IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 10

1. SAT vs UNSAT, NP y coNP

A estas alturas, ya conocemos que SAT, CNF-SAT y 3CNF-SAT son problemas NP-completos: sabemos que están en NP, y son tan difíciles como cualquier problema en NP. Ahora, si queremos implementar un algoritmo para SAT en un computador, sólo sabemos hacerlo en tiempo exponencial, por ejemplo usando DPLL.

En esta clase vamos a responder mejor una pregunta que ya tocamos hace unas semanas, y que tiene que ver con entender cuando usar DPLL y cuando usar resolución:

¿Qué pasa con el problema de ver si acaso una fórmula NO es satisfacible?

Para SAT, el problema era fácil de mostrar que está en NP: dada una fórmula φ, podíamos adivinar, no determinísticamente, una valuación que la satisfaga, y luego llamar a EVAL para verificar que esa valuación satisface a φ.

Sin embargo, ¿qué pasa si queremos en vez verificar que la formula φ NO es satisfacible? Acá no parece fácil mostrar que está en NP, no queda claro qué tendríamos que adivinar para ver que una fórmula no es satisfacible. Podríamos probar todas las valuaciones, pero eso toma tiempo exponencial. ¿Entonces, Hay algo que podamos adivinar en este caso? ¿podemos demostrar que el problema de ver cuando una fórmula NO es satisfacible está en NP?

En esta clase vamos a ver que el problema de UNSAT, o verificar si una fórmula no es satisfacible, pertenece a una clase de complejidad llamada coNP, en donde el co viene de la palabra *complemento*. Vamos a usar esta clase para argumentar por qué los algoritmos que resuelven SAT usando resolución no deberían poder funcionar rápido, desde un punto de vista de complejidad computacional.

1.1. La clase coNP

Recuerda que las clases de complejidad son conjuntos de lenguajes, y que cada lenguaje a su vez es un conjunto de palabras. De esta forma, dado un lenguaje L sobre un alfabeto $\mathbf{A}$, nos podemos referir al complemento $\bar{L}$ de ese lenguaje: son todas las palabras sobre $\mathbf{A}$ que no están en L.

Definición. Un problema L sobre un alfabeto $\mathbf{A}$ pertenece a coNP si su complemento $\bar{L}$ pertenece a NP.

Definamos formalmente ahora los lenguajes UNSAT, CNF-UNSAT y 3CNF-UNSAT

$$\text{UNSAT} = \{w \mid w = C(\varphi) \text{ para una fórmula } \varphi \text{ en } L(P), \text{ y } \varphi \text{ no es satisfacible } \}.$$

$$\text{CNF-UNSAT} = \{w \mid w = C(\varphi) \text{ para fórmula } \varphi \text{ en } L(P) \text{ y CNF, y } \varphi \text{ no es satisfacible }\}.$$

$$\text{3CNF-UNSAT} = \{w \mid w = C(\varphi) \text{ para una fórmula } \varphi \text{ en } L(P), \text{ en CNF y con tres variables por cláusula, y } \varphi \text{ no es satisfacible }\}.$$

Proposición. UNSAT, CNF-UNSAT y 3CNF-UNSAT están en coNP.

Para la demostración, por definición, sólo tenemos que probar que los complementos de estos lenguajes están en NP. Hay una cuestión técnica que tiene que ver con que los complementos de estos lenguajes no son exactamente SAT, CNF-SAT y 3CNF-SAT, por que necesito ubicar también a todas las palabras que no codifican fórmulas.

Demostración. Sean L_1, L_2 y L_3 los lenguajes

$$L_1 = \{w \mid w \text{ no es la codificación de una formula en } L(P)\}.$$

$$L_2 = \{w \mid w \text{ no es la codificación de una formula CNF en } L(P)\}.$$

$$L_3 = \{w \mid w \text{ no es la codificación de una formula 3CNF en } L(P)\}.$$

Notar que el complemento de UNSAT es SAT $\cup$ L_1, y el complemento de CNF-UNSAT es CNF-SAT $\cup$ L_2, y lo mismo con 3CNF-SAT. En otras palabras, el complemento $\overline{\text{UNSAT}}$ es el lenguaje de todos los strings que o no son codificaciones de fórmulas, o que son codificaciones de fórmulas pero esas fórmulas son satisfacibles. Lo mismo para $\overline{\text{CNF-UNSAT}}$: este es el lenguaje de todos los strings que o no son codificaciones de fórmulas en CNF, o que son codificaciones de fórmulas en CNF pero esas fórmulas son satisfacibles.

Ahora, para mostrar que SAT $\cup$ L_1, CNF-SAT $\cup$ L_2 y 3CNF-SAT $\cup$ L_3 están en NP, notar que para aceptar L_1, L_2 y L_3 sólo basta con saber parsear la codificación, lo que se puede hacer con una máquina de turing en tiempo polinomial. [1] Así, la máquina no-determinista que acepta a SAT $\cup$ L_1 primero llama a la máquina para L_1, y si L_1 rechaza, llama a la máquina para SAT. Lo mismo para CNF-SAT $\cup$ L_2. Con esto mostramos que son aceptados por una máquina no-determinista que funciona en tiempo polinomial, y luego están en NP. Finalmente, mostramos que los complementos de UNSAT y CNF-UNSAT están en NP, por lo que UNSAT y CNF-UNSAT están en coNP. Con esto termina la demostración.

