Teoría de la NP-completitud

Adrián Rodriguez Bazaga, Eleazar Díaz Delgado, and Rudolf Cicko

Universidad de La Laguna

20 de diciembre de 2016

Capítulo 1

Teorema de Cook

1.1. Introducción

El teorema de Cook también conocido como el teorema de Cook-Levin, se demuestra el primer problema $\mathcal{NP}$-Completo que permitirá que los demás problemas puedan ser transformados a él usando una reducción en tiempo polinomial.

El problema usado para probar su $\mathcal{NP}$-completitud es el de SATISFACTIBILIDAD (SAT).

1.2. Problema SAT

Demostrar que SAT $\in \mathcal{NP}$ es fácil, y se hará a continuación. La parte difícil de la prueba es demostrar que cualquier lenguaje $\mathcal{NP}$ es reducible a SAT en tiempo polinómico.

Para ello se construye una reducción polinómica para cada lenguaje A en los $\mathcal{NP}$ a SAT. La reducción de A coge una cadena w y produce una fórmula booleana ϕ que simula la máquina NP para el lenguaje A en la entrada w. Si la máquina no acepta, entonces la asignación no satisface ϕ. Por ello, w esta en A, si y sólo si, ϕ es satisfacible.

Actualmente, construir una reducción para trabajar de esta manera es una tarea conceptualmente simple, aunque piensa que se deben tener en cuenta muchos detalles. Una fórmula booleana puede que contenga las operaciones booleanas, AND, OR y NOT. Estas operaciones forman la base de la circuitería de los ordenadores electrónicos. De hecho podemos diseñar una fórmula booleana para simular una máquina de Turing.

El problema de la satisfabilidad booleana, es el problema de determinar si existe al menos una interpretación que satisfaga una fórmula booleana. Otra forma de verlo es que dado un conjunto de valores booleanos pasados a una fórmula booleana, hallar aquel conjunto que dé valores verdaderos.

1.3. Demostración del Teorema de Cook

ENTRADA: Un conjunto de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre un conjunto finito U de variables.
PREGUNTA: ¿Existe una asignación booleana para U, tal que satisfaga todas las cláusulas de C?

Teorema 1. *SAT es $\mathcal{NP}$-completo.*

Demostración. Es fácil comprobar que SAT $\in \mathcal{NP}$, ya que se puede encontrar un algoritmo para una Máquina de Turing No Determinista (NDTM) que reconozca el lenguaje $L(\text{SAT}, e)$, para un esquema de codificación e, en un número de pasos acotado por una función polinomial. De esta manera, el primer requerimiento para la prueba de $\mathcal{NP}$-completitud se cumple.

Para el segundo requerimiento debemos bajar a nivel de lenguaje, donde SAT se representa por un lenguaje $L_{SAT} = L[SAT, e]$ para algún esquema de codificación razonable e. Debemos demostrar que para todo lenguaje $L \in \mathcal{NP}$, $L \preceq L_{SAT}$. Los lenguajes de la clase $\mathcal{NP}$ son bastante diversos, se trata de una clase infinitamente grande, por lo que no esperemos presentar una demostración para cada uno de los lenguajes por separado.

No obstante, todo lenguaje de la clase $\mathcal{NP}$ puede describirse de una manera estándar: mediante un programa para una NDTM que lo reconozca en tiempo polinomial. Esto nos permitirá trabajar

con un programa genérico para una NDTM y derivar una transformación polinomial genérica desde el lenguaje que reconozca esta máquina genérica al lenguaje L_{SAT}. Esta transformación genérica, cuando se particularice para un programa de una NDTM M que reconozca el lenguaje L_M, dará la deseada transformación desde L_M a L_{SAT}. Así, en esencia, presentaremos una demostración simultánea para todos los lenguajes $L \in \mathcal{NP}$ de que $L \preceq L_{SAT}$.

Para empezar, denotemos por M un programa arbitrario en tiempo polinomial para una NDTM, especificado por Γ, Σ, b, Q, q_0, q_Y, q_N, y δ, que reconoce el lenguaje $L = L_M$. Además, sea $p(n)$ un polinomio que acota superiormente la función $T_M(n)$. Sin pérdida de generalidad se asume que $p(n) \geq n$ para todo $n \in \mathbb{Z}^+$. La transformación genérica f_L será derivada en términos de M, Γ, Σ, b, Q, q_0, q_Y, q_N, δ y p.

