Autómatas y Lenguajes Formales
Tarea 4

Rodríguez Torres Víctor Fidel
Rubí Rojas Tania Michelle

29 de noviembre de 2018

1. Demuestra que

   - Si M es una Máquina de Turing determinista con k cintas que corre en tiempo $O(f(n))$, existe una Máquina de Turing determinista M' con una única cinta y equivalente a M que corre en tiempo $O(f(n)^2)$.

     Demostración. Supongamos que M es una Máquina de Turing determinista con k cintas que corre en tiempo $O(f(n))$. Construyamos una Máquina de Turing determinista M' con una única cinta que sea equivalente a M y corra en tiempo $O(f(n)^2)$.
     Sabemos que M' es capaz de simular a M gracias al siguiente teorema visto en clase:

     Teorema 1. *Toda Máquina de Turing multicinta tiene una Máquina de Turing con una única cinta equivalente.*

     Para revisar esta simulación, recordemos que M' usa una única cinta para representar los contenidos que hay en todas las k cintas de M. Estas cintas se almacenan consecutivamente, con las posiciones de las cabezas de M marcadas en las casillas apropiadas.
     Inicialmente, M' pone su cinta en el formato que representa todas las cintas de M y luego simula los pasos de M. Para simular un paso, M' escanea toda la información almacenada en su cinta para determinar los símbolos bajo las cabezas de cinta de M. Luego, M' hace otro escaneo sobre su cinta para actualizar el contenido de la cinta y las posiciones de la cabeza. Si una de las cabezas de M se mueve hacia la derecha sobre la parte previamente no leída de su cinta, M' debe aumentar la cantidad de espacio asignado a esta cinta. Lo hace desplazando una parte de su propia cinta una celda a la derecha.
     Ahora, analizamos esta simulación. Para cada paso de M, la máquina M' realiza dos escaneos sobre la parte activa de su cinta. El primero obtiene la información necesaria para determinar el siguiente movimiento y el segundo lo lleva a cabo. La longitud de la parte activa de la de la cinta de M' determina cuánto tiempo se tarda en escanearla, por lo que debemos determinar un límite superior en esta longitud. Para ello, tomamos la suma de las longitudes de las partes activas de las k cintas de M. Cada una de estas porciones activas tiene la longitud a lo más de $f(n)$ porque M utiliza $f(n)$ celdas de cinta en $f(n)$ pasos si la cabeza se mueve hacia la derecha en cada paso, e incluso menos si una cabeza se mueve siempre a la izquierda. Por lo tanto, una exploración de la parte activa de la cinta M' utiliza $O(f(n))$ pasos.
     Para simular cada uno de los pasos de M, M' realiza dos escaneos y posiblemente hasta k desplazamientos hacia la derecha. Cada uno utiliza el tiempo $O(f(n))$, por lo que el tiempo total para que M' simule uno de los pasos de M es $O(f(n))$. Ahora, limitamos el tiempo total utilizado por la simulación. La etapa inicial, donde M' pone su cinta en el formato adecuado, utiliza $O(n)$ pasos. Después, M' simula cada uno de los $f(n)$ pasos de M, utilizando $O(f(n))$ pasos, por lo que esta parte de la simulación utiliza $f(n)xO(f(n)) = O(f(n)^2)$ pasos. Por lo tanto, toda la simulación de M usa $O(n) + O(f(n)^2)$ pasos.
     Hemos asumido que $f(n) \geq n$ (ya que M ni siquiera podía leer la entrada completa en menos tiempo). Por lo tanto, el tiempo de ejecución de M' es $O(f(n)^2)$.

     □

- Si M es una Máquina de Turing no-determinista que corre en tiempo $O(f(n))$, existe una Máquina de Turing determinista M' equivalente a M que corre en tiempo $O(2^{f(n)})$.

