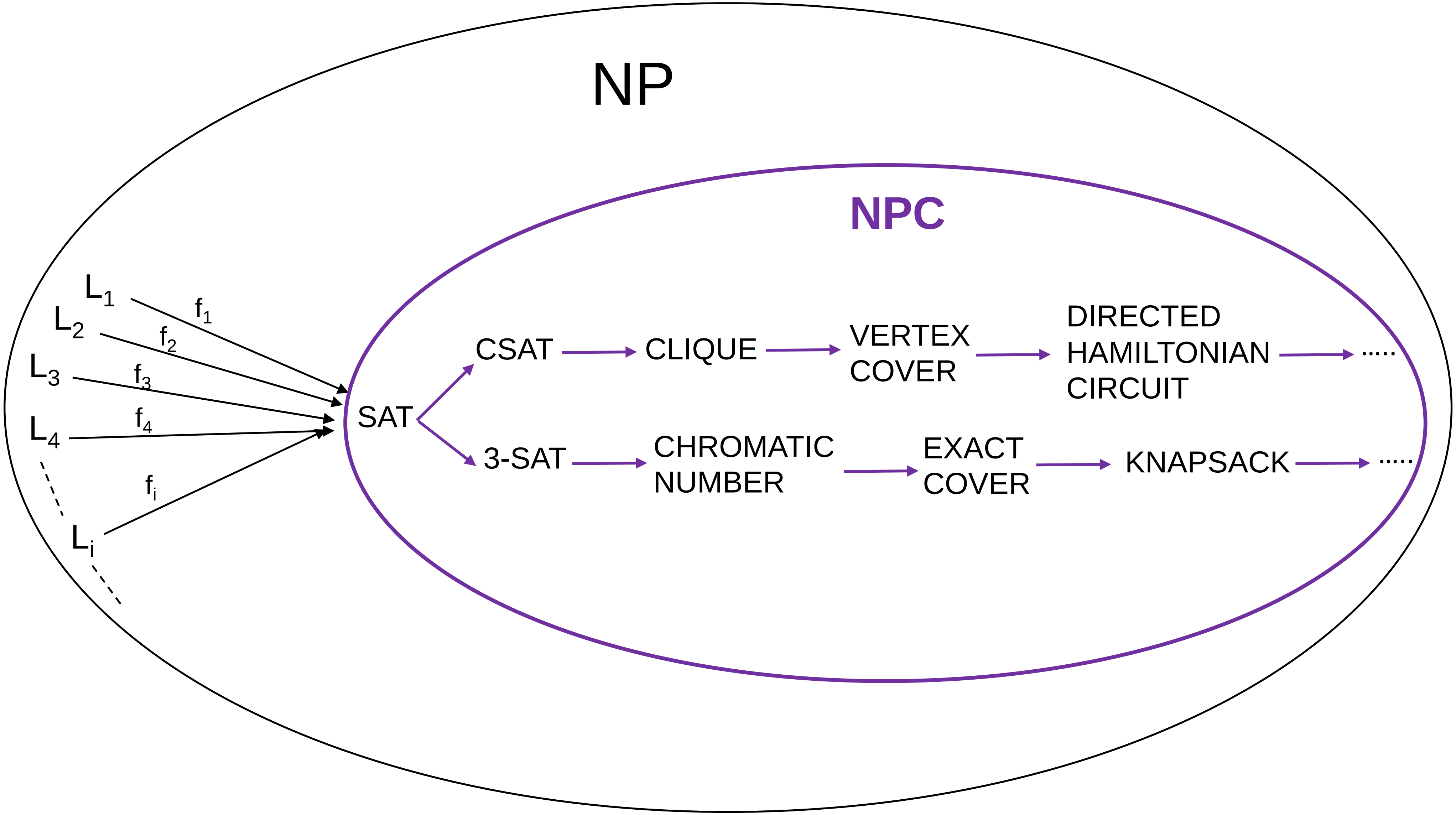
Se cumple que **L** es NP-completo:

- Sea **g** la composición de f_i con **f**
- La función **g** es una reducción polinomial de L_i a L (las reducciones polinomiales son transitivas)
- Como lo anterior vale para todo L_i de NP, entonces **L es NP-completo**

Repaso del mecanismo:

1) SAT $\in$ NPC 2) **L** $\in$ NP 3) SAT $\leq_p$ **L** $\longrightarrow$ 4) **L** $\in$ NPC

¿Cómo encontrar más lenguajes NP-completos?



<u>Mecanismo</u>:

1) $L \in NPC$
2) $L_1 \in NP$
3) $L \leq_p L_1$

---

4) $L_1 \in NPC$

21 lenguajes NP-completos encontrados a partir de SAT (R. Karp, 1972)

- CSAT: problema de satisfactibilidad de las fórmulas booleanas sin cuantificadores en la forma normal conjuntiva
  - INTEGER PROGRAMMING
  - CLIQUE: problema del clique
    - SET PACKING
    - VERTEX COVER
      - SET COVERING
      - FEEDBACK NODE SET
      - FEEDBACK ARC SET
      - DIRECTED HAMILTONIAN CIRCUIT: problema del circuito de Hamilton en grafos dirigidos
        - UNDIRECTED HAMILTONIAN CIRCUIT: problema del circuito de Hamilton en grafos no dirigidos
  - 3-SAT: problema CSAT con tres literales por cláusula
    - CHROMATIC NUMBER: problema de coloración de grafos
      - CLIQUE COVER
      - EXACT COVER
        - HITTING SET
        - STEINER TREE
        - 3-DIMENSIONAL MATCHING
        - KNAPSACK
          - JOB SEQUENCING
          - PARTITION
            - MAX-CUT

Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos

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

$$x_1 \lor (x_2 \land \neg x_3 \lor x_1) \land (x_3 \lor \neg x_1 \land x_3) \lor (x_5 \lor \neg x_1) \land \neg x_4$$

