IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 10

1. ¿CNF-SAT en PTIME?

Si bien aún no podemos mostrar que SAT o CNF-SAT están en PTIME, vamos a mostrar algo bastante fuerte: si CNF-SAT está en PTIME, entonces PTIME=NPTIME. Esto nos va a dar una luz: puede que SAT esté en PTIME, pero si lo está, es algo difícil de demostrar, pues equivale a mostrar que todas las palabras aceptadas por máquinas no-deterministas en tiempo polinomial pueden ser aceptadas por una máquina determinista en tiempo polinomial.

El camino para establecer el resultado de arriba es el siguiente.

- Definimos las nociones de hardness y completitud. La noción de hardness nos va a servir para capturar la intuición de que un problema es tan difícil como cualquier problema en ese clase. La noción de completitud nos sirve para establecer los problemas más difíciles de una clase.
- Vamos a mostrar después que si un problema es NP-completo y además está en PTIME, entonces PTIME=NP.
- Vamos a mostrar que CNF-SAT es NP-completo.

1.1. Hardness y completitud

El primer concepto es el de hardness, que da una noción de qué tan difícil es un problema con respecto a una clase. Intuitivamente, si un lenguaje L es hard para una clase de complejidad C, entonces L va a ser al menos tan difícil como cualquier lenguaje que pertenezca al conjunto C.

Definición. Sea C un conjunto de lenguajes que contiene a PTIME. Decimos que un lenguaje L es *hard* para C si para todo $L' \in C$ existe una reducción polinomial de L' a L.

El segundo concepto es el de completitud. Nos va a servir para saber la complejidad exacta de un problema.

Definición. L es completo para C si L es hard para C y a la vez $L \in C$.

Obviamente, si PTIME $\subsetneq C$ y L es completo para C entonces $L \notin$ PTIME.

Si L es hard para C, decimos que L es C-hard. Si es completo para C decimos que es C-completo.

Finalmente, podemos ver cómo nos sirven estas nociones, con la siguiente proposición.

Teorema. Sea C una clase de complejidad, tal que PTIME $\subseteq C$. Si un lenguaje L es C-completo y además L está en PTIME, entonces $C =$ PTIME.

Demostración. Para mostrar que $C =$ PTIME solo tenemos que probar que $C \subseteq$ PTIME (el otro lado lo tenemos del enunciado del Teorema). Sea entonces un lenguaje L_1 arbitrario que esté en C. Para mostrar que está en PTIME, tenemos que mostrar que hay una máquina de turing determinista, que funciona en tiempo polinomial, que acepta a L_1. La máquina se construye de la siguiente forma:

- Como L es C-hard, por definición hay una reducción polinomial f de L_1 a L. Supongamos que esa reducción funciona en tiempo n^k, es decir, hay una máquina M_f que la computa en ese tiempo.
- Como L está en PTIME, por definición hay una máquina de turing determinista M_L que acepta a L. Supongamos que esa máquina funciona en tiempo n^ℓ.
- La máquina para L_1 hace lo siguiente. Con input w, primero llama a M_f para computar $f(w)$. Luego le pasa $f(w)$ a M_L. Si M_L acepta a $f(w)$ entonces aceptamos a w, de lo contrario (si M_L no acepta a $f(w)$) no aceptamos a w.

Dado que (por definición de reducción) w pertenece a L_1 si y solo si $f(w)$ pertenece a L, tenemos que la máquina descrita arriba acepta a L_1. Más aún, lo hace en tiempo $((n^k)^\ell) = n^{k\ell}$, lo que es polinomial.

1.2. SAT es NP-completo, y 3-colorabilidad también!

El primer teorema no lo vamos a demostrar, pero es uno de los resultados más importantes del siglo pasado en computación. Ojo que ya sabemos cómo mostrar que SAT y CNF-SAT están en NP, lo que es mucho más difícil es mostrar que sonNP-hard!

Teorema (Cook-Levin). SAT, CNF-SAT y 3-CNF-SAT son NP-completos.

Ahora ya tenemos un lenguaje NP-completo, podemos pasar a demostrar otros lenguajes NP-completos. Para eso vamos a ayudarnos de un pequeño lema.

Lema. Sea C una clase de complejidad y L_1 y L_2 dos lenguajes. Si L_1 es C-hard, y además hay una reducción polinomial desde L_1 a L_2, entonces L_2 es C-hard también.