  Demostración. Supongamos que M es una Máquina de Turing no-determinista que corre en tiempo $O(f(n))$. Construyamos una Máquina de Turing determinista M' que simule a M como en la prueba del siguiente teorema visto en clase:

  Teorema 2. *Toda Máquina de Turing no determinista tiene una Máquina de Turing determinista equivalente.*

  buscando el árbol de cálculo no determinista de M. En una entrada de longitud n, cada rama del árbol de cómputo no determinista de M el árbol tiene una longitud de a lo más $f(n)$. Cada nodo del árbol puede tener a lo más k hijos, donde k es el número máximo de opciones legales dadas por la función de transición de M. Por lo que, el número total de hojas en el árbol es a lo más $k^{f(n)}$.
  La simulación procede explorando primero la amplitud de este árbol. En otras palabras, visita todos los nodos en profundidad p antes de pasar a cualquiera de los nodos en profundidad $p+1$. El algoritmo del teorema mencionado anteriormente comienza en la raíz y viaja hacia un nodo cada vez que visita ese nodo. El número total de nodos en el árbol es menor que el doble del número máximo de hojas, por lo que lo limitamos por $O(k^{f(n)}$. El tiempo que se tarda en arrancar desde la raíz y viajar hasta un nodo es $O(f(n))$. Por lo tanto, el tiempo de ejecución de M' es $O(f(n)k^{f(n)}) = O(2^{f(n)})$.

  □

2. Demuestra que

- La clase P es cerrada bajo unión, concatenación. ¿Es cerrada bajo la operación de complemento? Justifica tu respuesta.

  Demostración. Sean L_1 y L_2 lenguajes. Veamos que la clase P es cerrada bajo unión. Supongamos que $L_1, L_2 \in P$. Entonces existen M_1 y M_2 Máquinas de Turing que corren en tiempo polinomial y deciden a L_1 y L_2, respectivamente. En particular, digamos que M_1 tiene el tiempo de ejecución $O(n^i)$ y M_2 tiene el tiempo de ejecución $O(n^j)$, donde n es la longitud de la entrada w y i, j son constantes.
  La Máquina de Turing M_3 que decide $L_1 \cup L_2$ es la siguiente:

  En la entrada w:

  i) Ejecuta M_1 con la entrada w. If M_1 acepta, entonces acepta.
  ii) Ejecuta M_2 con la entrada w. If M_2 acepta, entonces acepta.
  En otro caso, rechaza.

  Así, M_3 acepta la cadena w si y sólo si M_1 ó M_2 (o ambos) aceptan a w. El tiempo de ejecución de M_3 es $O(n^i) + O(n^j) = O(n^{max(i,j)})$, que sigue siendo polinomial en n, i.e., la suma de dos polinomios también es polinomial. Por lo tanto, la clase P es cerrada bajo la operación unión.

  Ahora, vemos que la clase P es cerrada bajo la concatenación. Supongamos que $L_1, L_2 \in P$. Entonces existen N_1 y N_2 Máquinas de Turing que corren en tiempo polinomial y deciden a L_1 y L_2, respectivamente. En particular, digamos que N_1 tiene el tiempo de ejecución $O(n^k)$ y N_2 tiene el tiempo de ejecución $O(n^l)$, donde n es la longitud de la entrada w y k, l son constantes.
  La Máquina de Turing N_3 que decide $L_1 \cup L_2$ es la siguiente:

  En la entrada $w = a_1a_2...a_n$, con cada $a_i \in \Sigma$ un símbolo:

  i) Para $i = 0, 1, 2, ..., n$ hacer
     - Ejecuta N_1 con la entrada $w_1 = a_1a_2...a_i$ y ejecuta N_2 con la entrada $w_2 = a_{i+1}a_{i+2}...a_n$. Si ambas máquinas N_1 y N_2 aceptan, entonces acepta.
  ii) Si ninguna de las iteraciones en el paso anterior acepta, entonces rechaza.

  La Máquina de Turing N_3 verifica todas las formas posibles de dividir la entrada w en dos partes: w_1, w_2; y verifica si la primera parte w_1 es aceptada por M_1 (i.e. $w_1 \in L_1$) y si la segunda parte w_2 es aceptada M_2 (i.e. $w_2 \in L_2$) , por lo que la concatenación de $w_1w_2 \in L_1L_2$. Supongamos que la entrada w en N_3 tiene una longitud $|w| = n$. El primer paso de N_3 se ejecuta a lo más $n+1$ veces. Cada vez que

