MODELOS DE LA COMPUTACIÓN
PRÁCTICAS

SERGIO SAMANIEGO MARTÍNEZ
3º INGENIERÍA INFORMÁTICA
CURSO 16/17

Práctica 1 – Ejercicios Prácticos sobre Gramáticas y Lenguajes

1. Describir el lenguaje generado por las siguientes gramáticas en {0,1}*.

a) $S \rightarrow 0\,S_1\,1 \qquad S_1 \rightarrow 0\,S_1 \mid 1\,S_1 \mid \varepsilon$

$L = \{0\,a^i\,b^j\,1 \;/\; a,b \in \{0,1\} \text{ y } i,j \in \mathbb{N}\{0,1,2,\ldots,n\}\}$

Tal y como se describe anteriormente, el lenguaje generado por la gramática anterior es una cadena que comienza en 0 y termina en 1, el resto de la cadena puede estar formado por sucesiones de ceros y unos.

b) $S \rightarrow S1\ 101\ S1 \qquad S1 \rightarrow 0\ S1 \mid 1\ S1 \mid \varepsilon$

$L = \{a^i b^j\,101\,a^i b^j \mid a,b \in \{0,1\} \text{ y } i,j \in \mathbb{N}\{0,1,2,\ldots,n\}\}$

El lenguaje generado por la gramática consiste en una cadena que comieza con un determinado número de ceros, seguidos de otra sucesión de unos. En medio de la cadena está la subcadena 101, y para finalizar contiene la misma subcadena que había antes de la subcadena central 101.

c) $S \rightarrow 0\ S\ 1 \mid S1 \qquad S1 \rightarrow 1\ S1\ 0 \mid 1\ S2\ 0 \qquad S2 \rightarrow 0\ S2\ 1 \mid \varepsilon$

$L = \{0^i 1^j 0^k 1^k 0^j 1^i \mid i,j \in \mathbb{N}^* \text{ y } k \in \mathbb{N}\{0,1,2,\ldots,n\}\}$

Este lenguaje genera una cadena que podemos dividir en dos partes, la izquierda y la derecha. Hablando de la parte de la izquierda, generará primero una sucesión de ceros, le seguirá una sucesión de unos y para acabar la parte izquierda puede contener otra sucesión de ceros. La parte derecha contendrá el mismo número de símbolos que la izquierda pero cambiando los ceros por unos y los unos por ceros.

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”.

G = {V, T. P, S}   V = {S, S1, S2, A}   T = {1,0}   S = S

P contiene las siguientes reglas de producción:

$S \rightarrow 1\ S1\ 1 \qquad S1 \rightarrow 0\ S2\ 0$

$S2 \rightarrow A\ 0 \qquad A \rightarrow 0\ A$

$A \rightarrow 1\ A \qquad A \rightarrow \varepsilon$

Una gramática de tipo 3 que la genere puede ser la siguiente:

S → 1 S1 1 S1 → 0A 00

A → 0 A

A → 1 A A → ε

b) Palabras que tienen 2 o 3 "0".

G = {V, T. P, S} V = {S, S1, S2, A} T = {1,0} S = S

P contiene las siguientes reglas de producción:

S → A S1 0 S1 → S2 0

S2 → ε S2 → A 0

A → ε A → 1 A

Una gramática de tipo 3 puede ser la siguiente:

S → 0 A S1 S1 → 0 S2

S2 → ε S2 → 0 A

A → ε A → 1 A

c) Palabras que no contienen la subcadena "011".

G = {V, T. P, S} V = {S,S1} T = {1,0} S = S

P contiene las siguientes reglas de producción:

S → 0 S1 S1→ 1 S

S → ε

La gramática generada es de tipo 3

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?

