# PROBLEMA NP-COMPLETO: CSAT

**Alejandro Cárdenas Barranco**
**Juan Manuel Rodríguez Gómez**

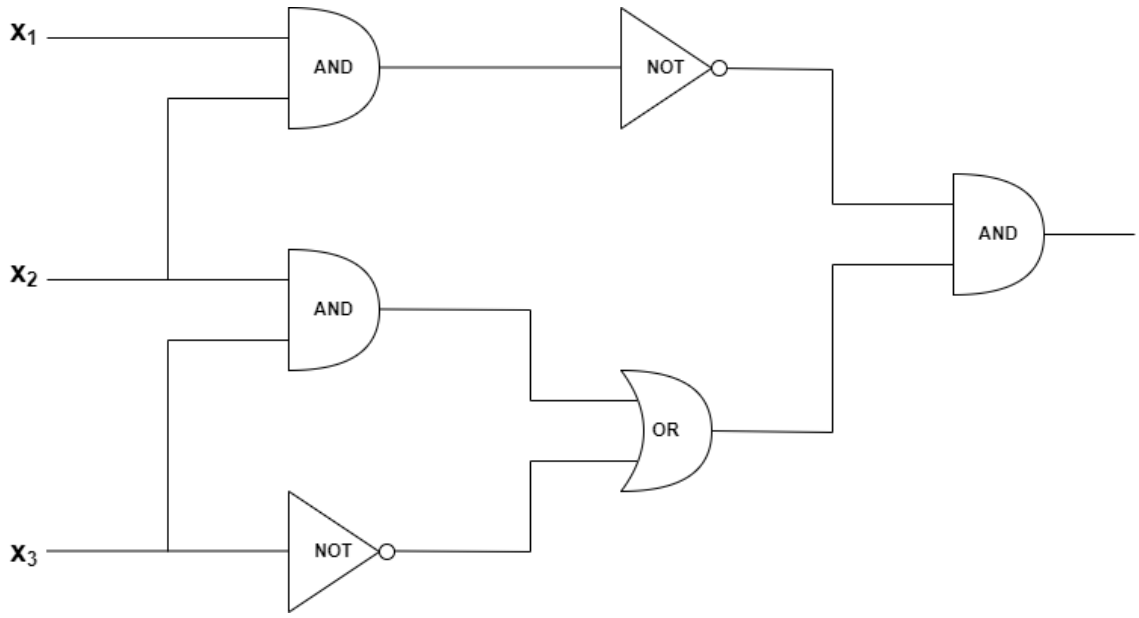
Modelos Avanzados de Computación
Curso 2022 – 2023


UNIVERSIDAD
DE GRANADA

# 1. Descripción del Problema CSAT

**Un circuito lógico es un grafo con entradas binarias, puertas AND, OR y NOT y una salida binaria. El problema CSAT es dado un circuito lógico, determinar si existe una entrada que hace que la salida sea verdad.**

Probaremos que el problema CSAT es NP-completo.




# 2. Demostración de la NP-Completitud del Problema CSAT

Para demostrar que el problema CSAT es NP-completo, necesitamos ver que CSAT está contenido en el conjunto NP y que todo problema NP es reducible a CSAT en espacio logarítmico.

Por tanto, realizaremos la demostración en dos partes:

1) **Probar que CSAT está en NP.**

2) **Probar que cualquier otro problema NP se reduce a él en espacio logarítmico.**