se ejecuta el paso uno, N_1 y N_2 se ejecutan en cadenas w_1 y w_2 con $|w_1| \leq |w| = n$ y $|w_2| \leq |w| = n$. Así, ejecutando N_1 en w_1 toma un tiempo de ejecución de $O(n^k)$ y ejecutar M_2 en w_2 toma un tiempo de ejecución de $O(n^l)$, por lo que el paso uno corre en tiempo $O(n^k) + O(n^l) = O(n^{max(k,l)})$, que es polinomial en n. Como el paso 1 se ejecuta a lo más $n+1$ veces, obtenemos que el tiempo de ejecución de N_3 es $O(n+1)O(n^{max(k,l)}) = O(n^{1+max(k,l)})$, que es polinomial en n. Por lo que, el tiempo de ejecución de N_3 es polinomial en n. Por lo tanto, la clase P es cerrada bajo la concatenación.

Finalmente, veamos que la clase P es cerrada bajo el complemento. Supongamos que $L_1 \in P$. Entonces existe una Máquina de Turing S que corre en tiempo polinomial que decide L_1. La Máquina de Turing S_2 que decide S^c es la siguiente:

En la entrada w:

- Ejecuta S_1 con la entrada w.
  Si S_1 acepta, entonces rechaza. En otro caso, acepta.

La Máquina de Turing S_2 sólo genera lo contrario de lo que hace S_1, por lo que S_2 decide L_1^c. Por lo que, debido a que S_1 corre en tiempo polinomial, entonces el tiempo de ejecución de S_2 también es polinomial. Por lo tanto, la clase P es cerrada bajo la operación complemento.

□

- La clase NP es cerrada bajo unión y concatenación. ¿Es cerrada bajo la operación de estrella? Justifica tu respuesta.

Demostración. Sean L_1 y L_2 lenguajes. Veamos que la clase NP es cerrada bajo la operación unión. Supongamos que $L_1, L_2 \in NP$. Para cada $i = 1, 2$, sea $V(w, c)$ un algoritmo que, para una cadena w y un posible certificado c, verifica si c es realmente un certificado para $w \in L_i$. Así, $V_i(w, c) = 1$ si el certificado c comprueba $w \in L_i$, y $V_i(w, c) = 0$ en otro caso. Por hipótesis sabemos que $V_i(w, c)$ termina en tiempo polinomial $O(|w|^k)$, para alguna constante k.
Para mostrar que $L_1 \cup L_2 \in NP$ construiremos un verificador V_3 de tiempo polinomial para la unión de los dos lenguajes. Dado que un certificado c para la unión tendrá la propiedad de que $V(w, c) = 1$ ó $V_2(w, c) = 1$, podemos construir fácilmente un verificador $V_3(w, c) = V_1(w, c) \vee V_2(w, c)$. Claramente, entonces $w \in L_1 \cup L_2$ si y sólo si hay un certificado c tal que $V_3(w, c) = 1$. Notemos que el nuevo verificador V_3 se ejecutará en tiempo $O(2(|w|^k))$, que es polinomial. Por lo que, $L_1 \cup L_2 \in NP$. Por lo tanto, la clase NP es cerrada bajo la unión.

Ahora, veamos que la clase NP es cerrada bajo la operación concatenación. Supongamos que los lenguajes $L_1, L_2 \in NP$ con los verificadores V_1 y V_2 descritos en la prueba de la parte anterior. Nuevamente, nuestro objetivo es construir un verificador de tiempo polinomial $V_4(w, c)$ para una cadena x y el posible certificado c. Supongamos que $|w| = n$. Podemos definir que $V_4(w, c) = 1$ si y sólo si $c = a\#y\#z$, donde $\#$ es un nuevo símbolo, $a \in \{0, 1, ..., n\}$, y

$$V_1(w_1...w_a, y) = 1 \text{ y } V_2(w_{a+1}...w_n, z) = 1$$

Notemos que a especifica la posición donde la cadena original w debería dividirse en dos partes, y y, z son los certificados para las dos partes. El verificador V_4 se ejecutará en tiempo $O(|w|^k)$ desde $|w_1...w_a| \leq |w|$ y $|w_{a+1}...w_n| \leq |w|$. También, $V_4(w, x) = 1$ si y sólo si w pertenece a la concatenación entre L_1 y L_2. Por lo que la concatenación entre $L_1, L_2 \in NP$ está en NP. Por lo tanto, la clase NP es cerrada bajo la concatenación.

