Examen de Computabilidad y Complejidad

(CMC)

15 de septiembre de 1998

(I) Cuestiones (justifique formalmente las respuestas)

1. Sea la familia de lenguajes $\mathcal{L}_{ncf}$ definida como la formada por aquellos lenguajes cuyo complementario es incontextual. ¿ Es $\mathcal{L}_{ncf}$ cerrada bajo: (a) Intersección y (b) Complementación ?

(1.5 ptos)

Solución

En primer lugar, recordemos que la clase de lenguajes incontextuales no es cerrada bajo complementario y sí que lo es bajo unión. A continuación, procederemos a analizar cada pregunta por separado.

(a) La clase $\mathcal{L}_{ncf}$ sí es cerrada bajo intersección. Tomemos dos lenguajes de $\mathcal{L}_{ncf}$, L_1 y L_2. Los complementarios de los anteriores lenguajes son incontextuales por definición. Aplicamos ahora la intersección y obtenemos $L_1 \cap L_2$. Para que $L_1 \cap L_2$ pertenezca a $\mathcal{L}_{ncf}$ su complementario debe ser incontextual. Se cumple que $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ que es incontextual al expresarse como la unión de dos lenguajes incontextuales.

(b) La clase $\mathcal{L}_{ncf}$ no es cerrada bajo complementario. Actuemos por contradicción y supongamos que sí lo fuera. Tomemos un lenguaje arbitrario incontextual L. Por definición $\overline{L}$ pertenece a $\mathcal{L}_{ncf}$ y, según nuestra hipótesis, $\overline{\overline{L}} = L$ también pertenece a $\mathcal{L}_{ncf}$. Ahora bien, si $L \in \mathcal{L}_{ncf}$ entonces $\overline{L}$ sería incontextual independientemente de qué lenguaje se trate. Como sabemos que la clase de los lenguajes incontextuales no es cerrada bajo complementario, entonces la clase $\mathcal{L}_{ncf}$ tampoco puede serlo ya que, en caso contrario, incurriríamos en contradicción.

2. Sea el lenguaje $L = \{x1y1z1 : x, y, z \in \Sigma^* \wedge |x| = |y| = |z|\}$. ¿ Es L incontextual ?

(1.5 ptos)

Solución

El lenguaje L no es incontextual. Partiremos de L y, mediante operaciones de cierre para la clase de los lenguajes incontextuales, llegaremos a un lenguaje que no sea incontextual. Asumiremos que el lenguaje L está definido para el alfabeto $\Sigma = \{0, 1\}$. Para alfabetos de más de un símbolo la solución que proponemos es similar. Por otra parte, si L está definido sobre un alfabeto de un único símbolo $\Sigma = \{1\}$, entonces la solución es totalmente distinta ya que, en ese caso, el lenguaje sería incontextual (se correspondería con el conjunto de cadenas de símbolos 1 con una longitud múltiplo de 3 que se podría generar con la gramática definida por las reglas $S \to 1A$; $A \to 1B$; y $B \to 1S|1$).

