UNIVERSIDAD DE GRANADA

Problema CSAT: NP-Completo

Pablo Martin Palomino

May 2026

1 Problema

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

Vamos a ver un ejemplo.

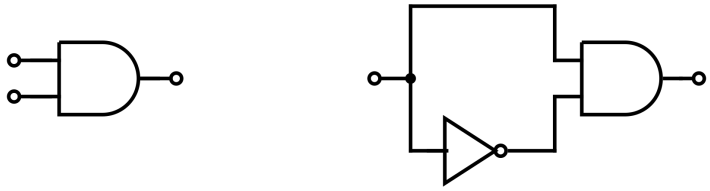

Figure 1: Imagen obtenida de `https://en.wikipedia.org/wiki/Boolean_circuit`.

La Figura 1 muestra dos circuitos. El circuito de la izquierda muestra una simple puerta AND. Tiene dos entradas distintas y una salida. La regla básica aquí es que la salida solo será verdadera si ambas entradas son verdaderas. Si lo analizamos aisladamente, este circuito sí es satisfacible en CSAT, ya que basta con poner ambas entradas a 1.

El circuito de la derecha tiene una sola entrada que se bifurca en dos cables o caminos:

- **El camino superior:** lleva la señal original directamente a la puerta AND.
- **El camino inferior:** pasa la señal por una puerta NOT (el triángulo con el pequeño círculo), la cual invierte la señal antes de entregarla a la puerta AND.

Si probamos con $x = 1$, el cable superior lleva un 1 y la puerta NOT invierte el cable inferior a 0; la puerta AND evalúa $1 \wedge 0$ y la salida es 0. Si probamos con $x = 0$, el cable superior lleva un 0 y la puerta NOT invierte el inferior a 1; la puerta AND evalúa $0 \wedge 1$ y la salida es 0. Como ninguna de las dos únicas entradas posibles produce salida 1, se concluye que es un circuito insatisfacible.

2 Demostración NP-Completitud

Para demostrar que CSAT es NP-Completo hemos de demostrar dos cosas:

- Demostrar que CSAT está en NP.
- Probar que cualquier otro problema de NP se reduce a él.

2.1 Demostración de CSAT en NP

Para demostrar que CSAT está en NP lo haremos de dos maneras. Aunque ambas son equivalentes, las incluimos las dos para que el trabajo sea más completo, apoyándonos en las dos caracterizaciones de la clase NP:

1. Un problema está en NP si existe una máquina de Turing no determinista que lo resuelve en tiempo polinómico.
2. Un problema está en NP si podemos verificar una solución candidata en tiempo polinómico mediante un algoritmo determinista.

2.1.1 Mediante una máquina de Turing no determinista

Diseñamos una MTND que, en su primera etapa, elige de forma no determinista una combinación de valores (0 y 1) para todas las entradas del circuito. Escribir estos valores toma un tiempo lineal respecto al número de entradas. Una vez fijadas las entradas, la máquina simplemente simula el circuito puerta por puerta hasta llegar a la salida, y acepta si y solo si dicha salida vale 1. Es claro que esta MTND resuelve CSAT: una rama acepta si y solo si la combinación de entradas elegida en esa rama satisface el circuito.

Veamos que el tiempo es polinómico. Sea n la longitud de la cadena que representa el circuito booleano en la cinta. Este tamaño n engloba el número total (finito) de entradas, puertas lógicas y conexiones. En la primera etapa, la MTND escribe en una cinta de trabajo una asignación de valores booleanos para las k entradas del circuito. Dado que el número de entradas nunca puede ser mayor que el tamaño total del propio circuito ($k \leq n$), escribir estos k bits requiere a lo sumo n pasos; por tanto, esta fase toma tiempo $O(n)$, que es polinómico.

La fase de evaluación también es polinómica: al ser el circuito un grafo dirigido acíclico, la máquina puede evaluarlo puerta por puerta siguiendo un

orden topológico. Evaluar una puerta toma tiempo constante y hay $O(n)$ puertas, de modo que evaluar todo el circuito toma tiempo $O(n)$.

