Universidad de Granada

Memoria de prácticas

Modelos de Computación

Andrés Molina López
31-1-2017

Índice:

Práctica 1: Ejercicios prácticos sobre Lenguajes y Gramáticas

1) **Describir el lenguaje generado por las siguientes gramáticas en {0,1}*.**
   a) **$S \rightarrow 0S_1 1 \quad S_1 \rightarrow 0S_1 \mid 1S_1 \mid \varepsilon$**
      $L(G) = \{0u1 / u \in A^*\}$
   b) **$S \rightarrow S_1 101 S_1 \quad S_1 \rightarrow 0S_1 \mid 1S_1 \mid \varepsilon$**
      $L(G) = \{u101v / u,v \in A^*\}$
   c) **$S \rightarrow 0S1 \mid S_1 \quad S_1 \rightarrow 1S_1 0 \mid 1S_2 0 \quad S_2 \rightarrow 0S_2 1 \mid \varepsilon$**
      $L(G) = \{0^n 1^m 0^i 1^i 0^m 1^n / n,i \geq 0 \ \ m \geq 1\}$

2) **Encontrar gramáticas de tipo 2 para los siguientes lenguajes sobre el alfabeto {0, 1}. En cada caso determinar si los lenguajes generados son de tipo 3, estudiando si existe una gramática de tipo 3 que los genera.**
   a) **Palabras que comienzan con la subcadena "10" y acaban en "001".**
      Directamente he encontrado una gramática de tipo 3, y como estás están incluidas en las de tipo 2, quiere decir que también se puede conseguir hacer con una de tipo 2, pero muestro la de tipo 3 ya que es mejor que las de tipo 2.
      $S \rightarrow 1S_1 \quad S_1 \rightarrow 0S_2 \quad S_2 \rightarrow 0S_2 \mid 1S_2 \mid 0S_3 \quad S_3 \rightarrow 0S_4 \quad S_4 \rightarrow 1$
   b) **Palabras que tienen 2 o 3 "0".**
      Al igual que en el apartado anterior también he conseguido diseñar una gramática de tipo 3, la cual es la siguiente:
      $S \rightarrow 1S \mid 0S_1 \quad S_1 \rightarrow 1S_1 \mid 0S_2 \quad S_2 \rightarrow 1S_2 \mid 0S_3 \mid \varepsilon \quad S_3 \rightarrow 1S_3 \mid \varepsilon$
   c) **Palabras que no contienen la subcadena "011".**
      Y en este último también he elaborado la siguiente gramática de tipo 3:
      $S \rightarrow 1S \mid 0S_1 \mid \varepsilon \quad S_1 \rightarrow 0S_1 \mid 1S_2 \mid \varepsilon \quad S_2 \rightarrow 0S_1 \mid \varepsilon$

3) **Como empleado de la empresa de desarrollo de videojuegos "MoreThanDungeons", se le ha pedido diseñar una gramática que represente los niveles de un juego de exploración de mazmorras y las salas de estas, con una serie de restricciones.**

   En cada nivel:

- **Existen salas grandes (g) y pequeñas (p) que deberán ser limpiadas de monstruos para avanzar. (Los niveles más sencillos tienen al menos una sala grande)**
- **Hay al menos una sala de tendero (t), donde recuperar fuerzas y comprar objetos.**
- **Habrá una sola sala secreta (x), siempre le precede una sala grande. Es decir, siempre habrá una "g" delante de "x".**
- **Cada nivel de la mazmorra debe acabar con una sala final de jefe (j).**

Por ejemplo, la cadena terminal "ppgxtj" representa el nivel en el que el jugador debe de pasar por dos habitaciones pequeñas "pp", seguidas de una grande "g". En esta, podrá encontrar la sala secreta "x". A continuación, podrá recuperar fuerzas en la tienda "t". Para finalmente, enfrentarse al jefe final "j" del nivel.

Elabore una gramática que genere estos niveles con sus restricciones. Cada palabra del lenguaje es UN SOLO NIVEL. ¿A qué tipo de la jerarquía de Chomsky pertenece la gramática que ha diseñado?

¿Podría diseñar una gramática de tipo 3 para dicho problema?

Lo primero que tenemos que hacer es comprender las restricciones que se nos plantean, de manera que vemos que estamos obligado a poner al menos una sala grande (g) por nivel, una sala secreta (x) que tiene que ir después de una sala grande de manera obligatoria, pero que solamente puede haber una por nivel, y por último tiene que haber como mínimo un tendero (t) por nivel, pero puede haber más, además de que obviamente la cadena tiene que cerrarse con un jefe final (j).

He conseguido diseñar la siguiente gramática de tipo 3 que se ajusta a todas las restricciones impuestas anteriormente. Para ello se siguen dos caminos distintos, uno en el cual el primer tendero aparece antes de la sala secreta, y el otro es en el que el primer tendero aparece después de la sala secreta. La gramática sería la siguiente:

$S \rightarrow pS \mid gS_1 \mid tS_3$

Ahora si no elegimos S -> tS_3 y ponemos la sala grande, pasamos a las siguientes opciones:

S_1 -> pS | gS_1 | xS_2 | tS_3

En la que nuevamente si no elegimos S_1 -> tS_3 y ponemos la sala secreta para seguir avanzando en la elaboración del nivel, ya solo nos faltaría por poner el tendero y la sala del jefe final de manera obligatoria, por lo que necesitamos forzar a que aparezca el tendero antes de poder cerrar el nivel, lo que conseguimos con las siguientes reglas:

S_2 -> pS_2 | gS_2 | tS_5

Una vez puesto el tendero ya podríamos pasar a cerrar el nivel, pero antes de ver esa regla, vamos a ver el otro camino posible que se consigue poniendo el primer tendero antes de la sala secreta, es decir eligiendo S -> tS_3 al principio o S_1 -> tS_3, de manera que ahora para cerrar el nivel necesitaríamos una sala secreta, y para forzar su aparición usamos las siguientes reglas:

S_3 -> pS_3 | gS_4 | tS_3

S_4 -> pS_3 | gS_4 | xS_5 | tS_3

Finalmente, con esto ya tendríamos todas las restricciones cumplidas a excepción de la de cerrar la cadena con un jefe final, para ello, vamos a usar las siguientes reglas, ya que antes de la sala de jefe final pueden aparecer todas las salas que quieran de las otras mientras no sean salas secretas. Por lo tanto, tenemos estas últimas reglas:

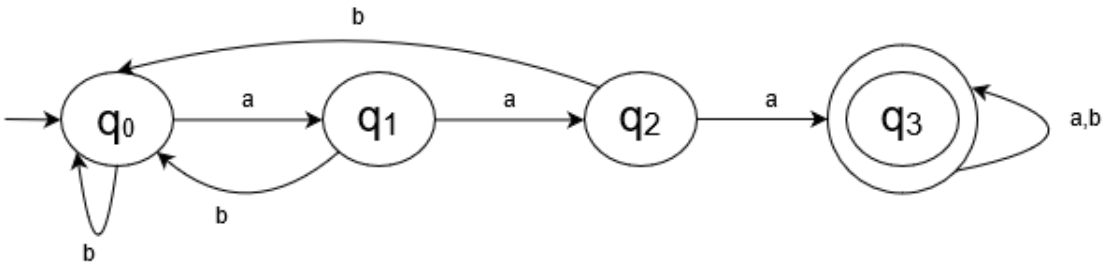
S_5 -> pS_5 | gS_5 | tS_5 | j

Como vemos todas las restricciones han sido cumplidas con éxito a la hora de diseñar el nivel, y la gramática elaborada es de tipo 3.

