Universidad Simón Bolívar
Teoría de la Computación
Tarea III

Fabiola Di Bartolo
Carnet: 09-87324

25 de marzo de 2010

Los siguientes ejercicios fueron resueltos consultando los libros [1] [2] [3] [4]

1. **Ejercicio 8.8.** Sea $EQ_{REX} = \{\langle R, S\rangle \mid R \text{ y } S \text{ son expresiones regulares equivalentes }\}$. Mostrar que $EQ_{REX} \in PSPACE$.

   Por el Teorema 1.54, en particular el Lema 1.55, se puede convertir una expresión regular a un autómata finito no determinístico (NFA) en tiempo polinomial y el cual es lineal en el tamaño de la expresión regular. Por lo tanto, se convertirán las expresiones regulares de entrada R y S en NFAs para realizar la demostración.

   Por otro lado, se sabe que $PSPACE$ es cerrado bajo el complemento y es equivalente a $NPSPACE$. Inicialmente, demostraremos que decidir si dos expresiones regulares no son equivalentes es $NPSPACE$, es decir, $\overline{EQ_{REX}} \in NPSPACE$. Para ello, construiremos una máquina de Turing no determinística que corra en espacio polinomial, simule el comportamiento de los dos autómatas y verifique si son equivalentes o no.

   Sean $N_R = (Q_R, \sum, \delta_R, q_{0R}, F_R)$ y $N_S = (Q_S, \sum, \delta_S, q_{0S}, F_S)$ los NFAs correspondientes a las expresiones regulares R y S respectivamente, antes de construir la máquina de Turing, se tiene que tener en cuenta lo siguiente para la simulación de estos autómatas:

   - ⋆ Para simular el funcionamiento de los autómatas, se colocarán marcadores en los estados para indicar en que estados se encuentran los autómatas en cada paso.
   - ⋆ Una configuración de un NFA será un subconjunto de los estados de ese NFA. Si una configuración se repite al simular un NFA en alguna entrada, la parte de la entrada procesada que generó la

repetición en los estados puede eliminarse sin que se vea afectada la salida. Por lo tanto, basta con probar con palabras que no generen configuraciones repetidas.

- ⋆ Como la función de transición de un NFA es $\delta : Q \times \sum_{\epsilon} \rightarrow \mathcal{P}(Q)$, es decir, que de un estado con la función de transición, se puede tener más de una opción para el siguiente estado al cual visitar, esto implica que hay en total 2^q ($q =\mid Q \mid$) subconjuntos diferentes de estados posibles o configuraciones. Por lo tanto, para indicar que existe una palabra que no es aceptada por un NFA, es suficiente considerar palabras de a lo más 2^q de longitud.
- ⋆ Debido a que se quiere probar si dos autómatas son inequivalentes, es suficiente con considerar palabras de a lo más $2^{q_R+q_S}$ de longitud (donde $q_R =\mid Q_R \mid$ y $q_S =\mid Q_S \mid$), ya que esa es la cantidad de formas distintas que existen para colocar marcadores en los estados de N_R y N_S y por lo tanto, es lo que se necesita para indicar si existe una palabra que un autómata rechaza y el otro no.

$M =$ En entrada $\langle R, S \rangle$, donde R y S son expresiones regulares.

1. Convertir R y S en sus equivalentes NFAs N_R y N_S
2. Colocar un marcador en los estados iniciales de N_R y N_S.
3. Repetir $2^{q_R+q_S}$ veces.
    4. Seleccionar no determinísticamente un símbolo de entrada $a \in \sum$ y mover las posiciones de los marcadores en los estados para simular la lectura del símbolo a.
5. Si en el paso anterior, se colocó un marcador en uno de los estados de aceptación de N_R, pero no en los de N_S o viceversa, ACEPTAR. Sino, RECHAZAR.

Si alguno de los dos NFAs entra en estado de aceptación, pero el otro no, M acepta porque existe una palabra que es aceptada por uno pero no por el otro y por lo tanto, los NFAs son inequivalentes y en consecuencia, las expresiones regulares que generaron estos NFAs son distintas.