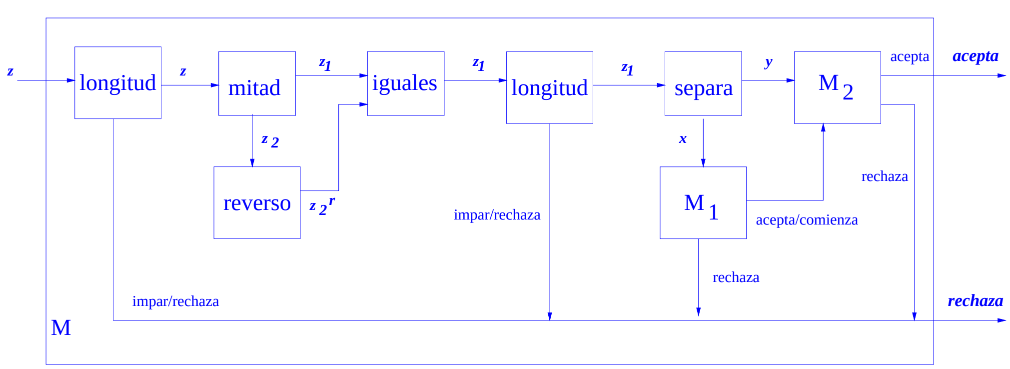
En primer lugar definimos $L_1 = L \cap 0^*10^*10^*1 = \{0^n10^n10^n1 : n \geq 0\}$. Tomemos ahora el homomorfismo h definido como $h(a) = 0$, $h(b) = 0$, $h(c) = 0$ y $h(d) = 1$. A partir de h y L_1 definimos $L_2 = h^{-1}(L_1) = \{\{a,b,c\}^n d\{a,b,c\}^n d\{a,b,c\}^n d : n \geq 0\}$. A continuación, definimos $L_3 = L_2 \cap a^*db^*dc^*d = \{a^n db^n dc^n d : n \geq 0\}$. Por último, tomemos el homomorfismo g definido como $g(a) = a$, $g(b) = b$, $g(c) = c$ y $g(d) = \lambda$ y definimos $L_4 = g(L_3) = \{a^n b^n c^n : n \geq 0\}$. El último lenguaje obtenido, L_4 se ha comprobado en numerosas ocasiones que no es incontextual y lo hemos obtenido a partir de L mediante operaciones de cierre para la clase de los lenguajes incontextuales. Por lo tanto, la conclusión es que L no es incontextual.

3. Se define la operación $mezcla(x,y) = x_1y_1x_2y_2 \cdots x_ny_nx_ny_n \cdots x_2y_2x_1y_1$ con $x = x_1x_2 \cdots x_n$ e $y = y_1y_2 \cdots y_n$. La operación se extiende a lenguajes de la forma $mezcla(L_1, L_2) = \{mezcla(x,y) : x \in L_1 \wedge y \in L_2\}$. ¿Es la clase de los lenguajes recursivos cerrada bajo la operación $mezcla$?

(1.5 ptos)

Solución

La clase de los lenguajes recursivos sí es cerrada bajo la operación $mezcla$. Partiendo de dos lenguajes recursivos, L_1 y L_2, definiremos un esquema de aceptación recursivo para el lenguaje $mezcla(L_1, L_2)$. Para ello, contaremos con dos módulos M_1 y M_2 basados en máquinas de Turing que establecen si una cadena de entrada pertenece o no a L_1 y L_2 respectivamente. Contaremos con el módulo *longitud* que se basa en una máquina de Turing que establece si la longitud de una cadena de entrada es par o no (esto se puede determinar de forma trivial con dos estados de la máquina) y, en caso de que sea par, copia a la salida su entrada. De igual forma, el módulo *mitad* se basa en una máquina de Turing multicinta que toma una cadena de entrada de longitud par y copia a la salida sus dos mitades (esto se realiza mediante un algoritmo trivial). El módulo *reverso* también se basa en una máquina de Turing multicinta que toma una cadena de entrada y copia a la salida su reverso. El módulo *iguales* se basa en una máquina de Turing multicinta que establece si, dadas dos cadenas de entrada, los símbolos impares de la primera coinciden con los pares de la segunda y los símbolos pares de la primera coinciden con los impares de la segunda. En caso afirmativo, *iguales* copia a la salida una de las dos cadenas de entrada. Por último, el módulo *separa* se basa en una máquina de Turing multicinta que toma como entrada una cadena de longitud par y copia a la salida dos cadenas formadas respectivamente por los símbolos pares y los símbolos impares de la cadena de entrada. A partir de los anteriores módulos, proponemos el siguiente esquema que acepta $mezcla(L_1, L_2)$ y garantiza siempre la parada