Por razones de conveniencia describiremos f_L como si fuera una función entre el conjunto de las cadenas de Σ hasta las entradas del problema SAT, mas que hasta las cadenas del alfabeto inducido por el esquema de codificación asociado a SAT, ya que los detalles de la demostración asociados al esquema de codificación pueden ser deducidos fácilmente.

De esta manera, f_L tendrá la propiedad de que para todo $x \in \Sigma^*$, $x \in L$ si, y sólo, $f_L(x)$ es satisfecho por una asignación booleana. La clave de las construcción de f_L está en demostrar cómo un conjunto de cláusulas puede ser usado para comprobar si una entrada x es aceptada por el programa para una NDTM M, es decir, si $x \in L$.

Si la cadena $x \in \Sigma^*$ es aceptada por M, entonces, habrá una secuencia de estados de M que acepte x acotada superiormente por un polinomio $p(n)$, donde $n = |x|$. Dicha secuencia no puede involucrar ninguna celda de la cinta excepto aquellas comprendidas entre $-p(n)$ y $p(n) + 1$ ya que la la cabeza de lectura/escritura comienza en la celda 1, y se mueve una única posición en cada paso. El estado de la computación en cualquier instante puede determinado completamente dando el contenido de las celdas, el estado actual, y la posición de la cabeza de lectura/escritura. Es más, puesto que no se llevarán a cabo más de $p(n)$ pasos durante la ejecución, habrá a los sumo $p(n)+1$ instantes que deben ser considerados. Esto nos permitirá describir la computación completamente usando sólo un número limitado de variables booleanas y una asignación booleana para las mismas.

A continuación se muestra cómo f_L construye el conjunto de variables a partir de M. Se etiquetarán los estados de Q como $q_0, q_1 = q_Y, q_2 = q_N, q_3, \ldots, q_r$, donde $r = |Q| - 1$, y los elementos de Γ como $s_0 = b, s_1, s_2, \ldots, s_v$, donde $v = |\Gamma| - 1$. Se crearán tres tipos de variables, cuyo sentido se especifica en la tabla 1.1.

Cuadro 1.1: Variables creadas por f_l

Variable	*Rango*	*Significado*
$Q[i,k]$	$0 \leq i \leq p(n)$ $0 \leq k \leq t$	En el instante i, M está en el estado q_k
$H[i,j]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n)+1$	En el instante i, la cabeza de lectura/escritura está examinando la celda j
$S[i,j,k]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n)+1$ $0 \leq k \leq v$	En el instante i, el contenido de la celda j es el símbolo s_k

El cómputo de M devuelve como resultado una asignación booleana para cada una de las variables. Si el programa se detiene antes del tiempo $p(n)$, la configuración permanece constante en todas las ejecuciones posteriores, manteniendo la posición de la cabeza, contenido de la cinta y el mismo estado de parada.

Inicialmente el contenido de la cinta consta de la entrada x escrita desde la celda 1 hasta n, y el resultado esperado escrito desde la celda -1 hasta la -$|w|$ con el resto de las celdas de la cinta vacías.

Por otro lado, una asignación booleana arbitraria de las variables no tiene por qué corresponder con un cómputo, menos aún con un cómputo de aceptación.

De acuerdo con una asignación booleana arbitraria, una cinta dada puede contener muchos símbolos al mismo tiempo, la máquina podría estar simultáneamente en varios estados diferentes, y la cabeza de lectura y escritura podría estar en cualquier subconjunto de posiciones $-p(n)$ hasta $p(n)$. La transformación funciona construyendo una colección de cláusulas, de modo que una asignación booleana es satisfactoria si y sólo si es la asignación booleana inducida por un cómputo de aceptación para x cuya etapa de verificación toma $p(n)$ o menos pasos y cuya cadena adivinada tiene longitud como máximo $p(n)$. Resumiendo, tenemos que:

$x \in L \iff$	si existe un estado de aceptación en M para x.
$\iff$	existe un estado de aceptación en M para x en $p(n)$ o menos pasos en la etapa de verificación y con una cadena adivinada w de longitud exactamente $p(n)$.
$\iff$	En el instante i, cada celda contiene exactamente un único símbolo de Γ.

Esto significa que f_L satisface una de las dos condiciones requeridas de una transformación polinomial. La otra condición, que f_L se pueda calcular en tiempo polinomial se verificará fácilmente una vez que hemos completado nuestra descripción de f_L.

Las cláusulas en f_L se pueden dividir en seis grupos, cada uno imponiendo un tipo separado de restricción en cualquier asignación de verdad satisfactoria como se muestra en el cuadro 1.2.