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

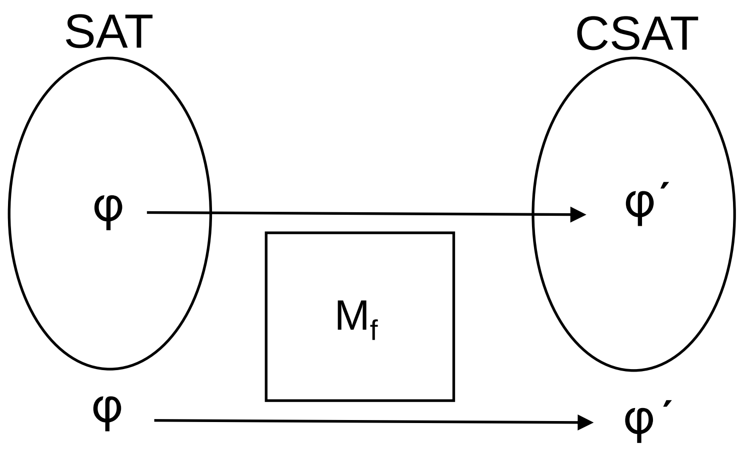
$$x_1 \land (x_1 \lor x_2 \lor x_5 \lor \neg x_3) \land (\neg x_5 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4 \lor \neg x_3 \lor x_3)$$

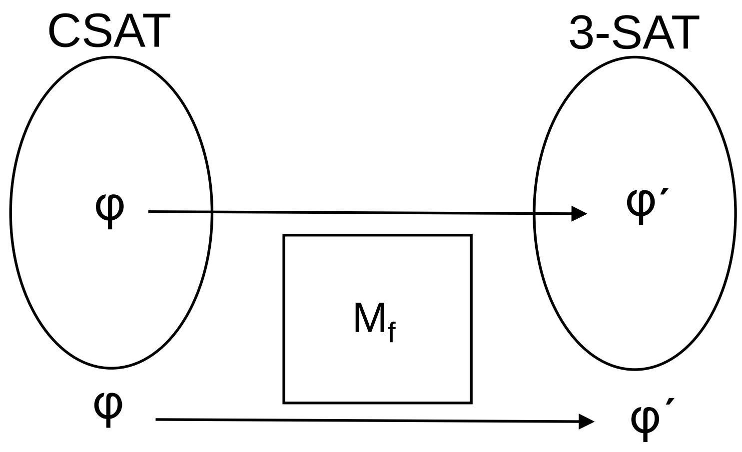
conjunciones de disyunciones de literales (variables o variables negadas), conocidas como **cláusulas**

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

$$(x_1 \lor \neg x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_4) \land (\neg x_2 \lor \neg x_3 \lor x_1)$$

cláusulas de tres literales



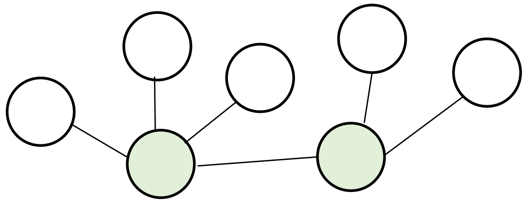


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

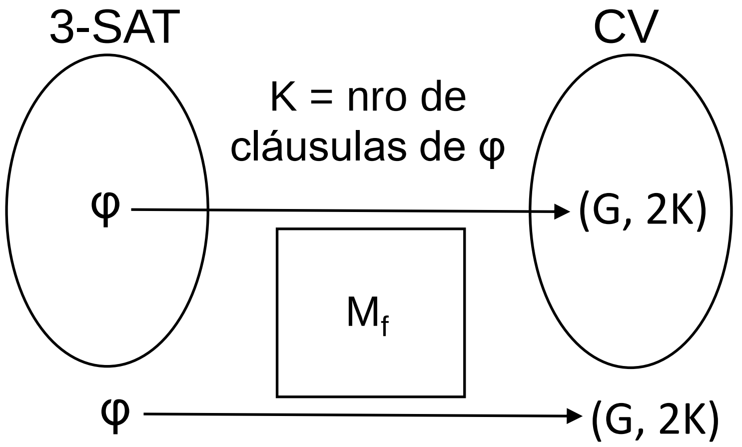
3-SAT = {φ | φ es una fórmula booleana sin cuantificadores en FNC con tres literales por cláusula y es satisfactible}

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

CV = {(G, K) | G es un grafo no dirigido y tiene un **cubrimiento de vértices** de tamaño K}



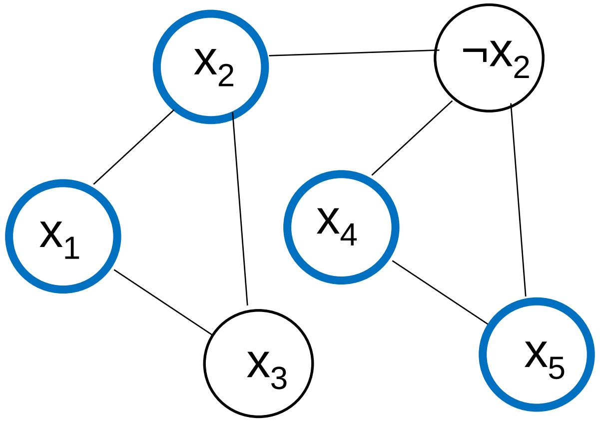
Ejemplo de cubrimiento de vértices de tamaño 2 (con 2 vértices se tocan todos los arcos)



Por ejemplo:

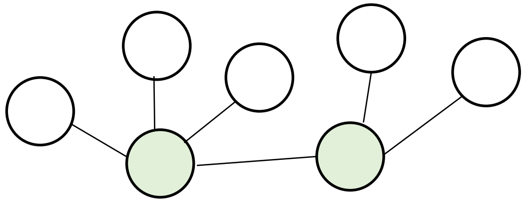
$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee \neg x_2)$$

Se genera un grafo con un cubrimiento de 4 vértices (φ tiene 2 clásulas)

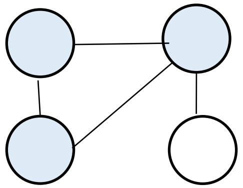


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

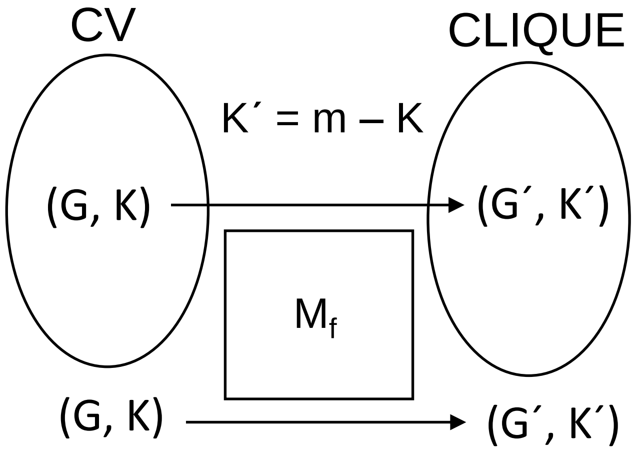
CV = {(G, K) | G es un grafo no dirigido de m vértices y tiene un cubrimiento de vértices de tamaño K}