En cuanto a la complejidad en espacio, para el paso 1, en la traducción de los autómatas se necesita espacio lineal en función de la entrada para almacenarlos. En el paso 3, se necesita llevar la cuenta del número de iteraciones y esto ocupa espacio logarítmico. En el paso 4, sólo se almacena el símbolo seleccionado, no toda la palabra con la que se esta probando, esto ocupa también espacio logarítmico. M corre en espacio $O(n)$. Por lo tanto, como $\overline{EQ_{REX}} \in NPSPACE$, por el Teorema de Savitch $\overline{EQ_{REX}} \in PSPACE$ y porque $PSPACE$ es cerrado bajo el complemento $EQ_{REX} \in PSPACE$

2. **Ejercicio 8.11.** Mostrar que si todo lenguaje $NP - hard$ es también $PSPACE - hard$, entonces $PSPACE = NP$.

($PSPACE \supseteq NP$). Por la Definición 8.6 y el Teorema 8.5 de Savitch se tiene que $PSPACE = NPSPACE$ y $NP \subseteq NPSPACE$, entonces $NP \subseteq PSPACE$.

($PSPACE \subseteq NP$). Por el Ejemplo 8.3 sabemos que SAT está en $SPACE(n)$ y por lo tanto en $PSPACE$, como por el Teorema 7.37 se sabe que SAT es $NP - completo$ y por la Definición 7.34 de $NP - completo$, SAT está NP y en $NP - hard$ y asumiendo la hipótesis, SAT es $PSPACE - hard$, por la Definición 8.8 de $PSPACE - completo$, SAT está en $PSPACE - completo$. Entonces, como SAT está en $PSPACE - completo$, cualquier lenguaje A en $PSPACE$ puede reducirse a SAT en tiempo polinomial, pero SAT también está en NP, por lo tanto A está en NP también. En consecuencia, $PSPACE \subseteq NP$.

Se demostró asumiendo la hipótesis que $PSPACE \supseteq NP$ y $PSPACE \subseteq NP$ lo que implica que $PSPACE = NP$

3. **Ejercicio 8.17.** Sea A el lenguaje de paréntesis balanceados. Por ejemplo (()) y (()(())) () están en A, pero)(no está. Mostrar que A está en L.

   Considere la siguiente máquina de Turing M:

   M = En entrada w, donde w es una palabra en $\sum = \{(,)\}$.

   1. Inicializar el contador de paréntesis izquierdos abiertos. $c := 0$
   2. Por cada s símbolo en w (de izquierda a derecha).
      3. Si s es un '(', $c := c + 1$. Sino, $c := c - 1$.
      4. Si $c < 0$, RECHAZAR.
   5. Si $c = 0$, ACEPTAR. Sino, RECHAZAR.

   La máquina de Turing M tomando la cinta de entrada de sólo lectura, cuenta el número de paréntesis que no han sido cerrados y por cada paréntesis que cierre descuenta uno al contador. Si en algún momento hay más paréntesis derechos que izquierdos, M rechaza porque la expresión no está balanceada ni lo estará, de la misma forma, si al final no se cerraron todos los paréntesis, es decir, hay más paréntesis izquierdos que derechos, M rechaza. Para esto, lo que se necesita es almacenar el contador, el cual, ocupa espacio logarítmico en función de la entrada $O(logn)$, ya que a lo más existen n paréntesis abiertos, donde $n =\mid w \mid$. Por lo tanto, M corre en espacio logarítmico $A \in L$.

4. **Ejercicio 8.25.** Un grafo no dirigido es bipartito si sus nodos pueden estar divididos en dos conjuntos tal que todas las aristas vayan de un nodo de un conjunto a otro nodo en el otro conjunto. Mostrar que un grafo es bipartito si y sólo si no contiene un ciclo con un número impar de nodos.

   Sea $BIPARTITE = \{\langle G\rangle \mid G$ es un grafo bipartito $\}$. Mostrar que $BIPARTITE \in NL$.

Inicialmente se demostrará que un grafo $G = (V, E)$ es bipartito si y sólo si no contiene un ciclo de tamaño impar. Y posteriormente se demostrará que $BIPARTITE \in NL$.