Cuadro 1.2: Grupos de cláusulas en $f_L(x)$ y las restricciones que imponen

Clase de grupo	*Restricción impuesta*
$G1$	En el instante i, M se encuentra exactamente en un estado.
$G2$	En el instante i, la cabeza de lectura/escritura está examinando exactamente una celda.
$G3$	En el instante i, cada celda contiene exactamente un único símbolo de Γ.
$G4$	En el instante 0 la máquina está en su configuración inicial para verificar la entrada x.
$G5$	En el instante $p(n)$, M ha cambiado al estado q_y y por lo tanto aceptado x.
$G6$	Para cada instante i, la configuración de M en el instante $i+1$ sigue las indicaciones de la función de transición δ para la configuración en el instante i.

Es sencillo observar que si todos los grupos de seis cláusulas realizan sus misiones previstas, entonces una asignación booleana satisfactoria tendrá que corresponder al estado de aceptación deseado para x. Así, todo lo que necesitamos mostrar es cómo pueden construirse los grupos de cláusulas que realizan estas misiones.

El grupo $G1$ consiste de las siguientes cláusulas:

$$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\}, 0 \leq i \leq p(n)$$
$$\{\overline{Q[i,j]}, \overline{Q[i,j']}\}, 0 \leq i \leq p(n), 0 \leq j \leq j^{'} \leq r$$

La primeras $p(n) + 1$ de estas cláusulas puede satisfacerse simultáneamente si y sólo si, para cada vez que i, M está en al menos un estado. Las cláusulas restantes $(p(n) + 1)(r + 1)(r/2)$ pueden satisfacerse simultáneamente si sólo en ningún momento i es M en más de un estado. Así $G1$ cumple su misión.

Los grupos $G2$ y $G3$ se construyen de manera similar, y los grupos $G4$ y $G5$ son bastante sencillos, cada uno de ellos consistente sólo en cláusulas literales. El cuadro 1.3 da una especificación completa de los cinco primeros grupos. Tenga en cuenta que el número de cláusulas en estos grupos y el número máximo de literales que ocurren en cada cláusula, está acotad por una función polinomial de n (puesto que r y v son constantes determinadas por M y por lo tanto por L).

Cuadro 1.3: Los cinco primeros grupos de cláusulas de f_L

Clase de grupo	*Restricción impuesta*
$G1$	$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\}, 0 \leq i \leq p(n)$ $\{\overline{Q[i,j]}, \overline{Q[i,j']}\}, 0 \leq i \leq p(n), 0 \leq j \leq j^{'} \leq r$
$G2$	$\{H[i,-p(n)], H[i,-p(n)+1], \ldots, H[i,p(n)+1]\}, 0 \leq i \leq p(n)$ $\{\overline{H[i,j]}, \overline{H[i,j']}\}, 0 \leq i \leq p(n), -p(n) \leq j \leq j^{'} \leq p(n)+1$
$G3$	$\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\}, 0 \leq i \leq p(n), -p(n) \leq j \leq p(n)+1$ $\{\overline{S[i,j,k]}, \overline{S[i,j,k']}\}, 0 \leq i \leq p(n), -p(n) \leq j \leq p(n)+1, 0 \leq k \leq k^{'} \leq v$
$G4$	$\{Q[0,0]\}, \{H[0,1]\}, \{S[0,0,0]\},$ $\{S[0,1,k_1]\}, \{S[0,2,k_2]\}, \ldots, \{S[0,n,k_n]\},$ $\{S[0,n+1,0]\}, \{S[0,n+2,0]\}, \ldots, \{S[0,p(n)+1,0]\}$, donde $x = s_{k_1} s_{k_2} \ldots s_{k_n}$
$G5$	$\{Q[p(n),1]\}$

El último grupo de cláusulas $G6$, que garantiza que cada sucesiva configuración en el cómputo sigue a partir de la anterior en cada paso del programa M, en realidad es más compleja ya que compuesta por dos subgrupos de cláusulas:

El primer subgrupo garantiza que si la cabeza lectura-escritura no está explorando la celda j de la cinta en el instante i, entonces el símbolo en la celda j no cambia entre los instantes i e $i + 1$. Las cláusulas de este subgrupo son las siguientes:

$$\{\overline{S[i,j,l]}, H[i,j], S[i+1,j,l]\}, 0 \leq i < p(n), -p(n) \leq j \leq p(n)+1, 0 \leq l \leq v$$

Para cualquier tiempo i, una celda j, y un símbolo s_l, si la cabeza de lectura-escritura no está escaneando dicha celda en dicho instante, y la celda j contiene s_l en tiempo i pero no en tiempo $i + 1$, entonces la cláusula basada en i, j, y l no se cumple. Así, las cláusulas $2(p(n) + 1)^2(v + 1)$ de este subgrupo realizan su misión.

