IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 8

1. Primeras clases de complejidad

1.1. Pasos y tiempo de ejecución de máquinas

Para una máquina de turing determinista M sobre un alfabeto A y donde M se detiene con cada input, y una palabra w, definimos $p_M(w)$ como la cantidad de pasos que ejecuta M con input w hasta que acepta o rechaza w. El *peor tiempo* para M con inputs de tamaño n se escribe como $t_M(n)$, y se define como el máximo de pasos que ejecuta M hasta que se detiene, sobre todas las palabras de largo n. Es decir, $t_M(n) = \text{máx}\{p_M(w) \mid |w| = n\}$.

Ejercicio. Construye una máquina de turing M determinista que acepte el lenguaje $L = \{w \in \{0,1\}^* \mid w \text{ empieza con 0 o } w \text{ termina con 0}\}$. Asegúrate que tu construcción satisface que $p_M(0101001010) = 1$ y $p_M(1101001010) = 10$. Concluye que $t_M(n) = n$

Definición. Sea $t : \mathbb{N} \to \mathbb{N}$ una función sobre los naturales. Decimos que un lenguaje L puede ser aceptado en tiempo t si existe una máquina de turing M tal que (1) M para en todas las entradas y (2) $t_M(n)$ es $O(t)$; en otras palabras, existe un entero n_0 y una constante real c tal que $t_M(n) \leq ct(n)$ para cada $n \geq n_0$. En este caso, decimos también que M funciona en tiempo t.

Ejercicio. Considera la función $f(x) = x$. La máquina n de arriba funciona en tiempo f. Generalmente usamos la notación $n, n^2, 2^n$, etc. para referirnos a $f(x) = x$, $f(x) = n^2$, $f(x) = 2^x$, respectivamente.

Definición. Para una función t, denotamos por $\text{DTIME}(t)$ al conjunto de todos los lenguajes que pueden ser aceptados en tiempo t. Por ejemplo, $\text{DTIME}(n)$ son todos los lenguajes que pueden ser aceptados en tiempo lineal.

1.2. PTIME y EXPTIME

Ahora podemos definir nuestras primeras clases de complejidad:

- $\text{PTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$,
- $\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k})$

Coloquialmente nos referimos a PTIME como el conjunto de problemas que pueden ser resueltos en tiempo polinomial, y a EXPTIME como el conjunto de problemas que pueden ser resueltos en tiempo exponencial.

1.3. SAT está en exptime

Veamos nuestro primer lenguaje en EXPTIME. Para poder hablar de SAT en complejidad computacional, vamos a usar una codificación de fórmulas proposicionales a el alfabeto $\Sigma = \{0, 1\}$. Para ir más rápido, vamos a asumir que esa codificación existe, y para una fórmula φ en $L(P)$ usamos $C(\varphi)$ para referirnos a esa codificación[1]. Definimos ahora el lenguaje:

$$\text{CNF-SAT} = \{w \in \{0,1\}^* \mid w = C(\varphi), \\ \text{para } \varphi \text{ fórmula de } L(P), \text{ en CNF, y tal que } \varphi \text{ es satisfacible}\}$$

Proposicion. CNF-SAT pertenece a la clase EXPTIME.

Para poder demostrar esto, necesitamos un resultado intermedio. Considera el siguiente lenguaje (usamos la notación $C(\tau)$ para referirnos a la codificación binaria de una valuación τ):

$$\text{CNF-EVAL} = \{w \in \{0,1\}^* \mid w = C(\tau)0000C(\varphi), \\ \text{para } \varphi \text{ fórmula de } L(P), \text{ en CNF, y } \tau \text{ una valuación tal que } \tau \models \varphi\}$$

Proposicion. CNF-EVAL pertenece a PTIME.

Demostración. Tenemos que mostrar que CNF-EVAL pertenece a DTIME(n^k), para algún k. Es decir, que CNF-EVAL puede ser aceptado por una máquina de turing que siempre toma $O(n^k)$ pasos para inputs de tamaño n. En la semana 5 vimos como resolver este algoritmo en PTIME, evaluando las proposiciones de la fórmula φ según la valuación τ, y de ahí propagando los $\neg$, $\vee$ y $\wedge$ hasta llegar a un 0 o un 1. Vimos también que podemos implementar un algoritmo que funciona en $O(n^2)$ y que hace esta evaluación, donde n es el tamaño de la fórmula. Cuando codificamos este algoritmo como una máquina de turing, cada operación del algoritmo se transforma en varios pasos de la máquina, pero esos pasos extra son constantes, no dependen de n. Eso nos dice que, para un input de tamaño n, la máquina M que primero decodifica el binario en $O(n^2)$ pasos y luego implementa el algoritmo de evaluación, toma $O(n^2)$ pasos en general: es decir, CNF-EVAL $\in$ DTIME(n^2) y por tanto CNF-EVAL pertenece a PTIME.