El tiempo total de la rama de aceptación es la suma del tiempo de la primera etapa más el de la evaluación. Como la suma de tiempos polinómicos $(O(n) + O(n))$ es polinómica, queda demostrado que CSAT se resuelve en tiempo no determinista polinómico y, por tanto, CSAT $\in$ NP.

2.1.2 Mediante un verificador

El certificado. Para CSAT, el certificado es una asignación específica de valores booleanos a los nodos de entrada del circuito. Su tamaño está acotado por n, luego es polinómico en la entrada.

El verificador. Dado el circuito y la asignación de entrada, evaluamos el circuito puerta por puerta avanzando desde las entradas hasta el nodo de salida, en orden topológico. Como el número de puertas es a lo sumo n y cada una se evalúa en tiempo constante, la evaluación completa toma tiempo $O(n)$. El verificador acepta si y solo si la salida vale 1.

Conclusión. Existe un certificado de tamaño polinómico tal que el verificador acepta si y solo si el circuito es satisfacible, y dicha verificación se realiza en tiempo polinómico. Por tanto, CSAT $\in$ NP.

2.2 CSAT es NP-Completo

Primero para este apartado voy a realizar un recordatorio de que entendemos por reducción:

Definición 1 (Reducción en espacio logarítmico)**.** Sean P_1 y P_2 dos problemas de decisión sobre los conjuntos X e Y, respectivamente. Decimos que P_1 es reducible a P_2, y escribimos $P_1 \propto P_2$, si y solo si existe un algoritmo determinista que, trabajando en espacio logarítmico, calcula una función $\text{REDU} : X \to Y$ tal que, para toda instancia $x \in X$,

$$P_1(x) = P_2(\text{REDU}(x)).$$

Probar que CSAT es NP-Completo significa demostrar que está en NP (lo cual ya hemos demostrado) y que todo problema de NP se reduce a CSAT. Para demostrar esto debemos tomar un problema NP-Completo conocido, como el problema 3-SAT exacto, y transformarlo en una instancia equivalente de CSAT (mediante una reducción).

Por ser 3-SAT NP-Completo sabemos que:

$$\forall L \in \text{NP}: \quad L \propto \text{3-SAT}.$$

Si construimos además una reducción 3-SAT $\propto$ CSAT, entonces, por transitividad de $\propto$,

$$\forall L \in \text{NP}: \quad L \propto \text{3-SAT} \propto \text{CSAT} \implies L \propto \text{CSAT}.$$

Es decir, una vez que sabemos que CSAT está en NP basta con reducir un único problema NP-Completo a CSAT para concluir que CSAT es NP-Completo.

2.2.1 Reducción

Recordemos que una instancia de 3-SAT viene dada por un conjunto de variables $U = \{x_1, \ldots, x_n\}$ y un conjunto de cláusulas $C = \{C_1, \ldots, C_m\}$, que representamos compactamente por la fórmula booleana en forma normal conjuntiva

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

donde cada cláusula $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ es una disyunción de exactamente tres literales, y cada literal ℓ_{ij} es una variable x_t o su negación $\neg x_t$. La pregunta es si existe una asignación de las variables que haga φ verdadera.

Construcción. Definimos la función REDU que, dada φ, produce un circuito booleano C_φ del siguiente modo:

(1) Por cada variable x_t se crea un *nodo de entrada* del circuito.

(2) Cada literal se traduce a un cable: si el literal es x_t, se usa directamente la entrada x_t; si es $\neg x_t$, se hace pasar la entrada x_t por una puerta NOT.

(3) Cada cláusula $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ se traduce a una puerta OR sobre sus tres literales, $\text{OR}(\ell_{i1}, \text{OR}(\ell_{i2}, \ell_{i3}))$, encadenando dos puertas OR binarias.

(4) El nodo de salida es la conjunción de todas las cláusulas, $\text{AND}(g_1, \ldots, g_m)$, construida encadenando $m - 1$ puertas AND binarias, donde g_i es la salida de la cláusula C_i.