El funcionamiento del anterior esquema se explica a continuación. Dada una cadena de entrada z se establece en primer lugar, mediante el módulo *longitud*, si su longitud es par. Si la longitud de la entrada es impar la cadena se rechaza (ya que no pertenece a $mezcla(L_1, L_2)$). En caso contrario, se proporciona la cadena de entrada al módulo *mitad* que obtiene las dos cadenas de igual longitud z_1 y z_2 que forman la cadena z. La cadena z_2 se procesa en el módulo *reverso* que proporciona como salida z_2^r. A continuación, al módulo *iguales* se le proporciona como entrada el par de cadenas z_1 y z_2^r y establece si los símbolos impares de z_1 coinciden con los pares de z_2^r y los símbolos pares de z_1 coinciden con los impares de z_2^r. Si las cadenas no coinciden entonces se rechaza la cadena z (ya que no pertenece a $mezcla(L_1, L_2)$). En caso contrario, se proporciona al módulo *longitud* la cadena de entrada z_1. El módulo *longitud* establece si z_1 es de longitud par. En caso negativo se rechaza la cadena de entrada z (ya que no pertenece a $mezcla(L_1, L_2)$) y, en caso afirmativo se pasa la cadena z_1 como entrada al módulo *separa*. El módulo *separa* obtiene las cadenas x e y que se analizan respectivamente en los módulos M_1 y M_2. Sólo en el caso de que ambos módulos acepten se aceptaría la cadena de entrada z. En caso contrario, la cadena de entrada se rechaza.

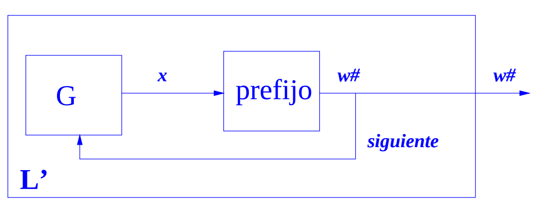
Tal y como se puede comprobar, el anterior esquema garantiza siempre la parada y acepta $mezcla(L_1, L_2)$. Por lo tanto, $mezcla(L_1, L_2)$ es recursivo y la operación $mezcla$ es de cierre para la clase de los lenguajes recursivos.

4. Sea $L = \{w_1\#w_2\# : w_1, w_2 \in \Sigma^* \wedge \# \notin \Sigma\}$ un lenguaje recursivamente enumerable y se define L' como el conjunto de prefijos de L que contienen sólo un símbolo $\#$ y éste se encuentra al final. ¿ Es L' recursivamente enumerable ?

(1.5 ptos)

Solución

El lenguaje L' es recursivamente enumerable. Para demostrarlo, proponemos un esquema generativo que genera L'. El esquema se basa en la utilización de dos módulos predeterminados. El módulo G genera el lenguaje L ya que L es recursivamente enumerable y, por lo tanto, puede ser generado por una máquina de Turing. El módulo *prefijo*, se basa en una máquina de Turing multicinta que copia la cadena de entrada en una cinta de salida hasta el primer símbolo $\#$. A partir de los anteriores módulos proponemos el siguiente esquema que genera L'



El funcionamiento del anterior esquema es sencillo de explicar. Cada vez que el módulo G genera una cadena de L, el módulo *prefijo* toma esa cadena como entrada, copia a la salida su prefijo hasta el primer símbolo $\#$ y vuelve a activar el módulo G. Dado que el anterior esquema genera L', entonces podemos concluir que L' es recursivamente enumerable.

(II) PROBLEMAS:

5. Sea la gramática G definida por las reglas $S \to aAbB \mid \lambda$; $A \to aA \mid ba$; y $B \to SB \mid b$. Se define el homomorfismo h como $h(a) = 10$ y $h(b) = 01$. Obtenga una gramática incontextual para el lenguaje definido por la operación $(L(G) \cup h(L(G)))^*$.