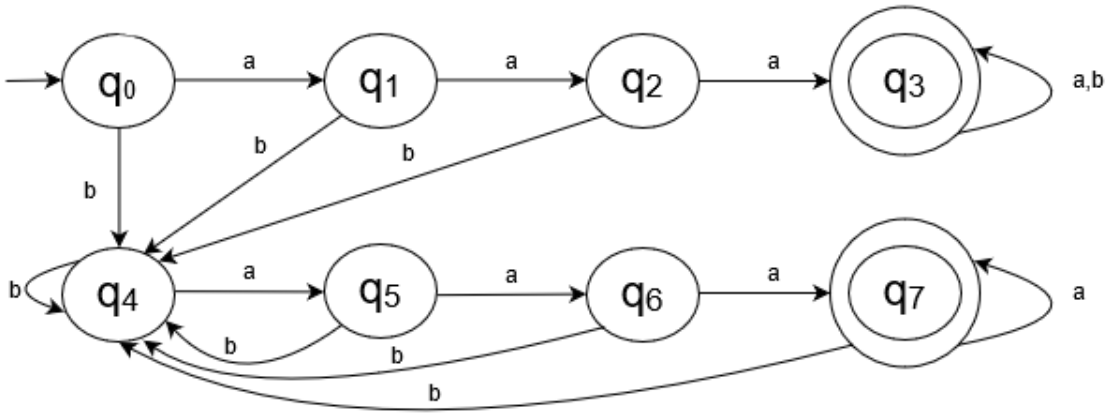
Práctica 2: AFD / AFND

1º) Construir un AFD que acepte cada uno de los siguientes lenguajes con alfabeto {a,b}:

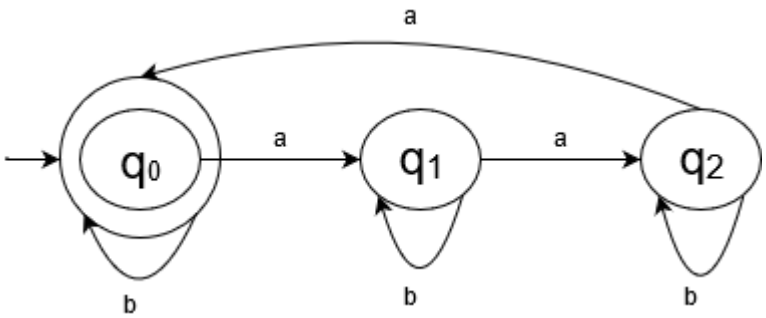
a) **El lenguaje de las palabras que contienen la subcadena aaa.**



b) **El lenguaje de las palabras que empiezan o terminan (o ambas cosas) en aaa.**

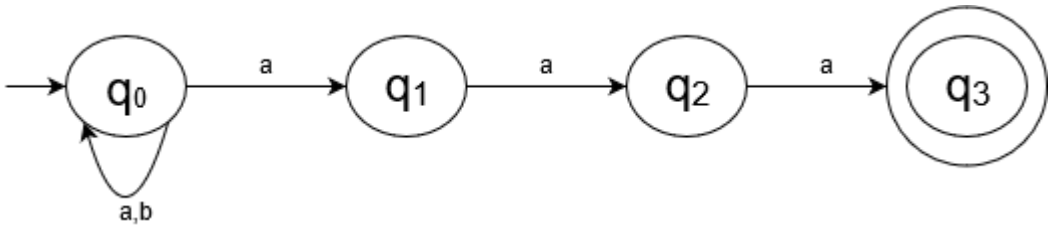


c) **El lenguaje formado por las cadenas donde el número de aes es divisible por 3.**

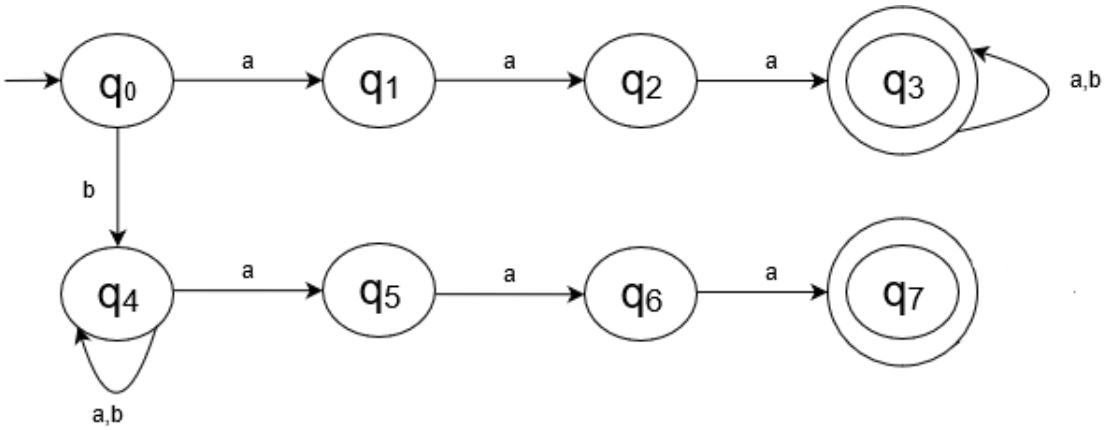


2º) <u>Construir un AFND que acepte cada uno de los siguientes lenguajes con alfabeto {a,b}:</u>

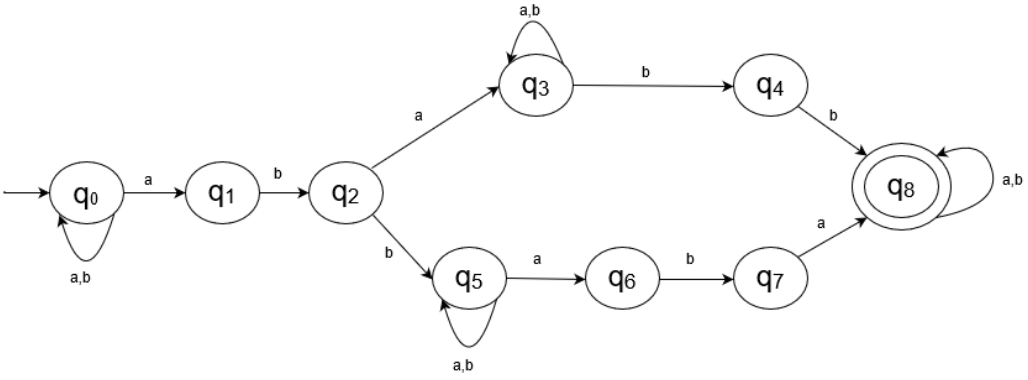
a) **<u>El lenguaje de las palabras que terminan en aaa.</u>**



b) **<u>El lenguaje de las palabras que empiezan o terminan (o ambas cosas) en aaa.</u>**