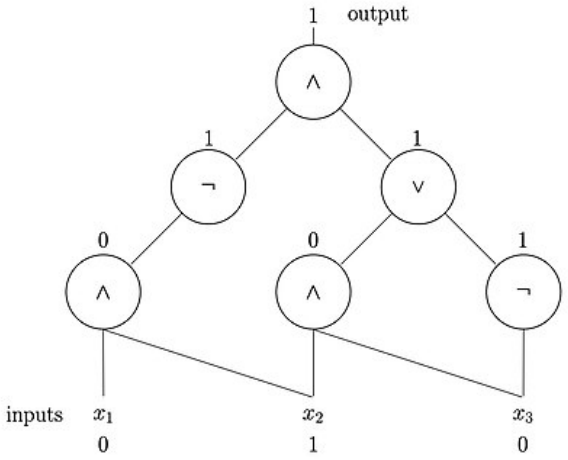
## 2.1. CSAT está en NP

**Un problema pertenece a NP si puede ser resuelto por una máquina de Turing no determinista en tiempo polinómico** o, equivalentemente, si dada una posible solución del problema podemos verificar en tiempo polinómico si la solución es correcta. Para ver esto, establecemos un algoritmo que genera de manera no determinista una sucesión de valores binarios, que serán las entradas del circuito lógico, y que comprueba si dicha sucesión de valores produce en el circuito dado da una salida verdadera.

Es claro que el algoritmo comentado anteriormente es no determinista, por lo que el problema CSAT puede ser resuelto por una máquina de Turing no determinista.

Además, el tiempo de resolución del algoritmo es polinómico ya que se dan los siguientes hechos:

- La entrada dada tiene un tamaño finito *x*.
- El grafo que represente al circuito lógico tendrá un número finito de nodos *n*.
- El tiempo para evaluar la salida de un nodo se hace en un tiempo constante *c*.




Teniendo en cuenta estos hechos, es claro que el tiempo de ejecución del algoritmo es polinómico. En conclusión, el problema **CSAT está en NP**.

## 2.2. Reducción del Problema 3-SAT Exacto a CSAT

Vamos a realizar una **reducción del problema 3-SAT exacto, el cual sabemos que es NP-completo, al problema CSAT**. De esta forma, quedaría demostrado que cualquier otro problema NP se reduce a CSAT y, por tanto, CSAT sería también NP-completo.

Cabe destacar que hay diferentes versiones del problema 3-SAT exacto. En nuestro caso, el problema 3-SAT exacto se enuncia como sigue: Dado un conjunto de variables $U$ y un conjunto de cláusulas en forma normal conjuntiva $C$ de longitud exactamente igual a 3 sobre dichas variables, determinar si se le puede asignar un valor de verdad a cada variable, de tal forma que en cada cláusula haya un literal que es cierto.

Por ejemplo, si tenemos el conjunto de variables $U = \{x, y, z\}$ y el conjunto de cláusulas $C = \{x \vee y \vee z, \neg x \vee y \vee \neg z\}$, la combinación $x = 1$, $y = 1$, $z = 0$ da una respuesta afirmativa al problema 3-SAT exacto.

**La idea general del algoritmo de reducción consiste en traducir el lenguaje lógico formal de las cláusulas en el lenguaje de puertas lógicas binarias. Una vez hecho esto, unimos todo con una puerta $\boldsymbol{AND}$**. A continuación, procedemos a formalizar dicha idea. Consideramos una entrada del problema 3-SAT exacto, es decir, un conjunto de variables $U = \{p_1, \dots, p_n\}$ y un conjunto de cláusulas $C = \{c_1, \dots, c_m\}$, donde cada cláusula es de la forma $c_i = = \ q_{i1} \vee q_{i2} \vee q_{i3}$, con $q_{ij} = \ p_k$ o $q_{ij} = \neg p_k$. Podemos construir cada cláusula con dos puertas $OR$ y en el caso de que alguna de las variables aparezca negada añadimos una puerta $NOT$. De esta forma, por cada cláusula siempre tendremos dos puertas $OR$ y, a lo sumo, tres puertas $NOT$. Tras esto, añadimos una puerta $AND$ con el fin de unir el resultado de cada una de las cláusulas. Dicha unión resultaría en un circuito lógico de la forma:

$$\boldsymbol{AND}(\boldsymbol{OR}(\boldsymbol{q_{11}}, \boldsymbol{q_{12}}, \boldsymbol{q_{13}}), \dots, \boldsymbol{OR}(\boldsymbol{q_{m1}}, \boldsymbol{q_{m2}}, \boldsymbol{q_{m3}})),$$

donde $q_{ij} = \ p_k$ o $q_{ij} = NOT(p_k)$, tal y como comentamos anteriormente.

