IIC2213 — Lógica para Ciencia de la Computación

# Notas Semana 9

## 1. La clase NP o NPTIME

### 1.1. Definiciones

Partimos por definir la noción de $t_M(w)$ para máquinas no-deterministas: el tiempo que se demora $M$ en procesar inputs de tamaño $w$.

Para una máquina de turing no-determinista $M$ sobre un alfabeto $A$ que para en todas las entradas y una palabra $w$ que es aceptada por $M$, definimos $p^*_M(w)$ como la cantidad de pasos que ejecuta $M$ con input $w$ hasta que acepta $w$, en la ejecución más corta dentro de todas las ejecuciones que aceptan a $w$ (es decir, todas las que con input $w$ se detienen en un estado final).

Nota que sólo definimos $p^*_M(w)$ para las palabras que acepta $M$.

El *peor tiempo* para $M$ con inputs de tamaño $n$ se escribe como $t_M(n)$, y se define esta vez como

$$t_M(n) = \text{máx}\big\{\{n\} \cup \{p^*_M(w) \mid w \in \mathbf{A}^*, |w| = n \text{ y } w \in L(M)\}\big\}.$$

Nota que ahora definimos $t_M(n)$ como el máximo entre $n$ y $p^*_M(w)$ para todas las palabras $w$ de largo $n$. La razón por la que hacemos esto (en vez de solo definir como el máximo de todos los $p^*_M(w)$ para $w \in L(M)$) es por que nos gusta que $T_M(n)$ sea continua, y para una máquina $M$ puede que para ciertos $n$ la máquina $M$ no acepte a ningún string de largo $n$, en cuyo caso $p^*_M(w)$ no estaría definido para ningún $w$ de ese largo.

**Definición**. Sea $t : \mathbb{N} \to \mathbb{N}$ una función sobre los naturales. Decimos que un lenguaje $L$ puede ser aceptado en tiempo $t$ por una máquina de turing no-determinista $M$ si (1) $L = L(M)$ 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 $n \geq n_0$.

**Definición**. Para una función $t$, denotamos por $\text{NTIME}(t)$ al conjunto de todos los lenguajes que pueden ser aceptados en tiempo $t$ por una máquina no-determinista.

Finalmente podemos definir la clase NPTIME:

- $\text{NPTIME} = \bigcup_{k\in\mathbb{N}} \text{NTIME}(n^k)$,

### 1.2. Relaciones con otras clases

**Teorema**.

- $\text{PTIME} \subseteq \text{NPTIME}$

- NPTIME $\subseteq$ EXPTIME

**Demostración**. El primer ítem sale automático del hecho que toda máquina no-determinista es también una máquina determinista.

Para el segundo ítem necesitamos refinar la transformación que hicimos para pasar de una máquina determinista a una no-determinista. Aunque los detalles se escapan un poco de este curso, podemos dar una intuición. La demostración sale del siguiente Lema. ¿Puedes ver por qué?

**Lema**. Sea $M$ una máquina no-determinista sobre un alfabeto $A$ y que para en todas las entradas, y tal que $t_M(n)$ es $O(n^k)$ para algún entero $k$ fijo. Entonces, es posible construir una máquina de turing determinista $M_D$ tal que (1) $L(M) = L(M_D)$, es decir que ambas máquinas aceptan el mismo lenguaje, y (2) $T_{M_D}(n)$ es $O(2^{(n^\ell)})$ para algún entero $\ell$.

Puedes revisar tus notas sobre la transformación de una máquina determinista en una no-determinista, y ver como sale el lema de esta transformación.

## 1.3. SAT y NP

Recordemos el lenguaje de CNF-SAT:

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

La semana pasada vimos que CNF-SAT estaba en EXPTIME, y que no sabíamos si estaba en PTIME. Ahora mostramos que también pertenece a la clase NPTIME!

**Proposicion**. El lenguaje CNF-SAT pertenece a NPTIME.

**Demostración**. Recuerda que mostramos que CNF-SAT pertenece a EXPTIME, y la idea era que podíamos iterar sobre todas las posibles valuaciones para la fórmula del input, hasta que alguna de esas valuaciones la satisfaga (luego de eso, ya sabemos que también tenemos otra máquina para resolver CNF-EVAL, el problema de saber si una valuación satisface a una fórmula, en tiempo polinomial).

Para mostrar que CNF-SAT está en NPTIME, vamos a hacer lo mismo, pero sustituyendo esta iteración por no-determinismo: nuestra máquina va a "adivinar" la valuación correcta, si esta existe. Para eso, vamos por parte.

**Observación**. Podemos construir, dado un número $n$ escrito como la palabra $1^n$, una máquina de turing no-determinista $M_b$ que tenga $2^n$ posibles ejecuciones, todas finales, pero tal que en cada una de esas ejecuciones la cinta de trabajo tiene, al final, un string binario de $n$ caracteres distinto.

La máquina es simple: tiene un estado $q_0$, y una única transición

$$\delta(q_0, 1) = \{(q_0, 1, \rightarrow), (q_0, 0, \rightarrow)\}.$$

**Construyendo la máquina para CNF-SAT**. Para esta máquina vamos a asumir que las codificaciones de valuaciones $C(\tau)$ a un conjunto de $n$ proposiciones es un string binario $a1, \ldots, a_n$, donde $\tau(p_i) = 0$ si y solo si el string tiene un 0 en la $i$-ésima posición, $a_i$.

Lo que hacemos es lo siguiente:

- Asumimos que tenemos como input la códificación de una fórmula $C(\varphi)$.
- Primero verificamos que el string tiene la forma $C(\varphi)$, y luego construimos el string $1^n BC(\varphi)$, donde $n$ es la cantidad de proposiciones en $\varphi$.
- Ahora llamamos a la máquina $M_b$. Se abren $2^n$ posibles ejecuciones.
- Cuando $M_b$ termina, la enlazamos con la máquina que resuelve CNF-EVAL, tomando como input el string computado por $M_b$ y $C(\varphi)$. Nota que el string escrito por $M_b$ es, en efecto, una valuación posible para las proposiciones de $\varphi$.

Sea $M_S$ la máquina descrita anteriormente. Para mostrar que acepta a CNF-SAT, tenemos que mostrar que $\varphi$ es satisfacible si y solo sí $M_S$ acepta con input $C(\varphi)$. Notar que no nos importa qué es lo que pasa con las fórmulas que no son satisfacibles, esto es clave en no-determinismo.

Si $\varphi$ es satisfacible, hay una valuación $\tau$ que lo satisface, y por tanto hay una ejecución de $M_b$ que produce el string $C(\tau)$. Esa ejecución termina aceptando, por que al llamar a la máquina que resuelve CNF-EVAL esta va a aceptar. Por otro lado, si esta máquina acepta, es por que hay al menos una ejecución para la que CNF-EVAL aceptó. Por construcción, el string computado por $M_b$ en esa rama de la ejecución va a satisfacer a $\varphi$.

Luego, la máquina funciona en tiempo polinomial: la construcción del string $1^n$ se puede hacer con un par de pasadas por $C(\varphi)$, $M_b$ toma $n$ pasos y CNF-EVAL toma tiempo $O(m^2)$, donde $m$ es el tamaño de $\varphi$. Sumando todo, la máquina funciona en tiempo $O(m + n + m^2)$.

Con esto construimos una máquina no-determinista que funciona en tiempo polinomial cuando acepta a un string, y tal que acepta a un string si y solo si ese string es $C(\varphi)$, con $\varphi$ satsifacible.