G = {V, T, P, S}   V = {S, S1, S2, S3, S4}   T = {g,p,t,x,j}   S = S

P contiene las siguientes producciones:

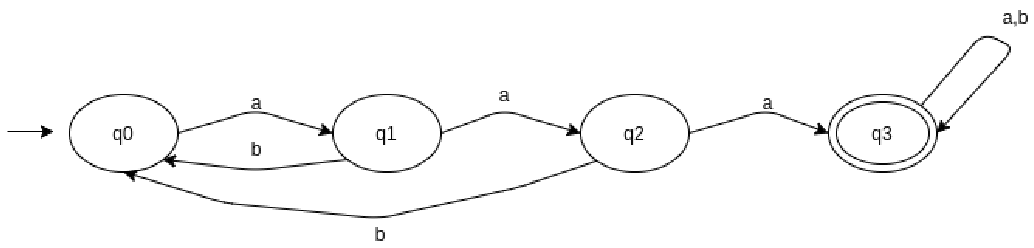
S → g S	S → p S	S → t S1
S1 → g S1	S1 → p S1	S1 → g S2
S2 → x S3		
S3 → j S4	S3 → p S3	S3 → g S3
S4 → ε		

La gramática que hemos generado es una gramática de tipo 3 o también llamada independiente de contexto.

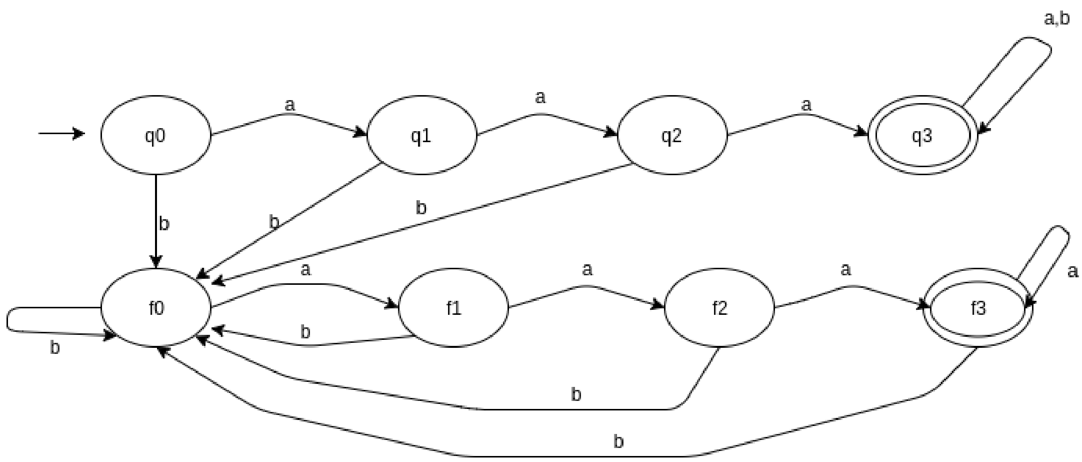
Práctica 2 – Autómatas Finitos Deterministas / Autómatas Finitos No Deterministas

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

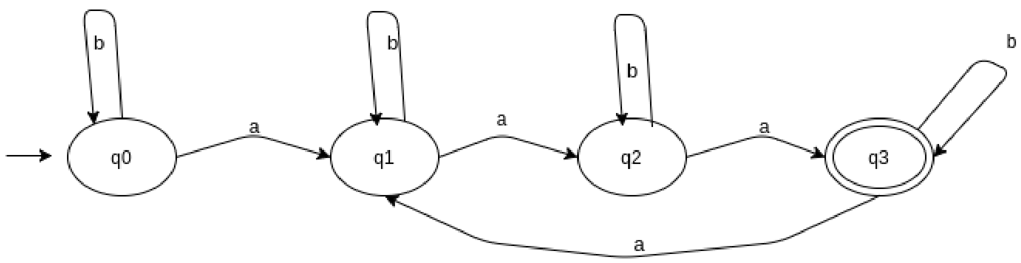
a. El lenguaje de las palabras que contienen la subcadena aaa.