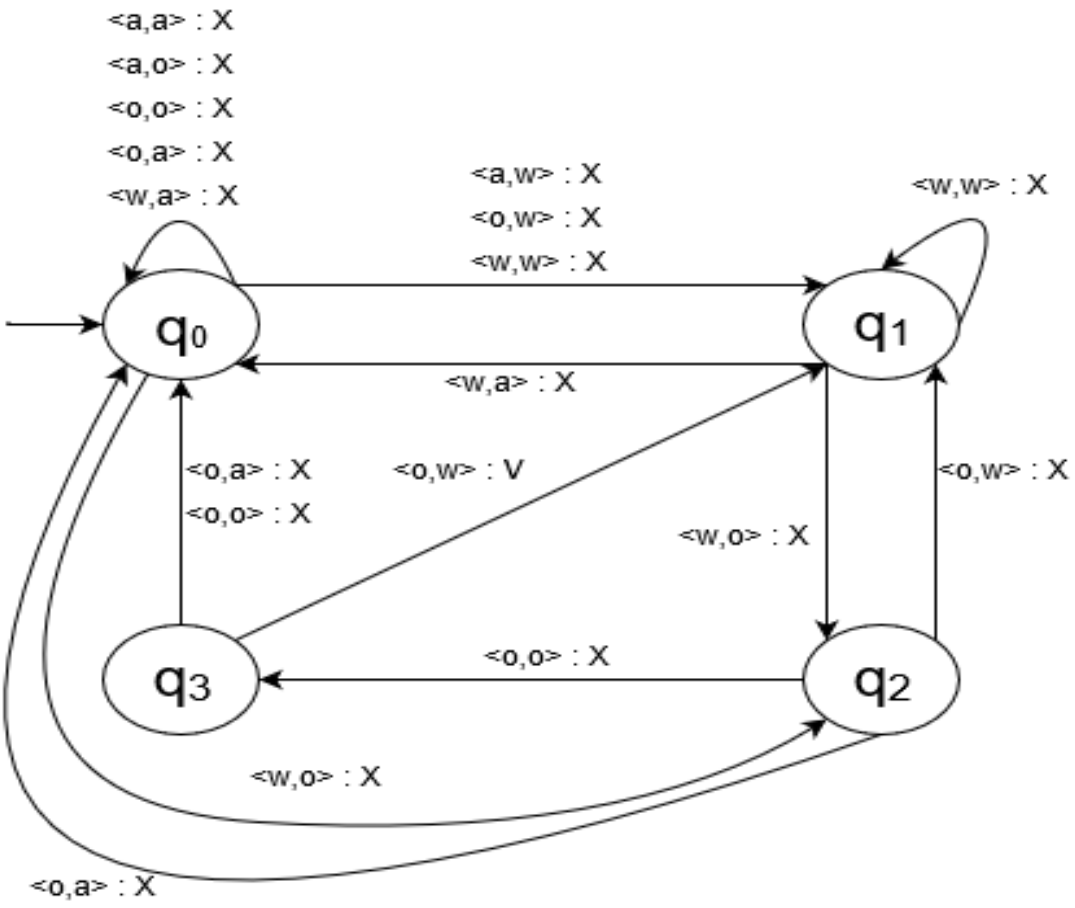
c) **<u>El lenguaje de las palabras que contengan, simultáneamente, las subcadenas aba y abb. Este AFND también acepta cadenas en la que estas subcadenas están solapadas (por ejemplo, las palabras "ababb" y "aaabbbaba" serían aceptadas).</u>**



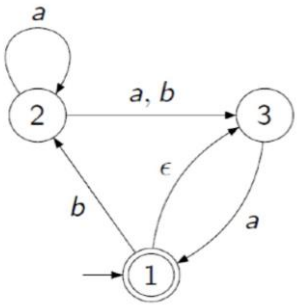
3º) <u>Diseña una Máquina de Mealy o de Moore que, dada una cadena usando el alfabeto A={'a','w','o'}, encienda un led verde (salida 'V') cada vez que se detecte la cadena "woow" en la entrada, apagándolo cuando lea cualquier otro símbolo después de esta cadena (representamos el led apagado con la salida "X"). El autómata tiene que encender el led verde (salida 'V'), tantas veces como aparezca en la secuencia "woow" en la entrada, y esta secuencia puede estar solapada. Por ejemplo, ante la siguiente entrada, la Máquina de Mealy/Moore emitirá la salida:</u>

entrada	aaawoawoowoowwoowa
salida	XXXXXXXXXVXXVXXXVX

He decidido hacer la siguiente Máquina de Mealy que satisface las indicaciones dadas en el enunciado:



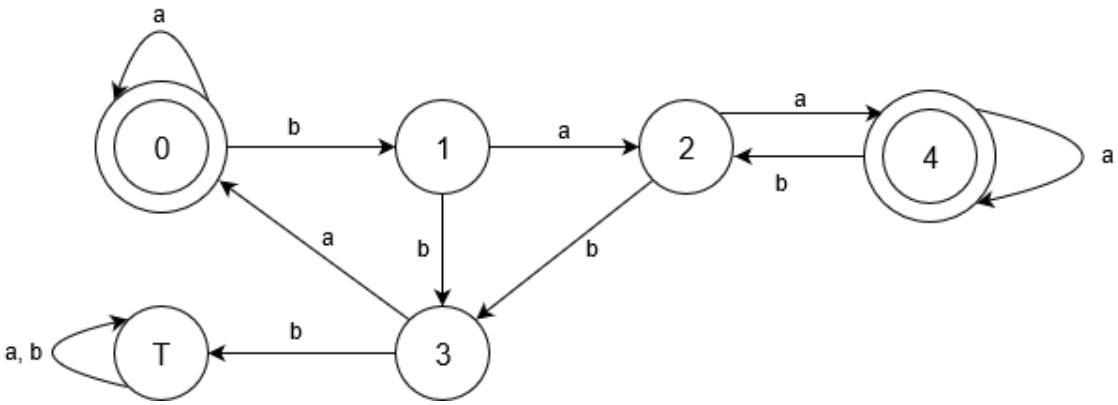
4º) <u>Obtener un AFD equivalente al AFND siguiente:</u>



Lo primero que vemos el tipo de palabras que acepta el AFND dado, las cuales son:

- Cadena vacía
- Infinitas aes
- $b^m a^n$ (y repetir esta secuencia infinitamente) m ϵ [0,2] n≥2

Por lo tanto, vemos que el AFD equivalente es:



Práctica 3: Lex

El programa que he implementado sirve para comprobar el precio de una carta del juego Magic: The Gathering en sus distintas ediciones, según la lista de precios de Metropolis Center la cual es una de las principales tiendas a nivel nacional. (Página de inicio -> http://www.mtgmetropolis.com/metropolistienda/index_tienda.asp).