CLIQUE = {(G, K) | G es un grafo no dirigido de m vértices y tiene un **clique** de tamaño K}



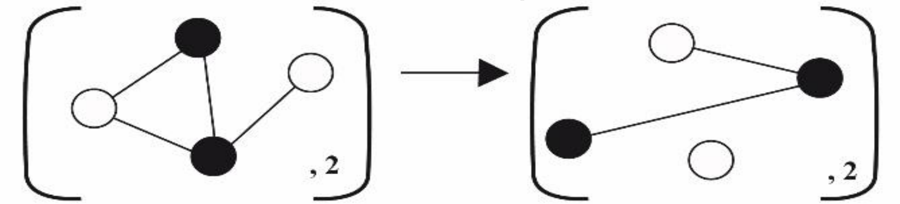
Ejemplo de clique de tamaño 3 (subgrafo completo de 3 vértices)



Por ejemplo:

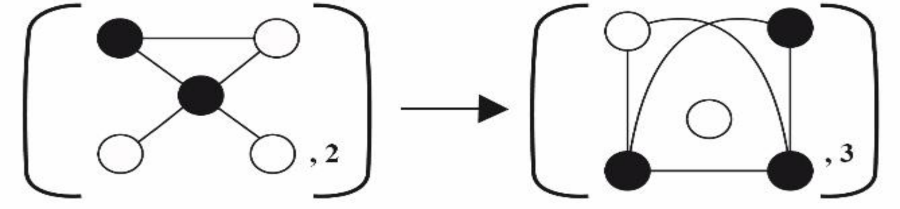
cubrimiento de tamaño 2

clique de tamaño 2 = 4 – 2



cubrimiento de tamaño 2

clique de tamaño 3 = 5 – 2

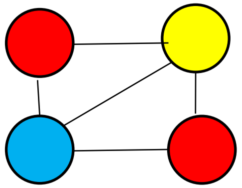


Ejemplos clásicos de reducciones polinomiales para encontrar lenguajes NP-completos (continuación)

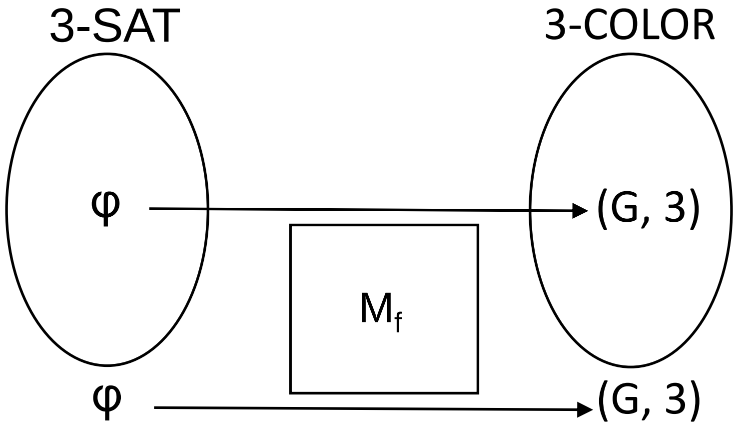
3-SAT = {φ | φ es una fórmula booleana sin cuantificadores en FNC con tres literales por cláusula y es satisfactible}

$$(x_1 \lor \neg x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_4) \land (\neg x_2 \lor \neg x_3 \lor x_1)$$

3-COLOR = {(G, 3) | G es un grafo no dirigido coloreable con 3 colores sin generar vértices vecinos con igual color}



Ejemplo de grafo coloreable con 3 colores



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

- Levin llamó a los problemas NP-completos **problemas universales**. La idea subyacente (¡y fascinante!) es que los miles de problemas NP-completos son en realidad pocos problemas, pero que se repiten y se expresan de distinta forma (por medio de grafos, fórmulas booleanas, números, conjuntos, etc).

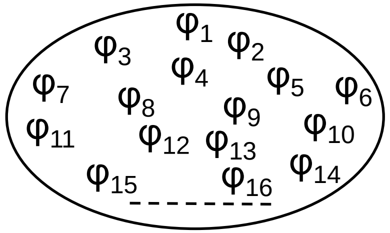
- <u>Observación 1</u>: Todo par de lenguajes L_1 y L_2 NP-completos conocidos son **p-isomorfos**:

$$L_1 \underset{f^{-1}}{\overset{f}{\rightleftarrows}} L_2$$

La función f y la función inversa f^{-1} son reducciones polinomiales

Se prueba que si todos los lenguajes NP-completos son p-isomorfos, se cumple P ≠ NP.

- <u>Observación 2</u>: Todos los lenguajes NP-completos conocidos son **densos**, es decir que tienen muchas cadenas (formalmente, para todo n tienen exp(n) cadenas de longitud a lo sumo n).



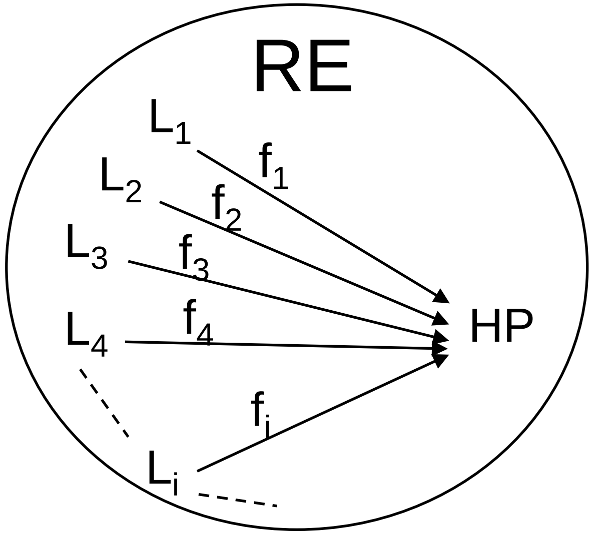
Lenguajes como SAT son densos, tienen muchas cadenas

A diferencia de lo indicado en la parte de computabilidad, en la complejidad computacional el tamaño de un lenguaje está muy ligado a su dificultad.