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

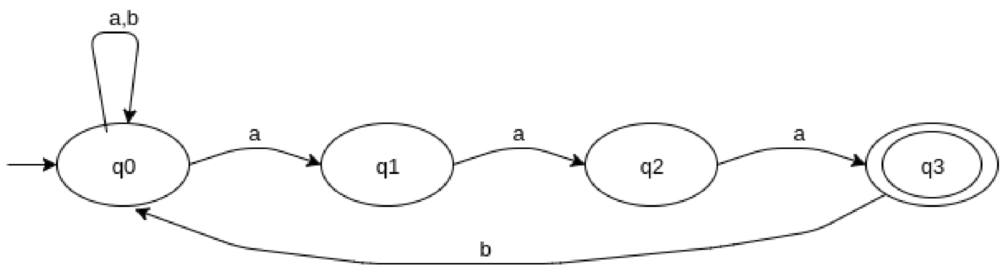


c. El lenguaje formado por las cadenas donde el número de a's es divisible por 3.

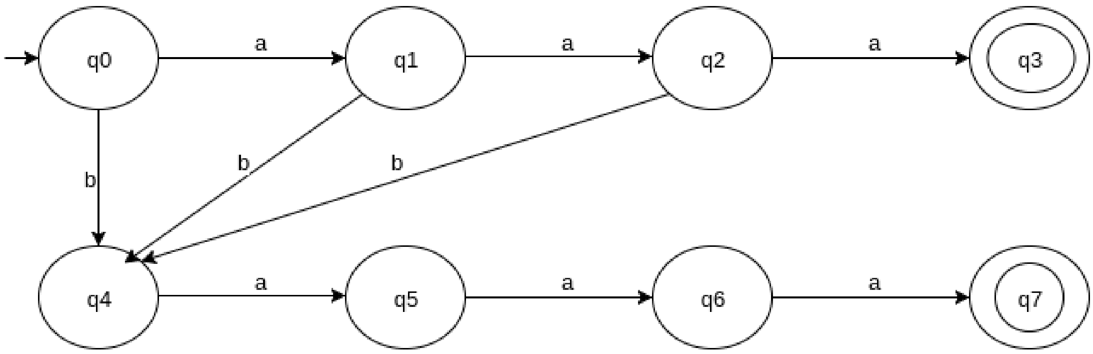


2. Construir un AFND que acepte cada uno de los siguientes lenguajes con alfabeto {a,b}:

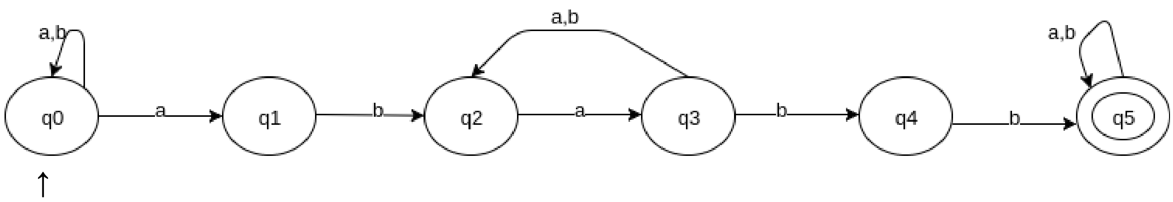
a. El lenguaje de las palabras que terminan en aaa.