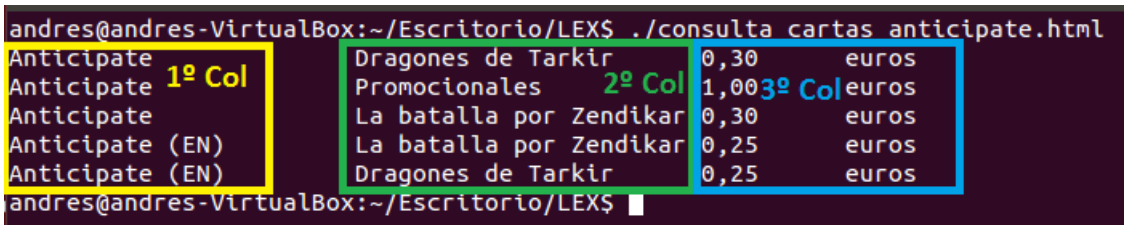
Aunque sería mejor que el programa recibiese el nombre de una carta y le hiciese un wget a la página buscando en la URL la carta pasada como argumento, no he podido hacerlo porque el tipo de URL que utiliza la página es .asp, y cuando buscamos una carta la URL sigue siendo .asp por lo que no puedo insertar la búsqueda directamente en la URL.

De modo que sobre lo que he trabajado para hacer el programa ha sido sobre HTMLs descargados con el navegador, es decir, busco una carta en la página, y la página que me muestra la descargo con el navegador (Firefox) dándole a guardar como…De esta manera consigo el HTML que me interesa, en el cual hay que buscar el nombre de la carta, la edición y el precio correspondiente a esa edición.

Ahora que ya sabemos cómo he conseguido el texto sobre el que trabajar, decir que la salida del programa es una tabla de 3 columnas que indican los siguiente:

- **1ª columna** -> el nombre de la carta en inglés, aunque a no ser que entre paréntesis ponga EN, la carta está en español. También pueden venir otras indicaciones como el estado de la carta, es decir, si ha sido jugada PLAYED, o ligeramente jugada SP, en caso de no venir estado, significa que la carta está nueva. Otra indicación sería si es FOIL o no, por defecto si no viene especificado significa que no lo es.
- **2ª columna** -> nombre de la edición de la carta. Esto se debe a que una carta puede estar publicada en varias ediciones distintas, y dependiendo de la edición a la que hagamos referencia el precio de la carta puede variar.
- **3º columna** -> precio de la carta en euros para el estado y edición que indica esa fila.

A continuación, vemos una ilustración que es un ejemplo de salida que daría el programa al buscar la carta llamada anticipar.



Como vemos nos encontramos que hay 5 tipos distintos de anticipar dependiendo de la edición y del idioma de la carta:

- La primera está en español, es de la edición Dragones de Tarkir y cuesta 0,3 euros.
- La segunda está en español, es de la edición de Promocionales y cuesta 1 euro.
- La tercera está en español, es de la edición La batalla por Zendikar y cuesta 0,3 euros.
- La cuarta está en inglés, es de la edición La batalla por Zendikar y cuesta 0,25 euros.
- La quinta está en inglés, es de la edición Dragones de Tarkir y cuesta 0,25 euros.

Todas las salidas son análogas a esta analizada.

En mi repositorio de GitHub se pueden encontrar varios HTMLs para probar el programa y darán una salida similar con sus datos correspondientes. Además, se puede ir a la página que he enlazado anteriormente, buscar cualquier carta de Magic, bajarse el HTML, pasárselo al programa como argumento y este tiene que dar un resultado correcto.

Las reglas definidas para encontrar lo que me interesa del HTML son las siguiente:

- style=.({especial}|{letra}|{digito})+.>[\b a-zA-Z(),]+<.a>
 { bool copia = false; string nombre; for(int i=0; i<yyleng; i++){ if(copia){nombre += yytext[i];} if(isspace(yytext[i])){copia = true;} } nombre.erase(nombre.end()-9,nombre.end()); nombres.push_back(nombre); } => sirve para encontrar la primera columna de la tabla (nombre, idioma y estado) meterlo en un string y añadir el string a un vector en el que iran todos los nombres escontrados.

- style=.({especial}|{letra}|{digito})+.>[\b a-zA-Z0-9]+<.a>\n **{ bool copia = false; string edicion; for(int i=0; i<yyleng; i++){ if(copia){edicion += yytext[i];} if(yytext[i] == '>'){copia = true;} } edicion.erase(edicion.end()-5,edicion.end()); ediciones.push_back(edicion); }** => sirve para encontrar la segunda columna (edición) meter la edición en un string y este en un vector de strings donde irán todas las ediciones en las que la carta está editada.

- align=.{letra}+.>[\.,0-9]+ **{ bool copia = false; string precio; int coincidencia = 0; for(int i=0; i<yyleng; i++){ if(copia && coincidencia>1){precio += yytext[i];} if(yytext[i] == '>'){copia = true; coincidencia++;} } precios.push_back(precio); }** => con esta encontramos el precio, lo introducimos en un string, este en un vector de strings donde irán todos los precios que tiene la carta con sus distintas características.

Además de estas reglas he tenido que dejar las que había en la plantilla de la práctica porque no me funcionaba lo de sobrescribir la regla por defecto, así que lo que he hecho ha sido sobrescribir las de la plantilla diciéndoles que no hagan nada. Así tendría las siguientes 3 reglas más:

- [^ \t\n]+ { }
- [\t]+ { }

- \n { }

Como vemos en las primeras 3 reglas, también he definido 3 alias, lo cuales son:

- letra [a-zA-Z]
- digito [0-9]
- especial [:#;]