Demostración. Para mostrar que L_2 es C-hard, sea L_3 un lenguaje arbitrario en C. Tenemos que mostrar que hay una reducción polinomial desde L_3 a L_2.

Como L_1 es C-hard, hay una reducción, digamos f, desde L_3 a L_1. Además, sea h la reducción polinomial desde L_1 a L_2. Tenemos que componiendo h y f en $h \circ f$ obtenemos la reducción de L_3 a L_2.

Para demostrar que $h \circ f$ es una reducción polinomial, tenemos dos cosas que hacer:

- Mostrar que $h \circ f$ es polinomial. Esto sale directo de la composición de dos polinomios: Si f toma tiempo n^k y h toma tiempo n^ℓ, entonces $h \circ f$ toma tiempo $(n^k)^\ell = n^{k\ell}$.

- Mostrar que w pertenece a L_3 si y solo si $h(f(w))$ pertenece a L_2. Sabemos que w pertenece a L_2 si y solo si $f(w)$ pertenece a L_1. Además, para cualquier palabra v, v pertenece a L_1 si y solo si $h(v)$ pertenece a L_2. Haciendo $f(w) = v$, tenemos que $f(w)$ pertenece a L_1 si y solo si $h(f(w))$ pertenece a L_2, que era lo que buscábamos.

Como la elección de L_3 fue arbitraria, acabamos de mostrar que para cualquier lenguaje L en C hay una reducción desde L a L_2, lo que prueba que L_2 es hard.

Ahora si. Supongamos una codificación estándar $C(G)$ para grafos $G = (V, E)$ en strings binarios[1]. Definimos el lenguaje de los grafos 3-coloreables como

$$\text{3COL} = \{w \mid w = C(G) \text{ y } G \text{ es 3-coloreable}\}$$

Proposicion (Karp) . El lenguaje 3COL es NP-completo.

Demostración. Tenemos que mostrar que 3COL está en NP y además que 3COL es NP-hard. Para mostrar que 3COL está en NP, procedemos como en SAT:

- Construimos una máquina sobre el lenguaje $\{r, b, a, 0, 1\}$.
- Primero revisamos que w sea la códificación de un grafo $C(G)$, y escribimos $1^n 0 C(G)$, donde n es la cantidad de nodos de G.
- Luego *adivinamos la coloración*: invocamos una máquina de turing no-determinista que reemplaza 1^n con todos los posibles strings de largo n construibles con r, b o a. Esos strings son la codificación de una coloración.
- Enlazamos esa máquina con una máquina que chequee que esa coloración es válida (es decir que asigne colores distintos a los nodos conectados por una arista).

Para mostrar que $3COL$ es NP-hard, vamos a hacer una reducción de CNF-SAT a $3COL$. Esta reducción, con input $C(\varphi)$, computa la codificación $C(G)$ de un grafo G, de forma que φ es satisfacible si y solo si G es 3-coloreable. Esta reducción es parte de un video.

Listo! ya tenemos dos problemas NP-completos. No sabemos si hay algoritmos polinomiales para estos problemas, pero lo que si sabemos es que responder eso no es fácil: equivale a responder si acaso P = NP.

2. Ejercicios propuestos

El siguiente resultado es un poco más fuerte que lo que está en las notas, pero es igual de útil. La demostración es muy similar, por lo que es un buen ejercicio.

[1]Una estándar: si G tiene n nodos $v_1, \ldots, v_n$, entonces $C(G) = 1^n 0 v$, donde v es un string de tamaño n^2 y tal que la posición i-ésima de v es un 1 cuando la arista (v_p, v_ℓ) está en E, donde p y ℓ son los enteros en $[1, n]$ tal que $i = n * p + \ell$.

Teorema. Sean C y C' clases de complejidad tales que $\text{PTIME} \subseteq C \subseteq C'$. Si un lenguaje L es C'-completo y además L está en C, entonces $C = C'$.

Considera el lenguaje de 4-coloración: son todos los grafos en los que puedo pintar los nodos con uno de cuatro colores, y tal que ninguna arista tiene sus dos nodos con el mismo color. Muestra que el lenguaje 4COL es NP-completo:

$$\text{4COL} = \{w \mid w = C(G) \text{ y } G \text{ es 4-coloreable}\}$$