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

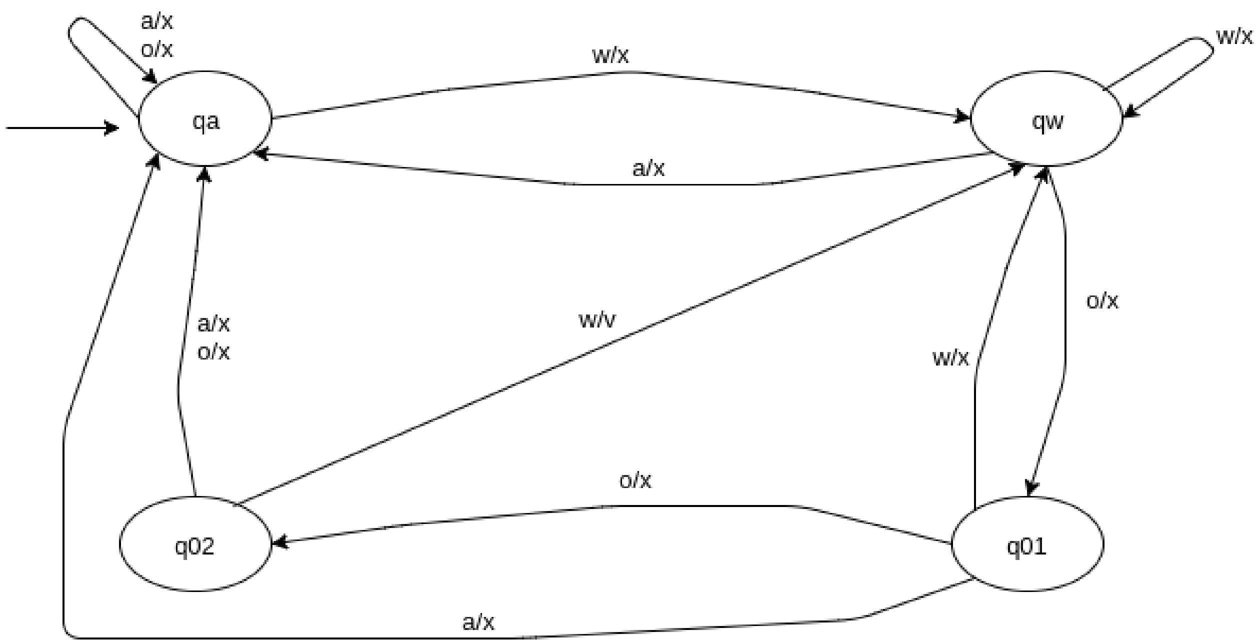


c. El lenguaje de las palabras que contengan, simultáneamente, las subcadenas aba y abb.
Este AFND también acepta cadenas en la que estas subcadenas están solapadas (por ejemplo, las palabras "ababb" y "aaabbbaba" serían aceptadas).