1.2. Problemas completos para coNP

El siguiente teorema es útil para relacionar UNSAT con otras clases:

Proposición. Si L es NP-completo, entonces su complemento $\bar{L}$ es coNP-completo.

Demostración. Suponemos que L es NP-completo. Por definición, $\bar{L}$ está en coNP. Para mostrar que $\bar{L}$ es coNP-hard, sea $\bar{L}'$ un lenguaje en coNP, de forma que L' está en NP.

[1] No somos tan formales en temas de codificaciones, pero pensar por ejemplo en como sabemos codificar/verificar formulas en CNF para implementar DPLL, o codificar/decodificar máquinas de turing. Normalmente sabemos si un string es una codificación solo con una pasada por el string.

Tenemos que mostrar una reducción desde $\bar{L}'$ a $\bar{L}$. Pero sabemos que hay una reducción f de L' a L, por que L es NP-hard. La misma reducción sirve! Como $w \in L'$ si y solo si $f(w) \in L$, tenemos también que $w \in \bar{L}'$ si y solo si $f(w) \in \bar{L}$. Con esto termina la demostración.

Así, tenemos una herramienta para poder mostrar algunos problemas coNP-completos. Por supuesto, también podemos ver la técnica que vimos la clase pasada: Si L es coNP-hard y hay una reducción polinomial de L a L', entonces L' también es coNP-hard.

El siguiente teorema sale de el teorema de Cook-Levin, pero tampoco lo vamos a demostrar acá. Nos da unos primeros lenguajes coNP-completos. La idea intuitiva es que para mostrar la dificultad de SAT y de CNF-SAT no nos importa si tiene a L_1 o L_2 o no, esos lenguajes son fáciles y por lo tanto no les prestamos mucha atención.

Proposición. SAT $\cup$ L_1, CNF-SAT $\cup$ L_2 y 3CNF-SAT $\cup$ L_3 son NP-completos.

De todo esto, obtenemos lo que queríamos: tanto UNSAT como CNF-UNSAT son coNP-completos. Ahora podemos hacer reducciones desde UNSAT y CNF-UNSAT para probar que otros problemas son coNP-duros también!

Ejercicio (propuesto). Muestra que el siguiente problema es coNP-completo:

$$\text{TAUT} = \{w \mid w = C(\varphi), \text{ para } \varphi \text{ una fórmula en L(P) y } \varphi \text{ es una tautología } \}.$$

1.3. Relaciones con otras clases

Ahora tenemos algunas herramientas para ver qué pasa con NP y coNP:

Proposición. Si P = NP, entonces NP= coNP.

Demostración. Supongamos que $P = \text{NP}$.

Mostramos primero que NP $\subseteq$ coNP. Para eso, sea L un lenguaje en NP (luego $\bar{L}$ está en coNP). Mostramos que $\bar{L}$ también está en NP, desde donde obtenemos que L está en coNP. Para mostrar que $\bar{L}$ está en NP, si $P = \text{NP}$ y L está en NP, entonces L está en ¶. luego hay una máquina de turing Mdeterminista que acepta a L, que se detiene en todas las entradas, y que toma tiempo polinomial. Pero como esta máquina es determinista, es fácil construir una máquina para aceptar a $\bar{L}$: simplemente tomamos los estados finales de M y los intercambiamos por los estados no finales. Más formalmente, supongamos que Q es el conjunto de estados de M y F es el conjunto de estados finales de M. Construimos a M' como una máquina idéntica a M, salvo que el conjunto F' de estados finales de M' es $Q \setminus F$. Por construcción verificamos que M' acepta al complemento $\bar{L}$ de L, desde donde obtenemos que $\bar{L}$ está en P y por tanto en NP.

Falta demostrar que coNP $\subseteq$ NP, pero eso se hace de forma idéntica al apartado anterior. Con esto termina la demostración.

Así, las clases NP y coNP sólo cobran sentido cuando $P \neq \text{NP}$, dado que solo ahí es cuando tenemos esta diferencia entre *adivinar* una valuación y *chequear todas las valuaciones*. ¿Qué pasa con P cuando NP = coNP? No sabemos mucho. Podría pasar, efectivamente,

que NP = coNP pero $P \neq$ NP. El siguiente resultado nos muestra como demostrar que NP = coNP. Esta pregunta también sigue abierta hasta el día de hoy.

Proposición. Si existe un lenguaje L en NP y además L es coNP-completo, entonces NP = coNP.

Demostración. Supongamos que tenemos un lenguaje $L \in$ NP y L coNP-completo.