Por construcción, para toda asignación a de las variables el circuito cumple $C_\varphi(a) = \varphi(a)$.

La fórmula φ es satisfacible si y solo si el circuito $C_\varphi = \text{REDU}(\varphi)$ es satisfacible.

Proof. ($\Rightarrow$) Supongamos que φ es satisfacible y sea a una asignación que la satisface. Alimentamos las entradas de C_φ con a. Como a satisface cada cláusula C_i, al menos uno de sus literales vale 1, por lo que la puerta OR correspondiente produce $g_i = 1$. La puerta AND final calcula $g_1 \wedge \cdots \wedge g_m = 1$, así que la salida es 1 y C_φ es satisfacible.

($\Leftarrow$) Supongamos que existe una entrada b con $C_\varphi(b) = 1$. Entonces la puerta AND final vale 1, lo que obliga a que cada $g_i = 1$. Por construcción, $g_i = 1$ significa que la cláusula C_i tiene algún literal verdadero bajo b, luego b satisface todas las cláusulas y, por tanto, satisface φ. □

Ejemplo. Para $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ se obtienen tres entradas x_1, x_2, x_3; puertas NOT que producen $\neg x_1, \neg x_2, \neg x_3$; una subpuerta OR por cada cláusula; y una puerta AND final que combina ambas. La asignación $x_1 = x_2 = x_3 = 1$ hace la primera cláusula $1 \vee 0 \vee 1 = 1$ y la segunda $0 \vee 1 \vee 0 = 1$, luego la salida es 1: el circuito es satisfacible, igual que la fórmula.

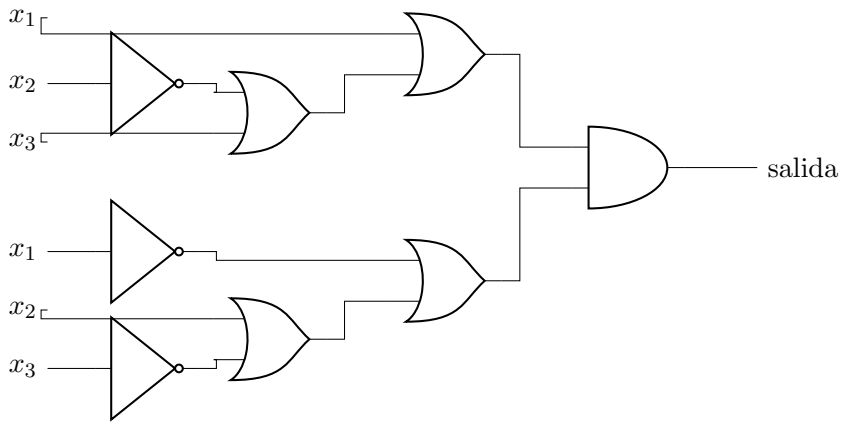

Figure 2: Circuito $C_\varphi = \text{REDU}(\varphi)$ para $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$. Cada cláusula se realiza con dos puertas OR (y las puertas NOT de sus literales negados) y todas se combinan con una puerta AND final. Las etiquetas de variable repetidas denotan la misma entrada del circuito.

Espacio logarítmico de la Reducción

Falta comprobar que REDU puede ser calculada por un algoritmo determinista que use solo $O(\log n)$ celdas de espacio de trabajo. Fijamos una codificación explícita de la entrada y de la salida.

Codificación de la entrada. Sea $b = \lceil \log_2 n \rceil$ el número de bits necesarios para identificar una de las n variables. Cada literal se codifica con $b+1$ bits: los b bits que identifican la variable seguidos de un bit de signo (1 si la variable aparece negada, 0 en caso contrario). Como cada cláusula consta de exactamente tres literales, se codifica con $3(b+1)$ bits, y la fórmula completa de m cláusulas ocupa $3m(b+1)$ bits.

