Problema CSAT

Demostración de un problema NP-completo

Irene Pérez Martínez Blanca Cano Camarero

Junio 2021

Problema CSAT

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

Queremos probar que CSAT es NP-completo.

Pasos de la demostración

Para demostrar que el problema CSAT es NP-completo necesitamos evidenciar que se cumpla que el CSAT esté contenido en el conjunto NP y que todo problema NP es reducible a CSAT en tiempo polinomial o espacio logarítmico.

Por eso, nuestra demostración estará dividida pro ende en dos partes:

1. Probar que CSAT está en NP.

2. Probar que cualquier otro lenguaje NP se reduce a él en espacio logarítmico.

CSAT es NP

Un problema pertenece a NP si pertece a la clase de complejidad polinómica no determinista en tiempo. Es decir, si puede ser resuelto en tiempo polinómico por una máquina de Turin no determinista. Para ver esto, establezcamos el algoritmo que genera de manera no determinista una sucesión de valores de verdad, que serán las entradas binarias a nuestro circuito lógico, y que comprueba si el circuito dado en el problema CSAT da salida verdadera.

Queda claro que el algoritmo es no determinista, por lo que el problema CSAT puede ser respondido por una máquina de Turin no determista. Ahora bien, debemos aclarar que el tiempo de resolución es polinómico, por lo que tendremos que tener en cuenta los siguientes hechos:

- La entrada dada tiene un tamaño finito x.
- Si establecemos un grafo que represente a nuestro circuito lógico, este contará con un número de nodos finito, n.
- El tiempo para evaluar la salida en un nodo de una entrada de tamaño x está acotado superiormente por k.

Teniendo en cuenta estos hechos, podríamos acotar el tiempo de ejecución del algoritmo superiormente con el producto xnk. Así pues, el problema CSAT está en NP.

Reducción NP a CSAT

Para llevar a cabo la demostración de esta parte hemos seleccionado un problema NP-completo, el problema 3-SAT. Si demostramos que este se puede reducir a nuestro problema quedaría demostrado que cualquier otro problema NP se puede deducir al nuestro, como queremos.

Descripción del problema 3-SAT

Dado un conjunto de símboloso proposicionales $U = p_1, ..., p_{n-1}$ y una colección de C de claúsusas sobre estos símbolos, el problema determina si son consistentes dichas claúsulas y da una respuesta verdadera o falsa dependiendo del caso.
Cabe destacar que hay múltiples versiones del problema, pero que, concretamente, el 3-SAT en que nos vamos a basar se caracteriza porque requiere determinar la satisfacibilidad de una fórmula en forma normal conjuntiva donde cada cláusula se limita a lo sumo tres literales.

Por poner un ejemplo que clarifique el problema, contamos con la cláusula siguiente para el conjunto $\{x, y\}$ de símbolos:

$$(x \vee x \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \wedge (\neg x \vee y \vee y)$$

La única combinación que hace satisfacible al problema es tomar x=FALSO e y=VERDADERO, luego para la entrada con estas claúsulas sí sería aceptada.

Reducción del problema 3-SAT a CSAT

Necesitamos ver que cualquier otro problema NP-completo se puede reducir a CSAT, lo que haremos será, como decíamos, probar que 3-SAT se puede reducir a CSAT y puesto que la reducción es transitiva esto nos garantizará que cualquier otro problema NP-completo se pueda reducir a él.

La idea general es traducir las cláusulas del leguaje lógico formal dl de pueras lógicas binarias, una vez que tengamos esto, como queremos que se cumplan que todas seas satisfacibles las uniremos con una puenta AND y ya estaría. Formalicemos la idea expuesta:

Dada un conjunto de variables $U = \{p_1, ..., p_{n-1}\}$ y una serie de cláusulas, $C = \{c_1, ..., c_m\}$.

Donde cada $c_i = q_{i1} \vee q_{i3} \vee q_{i3}$ con $q_{ij} = p_k$ o $q_{ij} = \neg p_k$

Es decir las claúsulas contarán con disyunciones de únicamente tres literales.

Cada cláusula se va a construir con dos puertas OR, que unirá a las variables involucradas. Estas variables pueden aparecer negadas, por lo que añadimos una puerta NOT en tal caso. Por tanto, tendremos el doble de puertas OR como cláusulas y en el peor de los casos, el tiple de puertas NOT.

Una vez hecho esto debemos unir las cláusulas, y para ello haremos uso de una puerta AND que multiplique el valor que den las puertas OR. Así pues, unimos el resultado de todas las cláusulas mediante esta puerta AND. El resultado queda como un circuito lógico similar a como se sigue:

$AND(ORD(x_{11}, x_{12}, x_{13}), ..., ORD(x_{m1}, x_{m2}, x_{m3}))$ Donde cada x_{ij} se corresponde a identifica a una variable q_{ij} si ésta negaba a algún literal bastará con anteponer un NOT.

Equivalencia de los problemas

Si hay una asignación de verdad que satisface a todas las claúsuclas de 3-SAT entonces, esa misma asignación hará verdadera la salida del circuito construído, en caso contrario no existirá ninguna.

Reducción en espacio logarítmico

Nótese que la transformación consiste en leer cada entrada e escribir su salida sustituyendo los símbolos como se ha explicado.

Por ejemplo, para un alfabeto de entrada $A = \{0, 1\}$ con la siguiente codificación:

- Para representar un símbolo:

  - Primer bit valor de verdad.
  - Si existe n variables $m = \lceil log(n) \rceil$ bits.

- Representación de las claúsulas:
  - Son agrupaciones de tres símpolos, luego si hay k claúsulas entonces habrá $N = 3k(m+1)$ bits de entrada a lo sumo.

La salida al algoritmo, si representamos con n, a, o a variables del alfabeto de salida a las respectivas puertas NOT,AND Y OR quedaría y contamos con los símbolos adicionales c, X que representa separación entre claúsulas y que se conectan las salidas de las puertas anteriores a otra X a una puerta del tipo que viene quedaría:

- Para representar símbolos:
  - Si primer valor 1 escribir n, si no nada.
  - Se copian los m bits restantes.
- Después de un símbolo de una misma claúsula, si viene otro, se añade la puerta OR, esto es, se escribe o en la cinta de salida.
- Tras cada claúsula se añade c si después viene otra claúsula.
- Tras acabar se añade los símbolos Xa.

Con esta codificación propuesta la transformación sería directa, por lo que cumple la condición de que requiera de espacio logarítmico en la entrada.

Ejemplo del algoritmo

En base a mismo ejemplo del principio:

$$(x \vee x \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \wedge (\neg x \vee y \vee y)$$

Sólo hay dos símbolos, x que codificaremos como 0, y codificado como 1.

Luego las claúsulas serán: - 000001, 101111, 100101 respectivamente.

- concatenando con las puertas ya explicada quedaría como salida: $0o0o1cn0n1n1cn011Xa$

Fuentes

(Esto habría que redactarlo según los cánones de referencias)

- Diapositivas de teoría.
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#3-satisfiability

- https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#Proof_of_NP-Completeness

- Libro Introducción a la teoría de autómatas lenguajes y computación. Tercera edición de Jopcroft, Motwani y Ullman. pag 373.