Para finalizar adjunto el código completo del programa, para que pueda ser probado, además en mi repositorio de GitHub (https://github.com/Andresmag/Modelos_de_Computacion_UGR) podemos encontrarlo junto con varios archivos HTML para probarlo.

```
     /*----- Seccion de Declaraciones --------------*/
%{
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

vector<string> nombres;
vector<string> ediciones;
vector<string> precios;

%}

digito     [0-9]
letra      [a-zA-Z]
especial   [:#;]

%%
     /*----- Seccion de Reglas ----------------*/
style=.({especial}|{letra}|{digito})+.>[\b a-zA-Z(),]+<.a><br>
          { bool copia = false; string nombre; for(int i=0;
i<yyleng;     i++){     if(copia){nombre     +=     yytext[i];}
if(isspace(yytext[i])){copia         =         true;}         }
nombre.erase(nombre.end()-9,nombre.end());
nombres.push_back(nombre); }

style=.({especial}|{letra}|{digito})+.>[\b a-zA-Z0-9]+<.a>\n
     { bool copia = false; string edicion; for(int i=0; i<yyleng;
i++){ if(copia){edicion += yytext[i];} if(yytext[i] == '>'){copia
=    true;}    }    edicion.erase(edicion.end()-5,edicion.end());
ediciones.push_back(edicion); }

align=.{letra}+.><strong>[\.,0-9]+                    {     bool
copia = false; string precio; int coincidencia = 0; for(int i=0;
i<yyleng; i++){ if(copia && coincidencia>1){precio += yytext[i];}
```

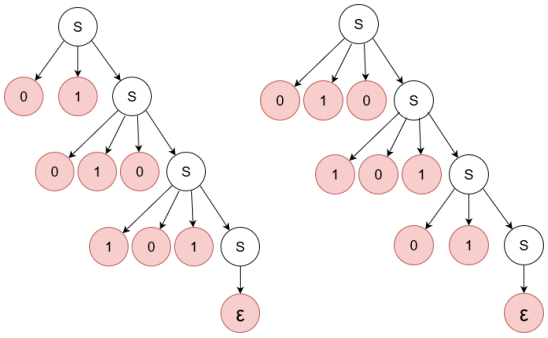
```
if(yytext[i]   ==   '>'){copia   =   true;   coincidencia++;}   }
precios.push_back(precio); }

[^ \t\n]+   { }
[ \t]+     { }
\n      { }

%%
      /*----- Seccion de Procedimientos --------*/
int main(int argc, char *argv[]){
      if(argc == 2){
            yyin = fopen (argv[1], "rt");
            if (yyin == NULL){
                  printf("El  fichero  %s  no  se  puede  abrir\n",
argv[1]);
                  exit (-1);
            }
      }
      else yyin = stdin;

      yylex();
      for(int i=0; i<nombres.size(); i++){
            cout << left << setw(20) << nombres[i] << "\t" << left
<<  setw(23)  <<  ediciones[i]  <<  "\t"  <<  left  <<  setw(10)  <<
precios[i] << "euros" << endl;
      }

      return 0;
}
```

Práctica 4: Ejercicios prácticos sobre Lenguajes libres de contexto

1) **Determinar cuáles de las siguientes gramáticas son ambiguas y, en su caso, comprobar si los lenguajes generados son inherentemente ambiguos. Justificar la respuesta.**

a) S -> 01S | 010S | 101S | ε

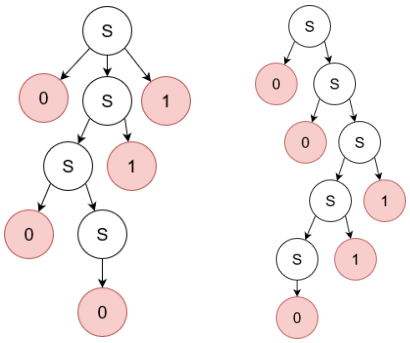
Esta gramática es ambigua, ya que para obtener la palabra 01010101 se pueden desarrollar dos árboles de derivación al menos.



Por su parte el lenguaje que nos da esta gramática es un lenguaje que puede empezar por 01 o 10, luego tener cualquier combinación de ceros y unos mientras no haya más de dos ceros o dos unos juntos, y tiene que terminar en 01 o 10. Todas las gramáticas que he encontrado para este lenguaje son ambiguas, por lo tanto, asumo que el lenguaje es inherentemente ambiguo.

b) S -> 0S1 | S1 | 0S | 0

Esta gramática es ambigua porque por ejemplo para la palabra 00011 hay al menos dos árboles de derivación.



El lenguaje que genera esta gramática es L(G) = $\{0^n1^m / n \geq 1\ m \geq 0\}$.

Pero para este lenguaje si he encontrado otra gramática que lo genera la cual no es ambigua, por lo tanto, el lenguaje no es inherentemente ambiguo. La gramática que he encontrado es la siguiente:

$S \rightarrow 0S \mid 0S_1 \qquad S_1 \rightarrow 1S_1 \mid \varepsilon$

c) S -> A1B A -> 0A | ε B -> 0B | 1B | ε

Esta gramática no es ambigua, ya que no he encontrado ninguna palabra que pueda tener dos árboles de derivación distintos, además de que el lenguaje que genera es $L(G) = \{0^n1u / n \geq 0 \ \ u \in A^*\}$.

Como vemos todas las palabras que puede generar están muy bien cuadradas, es decir, siempre va una secuencia de ceros (puede ser ninguno), los cuales solo se pueden poner con las reglas de derivación de la variable A y siempre se añaden a la izquierda de la palabra. Luego tiene que ir obligatoriamente un 1, y por último, después del 1 puede ir absolutamente cualquier combinación de ceros y unos (puede no ir ningún 0 ni ningún 1), pero esta combinación se genera solamente con las reglas de derivación de la variable B que como vemos ponen un símbolo terminar y luego otra vez la variable B de modo que las reglas siempre se van a ir disparando en el orden en que el aparezcan los símbolos en la palabra, sin poder dar lugar a que haya más de un árbol de derivación posible por palabra. Por lo tanto, podemos afirmar que es una gramática no ambigua.

2) <u>Eliminar símbolos y producciones inútiles. Realizar el procedimiento paso por paso, indicando las variables descartadas y el motivo.</u>

S -> moA; S -> cI; A -> dEs; A -> jBI;
B -> bb; B -> D; E -> elO; E -> Perl;
D -> de; C -> c; J -> kC; I -> fI;
O -> o; P -> oIa;

Lo primero que hacemos es identificar las reglas que derivan únicamente en símbolos terminales, meter en V_T la variable a la izquierda de la regla y "quitar" todas las reglas con esa variable a la izquierda de lista para no tenerlas en cuenta las siguientes pasadas. V_T queda de la siguiente manera:

$V_T = \{B, C, D, O\}$

El siguiente paso es volver a repasar la lista de reglas que nos quedan para ver si en su parte derecha todas las variables que tienen están en V_T, y en caso de que así sea, meter la variable a la izquierda de la regla en V_T y "quitar" todas las reglas con esa variable a la izquierda de la lista. Tras la segunda pasada a la lista V_T quedaría así:

$V_T = \{B, C, D, O, E, J\}$

Ahora volvemos a repetir el paso anterior, y seguiremos haciéndolo hasta que al recorrer la lista de reglas ninguna se añada a V_T.