(G es bipartito $\Rightarrow$ no contiene un ciclo de tamaño impar). Demostración por contradicción, sean A y B los conjuntos disjuntos de nodos que forman al grafo bipartito. Suponemos que existe un ciclo con un número impar de nodos formado por los nodos $u_1, u_2, .., u_{2k}, u_{2k+1}$ $(u_i \in V)$ para cualquier k, donde $< u_1, u_2 >, < u_2, u_3 >, ..., < u_{2k}, u_{2k+1} >, < u_{2k+1}, u_1 >$ $(< u_i, u_j > \in E)$ son las aristas del ciclo, entonces por definición de grafo bipartito (no pueden existir aristas entre nodos pertenecientes al mismo conjunto) si $u_1 \in A$, $u_2 \in B$, $u_3 \in A$ y así sucesivamente, $u_{2k} \in B$, $u_{2k+1} \in A$. Sin embargo, $u_{2k+1} \in A$, pero ya $u_1 \in A$ y existe la arista $< u_{2k+1}, u_1 >$, lo cual es una contradicción, la arista $< u_{2k+1}, u_1 >$ no puede existir, por lo tanto no se pueden tener ciclos de tamaño impar. Nótese que si el grafo tuviese ciclos de tamaño par seguiría siendo bipartito.

(Grafo G es bipartito $\Leftarrow$ no contiene un ciclo de tamaño impar). Se forman los conjuntos disjuntos A y B del grafo bipartito a partir de un grafo G sin ciclos de tamaño impar. Por cada componente conexa C en G:

- $\star$ Si $\mid C \mid = 1$, el nodo u perteneciente a C se agrega en cualquiera de los dos conjuntos A ó B.
- $\star$ Si $\mid C \mid > 1$, se toma un nodo u perteneciente a C, se coloca en A y se marca, se buscan los adyacentes v_i de u que no estén marcados y se colocan en el conjunto opuesto, en este caso B y se marcan, luego para cada v_i se buscan sus adyacentes x_i se agregan en el conjunto opuesto, A y se marcan, y continua, hasta que no queden nodos desmarcados. Nótese que dos nodos u y v adyacentes se almacenarán en conjuntos distintos, de lo contrario, se tendría una arista entre ellos que generaría un ciclo de tamaño impar. Al final, se tendrán los conjuntos disjuntos A y B de nodos que forman al grafo bipartito G.

Por lo tanto, si:

$ODD-CYCLE = \{\langle G\rangle \mid G \text{ tiene al menos un ciclo de tamaño impar}\}$

Entonces:

$BIPARTITE = \overline{ODD-CYCLE}$

($BIPARTITE \in NL$). Por el Teorema 8.27 $NL = CO-NL$, es decir que $\overline{BIPARTITE} \in NL \Leftrightarrow$BIPARTITE $\in NL$. Sabiendo que NL es cerrado bajo el complemento y que $BIPARTITE = \overline{ODD-CYCLE}$, se construye la siguiente máquina de Turing no determinística para $ODD-CYCLE$.

$M =$ En entrada $\langle G\rangle$.

1. Inicializar el contador de nodos, $c := 0$
2. Seleccionar no determinísticamente el nodo inicial y actualizar el apuntador v_{ini} con este nodo.
3. Encontrar no determinísticamente un nodo sucesor y asignar el apuntador v_{suc} a él y $c := c + 1$
4. Si no se encontró sucesor, RECHAZAR.
5. Si $v_{ini} = v_{suc}$, ACEPTAR.
6. Mientras $c \leq \mid V \mid$
    5. Si $v_{ini} = v_{suc}$ y *cesimpar*, ACEPTAR.
    6. Encontrar no determinísticamente un nodo sucesor y asignar el apuntador v_{suc} a él y $c := c + 1$
    7. Si no se encontró sucesor, RECHAZAR.
7. Si no aceptó antes, RECHAZAR.

M corre en espacio logarítmico en cualquier rama, ya que únicamente almacena en la cinta de trabajo en cada paso, los apuntadores al nodo inicial (v_{ini}), al nodo sucesor (v_{suc}) y el contador de nodos visitados (c) y todos estos ocupan espacio logarítmico en función de la entrada. M decide a $ODD-CYCLE$, por lo tanto $ODD-CYCLE \in NL$,

$\overline{ODD-CYCLE} \in NL$ y $BIPARTITE \in NL$.

5. **Ejercicio 8.27.** Un grafo dirigido es fuertemente conexo si existe un camino entre cada par de nodos.