Se prueba que si existe un lenguaje NP-completo no denso (disperso), se cumple P = NP.

- Los lenguajes completos de una clase (RE, NP u otra) **identifican la dificultad de toda la clase**. P.ej.:

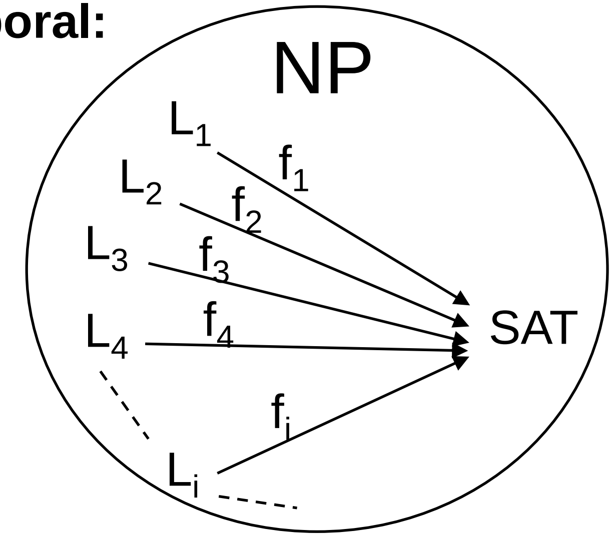
- **En la computabilidad:**



Todo lenguaje de RE se reduce a HP.
HP es **tan o más difícil** que todos los lenguajes de RE
(si HP fuera recursivo, entonces todos los lenguajes de RE serían recursivos, y así valdría R = RE).

HP (entre otros) identifica la dificultad de la clase RE.

- **En la complejidad temporal:**

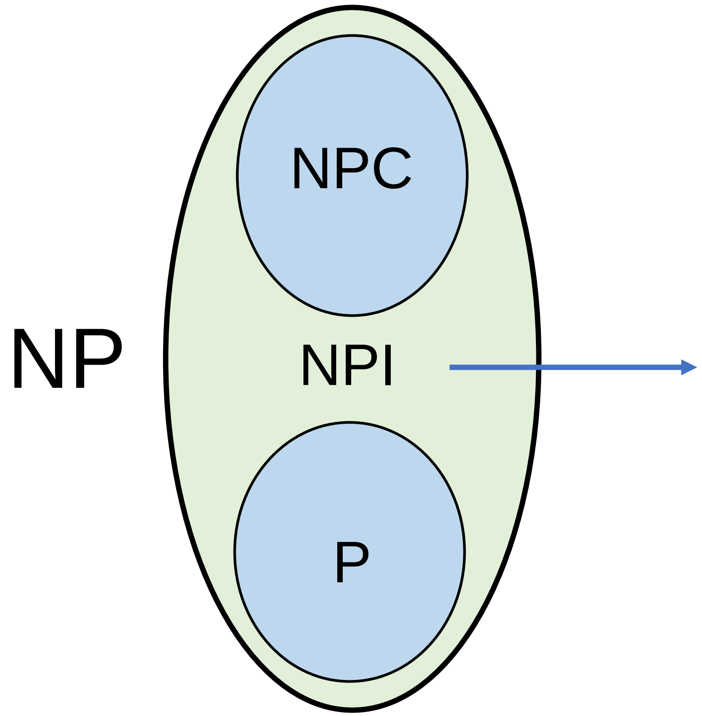


Todo lenguaje de NP se reduce polinomialmente a SAT.
SAT es **tan o más difícil** que todos los lenguajes de NP
(si SAT estuviera en P, entonces todos los lenguajes de NP estarían en P, y así valdría P = NP).

SAT (entre otros) identifica la dificultad de la clase NP.

La clase NPI

- Asumiendo P ≠ NP, entre las clases P y NPC de la clase NP se encuentra **la clase NPI**, llamada así porque identifica a los lenguajes de NP de **dificultad intermedia** (Teorema de Ladner, 1975).



Asumiendo P ≠ NP

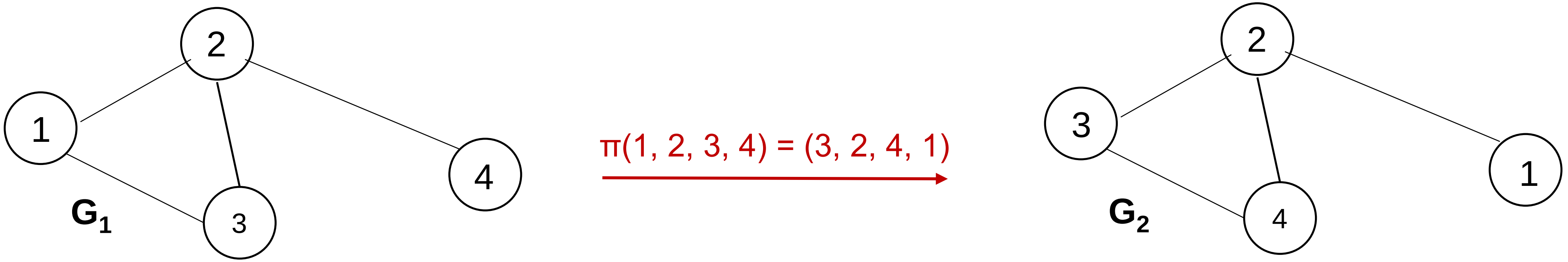
Hoy día, en NPI se ubican los lenguajes que:

- No cuentan con una MT de tiempo polinomial que los decida.

- No cuentan con una reducción polinomial desde un lenguaje NP-completo.