Ahora podemos demostrar la clase de complejidad de SAT:

Demostración que CNF-SAT $\in$ EXPTIME . Para mostrar que CNF-SAT está en EXPTIME, por la definición, tenemos que encontrar una máquina M que (1) acepta a SAT y que (2) siempre tome $O(2^{n^k})$ pasos con inputs de tamaño n, para algún k fijo. La máquina que acepte a SAT lo que tiene que hacer es lo siguiente.

1. Probar las $2^{|P|}$ valuaciones posibles.

[1] Una posible codificación sería usar algo como lo que usamos en la tarea, codificando una fórmula primero en un texto y después usando la codificación binaria de ese texto.

2. Para cada una de esas valuaciones τ, codificar τ e invocar la máquina que acepta a CNF-EVAL con τ y φ.

3. Si al menos una valuación funciona, retornamos true. De lo contrario, retornamos false.

Ya sabemos que CNF-EVAL pertenece a DTIME(n^2). Por lo tanto el costo mayor se lo lleva el loop con las $2^{|P|}$ valuaciones posibles. Recordemos que φ tiene tamaño n, lo que es siempre más grande que la cantidad de proposiciones en P. Luego probar todas las valuaciones está en $O(2^n)$, y por tanto la máquina que hace todo esto toma $O(2^n)$ pasos, y luego CNF-SAT está en EXPTIME.

1.4. Comparando PTIME y EXPTIME

Ejercicio. Muestra que PTIME $\subseteq$ EXPTIME. Para lograr esto, basta con mostrar que PTIME $\subseteq$ DTIME(2^n), lo que significa que cada máquina que funciona en tiempo polinomial también funciona en tiempo 2^n. ¿Como podrías demostrar esto?

En el curso de complejidad computacional mostramos que la contención es propia (nosotros no vamos a ver la demostración de este teorema, pero puedes buscarla en un libro de complejidad):

Teorema. Existe un lenguaje L que pertenece a EXPTIME pero no a PTIME.

2. Mostrando que problemas no pueden ser resueltos en tiempo polinomial

Como ya hemos discutido, sabemos que el problema de satisfacibilidad SAT pertenece a EXPTIME y tenemos todos los indicios de SAT no debería pertenecer a PTIME. Pero cómo podríamos demostrar esto? Nuevamente, vamos a utilizar el concepto de reducciones. Esta vez si, vamos a pedirles un poco más a las reducciones.

Decimos que una función f es computable en tiempo t si existe una máquina de turing que la computa y que funciona en tiempo t.

Definición. Dados lenguajes L_1 y L_2 sobre un alfabeto $\mathbf{A}$, decimos que L_1 puede ser reducido en tiempo polinomial a L_2 si existe una función f computable en tiempo polinomial y tal que para todo $w \in \mathbf{A}^*$ se tiene que $w \in L_1$ si y solo si $f(w) \in L_2$.

Nota que este es el mismo concepto de reducciones que usábamos antes, solo que ahora le pedimos también a f que sea computada en tiempo polinomial.

Ahora podemos usar reducciones polinomiales para obtener nuestro primer resultado sobre PTIME.

Teorema. Dados lenguajes L_1 y L_2 sobre un alfabeto $\mathbf{A}$, suponga que L_1 puede ser reducido en tiempo polinomial a L_2. Entonces:

- Si $L_2 \in$ PTIME entonces $L_1 \in$ PTIME.

- Si $L_1 \notin$ PTIME entonces $L_2 \notin$ PTIME.

Ejercicio. Demuestra el teorema (**Ayuda:** mira el Teorema de las notas de la semana 3, y la demostración de ese teorema).

Gracias a este teorema, sabemos que si queremos mostrar que L_2 no pertenece a PTIME, tendríamos que encontrar un lenguaje L_1 que no pertenece a PTIME y una reducción desde L_1 a L_2.