Mostramos que NP $\subseteq$ coNP. Para eso, tomemos un L_1 arbitrario que pertenece a NP. Tenemos que mostrar que L_1 pertenece a coNP, o lo que es igual, que $\bar{L}_1$ pertenece a NP. Para eso, mostramos que hay una reducción desde $\bar{L}_1$ a L. Esa reducción existe por lo siguiente:

- L es coNP-completo.
- Si L_1 está en NP, $\bar{L}_1$ está en coNP.

Mostramos que coNP $\subseteq$ NP. Para eso, usamos el hecho que NP $\subseteq$ coNP, lo que implica que hay un lenguaje L' que es NP-completo y que además pertenece a coNP(en particular, como NP $\subseteq$ coNP, todos los lenguajes NP-completo ahora están en coNP). Sea entonces un lenguaje L_2 arbitrario que está en coNP. Tenemos que mostrar que L_2 está en NP, o lo que es igual, que $\bar{L}_2$ está en coNP. Para eso, basta una reducción desde $\bar{L}_2$ hasta L'. Esa reducción existe, por que

- L' es NP-completo.
- Si L_2 está en coNP, $\bar{L}_2$ está en NP.

Con esto termina la demostración.

2. Resolución

En lógica proposicional vimos que SAT era difícil de resolver. Un método era buscar todas las valuaciones y ver una a una si satisfacían a la fórmula, lo que dio la inspiración para definir la clase NP. ¿Pero qué hay de la otra forma de resolver SAT, por ejemplo con resolución? Si una formula no es satisfacible, ¿cuál es el largo de la demostración?

Considera el siguiente resultado (también demostrado por Cook).

Teorema. Existe una familia de fórmulas insatisfacibles $\varphi_1, \varphi_2, \varphi_3, \ldots, \varphi_n, \ldots$ en $L(P)$ tales que, para cada $i \geq 1$, el largo de una demostración por resolución que pruebe que $\varphi_i \vdash \bot$ es siempre mayor que 2^n.

Eso nos muestra inmediatamente que resolución toma tiempo exponencial, dado que hay fórmulas en las que el largo de una demostración es siempre exponencial.

Pero ¿qué pasa si le agregamos otras reglas a resolución? ¿Habrá un sistema deductivo que siempre use tiempo polinomial? Vamos a ver ahora que la complejidad computacional nos puede dar pistas sobre eso. Necesitamos un par de definiciones. En adelante, hablaremos de un *sistema deductivo* como algo parecido a resolución: alguna forma de ir derivando fórmulas adicionales hasta llegar a una contradicción.

Definición. Un sistema deductivo Π para $L(P)$ es *polinomialmente verificable* si para fórmula φ y para cada demostración D que verifique $\varphi \vdash_\Pi \bot$, verificar si D es una demostración válida de acuerdo a las reglas de Π toma tiempo polinomial en el tamaño de φ.

Definición. Un sistema deductivo Π para $L(P)$ es *polinomialmente acotado* si existe un polinomio S tal que, para cada fórmula φ en $L(P)$ existe una demostración de que $\varphi \vdash_\Pi \bot$ tal que (recuerde que $|\varphi|$ es el tamaño de φ):

- cada subfórmula deducida, asumida o usada en la demostración es de largo menor que $S(|\varphi|)$.
- El número total de reglas es menor que $S(|\varphi|)$.

Estas dos condiciones resumen lo que buscamos: alguna forma de mejorar resolución para que el largo de todas las demostraciones sea polinomial (polinomialmente acotado, segunda condición), para que las fórmulas que podemos deducir sean de tamaño polinomial (polinomialmente acotado, primera condición) y para que podamos verificar todo esto en tiempo polinomial (polinomialmente verificable).

Ahora vamos a argumentar lo siguiente (las demostraciones no están con lujo de detalles, pero buscamos que se entienda la idea):

a) La existencia de un sistema deductivo para $L(P)$ que sea correcto, completo, polinomialmente verificable y polinomialmente acotado implica que el problema de verificar si una fórmula es una contradicción está en NP.

b) La existencia de un sistema deductivo para $L(P)$ que sea correcto, completo, polinomialmente verificable y polinomialmente acotado implica que NP = coNP.

Para a), la idea es mostrar que una máquina de turing puede generar todas las demostraciones en tiempo exponencial, ya que la cantidad de estas es exponencial. Luego, una máquina de turing no-determinista puede adivinar esa demostración, y luego verificarla en tiempo polinomial, tal como hacíamos con EVAL para SAT.

Para b), notar que ahora podemos resolver UNSAT en NP: Para a) tenemos una máquina de turing que resuelve si acaso una fórmula es una contradicción. Como ya sabemos que UNSAT es coNP-completo, invocamos uno de los resultados que ya mostramos, y obtenemos que NP = coNP.

¿Cuál es la moraleja de todo esto?

Resolución va a tomar tiempo exponencial, y cualquier mejora que le hagamos a resolución debería seguir siendo exponencial. En efecto, encontrar un algoritmo de resolución que siempre tenga demostraciones polinomiales es equivalente a mostrar que NP = coNP, lo que consideramos falso.