- El problema de los **grafos isomorfos** es candidato a pertenecer a NPI:

ISO = {(G_1, G_2) | G_1 y G_2 son grafos isomorfos, es decir son idénticos salvo por el nombre de sus arcos}

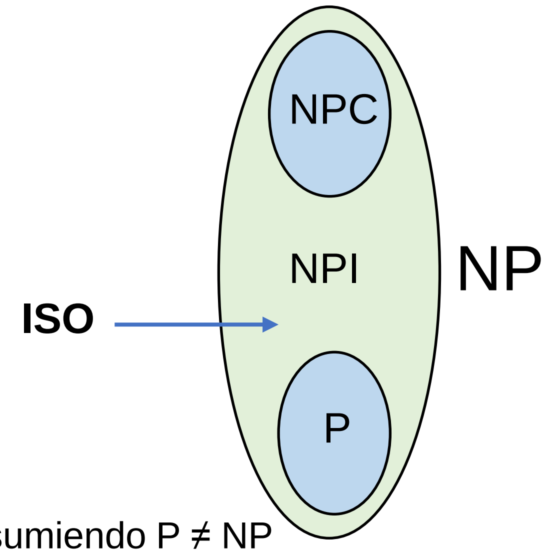


En este ejemplo, G_1 y G_2 son isomorfos: **la permutación π(1, 2, 3, 4) = (3, 2, 4, 1**) es un certificado de (G_1, G_2): (1, 2) es un arco de G_1 y (π(1), π(2)) = (3, 2) es un arco de G_2; (1, 3) es un arco de G_1 y (π(1), π(3)) = (3, 4) es un arco de G_2; etc.

ISO está en NP
Los certificados son permutaciones de vértices y se verifican eficientemente.

ISO no estaría en P
En el peor caso hay que probar con todas las permutaciones de vértices.

ISO tampoco estaría en NPC
No se ha encontrado ningún lenguaje NP-completo L, tal que $L \leq_P$ ISO.

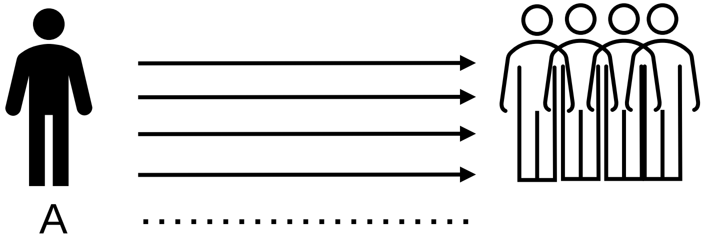


Asumiendo P ≠ NP

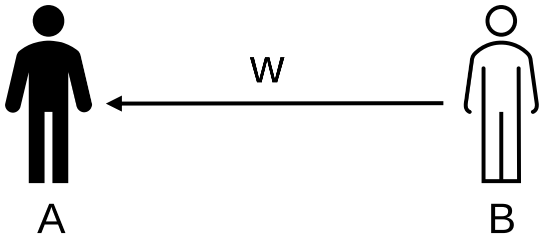
- El problema de **factorización** es otro candidato a pertenecer a la clase NPI:
  Definición: dado N, encontrar sus factores primos. Por ejemplo, **$180 = 2^2 \times 3^2 \times 5$.**
  Lenguaje asociado: **FACT = {(M, A, B) | existe un factor primo de M entre A y B}**

$180 \rightarrow \boxed{M} \rightarrow 2^2 \times 3^2 \times 5$

- Asumiendo FACT ∉ P, se lo utiliza para implementar un sistema de seguridad de **clave pública y privada.**
  Fundamento: si **$N = N_1 \times N_2$** es muy grande, y **N_1 y N_2 son números primos**,
  se necesitaría mucho tiempo (tiempo exponencial) para obtener N_1 y N_2 de N:

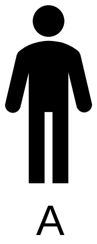


A permite que todos conozcan la **clave pública** (incluso los *hackers*), basada en **N**. Sólo A tiene la **clave privada**, basada en **N_1 y N_2.**



B le envía a A un mensaje w, encriptado con una función E:
v = E(w, N)

v = 111101000011101011100011111101010

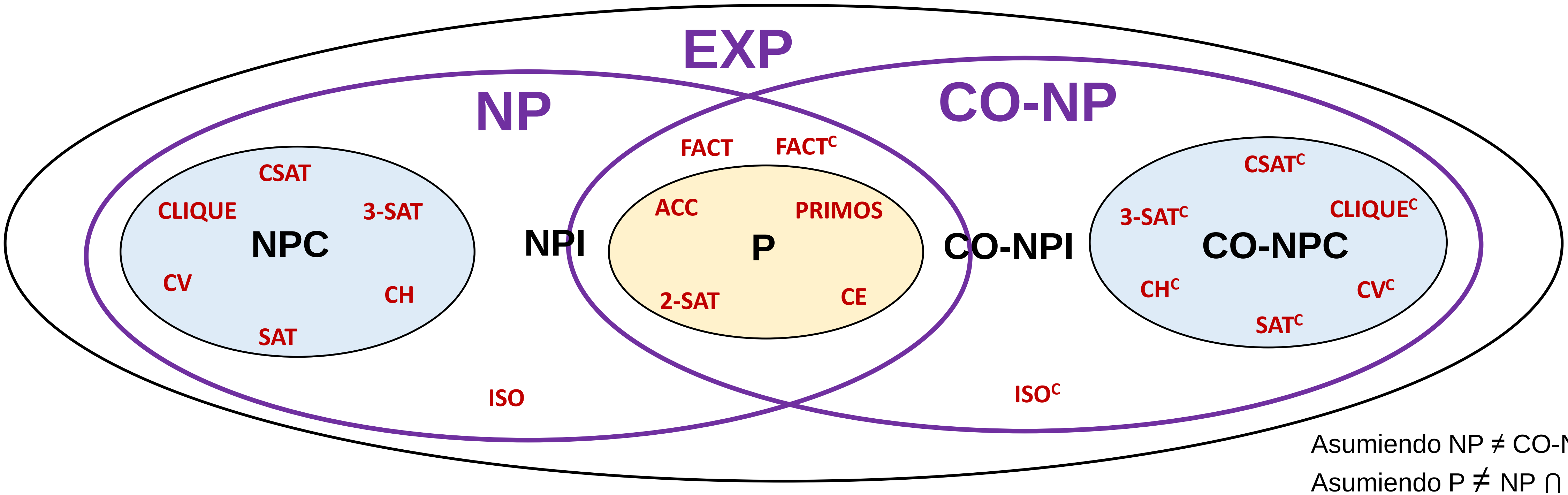


A desencripta el mensaje w, por medio de una función D:
$w = D(v, N_1, N_2)$

w = 101111000010101110111|101010111

En 1994, **P. Shor diseñó un algoritmo cuántico que factoriza en tiempo polinomial.**
Cuando las máquinas cuánticas sean una realidad, el esquema anterior no servirá más.
También puede suceder que la factorización pertenezca a P.