3ª pasada -> $V_T = \{B, C, D, O, E, J, A\}$

4º pasada -> V_T = {B, C, D, O, E, J, A, S}

En la quinta pasada ya no se añade ninguna variable a V_T, por lo que ahora lo que tenemos que hacer es eliminar las reglas que tengas las variables que no se han metido en V_T, es decir {I, P}, y por lo tanto las reglas que nos quedan son:

S -> moA; A -> dEs;
B -> bb; B -> D; E -> elO;
D -> de; C -> c; J -> kC;
O -> o;

Ahora vamos analizando las reglas que nos quedan una a una, y metiendo en V_s las variables analizadas, en J las variables a analizar, y en T_s los símbolos terminales.

Para empezar, tenemos J = {S} para empezar, así que analizamos su regla y metemos en V_s = {S}, J = {A} y en T_s = {m, o}.

Seguimos con la siguiente variable en J, la cual es A y la analizamos de modo que nos queda en V_s = {S, A}, J = {E} y en T_s = {m, o, d, s}.

Continuamos con la E y ahora tenemos en V_s = {S, A, E}, J = {O} y en T_s = {m, o, d, s, e, l}.

Y finalmente analizamos la última variable en J, la cual es la O, y se nos queda en V_s = {S, A, E, O}, J = {} y en T_s = {m, o, d, s, e, l}.

Por lo que las reglas que no tengan a su lado izquierdo una de las variables de V_s significa que son inútiles ya que no son accesibles, por lo que la gramática finalmente se nos quedaría de la siguiente forma:

S -> moA; A -> dEs; E -> elO; O -> o;

Que como vemos nos da el siguiente resultado:

S => moA => modEs => modelOs => modelos

3) Eliminar producciones nulas y unitarias, en el orden correcto. Realizar los procedimientos paso por paso, indicando las producciones descartadas en cada momento.

S -> XYZ S -> XYz X -> xxX X-> ε

Y -> yyY Y -> ε Z -> yxZ Z -> X

Lo primero que tenemos que hacer es meter en H el conjunto de variables anulables, las cuales son las que derivan en la cadena vacía, por lo tanto, para empezar, metemos en H = {X, Y}, y acto seguido metemos también Z por la regla de Z -> X, y como hemos metido Z, tenemos que meter S por la regla de S -> XYZ, de modo que H = {X, Y, Z, S}.

Lo siguiente es quitar las producciones que dan a la cadena vacía directamente, con lo que la gramática se nos queda de la siguiente forma:

S -> XYZ S -> XYz X -> xxX
Y -> yyY Z -> yxZ Z -> X

Ahora tenemos que para cada regla que nos queda crear otras a partir de ellas, probando a quitar las distintas combinaciones posibles con las variables en H, así de S -> XYZ saldrían

S -> XY S -> YZ S -> XZ S-> X S -> Y S -> Z

De S -> XYz salen: S -> Yz S -> Xz S -> z

De X -> xxX sale: X -> xx

De Y -> yyY sale: Y -> yy

Y por último de Z -> yxZ sale: Z-> yx

Así la gramática que tenemos ahora mismo es:

S -> XYZ S -> XYzX -> xxXY -> yyYZ -> yxZ Z -> X

S -> XY S -> YZ S -> XZ S-> X S -> Y S -> Z

S -> Yz S -> Xz S -> z X -> xx Y -> yy Z-> yx

Y ahora toca eliminar las producciones unitarias, para ello metemos en H las reglas que solo deriven a una variable, y metemos los dos lados de la producción. Quedando de la siguiente manera H = {(Z, X), (S, X), (S, Y), (S, Z)}

Y eliminamos dichas producciones de la gramática, pero para compensarlo tenemos que poner las distintas combinaciones que sale de dejar en el lado izquierdo la variable que estaba, y en el lado derecho poner las distintas derivaciones que se nos dan desde la variable que está en ese lado.

Así de Z -> X, al eliminarla, tenemos que añadir Z -> xxX Z -> xx

De S -> X tenemos que añadir S -> xxX S -> xx

De S -> Y tenemos que añadir S-> yyY S-> yy

Y de S -> Z tenemos que añadir S -> yxZ S -> yx

Por lo tanto, la gramática final resultante es:

S -> XYZ S -> XYzX -> xxXY -> yyYZ -> yxZ S -> XY S -> YZ S -> XZ

S -> Yz S -> Xz S -> z X -> xx Y -> yy Z-> yx

Z -> xxX Z -> xx S -> xxX S -> xx

S-> yyY S-> yy S -> yxZ S -> yx

4) Pasar la siguiente gramática a forma normal de Greibach:

S -> a |CD | CS

A -> a | b | SS

C -> a

D -> AS

Si analizamos la gramática que nos dan, vemos que está en forma normal de Chomsky, por lo tanto, para pasarla a la forma normal de Greibach solo hay que hacer el último paso, el cual es cambiar las producciones para que todas sean del tipo A -> aα, donde A es una variable, a un símbolo terminal, y α es un conjunto de variables (puede ser ninguna).

Comenzamos por la última producción, es decir, D -> AS, y lo que hacemos es eliminarla, sustituyendo A por sus posibles derivaciones. De esta manera nos aparecen:

D -> SSS D -> aS D -> bS

De las cuales solo D -> SSS no está en la forma que buscamos, así que repetimos el proceso sobre ella, es decir, la eliminamos y obtenemos las producciones correspondientes a las derivaciones de la primera S.

D -> aSS D-> CDSS D -> CSSS

Ahora son D -> CDSS y D -> CSSS sobre las cuales necesitamos hacer el proceso, y así obtenemos:

D -> aDSS D -> aSSS

Como ya todas las que tienen a la izquierda D están como queremos, pasamos a las que tiene la A a la izquierda, y vemos que A -> SS no está como nos interesa, así que aplicamos el procedimiento anterior y obtenemos:

A -> aS A -> CDS A ->CSS

Nuevamente, nos salen dos que no nos interesan, sobre las cuales aplicar el procedimiento para obtener:

A -> aDS A -> aSS

Y ya por último nos quedan las que tiene la S a la izquierda, de las cuales son S -> CD y S -> CS las que no nos interesan y las eliminamos obteniendo a cambio:

S -> aD S -> aS

Así que al final la gramática resultante en forma normal de Greibach es:

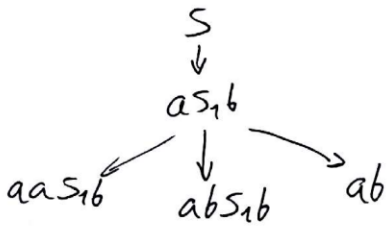
S -> a	S -> aD	S -> aS	A -> a	A -> b
A -> aS	A -> aDS	A -> aSS	C -> a	D -> aS
D -> bS	D -> aSS	D->aDSS	D -> aSSS	