Veamos ahora que los problemas son equivalentes, es decir:

**3-SAT(U, C) = CSAT( RED(U, C) )**

- Si 3-SAT(U, C) = "SÍ", entonces existe un conjunto de valores de verdad asociado a las variables que satisfacen todas las cláusulas. Por el procedimiento explicado anteriormente, es claro que CSAT( RED(U, C) ) = = "SÍ", puesto que para cada cláusula obtenemos una salida verdadera de una puerta OR y, como se satisfacen todas las cláusulas, la salida de la puerta AND será verdadera.
- Si CSAT( RED(U, C) ) = "SÍ", entonces existe una entrada binaria que hace que la salida sea verdadera. Si la salida final es verdadera, entonces la salida de cada una de las puertas OR es verdadera y, por tanto, se satisfacen todas las cláusulas. De esta forma 3-SAT(U, C) = "SÍ".

Finalmente, veamos que **la reducción descrita se realiza en espacio logarítmico**:

- Con respecto a la **codificación de la entrada del algoritmo de reducción**:

    - Si tenemos n variables, entonces para representar cada una de ellas usaremos $m + 1$ bits, donde el bit más significativo nos indicará si la variable está negada (1) o no (0) y los $m = \lceil log(n) \rceil$ bits restantes identifican a la variable.

    - Así, para representar un conjunto de $k$ cláusulas, necesitaríamos $3k(m + 1)$ bits ya que cada cláusula cuenta con 3 variables.

- Con respecto a la **codificación de la salida del algoritmo de reducción**:

    - Para representar las puertas $NOT$ usaremos el símbolo $n$, para las puertas $OR$ el símbolo $o$ y para las puertas $AND$ el símbolo $a$.

    - Usaremos el símbolo adicional $c$ para representar la separación entre las codificaciones de las cláusulas.

    - Usaremos el símbolo adicional $X$, el cual irá acompañado a la derecha de un símbolo de operación lógica $n/o/a$, indicando que se le aplica dicha operación lógica al conjunto de codificaciones de cláusulas que se encuentran a la izquierda del símbolo $X$.

- De esta forma, a partir de la entrada codificada y siguiendo la idea explicada anteriormente, el **algoritmo de reducción** produciría una salida codificada de la siguiente forma:

    1) Dada la codificación de entrada de una variable perteneciente a una cláusula, si su primer bit es un 1, escribir $n$ (indicando que la variable está negada). Si no, no escribir nada. Tras ello, copiar los $m$ bits restantes que identifican a la variable.

    2) Añadir la codificación de una puerta $OR$, escribiendo $o$.

    3) Repetir el paso 1 para la segunda variable de la cláusula.

    4) Añadir la codificación de una puerta $OR$, escribiendo $o$.

    5) Repetir el paso 1 para la tercera variable de la cláusula.

    6) Como cada cláusula está formada por exactamente 3 variables, escribir una $c$ para indicar el final de la cláusula.

7) Si hay más cláusulas, repetir los pasos 1-6 dicha cláusula.

8) Si no hay más cláusulas, escribir los símbolos $Xa$.

De esta forma, es claro que la reducción se realiza en espacio logarítmico.

- Un **ejemplo** del algoritmo de reducción sería el siguiente:
  - Conjunto de variables: $U = \{x, y, z\}$
  - Conjunto de cláusulas: $C = \{x \vee y \vee z, -x \vee y \vee -z\}$
  - Codificación de las variables:
    - $x \rightarrow 00$
    - $y \rightarrow 01$
    - $z \rightarrow 10$
  - Entrada del algoritmo (cláusulas codificadas):
    - $x \vee y \vee z \rightarrow 000001010$
    - $\neg x \vee y \vee \neg z \rightarrow 100001110$
  - Salida del algoritmo: $00o01o10cn00o01on10Xa$