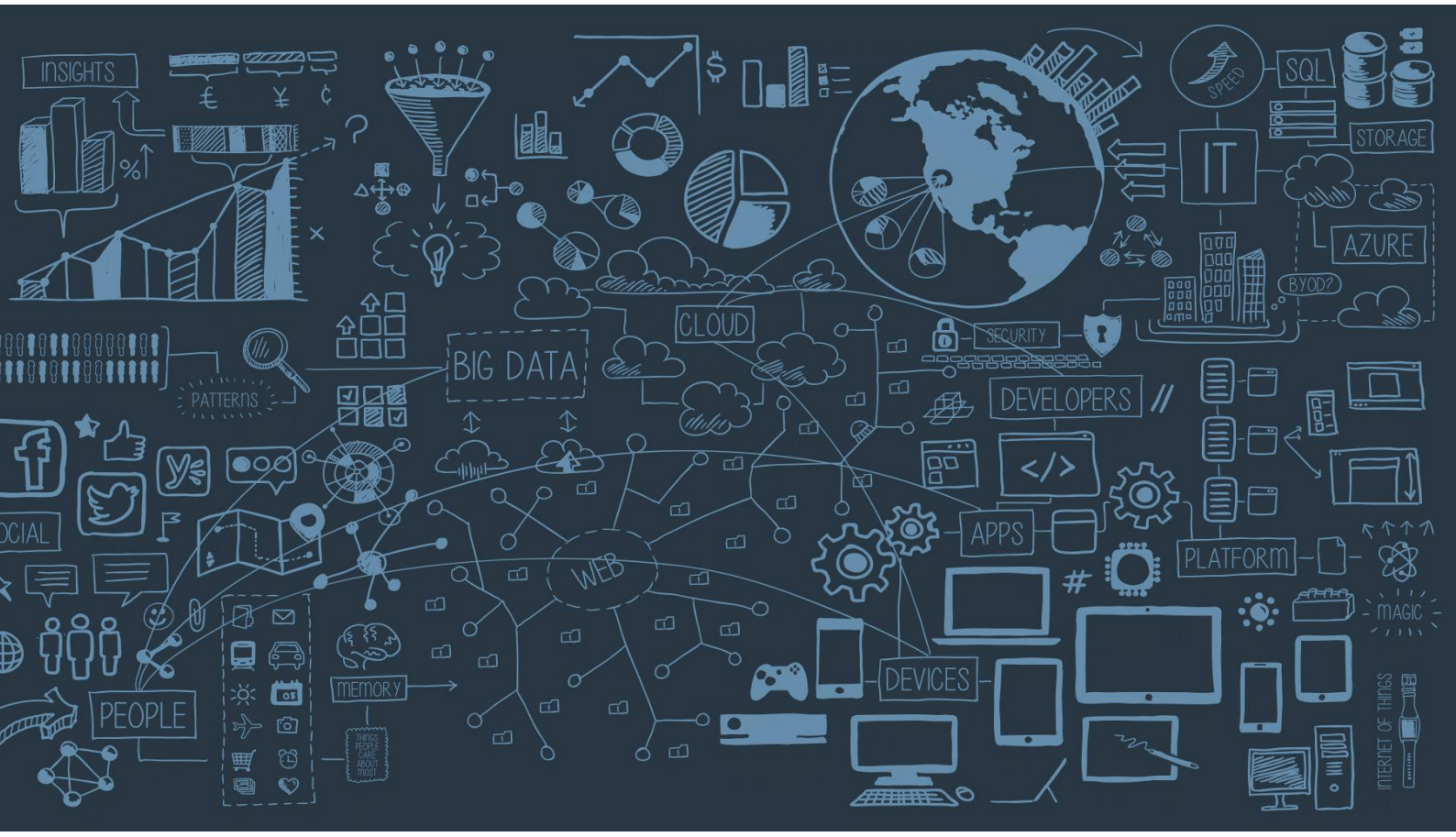
Prácticas de
Modelos de Computación.

Curso 2016/2017



Francisco Javier Caracuel Beltrán

Índice

Práctica 1. Ejercicios Prácticos sobre Lenguajes y Gramáticas.

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

a. **$S \rightarrow 0\ S_1\ 1$** (a) **$S_1 \rightarrow 0\ S_1$** (b) **$\mid 1\ S_1$** (c) **$\mid \varepsilon$** (d)

$S \rightarrow_{(a)} 0\ S_1\ 1 \rightarrow_{(b)} 00\ S_1\ 1 \rightarrow_{(b)} 000\ S_1\ 1 \rightarrow \dots$

$\rightarrow_{(c)} 001\ S_1\ 1 \rightarrow_{(c)} 0011\ S_1\ 1$

$\rightarrow_{(c)} 001\ S_1\ 1 \rightarrow_{(b)} 0010\ S_1\ 1$

$\rightarrow_{(c)} 001\ S_1\ 1 \rightarrow_{(b)} 0010\ S_1\ 1$

$L = \{ 0u1 \mid u \in \{0,1\}^* \}$

Siempre empieza con 0 y termina con 1. Se pueden mezclar tantos 0 y 1 entre ellos como sea necesario.