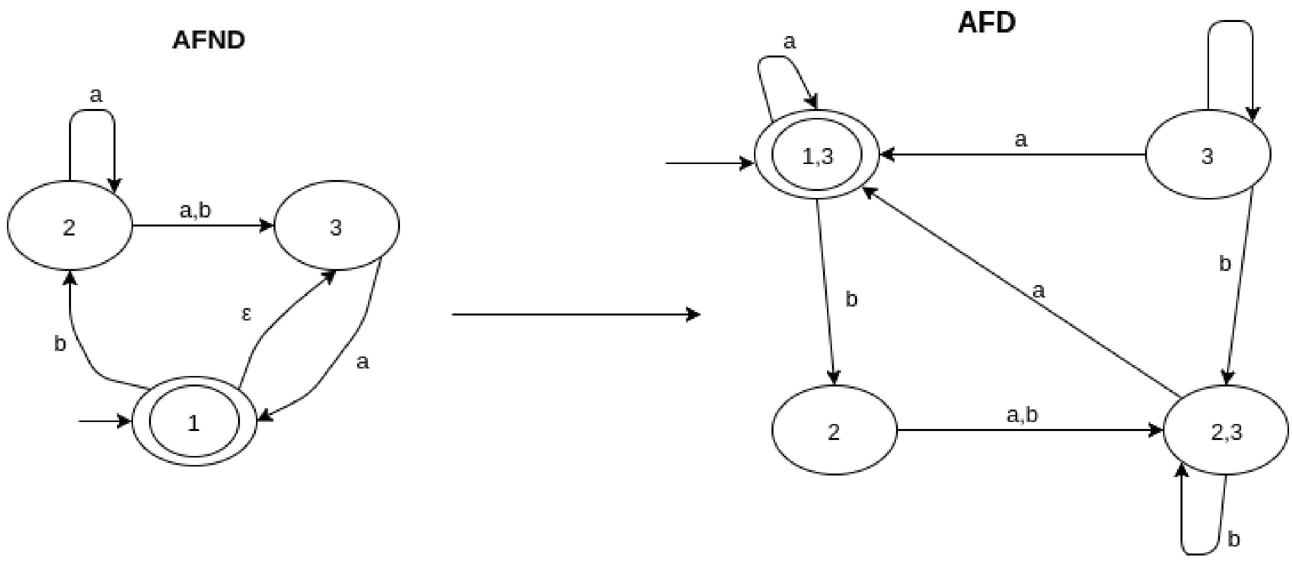


3. Diseñar una Máquina de Mealy o de Moore que, dada una cadena usando el alfabeto A={'a', 'w', 'o'}, encienda un led verde (salida 'V') cada vez que se detecte la cadena "woow" en la entrada, apagándolo cuando lea cualquier otro símbolo después de esta cadena (representamos el led apagado con la salida"X"). 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:

Entrada	aaawoawoowoowwoowa
salida	XXXXXXXXXVXXVXXXVX



4. Obtener un AFD equivalente al AFND siguiente:



<u>Tabla de transiciones</u>

	a	**b**
1,3	1,3	2
2,3	1,3	2,3
2	2,3	2,3
3	1,3	1,3

Práctica 3 - Lex como localizador de expresiones regulares con acciones asociadas

1. Pensar un problema original de procesamiento de textos. Para la resolución de este problema debe ser apropiado el uso de Lex, o sea, se debe resolver mediante el emparejamiento de cadenas con expresiones regulares y la asociación de acciones a cada emparejamiento. Se presentará una descripción por escrito del problema. Consultar al profesor de prácticas acerca de la complejidad del problema propuesto.

Necesitamos conocer las imágenes que hay en una página web determinada, en nuestro caso las imágenes del diario deportivo as.com, debido a que necesitamos imágenes deportivas para realizar un nuevo diario.
Para ello, nosotros tenemos el código HTML de dicha página en un fichero de texto (esta parte se ha realizado manualmente, copiando y pegando el código en el archivo de texto).
El problema es que hay mucho contenido en este código para poder diferenciar bien cuáles son las imágenes que hay.

Para ello vamos a usar Lex, de forma que analizaremos el texto y presentaremos solamente las imágenes existentes.

2. Resolver el problema propuesto usando Lex.

Para resolver este problemas hemos realizado un programa haciendo uso de Lex. Como vemos a continuación:

```
/* -------- Sección de Declaraciones ----------- */
%{
#include <stdio.h>
#include <string.h>


int num_ima_jpg;
int num_ima_png;
int hide;

void escribir_datos(int dato1, int dato2);
void hide_image();

%}

/* -------- Sección de Reglas ----------------- */
%%


.                       { hide_image();}
(img src.)              { hide = 1; hide_image();}
(data-src=.)            { hide = 1; hide_image();}
(jpg.)                  { num_ima_jpg++; hide_image(); hide = 0;}
(png.)                  { num_ima_png++; hide_image(); hide = 0;}


%%
```


ejercicio.l (~/UGR/MC/P3) - gedit

```
%%
/* -------- Sección 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{
        printf("Debes pasar un archivo de texto que se pueda analizar.\n");
        exit (-1);
}

num_ima_jpg = num_ima_png = 0;
pos = 0;
hide = 0;
yylex ();
escribir_datos(num_ima_jpg,num_ima_png);
return 0;
}

void escribir_datos (int dato1, int dato2) {
        printf ("Imágenes jpg=%d\nImágenes png=%d\n", dato1,dato2);
}

void hide_image(){
        if(hide == 0){
                yytext = "";
        }
        else{
                printf("%s", yytext);
        }

}
```


3. Realizar un documento presentando el problema y la solución con Lex.

Para resolver la práctica, hemos declarado una serie de reglas, las cuales son:

- (img src.)   {hide = 1; hide_image();}

  Con esta regla lo que hacemos es buscar en el texto aquellas cadenas que sean iguales a "img src".

- (data-src=.)   {hide = 1; hide_image();}

  En esta regla buscaremos la cadena data-src= seguida de cualquier otro carácter.

- (jpg.)   {num_ima_jpg++; hide_image(); hide = 0;}

  Buscaremos cadenas que contengan jpg. Ya que será el formato que nos ayudará a identificar la imagen.