(1 pto)

Solución

En primer lugar obtendremos una gramática para $h(L(G))$. La gramática queda definida por las siguientes reglas

$S_h \to 10A_h01B_h \mid \lambda$

$A_h \to 10A_h \mid 0110$

$B_h \to S_hB_h \mid 01$

A continuación, una gramática para $L(G) \cup h(L(G))$

$S_1 \to S \mid S_h$

$S \to aAbB \mid \lambda$

$A \to aA \mid ba$

$B \to SB \mid b$

$S_h \to 10A_h01B_h \mid \lambda$

$A_h \to 10A_h \mid 0110$

$B_h \to S_hB_h \mid 01$

Por último, la gramática que se solicita en el enunciado del problema para $(L(G) \cup h(L(G)))^*$

$S_2 \to S_1S_2 \mid \lambda$

$S_1 \to S \mid S_h$

$S \to aAbB \mid \lambda$

$A \to aA \mid ba$

$B \to SB \mid b$

$S_h \to 10A_h01B_h \mid \lambda$

$A_h \to 10A_h \mid 0110$

$B_h \to S_hB_h \mid 01$

6. Dada la gramática G definida por las siguientes producciones obtenga una gramática en Forma Normal de Chomsky que genere $L(G) - \{\lambda\}$

$$\begin{array}{lll} S \to aAbB \mid \lambda & A \to aA \mid ba \mid \lambda & B \to SB \mid b \mid CS \\ C \to aCa \mid CD & D \to DC \mid bDa & E \to aBAb \mid b \end{array}$$

(1 pto)

Solución

En primer lugar, simplificaremos G para facilitar la obtención de la gramática en Forma Normal de Chomsky.

Eliminación de símbolos no generativos

Símbolos no generativos: $\{C, D\}$

Gramática sin símbolos no generativos

$S \to aAbB \mid \lambda$

$A \to aA \mid ba \mid \lambda$

$B \to SB \mid b$

$E \to aBAb \mid b$

Eliminación de símbolos no alcanzables

Símbolos no alcanzables: $\{E\}$

Gramática sin símbolos no alcanzables

$S \to aAbB \mid \lambda$

$A \to aA \mid ba \mid \lambda$

$B \to SB \mid b$

Eliminación de producciones vacías

Símbolos anulables: $\{S, A\}$

Gramática sin producciones vacías

$S \to aAbB \mid abB$

$A \to aA \mid a \mid ba$

$B \to SB \mid B \mid b$

Eliminación de producciones unitarias

$\mathcal{C}(S) = \{S\}$ $\mathcal{C}(A) = \{A\}$ $\mathcal{C}(B) = \{B\}$

Gramática sin producciones unitarias

$S \to aAbB \mid abB$

$A \to aA \mid a \mid ba$

$B \to SB \mid b$

La gramática anterior ya está totalmente simplificada ya que todos sus símbolos son útiles (generativos y alcanzables).

Paso a Forma Normal de Chomsky

Sustitución de símbolos terminales

$S \to C_aAC_bB \mid C_aC_bB$

$A \to C_aA \mid a \mid C_bC_a$

$B \to SB \mid b$

$C_a \to a$

$C_b \to b$

Factorización de las producciones y obtención de la gramática definitiva en FNC

$S \to C_aD_1 \mid C_aD_3$

$D_1 \to AD_2$

$D_2 \to C_bB$

$D_3 \to C_bB$

$A \to C_aA \mid a \mid C_bC_a$

$B \to SB \mid b$

$C_a \to a$

$C_b \to b$

7. Escriba un módulo *Mathematica* que, tomando como entrada una lista con símbolos a y b, produzca como salida otra lista cambiando a por b y b por ba. (Ej Entrada = $\{a, b, b\}$ Salida = $\{b, b, a, b, a\}$)

(1 pto)

Solución

```
Solucion[entrada_List]:=Module[{ salida, k },
    salida = {};
    For[k=1, k ≤ Length[entrada], k++,
```

```
    If[entrada[[k]] == a, salida=Join[salida,{ b }], salida=Join[salida,{ b,a }]]
  ];
  Return[salida]
]
```