   $STRONGLY-CONNECTED = \{\langle G\rangle \mid G$ *es un grafo dirigido fuertemente conexo* $\}$.

   Mostrar que $STRONGLY-CONNECTED$ es $NL-$*completo*.

   Un lenguaje está en $NL-$*completo* si está en NL y cualquier otro lenguaje puede reducirse en tiempo polinomial a éste.

   ($STRONGLY-CONNECTED \in NL$). Como $NL = Co-NL$ por el Teorema 8.27, se probará entonces que $\overline{STRONGLY-CONNECTED} \in NL$ y para ello, se utilizará el lenguaje $PATH$:

   $PATH = \{\langle G, s, t\rangle : G$ *es un grafo dirigido que tiene un camino entre* s *y* t $\}$

   Por el Teorema 8.25 se sabe que $PATH \in NL-$*completo*, por lo tanto, se puede utilizar en la construcción de una máquina no determinística en espacio logarítmico para $\overline{STRONGLY-CONNECTED}$, con lo cual se tendrá que $\overline{STRONGLY-CONNECTED} \in NL$ y en consecuencia $STRONGLY-CONNECTED \in NL$.

   Sea M_P la máquina que decide a $PATH$ en espacio logarítmico, la siguiente máquina decide a $\overline{STRONGLY-CONNECTED}$:

   $M =$ En entrada $\langle G\rangle$.

   1. Seleccionar no determinísticamente dos vértices cualesquiera s y t de G.
   2. Simular M_P en entrada $\langle G, s, t\rangle$ si acepta, RECHAZAR. Si rechaza, ACEPTAR.

Si M_P rechaza, significa que no encontró un camino entre s y t, por lo que el grafo no sería fuertemente conexo, de lo contrario se tendrían caminos entre cada par de nodos, así que M acepta. M decide a $\overline{STRONGLY-CONNECTED}$ en espacio logarítmico, ya que sólo necesita almacenar los apuntadores a los nodos s y t que ocupan espacio logarítmico en función de la entrada, y también se sabe que M_P corre en espacio logarítmico porque chequea si existe el camino sin guardarlo completo, esto es, sólo llevando un apuntador con el último nodo visitado en el camino. Por lo tanto, $\overline{STRONGLY-CONNECTED} \in NL$ lo que implica que $STRONGLY-CONNECTED \in NL$

($L \in NL \Rightarrow L \leq_L STRONGLY-CONNECTED$). Todo lenguaje $L \in NL$ debería ser en espacio logarítmico reducible a $STRONGLY-CONNECTED$, por lo tanto, la demostración se realizará con $PATH$ ya que se sabe que como es $NL-completo$, todo lenguaje $L \in NL$ es reducible a él. Por la definición 8.21, encontrar una reducción $PATH \leq_L STRONGLY-CONNECTED$ es equivalente a hallar un transductor en espacio logarítmico que compute para todo $w \in PATH$ ssi $f(w) \in STRONGLY-CONNECTED$.

La idea para esta demostración, es construir un grafo fuertemente conexo $G' = (V, E')$ a partir de un grafo $G = (V, E)$. Suponga que G al menos tiene un camino entre dos nodos s y t (donde $s, t \in V$) . Si se se agregan arcos desde todos los nodos $x_i \in V$ (tal que $x_i \neq s$ y $x_i \neq t$) arcos hasta s, se tendrán caminos desde todos los nodos hasta t, notése que si el camino de s a t no existe, tampoco existirían estos caminos ya que parte del tramo sería el camino de s a t. Análogamente, se agregan arcos desde t a todos los nodos $x_i \in V$ (tal que $x_i \neq s$ y $x_i \neq t$), con esto, se tendrán caminos desde t hacia todos los nodos, de la misma forma, si el camino de s a t no existe, no existirían los de t a s. Formalmente, $E' = E \cup \{\langle x_i, s\rangle \mid x_i \in V - \{s, t\}\} \cup \{\langle t, x_i\rangle \mid x_i \in V - \{s, t\}\}$.

Primero, se demostrará que G' es fuertemente conexo si y sólo si G tiene un camino de s a t, y luego se mostrará el transductor que utiliza este teorema para computar G'.

($G' \Rightarrow G$). Si G' es fuertemente conexo, entonces existe un camino de s a t y por lo tanto existe el mismo camino en G.