- (png.)   {num_ima_png++; hide_image(); hide = 0;}

  Buscaremos las cadenas que contengan la cadena png, ya que nos ayudará a identificar las imágenes, debido a que es un formato de ellas.

Además contamos el número de imágenes de cada tipo que hay, para ello utilizamos las variables num_ima_jpg y num_ima_png, las cuales se van sumando cada vez que encontramos una cadena con la extensión .jpg o .png.

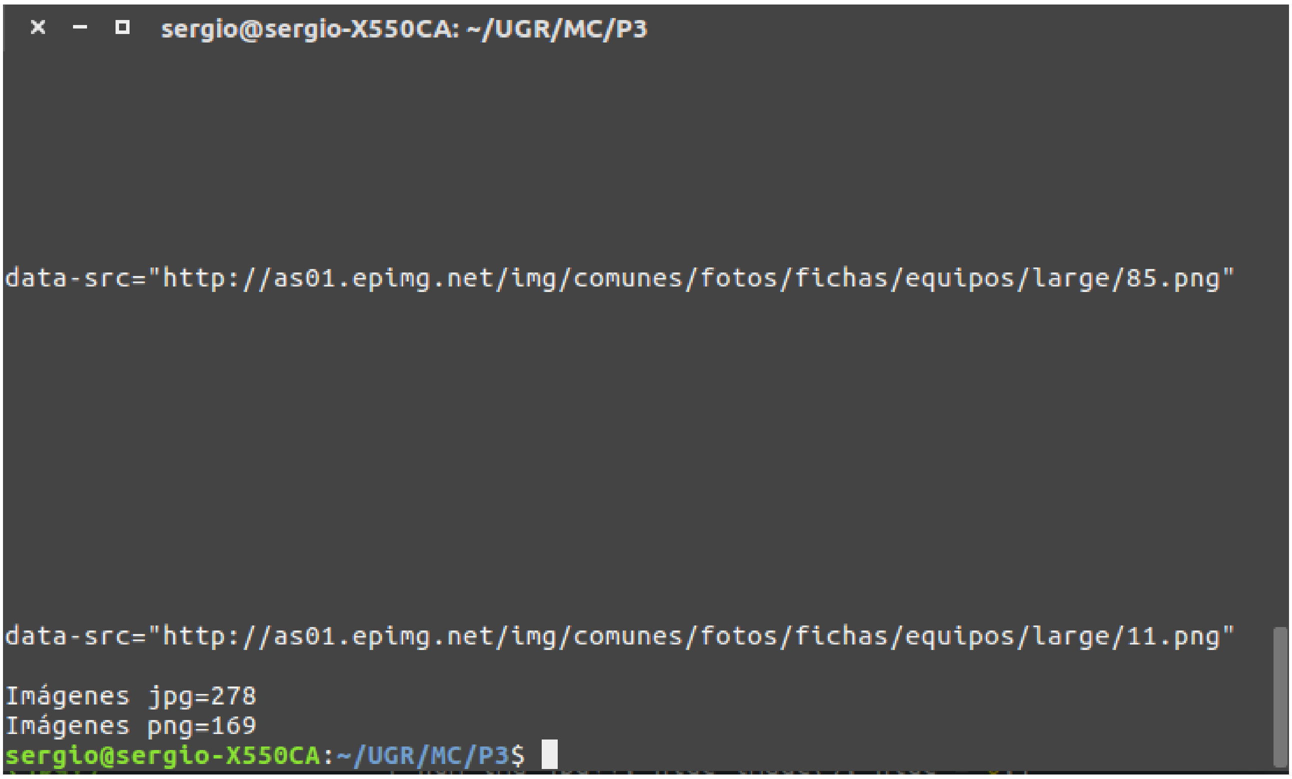
Por último creamos dos métodos, uno que se usará al terminar de recorrer el texto que nos mostrará el número total de imágenes de cada tipo leídas, que será el método escribir_datos.

Y el otro método que usaremos será hide_image, el cual con ayuda de la variable “hide” nos servirá para escribir por pantalla solamente las imágenes, y que no nos moleste el resto de código.