Finalmente, veamos que la clase NP es cerrada bajo la operación estrella. Supongamos que el lenguaje $L \in NP$. Entonces existe una Máquina de Turing no-determinista M tal que decide a L de manera no determinista en tiempo polinomial. La siguiente Máquina de Turing no-determinista M' decide a L^* de manera no determinista en tiempo polinomial:

$M'(w) =$

i) Si $w = \epsilon$, entonces acepta.

ii) En otro caso, de manera no determinista, particionar la cadena w de cualquier forma entre 1 a $|w|$ partes.

iii) Usar M para comprobar que cada una de las partes está en L. Si alguna parte no lo es, entonces rechaza.

Es sencillo comprobar que esto se ejecuta en tiempo polinomial. Un hecho que es clave es que en cada hilo de computación usamos M unicamente hasta $|w|$ veces (una cantidad lineal de veces). Como M' decide a L^* en tiempo polinomial de manera no determinista, entonces podemos concluir que la clase NP es cerrada bajo la operación estrella.

□

3. Muestra que los siguientes problemas están en P:

- $ALL_{AFD} = \{\langle M \rangle \mid M \text{ es un } AFD \text{ tal que } L(M) = \Sigma^*\}$

  Demostración. Sea M un AFD. Tenemos que averiguar si M acepta todas las cadenas de Σ^*. Notemos que es fácil comprobar si M rechaza alguna cadena. Para ello realizamos una búsqueda de amplitud o profundidad (BFS o DFS) en el AFD desde el estado inicial, lo cual nos toma tiempo polinomial. Si se puede alcanzar un estado de no aceptación en M, seguramente hay una cadena que no pertenece a $L(M)$, por lo que $L(M) \notin \Sigma^*$ y $\langle M \rangle \notin ALL_{AFD}$, y si sólo se puede acceder a los estados de aceptación en M, entonces $L(M) \in \Sigma^*$ y $\langle M \rangle \in ALL_{AFD}$.
  Como acabamos de describir un algoritmo de tiempo polinomial de manera determinista para decidir ALL_{AFD}, entonces podemos concluir que $ALL_{AFD} \in P$.

  □

- $\{\langle G \rangle \mid G \text{ es una gráfica simple tal que } G \text{ es conexa}\}$

  Demostración. Sea $L = \{\langle G \rangle \mid G \text{ es una gráfica simple tal que } G \text{ es conexa}\}$. Construyamos una Máquina de Turing M que decide a L en tiempo polinomial:

  $M =$ En la entrada $\langle G \rangle$:

  i) Elegir el primer nodo de G y marcarlo.

  ii) Repetir el siguiente paso hasta que no se marquen nuevos nodos:

  iii) Para cada nodo en G, marcarlo si es vecino de un nodo que ya está marcado.

  iv) Escanear todos los nodos de G para determinar si están marcados. Si lo están, acepta. En otro caso, rechazar.

  Ahora, veamos que este algoritmo se ejecuta en tiempo polinomial. El paso 1 se ejecuta en tiempo constante, es decir, $O(1)$. El paso 2 ejecuta a lo más $(n+1)$ repeticiones (ya que, a excepción de la última repetición, cada repetición marca al menos un nodo adicional). El paso 3 se ejecuta en $O(n^2)$ (ya que hay que examinar todos los vecinos del nodo actual para ver si alguno ha sido marcado). El paso 4 se ejecuta en tiempo lineal, es decir, $O(n)$ (ya que escanea todos los nodos). Por lo que, el algoritmo de ejecuta en $O(n^3)$.
  Como hemos mostrado un algoritmo que se ejecuta en tiempo polinomial de manera determinista que decide a L, entonces podemos concluirque $L \in P$.

  □

4. Demuestra que los siguientes problemas están en NP. Debes usar la definición de certificados y la definición que ocupa Máquinas de Turing no-deterministas:

- $\{\langle n \rangle \mid n \in \mathbb{N} \text{ y existen } a, b \in \mathbb{N} \text{ números primos tales que } n = ab\}$
  - Usando definición de certificado.

    Demostración. Sea V una MT total con entrada $\langle n, c \rangle$ donde $n \in \mathbb{N}$ y potencialmente puede estar en A,el lenguaje, además c está descrito como $c = \langle a, b \rangle$, de tal forma que $a, b \in \mathbb{N}$.