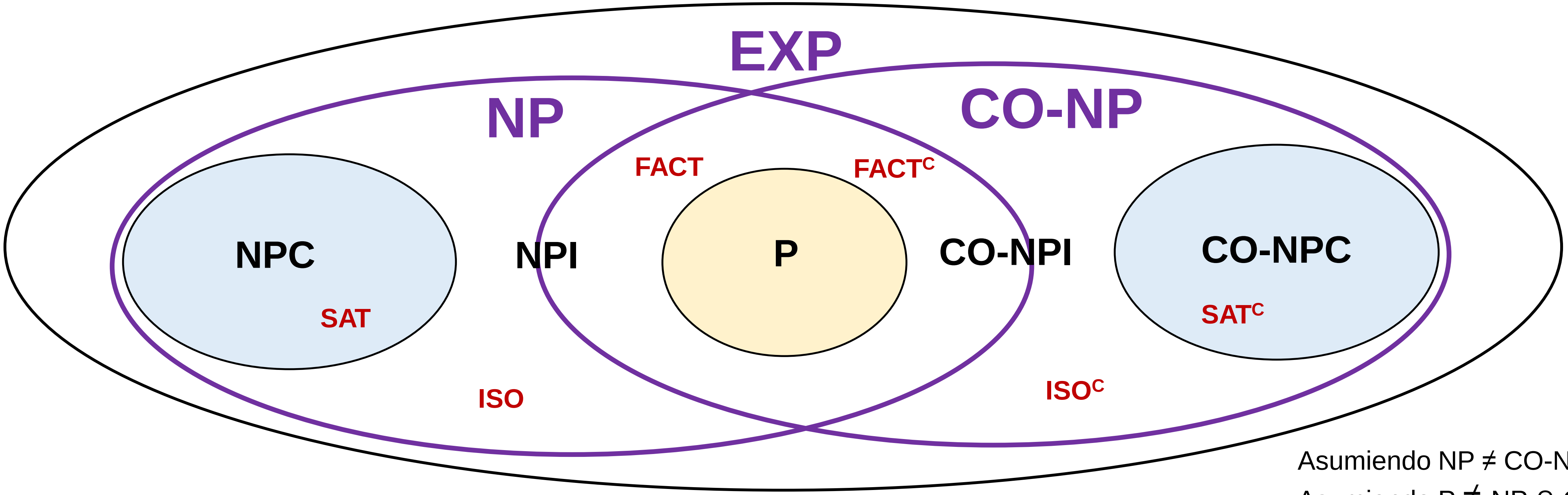
Ultima versión de la jerarquía temporal



Asumiendo NP ≠ CO-NP
Asumiendo P ≠ NP ∩ CO-NP

- Se cumple que **NP ≠ CO-NP implica P ≠ NP** (ejercicio)
- Se cumple que **NPC y NP ∩ CO-NP son disjuntos** (ejercicio)

- El **problema de primalidad** (lenguaje **PRIMOS):** ¿N es un número primo?, tuvo una evolución muy interesante:
  - En 1975 se probó su pertenencia a la clase NP.
  - Entre 1977 y 1992 se encontraron algoritmos probabilísticos eficientes para resolverlo.
  - Finalmente, en 2002 se demostró que pertenece a la clase P (Agrawal, Kayal y Saxena, 2002).



Asumiendo NP ≠ CO-NP
Asumiendo P ≠ NP ∩ CO-NP

Regiones con grado de dificultad creciente:

- La clase P
- La clase (NP ∩ CO-NP) – P
- La clase NPI – (NP ∩ CO-NP)
- La clase NPC
- La clase CO-NPI – (NP ∩ CO-NP)
- La clase CO-NPC

- ISO e ISO^C no estarían en (NP ∩ CO-NP) – P, pero FACT y $FACT^C$ sí. En este sentido, **FACT sería más fácil que ISO**.

  Esto se justificaría por el hecho de que **ISO puede tener cero o varias soluciones**, mientras que **FACT siempre tiene una** (por el Teorema Fundamental de la Aritmética).

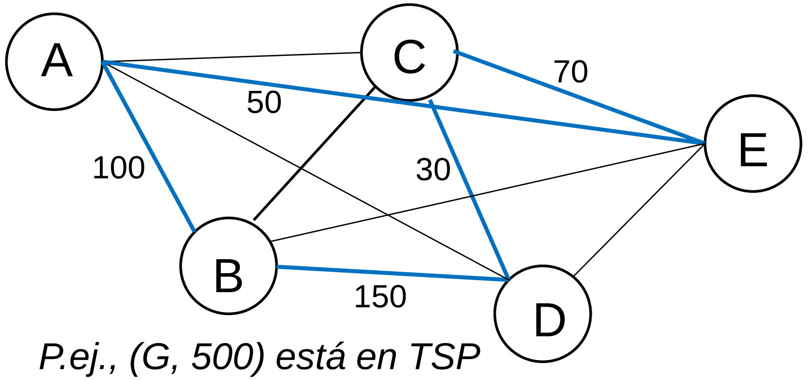
- Así como SAT está entre los lenguajes más difíciles de NP, **SAT^C está entre los lenguajes más difíciles de CO-NP**.

Anexo

El problema del viajante de comercio

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

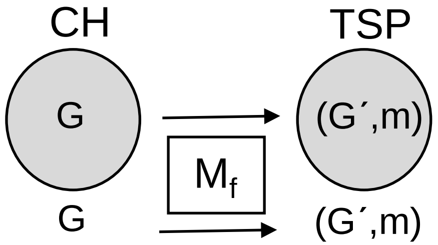
TSP (por *Travelling Salesman Problem*) representa el problema del viajante de comercio: un vendedor debe recorrer varias ciudades y volver a la inicial, de modo tal que en su recorrido no haga más que una distancia de K kilómetros.