Para ello lo que hacemos es, si encontramos una de las reglas, ya sea data-src o img src, cambiaremos el valor de hide, el cual nos indicará que ahí comienza una imagen y que terminará cuando encuentre la cadena “jpg” o “png” donde volveremos a cambiar el valor de hide.

Al cambiar el valor de hide lo que hacemos es, permitir que se imprima por pantalla yytext, o cambiar el valor de yytext para que por pantalla no aparezca nada.

Un ejemplo de funcionamiento del programa es el siguiente:



Como vemos en la imagen, ha reconocido en el texto 278 imágenes de formato jpg y 169 imágenes de formato png, y sólo nos muestra los enlaces de dichas imágenes, eliminando el resto de texto.

En el siguiente enlace se deja el código del archivo lex, así como el archivo de texto con código fuente de la página web mencionada.

https://drive.google.com/file/d/0B2SznM7I-gDYdlhXZEJxb3NhQk0/view?usp=sharing

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 -> 01 S | 010 S | 101 S | ε

b) S -> 0 S 1 | S 1 | 0 S | 0

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

Nota: Explicar/Demostrar cuidadosamente si la gramática no es ambigua (con lenguaje natural).

a)
La gramática que se nos presenta es una gramática ambigua, ya que para generar la palabra "010101" puede seguir distintos árboles de derivación.

```
      S                      S
      |                      |
     01S                    010S
      |                      |
    0101S                  010101S
      |                      |
   010101S                 010101
      |
    010101
```

Ahora debemos buscar una gramática que genere el mismo lenguaje que genera dicha gramática, pero que no sea ambigua, y así demostramos que el lenguaje no es inherentemente ambiguo.

b) El lenguaje generado es el siguiente.

$$L = \{0^i 1^j \mid i \in \mathbb{N}-\{0\} \ \ y \ \ j \in \mathbb{N}\}$$

Pero la gramática que hay es una gramática ambigua, ya que para generar la palabra "O" podemos seguir varios caminos

```
S        S
|        |
0        0S
         |
         S → ε
```

|
0

Para ver que el lenguaje no es inherentemente ambiguo, vamos a buscar una gramática que genere dicho lenguaje pero que no sea ambigua.
Una posible solución es:

S → 0 | AB

A→ 0 | 0A

B → 1 | B1

c) La gramática que se nos presenta no es una gramática ambigua ya que sólo existe un árbol de derivación para cada palabra que se genere, por lo tanto el lenguaje no es inherentemente ambiguo.

Esta gramática no es ambigua debido a que para las variables S y A sólo existe una única producción, por lo que ellas siempre generarán dicha producción, y sólo podemos variar con la variable B, la cual se llama a si misma recursivamente, y no da pie a volver hacia atrás y así poder crear ambigüedad.

2. Eliminar símbolos y producciones inútiles. Realizar el procedimiento paso por paso, indicando las variables descartadas y el motivo.

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;

Primero vamos a añadir las Producciones cuya parte derecha sean sólo símbolos terminales en el conjunto Vt, una vez hecho esto, vamos viendo cada producción, de forma que si en su parte derecha aparece alguna variable ya presente en el conjunto Vt, dicha producción también será útil, por lo que también se meterá.

Vt = {B, D, C, O}

Una vez metidas las producciones que generan símbolos terminales, vamos buscando la condición recursiva, dicha anteriormente.

Primero se añade A ya que tiene una producción A→ jBI, la cual tiene a B que está presente en Vt

Vt = {B, D, C, O,A}

Segundo meteríamos E, debido a que tiene una producción E→clO, y O está en Vt, por lo que metemos E y eliminamos todas sus producciones.

Vt = {B, D, C, O ,A, E}

Después añadimos J, debido a su producción J → kC

Vt = {B, D, C, O, A, E, J}

Volvemos a darle otra vuelta a la gramática, ya que hemos añadido variables en esta vuelta anterior.

Como vemos, ahora podemos añadir a Vt la variable S, ya que tiene en una producción a la variable A que ya está metida.