($G' \Leftarrow G$). Si existe un camino $C_{s,t}$ de s a t en G, entonces por la transformación explicada anteriormente, para ir de cualquier nodo u a cualquier otro nodo v con $u, v \in V - \{s, t\}$ se tiene en G' el camino $\langle u, s\rangle, C_{s,t}, \langle t, v\rangle$. Para ir de s a cualquier otro nodo $u \in V - \{t\}$ se tiene en G' el camino $C_{s,t}, \langle t, u\rangle$, para ir de t a cualquier otro nodo $u \in V - \{s\}$ se tiene en G' el camino $\langle t, u\rangle$ y para ir de t a s se tiene en G' el camino $\langle t, u\rangle, \langle u, s\rangle$. Por lo tanto G' es fuertemente conexo. Nótese que si no existiese el camino de s a t en G, tampoco existiría el mismo en G' y por lo tanto, G' no sería fuertemente conexo.

A continuación, se muestra el transductor que realiza el procedimiento anterior:

$M_T =$ En entrada $\langle G, s, t\rangle$.

1. Copiar $\langle G\rangle$ de la cinta de entrada a la cinta de salida.
2. Por cada vértice $x_i \in V - \{s, t\}$
    3. Añade los arcos $\langle x_i, s\rangle$ y $\langle t, x_i\rangle$ a la cinta de salida.

El transductor corre en espacio logarítmico porque únicamente almacena en cada iteración el apuntador al nodo x_i con el cual está trabajando para llevar la cuenta.

En la cinta de salida se tendrá G'. Si se representa G con una matríz de adyacencias m, para crear G' a partir de m, la transformación sólo agregaría 1's en todas las casillas $m[t, i]$ y 1's en todas las casillas $m[i, s]$ con $i \neq s$ y $i \neq t$.

Por las demostraciones anteriores, $STRONGLY - CONNECTED \in NL$, $PATH \in NL$, $PATH \leq_L STRONGLY - CONNECTED$, entonces $STRONGLY - CONNECTED \in NL - completo$

6. **Ejercicio 9.12.** Describir el error en la siguiente prueba engañosa de $P \neq NP$. Asuma que $P = NP$ y obtenga una contradicción. Si $P = NP$ entonces $SAT \in P$ y para algún k, $SAT \in TIME(n^k)$. Debido a que todo lenguaje en NP es reducible en tiempo polinomial a SAT, tenemos $NP \subseteq TIME(n^k)$. Por lo tanto, $P \subseteq TIME(n^k)$, pero por el Teorema de Jerarquía de Tiempo, $TIME(n^{k+1})$ contiene un lenguaje que no está en $TIME(n^k)$, lo cual contradice que $P \subseteq TIME(n^k)$. Por lo tanto, $P \neq NP$.

   El error (asumiendo $P = NP$) está en decir que como para algún k, $SAT \in TIME(n^k)$, entonces como todo lenguaje NP es reducible en tiempo polinomial a SAT, se tiene que $NP \subseteq TIME(n^k)$. La reducción en tiempo polinomial no implica necesariamente que se realizará en el mismo tiempo $O(n^k)$, es más, podría demorar más tiempo. Por lo tanto, no se puede concluir que $NP \subseteq TIME(n^k)$, y tampoco que $P \subseteq TIME(n^k)$ (asumiendo hipótesis) con lo cual, no se llegaría a una contradicción.

7. **Ejercicio 9.19.** Defina el problema unique-sat como:

   $USAT = \{\langle\phi\rangle \mid \phi$ *es una fórmula Booleana que tiene una sóla asignación satisfacible* $\}$.

   Mostrar que $USAT \in P^{SAT}$.

   La idea es asignarle valores de cierto o falso a cada variable x_i de la fórmula Booleana ϕ de forma iterativa, y preguntarle al oráculo SAT si las asignaciones hacen que ϕ sea satisfacible o no. Si se encuentra que con ambos valores posibles de la variable x_i la fórmula es satisfacible, se rechaza debido a que se tendría mas de una asignación posible con la que ϕ podría ser cierta; si con ambos valores posibles de x_i el oráculo dice que ϕ no se satisface, entonces se rechaza, no hay una asignación posible que satisfaga a ϕ. Por lo tanto, continuará con la evaluación de la siguiente variable únicamente en el caso de que sólo una de todas

las asignaciones probadas hasta el momento haga que ϕ sea satisfacible.