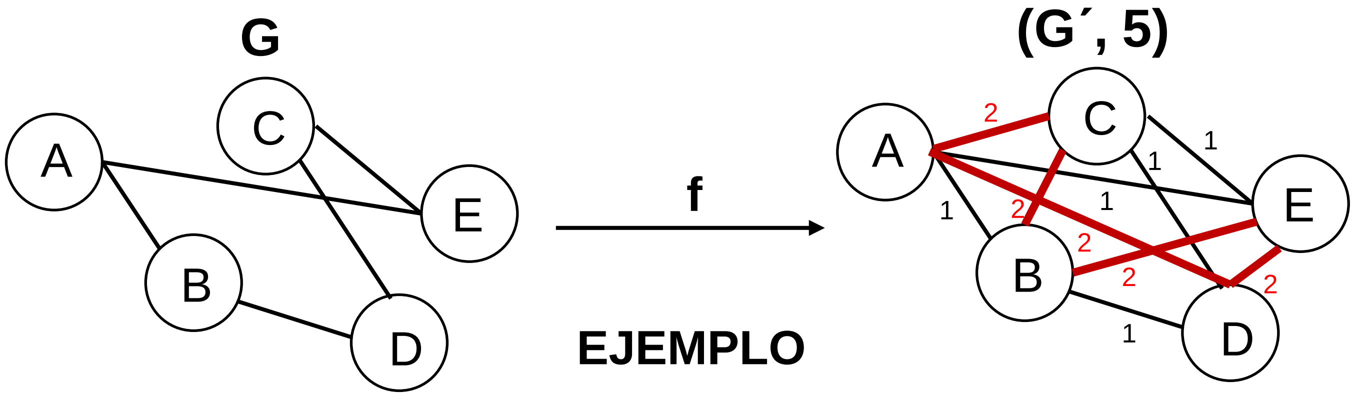
P.ej., (G, 500) está en TSP

Existe una reducción polinomial de CH a TSP

CH = {G | G es un grafo de m vértices y tiene un Circuito de Hamilton}



G y G´ tienen m vértices
G´ es ponderado (arcos con números) y completo (todos sus vértices están conectados)
Los arcos que están en G **tienen un 1** en G´, y los arcos que no están en G **tienen un 2** en G´



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

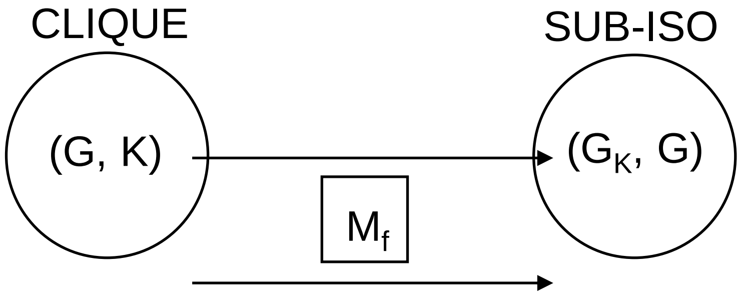
G ∈ HC sii (G´, m) ∈ TSP:
Si G tiene un CH, G' tiene un CdeH que mide: **m**
Si G no tiene un CH, todo CdeH mide al menos: **m + 1**

Algo más sobre la relación entre la NP-completitud y el tamaño

- **ISO** = {(G_1, G_2) | G_1 y G_2 son dos grafos isomorfos} **no sería NP-completo**, pero en cambio:

  SUB-ISO = {(G_1, G_2) | G_1 es un grafo isomorfo a un subgrafo de G_2} **sí lo es**.

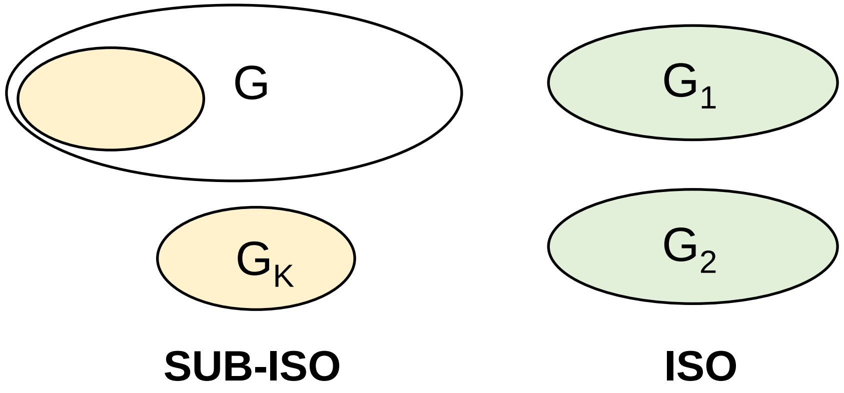
- La NP-completitud de SUB-ISO se puede probar con una reducción polinomial desde el lenguaje NP-completo CLIQUE:



Se le antepone a G un grafo completo G_K de K vértices

- Claramente, (G_K, G) se puede generar en tiempo polinomial, y (G, K) ∊ CLIQUE sii (G_K, G) ∊ SUB-ISO (ejercicio)

- Intuitivamente, SUB-ISO sería más difícil que ISO por su **redundancia de información**:
  - Modificando de varias maneras el grafo G en un par (G_K, G), G_K sigue siendo isomorfo a un subgrafo de G.
  - En cambio, cualquier alteración (no trivial) en un grafo de un par de grafos isomorfos (G_1, G_2) rompe el isomorfismo.



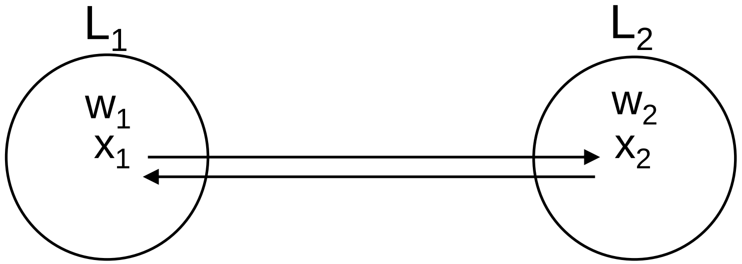
Algo más sobre la estrecha relación entre los lenguajes NP-completos

- No sólo todo par de lenguajes NP-completos conocidos L_1 y L_2 son **p-isomorfos**:

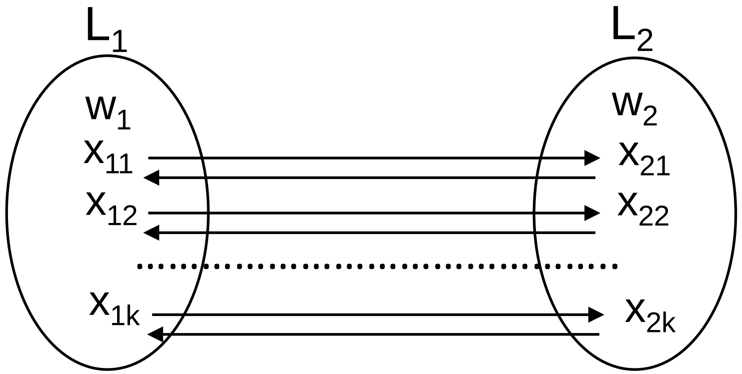
$$L_1 \underset{f^{-1}}{\overset{f}{\rightleftarrows}} L_2$$