    V hace lo siguiente:

1) Si es verdad que se cumple $esPrimo(a)$ y si se cumple $esPrimo(b)$ se pasa a 2), en otro caso se devuelve false.
2) Se devuelve true si se cumple que $n = ab$, se devuelve false en otro caso.

Como se hizo la certificación en tiempo $O(n)$, llegamos a que $A \in NP$.

□

- Usando definición de MT no determinista

Demostración. Sea N una MT no determinista con entrada $\langle n \rangle$.

N hace los siguiente:

1) Asigna a a un número natural seleccionado al azar en el rango $[1, \infty)$.
2) Asigna a b un número natural seleccionado al azar en el rango $[1, \infty)$.
3) Sea $c = \langle a, b \rangle$, ejecuta $V\langle a, b \rangle$, si V acepta entonces N devuelve true, en otro caso devuelve false.

Como dimos una MT no determinista que decide a A en tiempo $O(n)$, llegamos a que $A \in NP$.

□

- $\{\langle S, S' \rangle \mid S$ es un conjunto finito de números enteros, $S' \subseteq S$ y $\sum_{x \in S'} x = 0\}$
  - Usando definición de certificado.

Demostración. Sea V una MT total con entrada $\langle n, c \rangle$, donde $n = \langle s, s' \rangle$, con $s, s' \subseteq \mathbb{Z}$ y $c = 0$.
V hace lo siguiente:

1) Verifica que se cumpla que $s' \subseteq \mathbb{Z}$, si no se cumple se devuelve false.
2) Recorre cada elemento en s' y lo suma con los demás en s', si la suma final es igual a c devuelve true, en otro caso devuelve false.

Como se hizo la certificación en tiempo $O(n)$, llegamos a que $A \in NP$.

□

- Usando definición de MT no determinista

Demostración. Sea N una MT no determinista con entrada $\langle s, s' \rangle$.

N hace los siguiente:

1) Define $a = |s|$ y a $b = |s'|$.
2) Toma a números al azar en $\mathbb{Z}$ y los guarda en el conjunto S_1
3) Toma b números al azar en $\mathbb{Z}$ y los guarda en el conjunto S_2
4) Corre V con entrada $\langle \langle s_1, s_2 \rangle, c \rangle$ y si V acepta N devuelve true, en otro caso devuelve false.

Como dimos una MT no determinista que decide a A en tiempo $O(n^2)$, llegamos a que $A \in NP$.

□

- $\{\langle G, G' \rangle \mid G$ y G' son gráficas simples isomorfas$\}$, donde G y G' son isomorfas si existe una biyección $f : V(G) \to V(G')$ tal que $(v_1, v_2) \in E(G) \iff (f(v_1), f(v_2)) \in E(G')$.
  - Usando definición de certificado.

Demostración. Sea V una MT total con entrada $\langle n, c \rangle$, donde $n = \langle G_1, G_2 \rangle$, siendo G_1 y G_2 gráficas, además $c = f : V(G_1) \to V(G2)$.
V hace lo siguiente:

1) Para cada vertice v en G_1 aplicar $f(v)$.
2) Cada vez que se aplica $f(v)$ se reviza si los vecinos de v son iguales a los vecinos de $f(v)$, si es falso que los vecinos de ambos vértices son iguales, se devuelve false, en otro caso se siguen revizando los vértices de G_2
3) Al terminar de iterar sobre los vértices de G_1 se devuelve true.

Como se hizo la certificación en tiempo $O(n)$, llegamos a que $A \in NP$.

□

- Usando definición de MT no determinista

  Demostración. Sea N una MT no determinista con entrada $\langle G_1, G_2 \rangle$.

  N hace los siguiente:

  1) Se definen
     $A =$ el conjunto de todos los vértices de G_1
     $B =$ el conjunto de todos los vértices de G_2

  2) Genera al azar una función $f : A \to B$
  3) Ejecuta V con entrada $\langle \langle G_1, G_2 \rangle, f \rangle$, si v devuelve true, entonces N devuelve true, en otro caso devuelve false.

  Como dimos una MT no determinista que decide a A en tiempo $O(n)$, llegamos a que $A \in NP$.

