Problemas Teoría de Automátas y Lenguajes Formales

Tema 1

1. Dado los siguientes alfabetos: $\Sigma_1 = \{1,2,3,4,5,6,7,8\}$ y $\Sigma_2 = \{a,b,c,d,e,f,g,h\}$ y los siguientes lenguajes $L_1(\Sigma_1) = \{x|x \in \Sigma_1\}$ y $L_2(\Sigma_2) = \{x|x \in \Sigma_2\}$. Definir los lenguajes $L_1 \cup L_2$, $L_1 \bullet L_2$ y $(L_1 \bullet L_2)^2$.

2. Dada la siguiente gramática $G = (\{0,1\}, \{A,B\}, A, \mathcal{P})$ donde $\mathcal{P}$ es:

$$\begin{array}{ll} \text{A} \rightarrow & \text{1 B 1} \mid \text{0 B 0} \\ \text{B} \rightarrow & \text{A} \mid \text{1} \mid \text{0} \mid \epsilon \end{array}$$

   Obtener la derivación más a la derecha y más a la izquierda de la cadena 111 y 10001. Dibujar el árbol de derivación.

3. Supongáse la siguiente gramática que permite generar expresiones artiméticas:

$$\begin{array}{ll} \text{E} \rightarrow & \text{E + E} \mid \text{E - E} \mid \text{E * E} \mid \text{(E)} \mid \text{Letra} \mid \text{Digito} \\ \text{Letra} \rightarrow & \text{a} \mid \text{b} \mid \ldots \mid \text{z} \\ \text{Digito} \rightarrow & \text{0} \mid \text{1} \mid \ldots \mid \text{9} \end{array}$$

   Obtener la derivación más a la izquierda de la cadena $a * 8 + 5$. ¿ Es una gramática ambigua? Dibuja el árbol de derivación correcto en base a las prioridades de operadores habituales.

4. Dada la siguiente gramática $G = \{\Sigma_T, \Sigma_N, S, \mathcal{P}\}$ donde $\Sigma_T = \{0,1\}$, $\Sigma_N = \{S,A,B,C,D,E,F\}$, S es el axioma y $\mathcal{P}$ es:

$$\begin{array}{ll} \text{S} \rightarrow & \text{A B} \mid \text{A} \mid \text{C S 1} \mid \text{0 E} \\ \text{A} \rightarrow & \text{0 A S} \mid \epsilon \mid \text{A 0} \mid \text{C} \\ \text{B} \rightarrow & \text{B 1} \mid \text{1} \\ \text{D} \rightarrow & \text{B 1} \mid \epsilon \mid \text{1 F} \\ \text{E} \rightarrow & \text{E 1} \\ \text{F} \rightarrow & \text{0 D} \end{array}$$

   depurarla hasta conseguir una gramática bien formada.

Tema 4

5. Definir una gramática independiente del contexto que genere el lenguaje: $L = \{x^n y^m \; tal \; que \; m,n \geq 0, m = n \; o \; bien \; m = 2n\}$

6. Diseñar una gramática independiente del contexto que genere el lenguaje: $L = \{({}^n a)^n \; tal \; que \; n \geq 0\}$

7. Diseñar una gramática independiente del contexto para $\Sigma = \{a, b\}$ que reconozca cadenas tales que la relación entre el número de $a's$ y de $b's$ sea de 2 a 1.

8. Encontrar una gramática independiente del contexto para $\Sigma = \{0, 1\}$ que genere aquellas cadenas tales que el número de $0's$ sea igual al número de $1's + 1$.

9. Diseñar una gramática independiente del contexto que genere los números romanos del uno al mil. Tened en cuenta que la letra I es el 1, la letra V es 5, la letra X es 10, la letra L es 50, la letra D es 500 y la letra M es 1000. Además, para generar números romanos correctos, no puede haber más de tres símbolos iguales seguidos y los símbolos V, L, D no se pueden repetir. Finalmente, a la hora de componer números que impliquen restar dos símbolos contiguos, éstos no pueden diferir en un orden de magnitud. Por ejemplo: CDV (405), la C y la D difieren en un orden de magnitud.

10. Diseñar una gramática independiente del contexto que genere las sentencias de llamadas a función. El número de argumentos de una función puede ser cero o más de uno. Un argumento de una función puede ser la llamada a otra función. Los argumentos vienen separados por comas. La lista de argumentos viene encerrada entre paréntesis. Los nombres de las funciones y argumentos vienen definidos por el símbolo terminal id. Un ejemplo de llamada a una función es $id(id, id(id, id), id())$.