La función f y la función inversa f^{-1} son reducciones polinomiales

- Sino que también en su mayoría, dos lenguajes NP-completos conocidos L_1 y L_2 cumplen que **los certificados de sus cadenas se pueden transformar eficientemente en uno y en el otro sentido,**



y más aún, **el número de certificados de uno coincide con el número de los certificados del otro:**



Las reducciones con esta propiedad se conocen como **parsimoniosas**

El problema P vs NP revisitado

- La conjetura ampliamente aceptada es que P ≠ NP, pero no se descartan otras posibilidades.

1. **Que haya algoritmos eficientes para los lenguajes NP-completos, pero con exponentes muy grandes, del tipo $O(n^{100})$.** En un escenario así, el costo para obtener una solución seguiría siendo mucho mayor que el costo para verificarla.

2. **Que se demuestre que hay un algoritmo eficiente para un lenguaje NP-completo, pero con una prueba no constructiva**. En este caso, aún sabiendo de su existencia, no tendríamos idea alguna sobre la estructura del algoritmo.

3. **Que las conjeturas P ≠ NP y P = NP no puedan demostrarse con la lógica existente**. En esta situación, aun contando con un algoritmo eficiente para un lenguaje NP-completo, no lo podríamos comprobar.

Clase práctica 6

Ejemplo 1

DSAT = {φ | φ es una fórmula booleana sin cuantificadores en la forma normal disyuntiva (FND) y es satisfactible}

La forma FND consiste en disyunciones de conjunciones de literales (variables o variables negadas). P. ej.:

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

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

Dada una fórmula booleana φ, M hace:

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

2) Chequea si existe una conjunción que no tenga al mismo tiempo variables y variables negadas x_i y $\neg x_i$.
   Si existe una conjunción así, significa que φ es satisfactible, y acepta; en caso contrario, rechaza.
   Tiempo poly(n) (ejercicio)

(sigue en la hoja siguiente)

Sea ahora:

DSAT_{NT} = {φ | φ es una fórmula booleana sin cuantificadores en la forma FND y tiene al menos una asignación que no la satisface - no es una tautología -}

No pareciera que $\mathrm{DSAT}_{NT} \in P$:

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) (ejercicio)

Más aún, se prueba que DSAT_{NT} es NP-completo:

1) Se prueba fácilmente que $\mathrm{DSAT}_{NT} \in NP$ (ejercicio)

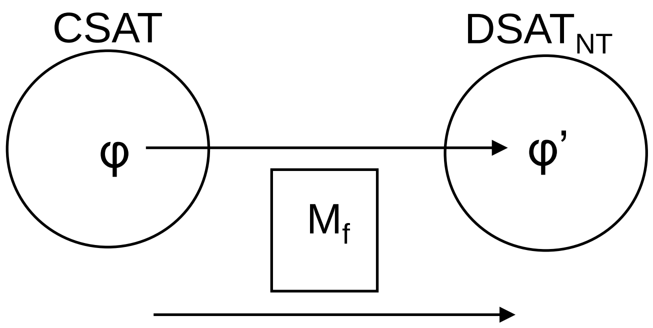
2) Se cumple que DSAT_{NT} es NP-difícil:

(sigue en la hoja siguiente)

Prueba que $DSAT_{NT}$ es NP-difícil

La siguiente función f es una reducción polinomial de CSAT (que es NP-completo) a $DSAT_{NT}$:

$f(\varphi) = \varphi'$, tal que f niega la fórmula φ.



Por ejemplo:
Dado $\varphi = (x_1 \vee x_2) \wedge (x_4 \vee \neg x_4) \wedge (\neg x_3 \vee x_5 \vee x_6)$,
queda $\varphi' = (\neg x_1 \wedge \neg x_2) \vee (\neg x_4 \wedge x_4) \vee (x_3 \wedge \neg x_5 \wedge \neg x_6)$.

1) Existe una MT M_f que computa f en tiempo polinomial:

M_f transforma φ en φ' según las leyes de De Morgan (si φ no es correcta sintácticamente, tampoco lo es φ').
Tiempo lineal (ejercicio)

2) $\varphi \in CSAT$ sii $f(\varphi) = \varphi' \in DSAT_{NT}$

$\varphi \in CSAT$ sii
φ está en la forma FNC y existe una asignación $\mathcal{A}$ que la satisface sii
φ' está en la forma FND y existe una asignación $\mathcal{A}$ que no la satisface sii
$\varphi' \in DSAT_{NT}$

Ejemplo 2

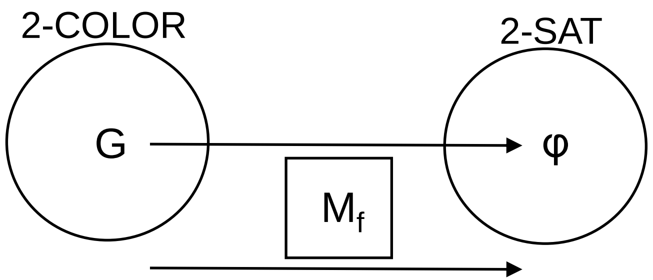
Existe una reducción polinomial de 2-COLOR a 2-SAT, siendo:

2-COLOR = {G | G es un grafo no dirigido 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, en la forma FNC con dos literales por cláusula, y es satisfactible}

Reducción:

- A todo grafo válido G se le asigna una fórmula booleana φ, tal que por cada arco (i, j) se construye $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$.
- A todo grafo inválido se le asigna una cadena inválida, p. ej. 1.



La reducción es 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 distintos c_1 y c_2 con los valores de verdad *verdadero* y *falso*, respectivamente, es fácil comprobar que los vértices i y j de (i, j) tienen colores distintos sii $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$ es satisfactible.

Ejercicio. Sabiendo que 2-SAT ∈ P, ¿qué se puede decir de 2-COLOR?