Pauta de Corrección

Segundo Certamen
Introducción a la Informática Teórica

7 de septiembre de 2013

1. Cada cual por turno.

   a) Un lenguaje es *regular* si es representado por una expresión regular, es aceptado por un autómata finito o es generado por una gramática regular (tipo 3 en la jerarquía de Chomsky).

   b) Un *problema* es un lenguaje $\mathcal{L}$. Está en NP si hay una máquina de Turing no determinista que acepta $\omega \in \mathcal{L}$ en un número de pasos acotado por un polinomio en el largo de ω.

   Para fines prácticos se usa la definición equivalente: Dada una solución al problema (o, más generalmente, un *certificado* de la solución) es posible verificar la solución mediante un algoritmo determinista que toma tiempo polinomial en $|\omega|$. ("Adivinar la solución y verificarla deterministamente en tiempo polinomial.")

   c) Un lenguaje es *recursivamente enumerable* si es aceptado por una máquina de Turing. Alternativamente, es generado por una gramática de tipo 0 en la jerarquía de Chomsky.

   d) Un lenguaje se dice *recursivo* si es aceptado por una máquina de Turing que siempre se detiene.

   e) Una *reducción* del problema A al problema B es un algoritmo determinista que traduce cualquier instancia α del problema A a una instancia β del problema B que tiene la misma respuesta. Una reducción es *polinomial* si el tiempo que toma tal traducción está acotado por un polinomio en $|\alpha|$.

Puntajes

Total			20
a) Cualquiera de las tres alternativas		5	
b)		5	
Definir problema	2		
Precisar problema en NP	3		
c) Definición		5	
d) Definición		5	
e)			
Reducción	3		
Especializar a polinomial	2		

2. Un lenguaje se dice *decidible* si es recursivo, hay un algoritmo que responde *si* o *no* en un tiempo finito.

   Dado el DFA M, podemos simularlo sobre el string ω. Simular cada movida del DFA tomará un tiempo finito, y en cada movida se consume un símbolo de ω, por lo que el proceso toma un tiempo finito en entregar la respuesta.

Puntajes

Total		15
– Definición de *decidible* (puede ser implícita)	3	
– Cada movida del DFA toma tiempo finito	5	
– Son finitas movidas	5	
– Conclusión	2	

3. La demostración es por contradicción. Supongamos que contamos con una máquina de Turing M_{EQ} que siempre se detiene y que reconoce $\mathsf{EQ}_{\mathrm{TM}}$. Dada una máquina de Turing $M_{\varnothing}$ que no acepta nada (su construcción queda de ejercicio), podemos construir una máquina de Turing que decide E_{TM}, ver la figura 1. Pero

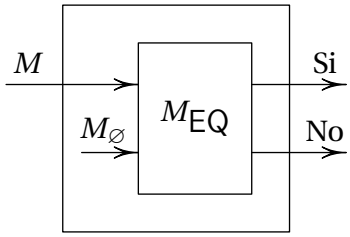

Figura 1: Máquina de Turing que decide E_{TM}

sabemos que eso es imposible, luego no existe M_{EQ}.

Puntajes

Total	20
Demostración por contradicción	5
Idea de reducir E_{TM} a $\mathsf{EQ}_{\mathrm{TM}}$	5
Construcción más formal	10

4. Primero las definiciones del caso: Un lenguaje es *recursivo* si hay una máquina de Turing que siempre se detiene que acepta el lenguaje (lo *decide*). Es *recursivamente enumerable* si es aceptado por una máquina de Turing. La idea es que podemos construir una máquina de Turing que simula la máquina de Turing que acepta el lenguaje "en diagonal" sobre los strings, y va escribiendo en una cinta de salida aquellos que son aceptados. Claramente los lenguajes recursivos son recursivamente enumerables.

   Tenemos tres situaciones que debemos cubrir:

   $\mathcal{L}$ es recursivo: En este caso, $\overline{\mathcal{L}}$ es recursivo también. Decidir si $\omega \in \mathcal{L}$ automáticamente decide si $\omega \in \overline{\mathcal{L}}$.