□

5. Demuestra que si $A \leq_P B$ y C es cualquier lenguaje finito, entonces $A \cup C \leq_P B \cup C$.

   Demostración. Si $w \in A \cup C$, entonces tenemos dos casos

   - Si $w \in A$, entonces se aplica $f(w) \Leftrightarrow f(w) \in B \cup C$
   - Si $w \in C$, entonces se aplica la identidad $i(w) = w \Leftrightarrow w \in C \Leftrightarrow w \in B \cup C$

   En ambos casos llegamos a los dos lados de la reducción por medio de las funciones computables f e i, por lo tanto $A \cup C \leq_P B \cup C$

   □

6. Muestra que $SAT \leq_P 3 - SAT$.

   Demostración. Se dará el algoritmo para pasar de una formula en SAT a una formula equivalente en $3-SAT$

   Sea $C_1, C_2, ..., C_n$ una formula en SAT, entonces revisamos cada cláusula de la siguiente manera:

   - Si C_i contiene tres variables, entonces no la modificamos.
   - Si $C_i = z$ está compuesta por una sola variable, entonces agregamos dos nuevas variables x y y y se cambia C_i por $(z \vee x \vee y), (z \vee x \vee \neg y), (z \vee \neg x \vee y), (z \vee \neg x \vee \neg y)$
   - Si $C_i = z \vee w$, entonces se añade una x y se cambia C_i por $(z \vee w \vee x), (z \vee w \vee \neg x)$
   - Si $C_i = z_1 \vee z_2 \vee ... \vee z_k$ con $k > 3$ variables, entonces se añade $k - 3$ nuevas variables $x_1, x_2, ..., k_{k-3}$ y se cambia C_i por

   $$(z_1 \vee z_2 \vee x_1),$$

   $$(\neg x_1 \vee z_3 \vee x_2)$$
   $$(\neg x_2 \vee z_4 \vee x_3)$$
   $$.$$
   $$.$$
   $$.$$
   $$\neg x_{k-4} \vee z_{k-2} \vee x_{k-3}$$
   $$\neg x_{k-3} \vee z_{k-1} \vee x_{k-k}$$

   □

7. Demuestra que los lenguajes $\varnothing$ y Σ^* están en NP pero no son $NP-$completos.

- P.d. el lenguaje $\varnothing \in NP$ y no es $NP-$completo.

  Demostración. Sea M una MT total con entrada $\langle w \rangle$

  M se devuelve true en caso de que $|w| = 0$, en otro caso devuelve false.

  La complejidad de M es $O(1)$ por lo que el lenguaje $\varnothing$ está en P y como $P \subseteq NP$ llegamos a que el lenguaje vacío está en NP

  Ahora para demostrar que $\varnothing$ no es $NP-$Completo se hará la demostración por reducción al absurdo, así que supongamos que el lenguaje $\varnothing$ es $NP-$Completo.

  Como el lenguaje $\varnothing$ es $NP-$ Completo $\Rightarrow \varnothing \leq_P A$, para algún problema A $NP-$Completo, pero como no hay ninguna cadena en el lenguaje $\varnothing$, entonces ninguna cadena está en el lenguaje $\varnothing$, por lo tanto no se puede reducir el lenguaje $\varnothing$ a un problema $NP-$Completo.

  □

- P.d. el lenguaje $\Sigma^* \in NP$ y no es $NP-$completo.

  Demostración. Sea M una MT total con entrada $\langle w \rangle$

  M se devuelve true en caso de que $|w| > 0$, en otro caso devuelve false.

  La complejidad de M es $O(1)$ por lo que el lenguaje Σ^* está en P y como $P \subseteq NP$ llegamos a que el lenguaje vacío está en NP.

  Ahora para demostrar que Σ^* no es $NP-$Completo se hará la demostración por reducción al absurdo, así que supongamos que el lenguaje Σ^* es $NP-$Completo.

  Como el lenguaje Σ^* es $NP-$ Completo $\Rightarrow \Sigma^* \leq_P A$, para algún problema A $NP-$Completo, pero como es válido que cualquier cadena esté en el lenguaje Σ^*, entonces no se puede dar ninguna cadena que no esté en el lenguaje Σ^*, por lo tanto no se puede reducir el lenguaje Σ^* a un problema $NP-$Completo.

  □