11. Diseñar una gramática independiente del contexto que genere bloques de sentencias anidados. El conjunto de símbolos terminales es {, }, s, donde { es inicio de bloque, } es fin de bloque y s representa cualquier sentencia. Se tiene que cumplir que: i) El número de llaves de apertura y de cierre son iguales; ii) No pueden existir bloques de sentencias vacíos; iii) Los bloques de sentencias están formados por sentencias y otros bloques de sentencias. Ejemplo de cadena a generar: $\{ss\{s\}s\}$.

12. Transforma a Forma Normal de Greibach la siguiente gramática:
$G = (\{0, 1, 2\}, \{A, B, C\}, A, \mathcal{P})$ donde $\mathcal{P}$ es:

$$\begin{array}{ll} \text{A} \rightarrow & \text{C B 2} \mid \text{1 B} \mid \epsilon \\ \text{B} \rightarrow & \text{B C} \mid \text{1} \\ \text{C} \rightarrow & \text{2} \end{array}$$

13. Construye el automata a pila equivalente a la siguiente gramática independiente del contexto que funcione por criterio de aceptación por vaciado de pila.

$$\begin{array}{ll} \text{A} \rightarrow & \text{2 B C} \mid \text{1 B} \mid \epsilon \\ \text{B} \rightarrow & \text{1 B'} \mid \text{1 C} \mid \text{1} \\ \text{B'} \rightarrow & \text{2 B'} \mid \text{2 C} \\ \text{C} \rightarrow & \text{2} \end{array}$$

14. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que funcione por criterio de aceptación por vaciado de pila.

$$S \rightarrow \quad S + S \mid (S) \mid a$$

15. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que permite generar declaraciones de variables en C/C++ que funcione por criterio de aceptación por vaciado de pila.

$$\begin{array}{ll} D \rightarrow & T \text{ id } V \text{ ;} \\ T \rightarrow & \text{int} \\ V \rightarrow & \text{, id } V \mid \epsilon \end{array}$$

16. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que genera expresiones relacionales y booleanas que funcione por criterio de aceptación por vaciado de pila.

$$\begin{array}{ll} S \rightarrow & [\text{ C and C }] \mid [\text{ C or C }] \mid [\text{ not C }] \\ C \rightarrow & [\text{ V} = \text{V }] \mid [\text{ V} > \text{V }] \mid [\text{ V} < \text{V }] \\ V \rightarrow & \text{id} \mid \text{num} \end{array}$$

17. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que por criterio de aceptación de estado final.

$$S \rightarrow \quad x\ S\ y \mid x\ y$$

18. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que por criterio de aceptación de estado final.

$$\begin{array}{ll} \text{Operacion} \rightarrow & E \mid P \\ P \rightarrow & \int E\ d\ V \mid d\ E\ V \\ E \rightarrow & T + E \mid T - E \mid T \\ T \rightarrow & F \uparrow T \mid F \\ F \rightarrow & V \mid N \mid (E) \mid \sin (V) \mid \cos (V) \\ N \rightarrow & 0 \mid 1 \mid \ldots \mid 9 \\ V \rightarrow & x \end{array}$$

19. Construye el automata a pila que reconozca la siguiente gramática independiente del contexto que permite generar un pequeño texto en HTML por criterio de aceptación de estado final.

$$\begin{array}{ll} \text{Doc} \rightarrow & \text{Ele Doc} \mid \epsilon \\ \text{Car} \rightarrow & a \mid b \mid \ldots \mid z \\ \text{Texto} \rightarrow & \text{Car Texto} \mid \epsilon \\ \text{Ele} \rightarrow & \text{Texto} \mid < \text{OL} > \text{Lista} < /\text{OL} > \mid < \text{EM} > \text{Texto} < /\text{EM} > \mid < \text{P} > \text{Texto} \\ \text{Lista} \rightarrow & \text{EleLista Lista} \mid \epsilon \\ \text{EleLista} \rightarrow & < \text{LI} > \text{Texto} \end{array}$$

20. La siguiente gramática independiente del contexto permite generar árboles binarios

$$\text{árbol} \rightarrow (\ \text{num árbol árbol}\ ) \mid \text{NULL}$$

Diseñar un AP que reconozca el lenguaje generado por dicha gramática por vaciado de pila. Hacer la traza con los movimientos del autómata, indicando el estado al que se transita, la configuración de la pila y de la cadena de entrada para la cadena (*num* (*num NULL NULL*)*NULL*). ¿Qué ocurre con la cadena ((*num NULL NULL*) *NULL*)? Indica el mensaje que emitiría el autómata.

21. Dada la siguiente gramática que permite generar declaraciones de vectores en C/C++. a) Construir el Autómata a Pila que reconozca dicho lenguaje por criterio de aceptación de estado final. b) Hacer la traza con los movimientos del autómata, indicando el estado al que se transita, la configuración de la pila y de la cadena de entrada para la cadena char id[n][n];

S →	T V ;
V →	id Vprima
Vprima →	[n] Vprima | ϵ
T →	int | float | char