Vt = {B, D, C, O,A, E, J, S}

Por último damos otra vuelta y ya no podemos añadir más.
Por lo tanto, vemos que tenemos dos variables inútiles, que son I y P, por lo que se pueden eliminar todas las producciones que tengan alguna de estas variables.

Así, la gramática final nos quedaría:

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

Finalmente, vamos a eliminar las producciones que no sean accesibles desde el símbolo inicial S.

Por lo que sólo se nos quedarían las producciones:

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

Y la única derivación posible sería la generación de la palabra “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

Primero vamos a eliminar las producciones nulas, para ello para cada producción que presente una variable la cual tenga una producción nula, la eliminaremos.

Las variables con producciones nulas son X, Y, y por ello también lo son Z y S

Eliminamos las producciones X → ε Y → ε y añadimos las siguientes

Primero las que salen de eliminar X
S→ YZ S → Yz X→ xx

Después las que salen de eliminar Y

S→ XZ S → Xz Y → yy

Por último las de eliminar Z

S→ XY Z → yx

Una vez hecho esto, a las nuevas producciones hay que seguir eliminando en ellas las variables que sean anulables, por lo tanto, en total generamos la siguiente gramática.

S → XYZ	S → XYz	X → xxX	Y→ yyY	Z → yxZ	Z → X
S→ YZ	S → Yz	X→ xx			
S→ XZ	S → Xz	Y → yy			
S→ XY	Z → yx				
S→ Z	S → Y	S → z	S→ X		

Ahora debemos eliminar las producciones unitarias, por lo que añadiremos a H todas aquellas producciones unitarias.

H = {(S,Z),(S,Y),(Z,X),(S,X)};

Hecho esto, debemos añadir las producciones de cada pareja (A,B), para cada producción que tenga A, añadirla sustituyendo A por B

S → yxZ	S → yyY	X→ yxZ	S→ yx	S → yy	X → yx
X → xyz	X → X Y z	X → Y Z	X→ Y z	X → z	X → X Z
X → Xz	X → XY				

Por lo que la gramática resultante será:

S → XYZ	S → XYz	X → xxX	Y→ yyY	Z → yxZ
S→ YZ	S → Yz	X→ xx		
S→ XZ	S → Xz	Y → yy		
S→ XY	Z → yx			

S → z

S → yxZ	S → yyY	X→ yxZ	S→ yx	S → yy	X → yx
X → xyz	X → X Y z	X → Y Z	X→ Y z	X → z	X → X Z
X → Xz	X → XY				

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

S → a | CD | CS   A → a | b | SS   C → a   D → AS

Partimos de que la gramática ya está en forma normal de Chomsky.

Primero vamos a renombrar variables:

S = A1   C = A2   D = A3   A = A4

A1 → a	A1 → A2 A3	A1 → A2 A1	A4 → a
A4 → b	A4 → A1 A1	A2 → a	A3→ A2 A1

Ahora debemos aplicar Elimina1(o sustitución) a aquellas producciones del tipo, Ai → Aj donde j>i

Por lo tanto se le aplica a las producciones A4 → A1 A1   A3 → A2 A1

A4 → A1 A1

A4 → a A1

A4 → A2 A3   A4 → a A3

A4 → A2 A1   A4 → a A1

A3 → A2 A1

A3 → a A1

Por lo que nos quedaría la siguiente gramática:

$A1 \rightarrow a$	$A1 \rightarrow A2\ A3$	$A1 \rightarrow A2\ A1$	$A4 \rightarrow a$	$A4 \rightarrow b$
$A4 \rightarrow a\ A1$	$A4 \rightarrow a\ A3$	$A2 \rightarrow a$	$A3 \rightarrow a\ A1$	

Volviendo a los nombres originales de las variables se quedaría tal que así:

$S \rightarrow a$	$S \rightarrow C\ D$	$S \rightarrow C\ S$	$A \rightarrow a$	$A \rightarrow b$
$A \rightarrow a\ S$	$A \rightarrow a\ D$	$C \rightarrow a$	$D \rightarrow a\ S$	