Ejercicios hechos en clase

Día 7/10/16

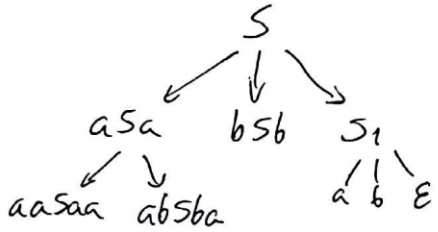
Gramáticas

a) $S \to aS_1b \quad S_1 \to aS_1 \mid bS_1 \mid \varepsilon \quad A = \{a,b\}$



$L(G) = \{ aub \ / \ u \in A^* \}$

b) $S \to aSa \mid bSb \mid S_1 \quad S_1 \to a \mid b \mid \varepsilon \quad A = \{a,b\}$



$L(G) = \{ uwu^{-1} \ / \ u \in A^* \ w \in \{a,b,\varepsilon\} \}$

c) $S \to aSb \mid aS_1b \quad S_1 \to cS_1d \mid \varepsilon$

$L(G) = \{ a^i c^j d^j b^i \ / \ i \geq 1 \ j \geq \emptyset \}$

d) $S \to S_1bbS_1 \quad S_1 \to aS_1 \mid bS_1 \mid \varepsilon$

$L(G) = \{ ubbv \ / \ u,v \in A^* \}$

e) $S \to BA \quad B \to \varepsilon \mid bB \quad A \to \varepsilon \mid aA$

$L(G) = \{ b^i a^j \ / \ i,j \geq \emptyset \}$

•) Modificar la gramática anterior (e)) para que sea regular, pero siga generando el mismo lenguaje.

$S \to bS_1 | aS_2 | \varepsilon \quad S_1 \to bS_1 | aS_2 | \varepsilon \quad S_2 \to aS_2 | \varepsilon$

Día 14/10/16

a) $L(G) = \{a^i b^j c^{i+j}\}$

$S \to aSc | B \quad B \to bBc | \varepsilon$

b) $L(G) = \{ab^i c^j d\}$

$S \to aB \quad B \to bB | C \quad C \to cC | d$

•) $A = \{a, b\}$ Crear una gramática que genere palabras que no contengan la subcadena bbb

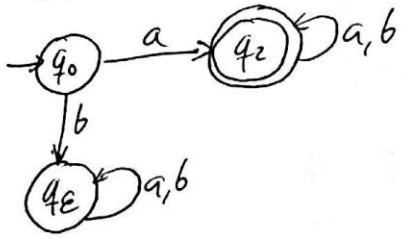
$S \to aS | bS_1 | \varepsilon \quad S_1 \to aS | bS_2 | \varepsilon \quad S_2 \to aS | \varepsilon$

Día 21/10/16

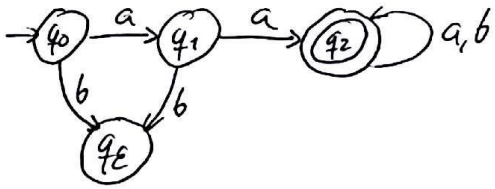
AFD

$A = \{a, b\}$

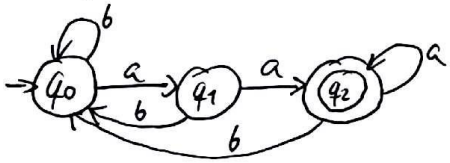
a) Autómata finito determinista que acepte las palabras que empiezan en a.



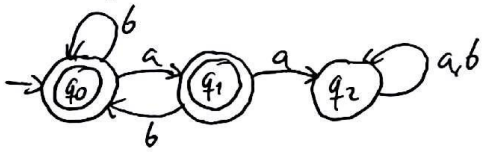
b) Palabras que empiezan por aa



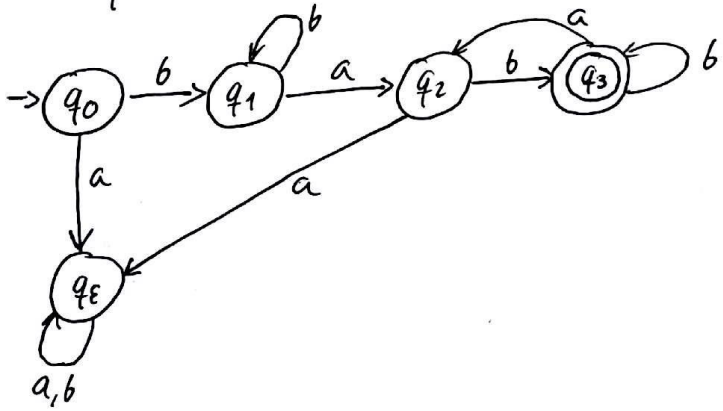
c) Palabras que terminan en aa



d) Palabras que no contienen aa



e) Palabras que cada a vaya precedida y seguida por una b, y tiene que contener al menos una a

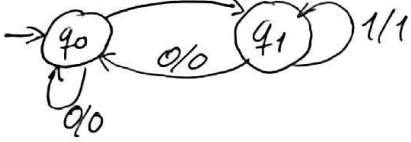


Día 4/11/16

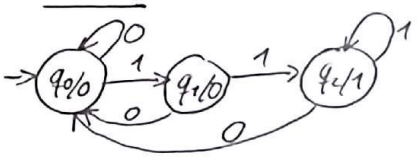
Máquinas de Mealy y Moore

•) Máquina de Mealy o Moore que de salida 1 mientras la cadena esté terminando en 11, en otro caso da salida 0.