Codificación de la salida. Describimos el circuito en *notación posfija* (los operadores se escriben tras sus operandos). Las puertas `n` (NOT) actúan sobre un operando, y `o` (OR) y `a` (AND) sobre exactamente dos; gracias a ello la cadena es autodelimitada y no necesita paréntesis ni marcas de fin de puerta. Un literal positivo se escribe como sus b bits de índice; uno negado, como esos b bits seguidos de `n`. Una cláusula $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ se realiza encadenando dos puertas OR binarias, $\mathrm{OR}(\mathrm{OR}(\ell_{i1}, \ell_{i2}), \ell_{i3})$, que en posfija es

$$g_i \;=\; L_{i1}\, L_{i2}\, \mathtt{o}\, L_{i3}\, \mathtt{o},$$

donde cada L_{ij} es la codificación (con o sin `n` final) del literal j-ésimo de la cláusula i. El circuito completo es la conjunción de las m cláusulas, encadenando $m-1$ puertas AND binarias asociadas a la izquierda,

$$\mathrm{AND}(\ldots \mathrm{AND}(\mathrm{AND}(g_1, g_2), g_3) \ldots, g_m)$$

, que se emite de forma incremental:

$$\underbrace{L_{11}L_{12}\mathtt{o}L_{13}\mathtt{o}}_{g_1}\ \underbrace{L_{21}L_{22}\mathtt{o}L_{23}\mathtt{o}}_{g_2}\ \mathtt{a}\ \underbrace{L_{31}L_{32}\mathtt{o}L_{33}\mathtt{o}}_{g_3}\ \mathtt{a}\ \cdots\ \underbrace{L_{m1}L_{m2}\mathtt{o}L_{m3}\mathtt{o}}_{g_m}\ \mathtt{a}.$$

Es decir, tras la primera cláusula, cada cláusula posterior va seguida de un único `a`, de modo que se acumulan exactamente $m-1$ puertas AND. (Si $m = 1$, la salida es simplemente g_1.)

El algoritmo de reducción. Recorremos la entrada cláusula a cláusula con el cabezal de lectura, manteniendo un único bit que indica si ya se ha emitido alguna cláusula:

(1) Para cada cláusula de la entrada:

(a) Emitir, en este orden, el primer literal, el segundo literal, el símbolo `o`, el tercer literal y de nuevo `o`. Para emitir un literal: copiar a la salida sus b bits de índice y, a continuación, leer su bit de signo; si es 1, escribir `n`.

(b) Si esta no es la primera cláusula emitida, escribir `a`.

(c) Marcar que ya se ha emitido al menos una cláusula.

(2) Detenerse al agotar la entrada.

El espacio es logarítmico. El algoritmo no almacena el circuito: lo *emite por flujo* en una sola pasada, sin mirar hacia adelante ni contar de antemano el número de cláusulas. Su memoria de trabajo se reduce a unos pocos elementos: el bit que indica si ya se emitió alguna cláusula, un contador acotado de la posición del literal dentro de la cláusula y un contador para copiar los b bits de un índice. Este último, al contar hasta $b = \lceil \log_2 n \rceil$, ocupa $O(\log b)$ celdas. Por tanto, el espacio de trabajo es $O(\log n)$, y la reducción es en espacio logarítmico como queríamos.

3 Conclusión

Hemos demostrado que CSAT es NP-Completo en dos pasos. Primero, que CSAT $\in$ NP, tanto exhibiendo una máquina de Turing no determinista polinómica como un verificador determinista polinómico. Segundo, que CSAT es NP-difícil, construyendo una reducción en espacio logarítmico 3-SAT $\propto$ CSAT y usando la transitividad de $\propto$ para cubrir todos los problemas de NP. La reducción explota una idea sencilla, una fórmula booleana es, en el fondo, un circuito, por lo que satisfacer la fórmula equivale a satisfacer el circuito asociado. En este enlace `https://en.wikipedia.org/wiki/Circuit_satisfiability_problem` se puede encontrar de donde surgió la idea de la reducción.