M = En entrada $\langle\phi\rangle$ donde $V = \{x_1, x_2, ..., x_n\}$ son variables de ϕ

1. Chequear que ϕ sea una formula Booleana, sino RECHAZAR.
2. Si $\mid V \mid = 0$, ACEPTAR.
3. Para $i = 1...n$:
   - 3.1. Sustituir $x_i = 0$ en ϕ y almacenar la fórmula resultante en ϕ_1.
   - 3.2. Sustituir $x_i = 1$ en ϕ y almacenar la fórmula resultante en ϕ_2.
   - 3.3. Consultarle al oráculo SAT si ϕ_1 es satisfacible.
   - 3.4. Consultarle al oráculo SAT si ϕ_2 es satisfacible.
   - 3.5. Si ambas son satisfacibles, entonces RECHAZAR.
   - 3.6. Si ninguna de las dos son satisfacibles, entonces RECHAZAR.
   - 3.7. Si sólo una es satisfacible, entonces asignarle a ϕ la fórmula (ϕ_1 ó ϕ_2) que es satisfacible.

4 Si los pasos anteriores concluyeron exitosamente, ACEPTAR.

La máquina de Turing M termina en n pasos, donde n es el número de variables en ϕ, todos los pasos corren en tiempo polinomial. El oráculo siempre devolverá una respuesta, por lo que al final de la ejecución se sabrá si la fórmula Booleana tiene una única asignación satisfacible o no. Por lo tanto, M decide a $USAT$ y $USAT \in P^{SAT}$.

8. **Ejercicio 9.21.** Una k-query oracle Turing machine es una máquina de Turing oráculo que sólo se le permite hacer a lo más k consultas en cada entrada. Un k-query TM M con un oráculo para A se escribe de la forma $M^{A,K}$ y $P^{A,K}$ es la colección de lenguajes que son decidibles en tiempo polinomial por k-query TMs que usan el oráculo A:

   a) Muestre que $NP \cup coNP \subseteq P^{SAT,1}$

   Por definición se tiene que $coNP \subseteq NP$, por lo tanto, encontrar un lenguaje L en $NP - completo$ tal que $L \in P^{SAT,1}$ es suficiente para la demostración. Si L es $NP - completo$ todos los lenguajes en NP y$coNP$ son reducibles a L y en consecuencia, estarían en $P^{SAT,1}$.

   Se sabe que $3SAT \in NP - completo$, es fácil de verificar, que puede ser decidido por una $M^{SAT,1}$ polinomial en tiempo, es decir, sólo necesita hacerle una sola consulta a SAT para chequear que la fórmula sea satisfacible. Por lo tanto, $3SAT \in P^{SAT,1}$, y en consecuencia, $NP \cup coNP \subseteq P^{SAT,1}$.

   b) Asuma que $NP \neq coNP$. Muestre que $P \cup coNP \subset P^{SAT,1}$

   Como P es cerrado bajo el complemento se tiene que $P = coP$, además se sabe que $P \subseteq NP$, pero asumiendo la hipótesis $NP \neq coNP$ implica que NP no es cerrado bajo el complemento, es decir que P y NP no se comportan igual, lo que lleva a que $P \neq NP$. Entonces, de la hipótesis se deduce que $P \subset NP$ y $coNP \subset NP$, así que $P \cup coNP \subset NP$ y como en el teorema anterior se demostró que $NP \cup coNP \subseteq P^{SAT,1}$, es decir, NP también es contenido por $P^{SAT,1}$, se puede concluir que $P \cup coNP \subset P^{SAT,1}$.

Bibliografía

[1] M. Sipser, *Introduction to the Theory of Computation; 2nd ed.* Cambridge: Thomson Course Technology, 2006.

[2] J. D. U. John E. Hopcroft, Rajeev Motwani, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, second ed., 2001.

[3] N. Cutland, *Computability: An Introduction to Recursive Function Theory.* Cambridge: Cambridge Univ. Press, 1980.

[4] J. E. Savage, *Models of computation : exploring the power of computing.* Addison Wesley, 1998.