Mealy

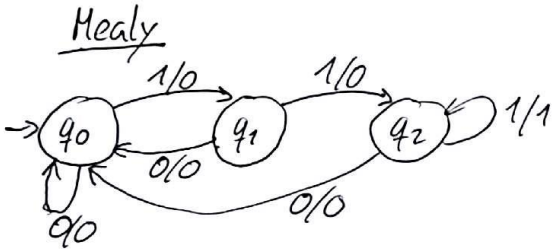


Moore

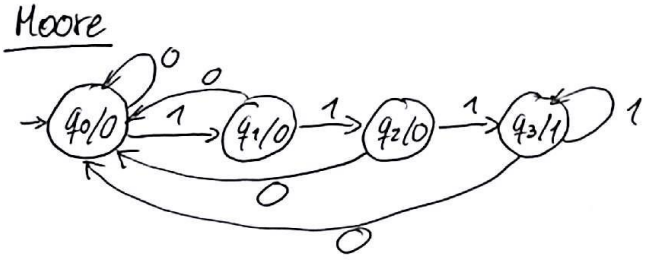


Y si es con 111

Mealy

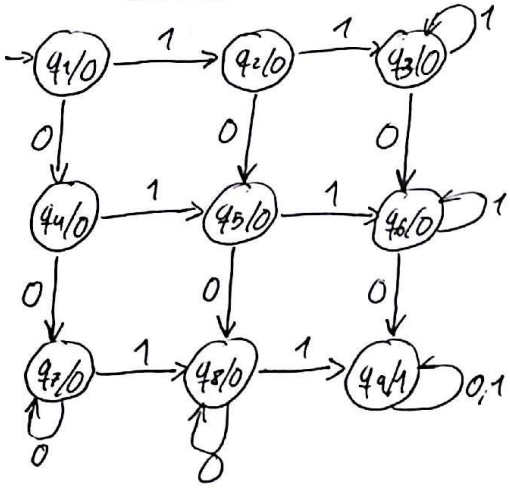


Moore

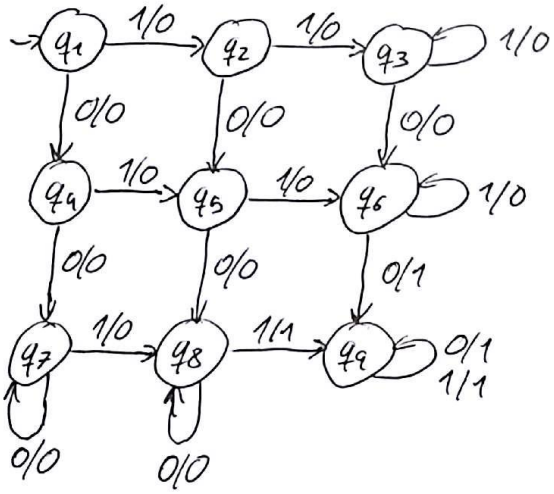


•) Máquina de Mealy o Moore, si llega a 2 ceros y 2 unos en total, en cualquier orden, la salida se activa y ya solo da 1 de salida

Moore



Mealy



Día 2/12/16

Lema de Bombeo

a) $A = \{a^n b^n / n \in \mathbb{N}\}$

$N = \underbrace{a^N b^N}_{xy\ z} = x\ y\ z \rightarrow a^{N-r-s} b^N$

a^r a^s

$r+s \leq N$

$s \geq 1$

Si ahora en xy^iz hacemos $i=2$ obtenemos $\Rightarrow a^r a^s a^s a^{N-r-s} b^N =$

$= a^{N+s} b^N$

$\hookrightarrow xy > N \Rightarrow$ No cumple la restricción del Lema de bombeo

Por lo tanto, demostramos que $xy^iz \notin A$

b) $A=\{0,1\}$ $L=\{uu \mid u\in A^*\}$

$w=0^N1^N0^N1^N = xyz \to 0^{N-r-s}1^N0^N1^N$

xy | z — 0^r 0^s $r+s\leq N$ $s\geq 1$

Hacemos $i=2 \Rightarrow 0^r0^s0^s0^{N-r-s}1^N0^N1^N = 0^{N+s}1^N0^N1^N$

Por lo tanto vemos que $xy^2z \notin L$

c) $A=\{0,1\}$ $L=\{u \mid u\in A^*, u=u^{-1}\}$

$w=0^N10^N = xyz \to 0^{N-r-s}10^N$

xy | z — 0^r 0^s $r+s\leq N$ $s\geq 1$

Hacemos $i=2 \Rightarrow 0^r0^s0^s0^{N-r-s}10^N = 0^{N+s}10^N$

Por lo tanto vemos que $xy^2z \notin L$

Día 9/12/16

Eliminar símbolos y producciones inútiles, transiciones nulas, producciones unitarias.

•) $S\to A \mid BCa \mid aDcd \mid EDF$ $A\to aAb \mid c$ $B\to CD \mid ECd \mid Ad \mid \varepsilon$

$C\to Cc \mid Bb \mid AaE \mid c$ $D\to aDd \mid Dd \mid \varepsilon$ $E\to aaEB \mid EFG$

A)

$V_T=\{A,B,C,D,S\}$

1º) $V-V_T$

$S\to A \mid BCa \mid aDcd$

$A\to aAb \mid c$

$B\to CD \mid Ad \mid \varepsilon$

$C\to Cc \mid Bb \mid c$

$D\to aDd \mid Dd \mid \varepsilon$

2º) $V_S=\{S,A,B,C,D\}$

$J=\{A,B,C,D\}$

$T=\{a,c,d,\varepsilon\}$

B)

$S \to A \mid BCa \mid aDcd$
$A \to aAb \mid c$
$B \to CD \mid Ad \mid \varepsilon$
$C \to Cc \mid Bb \mid c$
$D \to aDd \mid Dd \mid \varepsilon$

$\to H = \{B, D\} \to$

$S \to A \mid BCa \mid aDcd \mid Ca \mid acd$
$A \to aAb \mid c$
$B \to CD \mid C \mid Ad$
$C \to Cc \mid Bb \mid b \mid c$
$D \to aDd \mid ad \mid Dd \mid d$

C)

$H = \{(S, A), (B, C)\}$

$S \to BCa \mid aDcd \mid Ca \mid acd \mid aAb \mid c$
$A \to aAb \mid c$
$B \to CD \mid Ad \mid Cc \mid Bb \mid b \mid c$
$C \to Cc \mid Bb \mid b \mid c$
$D \to aDd \mid ad \mid Dd \mid d$

Día 16/12/16

Autómatas con pila - criterio de estados finales

•) $B = \{R, X, Y\}$ $L = \{0^i 1^j 0^j 1^i \mid i, j > 0\}$

$\delta(q_1, 0, R) = (q_1, XR)$
$\delta(q_1, 0, X) = (q_1, XR)$
$\delta(q_1, 1, X) = (q_2, YX)$
$\delta(q_2, 1, Y) = (q_2, YY)$
$\delta(q_2, 0, Y) = (q_3, \varepsilon)$
$\delta(q_3, 0, Y) = (q_3, \varepsilon)$
$\delta(q_3, 1, X) = (q_4, \varepsilon)$
$\delta(q_4, 1, X) = (q_4, \varepsilon)$
$\delta(q_4, \varepsilon, R) = (q_4, \varepsilon)$

Día 13/1/17

•) $L=\{0^i 1^j 2^k 3^{i+j+k} 4 \ / \ i,j,k \geq 0\}$

Hacer gramática y autómata con pila

$S \to Ae \quad A \to B \mid aAd \quad B \to C \mid bBd \quad C \to \varepsilon \mid cCd$

$0 \equiv a \quad 1 \equiv b \quad 2 \equiv c \quad 3 \equiv d \quad 4 \equiv e$

$\delta(q,a,a) = (q,\varepsilon)$
$\delta(q,b,b) = (q,\varepsilon)$
$\delta(q,c,c) = (q,\varepsilon)$
$\delta(q,d,d) = (q,\varepsilon)$
$\delta(q,e,e) = (q,\varepsilon)$

$\delta(q,\varepsilon,S) = (q,Ae)$
$\delta(q,\varepsilon,A) = \{(q,B),(q,aAd)\}$
$\delta(q,\varepsilon,B) = \{(q,C),(q,bBd)\}$
$\delta(q,\varepsilon,C) = \{(q,\varepsilon),(q,cCd)\}$