   $\mathcal{L}$ es recursivamente enumerable: Hay dos posibilidades:

   - Si $\overline{\mathcal{L}}$ también es recursivamente enumerable, podemos correr en paralelo las máquinas de Turing para $\mathcal{L}$ y $\overline{\mathcal{L}}$. Una de las dos aceptará en tiempo finito, con lo que tenemos cómo decidir $\mathcal{L}$ (y $\overline{\mathcal{L}}$ por el caso anterior). Ambos son recursivos.
   - Si $\overline{\mathcal{L}}$ no es recursivamente enumerable, $\mathcal{L}$ es recursivamente enumerable, pero no recursivo.

   $\mathcal{L}$ no es recursivamente enumerable: Por el punto anterior, $\overline{\mathcal{L}}$ puede ser recursivamente enumerable, pero no recursivo. Pero también puede darse el caso que $\overline{\mathcal{L}}$ no sea recursivamente enumerable.

Puntajes

Total		25
Definiciones (pueden estar implícitas)	5	
$\mathcal{L}$ recursivo	4	
$\mathcal{L}$ recursivamente enumerable	8	
$\mathcal{L}$ no recursivamente enumerable	8	

5. La demostración es por contradicción. La gramática dada produce strings de las formas:

$$a_{i_k} a_{i_{k-1}} \cdots a_{i_1} a_0 \alpha_{i_1} \alpha_{i_2} \cdots \alpha_{i_k}$$
$$a_{i_k} a_{i_{k-1}} \cdots a_{i_1} a_0 \beta_{i_1} \beta_{i_2} \cdots \beta_{i_k}$$

Es claro que un string aparecerá en ambas posibilidades, vale decir, ese string tiene dos derivaciones diferentes, si y sólo si esa instancia del problema de Post tiene solución. Si pudiéramos determinar si una gramática de contexto libre es ambigua, por esta vía tendríamos cómo resolver el problema de Post. Pero eso es imposible en general, luego tampoco es posible determinar si una gramática es ambigua.

Puntajes

Total		20
Demostración por contradicción	5	
Estamos reduciendo PCP a ambigüedad de CFGs	10	
PCP no es decidible, tampoco lo es determinar ambigüedad	5	

6. Es claro que una vez puestas las tarjeta en la caja podemos verificar rápidamente si hay algún agujero libre. Formalmente, podemos describir m tarjetas con n filas cada una de ellas mediante $m \cdot 2n$ símbolos 0 o 1, y $m-1$ signos de puntuación para separar tarjetas, para un total de $m(2n+m-1)$ símbolos. Revisar si hay algún agujero libre es revisar $2n$ agujeros, lo cual está acotado por un polinomio en $m(2n+m-1)$. O sea, el problema está en NP.

   Al dar vuelta la tarjeta queda ya sea la columna izquierda o derecha definiendo si se ve el fondo. Siguiendo la pista, vemos que conviene representar verdadero por falta de agujero, y tener agujero en una fila en ambas columnas hace que en esa fila la tarjeta no influya. Reduciremos SAT a PUZZLE, como sabemos que SAT es NP-completo, PUZZLE es NP-duro. Si agregamos a la colección una tarjeta como la de la figura 2 estamos

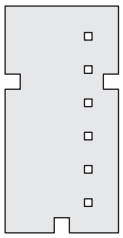

Figura 2: Tarjeta que fuerza una columna

   forzando a que de haber forma de ver el fondo de la caja sea a un lado, sin pérdida de generalidad podemos suponerla ubicada con los agujeritos a la derecha. Basta entonces definir una fila por cláusula, y la tarjeta correspondiente a la variable x tendrá un agujero solamente a la derecha en la fila i si $\overline{x}$ aparece en la cláusula C_i, solamente a la izquierda si x aparece en la cláusula C_i, y en ambas si no aparecen x ni $\overline{x}$. La caja fuerza a elegir una de las dos columnas, la tarjeta de x obstaculiza la vista al fondo si x es verdadero y la tarjeta está al derecho, o si $\overline{x}$ es verdadero y la tarjeta está al revés.

   Sabemos que 3SAT es NP-completo. Por lo tanto, para 3 o más filas por tarjeta PUZZLE es NP-duro. Como PUZZLE está en NP y es NP-duro, es NP-completo.

Puntajes

Total		25
– Argüir que está en NP		8
– Argüir que es NP-duro		15
Agujero corresponde a verdadero	3	
Fila es una cláusula	3	
Participación de la variable en la cláusula	4	
Se ve el fondo si y sólo si es satisfacible	5	
– Por las anteriores, es NP-completo		2