El subgrupo restante de $G6$ garantiza que los cambios de una configuración a la siguiente cumplen las especificaciones de la función de transición δ para M. Para cada tupla de tamaño 4 $(i, j, k, l), 0 \leq i < p(n), -p(n) \leq j \leq p(n) + 1, 0 \leq k \leq r, y0 \leq l \leq v$, este subgrupo contiene las siguientes tres cláusulas:

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, H[i+1, j+\Delta]\}$$
$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, Q[i+1,k^{'}]\}$$
$$\{H[i,j], \overline{Q[i,k]}, S[i,j,l], S[i+1,j,l^{'}]\}$$

donde si $q_k \epsilon Q - \{q_Y, q_N\}$, entonces los valores de $\Delta, k^{'}$, y $l^{'}$ corresponden a $\delta(q_k, s_l) = (q_{k'}, s_{l'}, \Delta)$, y si además $q_k \epsilon \{q_Y, q_N\}$, entonces $\Delta = 0$, $k^{'} = k$ y $l^{'} = l$.

A pesar de que puede que cueste entenderlo, no es complicado ver que estas $6(p(n))(p(n)+1)(r+1)(v+1)$ cláusulas imponen la restricción deseada para satisfacer las asignaciones booleanas.

Ahora ya tenemos demostrado cómo se construyen todas estas seis cláusulas realizando transformaciones previamente establecidas. Si $x \epsilon L$, entonces existe un estado de aceptación para M con la entrada x de longitud $p(n)$ o menos, y este cómputo, provoca la satisfacción de todas las cláusulas en $C = G1 \cup G2 \cup G3 \cup G4 \cup G5 \cup G6$.

Por el contrario, la construcción de C es tal que cualquier asignación booleana satisfactoria para C debe corresponder con un estado de aceptación de M sobre x. Se sigue que $f_L(x)$ tiene una asignación booleana satisfactoria $\iff x \epsilon L$.

Todo lo que queda por demostrar es que, para cualquier lenguaje fijo L, $f_L(x)$ puede construirse a partir de x en un tiempo acotado por una función polinomial con $n = |x|$. Dado L, elegimos una Máquina de Turing No Determinista M particular que reconoce L en ese tiempo (no necesitamos encontrar esa NDTM en tiempo polinomial, ya que solo estamos demostrando que existe la transformación deseada f_L). Una vez que tenemos una NDTM M específica y un polinomio específico p, la construcción del conjunto U de variables y la colección C de cláusulas equivale a poco más que llenar los espacios en blanco en una fórmula estándar (aunque complicada). Los límites polinomiales serán conocidos una vez que se conozca la longitud $[f_L(x)]$, que está limitada por una función polinomial de n, donde longitud $[I]$ refleja la longitud de una cadena que codifica la instancia I bajo un esquema de codificación razonable.

Esta función de longitud razonable para SAT se da, por ejemplo mediante $|U| \cdot |C|$. Ninguna cláusula puede contener más de $2|U|$ literales (que son todos los literales que hay), y el número de símbolos necesarios para describir un literal individual sólo tiene que añadir $log|U|$ Factor, que puede ser ignorado cuando todo lo que está en cuestión es límite polinomial. Puesto que r y v se fijan de antemano y sólo pueden aportar factores constantes a $|U|$ y $|C|$, tenemos $|U| = O(p(n)^2)$ y $|C| = O(p(n)^2)$. Por lo tanto, longitud $[f_L(x)] = |U| \cdot |C| = O(p(n)^4)$, y está limitado por una función polinomial de n como se desee.

Así, la transformación f_l se puede calcular mediante un algoritmo en tiempo polinomial (aunque el polinomio particular al que obedece dependerá de L y de nuestras elecciones para M y p), y concluimos que para cada $L \epsilon \mathcal{NP}$, f_L es una transformación polinómica De L a SAT (técnicamente, por supuesto, de L a L_{SAT}). Por lo tanto, se afirma que SAT es $\mathcal{NP}$-completo. □

Bibliografía

[CIG] Michael R. Garey, David S. Johnson. Computers and Intractability A Guide

[Sipser] Michael Sipser. Introduction to the Theory of Computation (3rd ed.)