b. **$S \rightarrow S_1\ 101\ S_1$** **$S_1 \rightarrow 0\ S_1 \mid 1\ S_1 \mid \varepsilon$**

$L = \{ u101v \mid u, v \in \{0,1\}^* \}$

c. **$S \rightarrow 0\ S\ 1$** (a) **$\mid S_1$** (b) **$S_1 \rightarrow 1\ S_1\ 0$** (c) **$\mid 1\ S_2\ 0$** (d) **$S_2 \rightarrow 0\ S_2\ 1$** (e) **$\mid \varepsilon$** (f)

$S \rightarrow_{(a)} 0\ S\ 1 \rightarrow_{(a)} 00\ S\ 11 \rightarrow_{(a)^*} 00..0\ S\ 1..1$

$\rightarrow_{(b)} 0^i\ S\ 1^i \rightarrow_{(c)} 0^i\ 1S_1\ 01^i \rightarrow_{(c)} 0^i1^j\ S_1\ 0^j1^i$

$\rightarrow_{(d)} 0^i1^j\ 1\ S_2\ 00^j1^i \rightarrow_{(e)} 0^i1^j\ 0^k\ S_2\ 1^k0^j1^i$

$\rightarrow_{(f)} 0^i1^j\ 0^j1^i$

$S \rightarrow_{(b)} S_1 \rightarrow_{(c)^i} 1^i\ S_1\ 0^i \rightarrow_{(e)} 1^i0^j\ S_1\ 1^j0^i$

$\rightarrow_{(d)} 1\ S_2\ 0 \rightarrow_{(e)} 10^j\ S_2\ 1^j0$

$L = \{ 0^i1^j0^k1^k0^j1^i \mid i,j,k \in \mathbb{N},\ j>0 \}$

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".

- Tipo 2:
  $S \rightarrow 10\ S_1\ 001 \mid 1001 \qquad S_1 \rightarrow 1\ S_1 \mid 0\ S_1 \mid \varepsilon$

- Tipo 3:
  $S \rightarrow 10\ S_1 \mid 1001 \qquad S_1 \rightarrow 1\ S_1 \mid 0\ S_1 \mid 001$

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

- Tipo 2:
  $S \rightarrow S_1\ 0\ S_2 \quad S_1 \rightarrow S_3\ 0\ S_3 \quad S_2 \rightarrow S_3\ 0\ S_3 \mid S_3 \mid \varepsilon \quad S_3 \rightarrow 1\ S_3 \mid \varepsilon$

- Tipo 3:
  $S \rightarrow 0\ S_1 \mid 1\ S_1 \mid 1\ S \quad S_1 \rightarrow 1\ S_1 \mid 0\ S_2 \quad S_2 \rightarrow 1\ S_2 \mid 0\ S_3 \mid 0 \quad S_3 \rightarrow 1\ S_3 \mid 1$

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

- Tipo 2:
  $S \rightarrow 1S \mid 0\ S_1\ S_2 \mid \varepsilon \quad S_1 \rightarrow 0\ S_1 \mid 1\ S_2 \mid \varepsilon \quad S_2 \rightarrow 0\ S_1 \mid \varepsilon$

- Tipo 3:
  $S \rightarrow 1\ S \mid 0\ S_1 \mid \varepsilon \quad S_1 \rightarrow 0\ S_1 \mid 1\ S_2 \mid 1 \mid \varepsilon \quad S_2 \rightarrow 0\ S_1$

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 éstas, 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 ésta, 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?

$S \rightarrow pS \mid gS \mid tS_3 \mid gxS_2$

$S_2 \rightarrow pS_2 \mid gS_2 \mid tS_4$

$S_3 \rightarrow pS_3 \mid tS_3 \mid gS_3 \mid gxS_4$

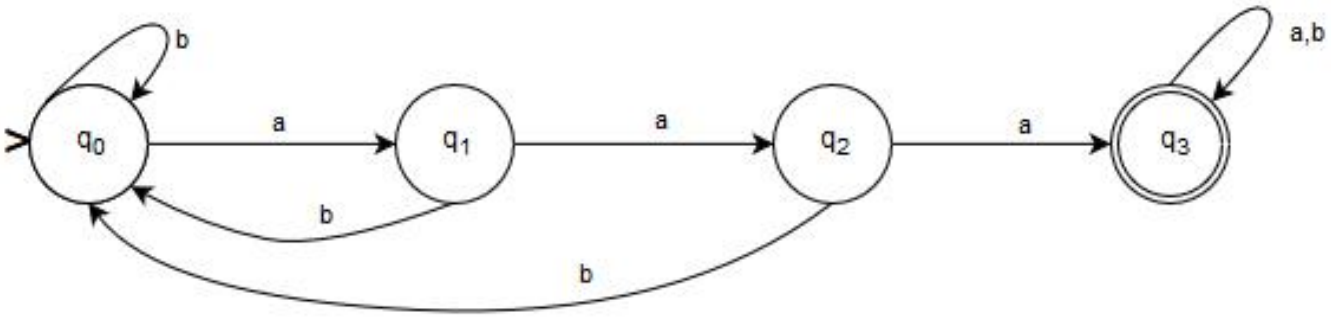
$S_4 \rightarrow pS_4 \mid gS_4 \mid tS_4 \mid j$

Gramática de tipo 3

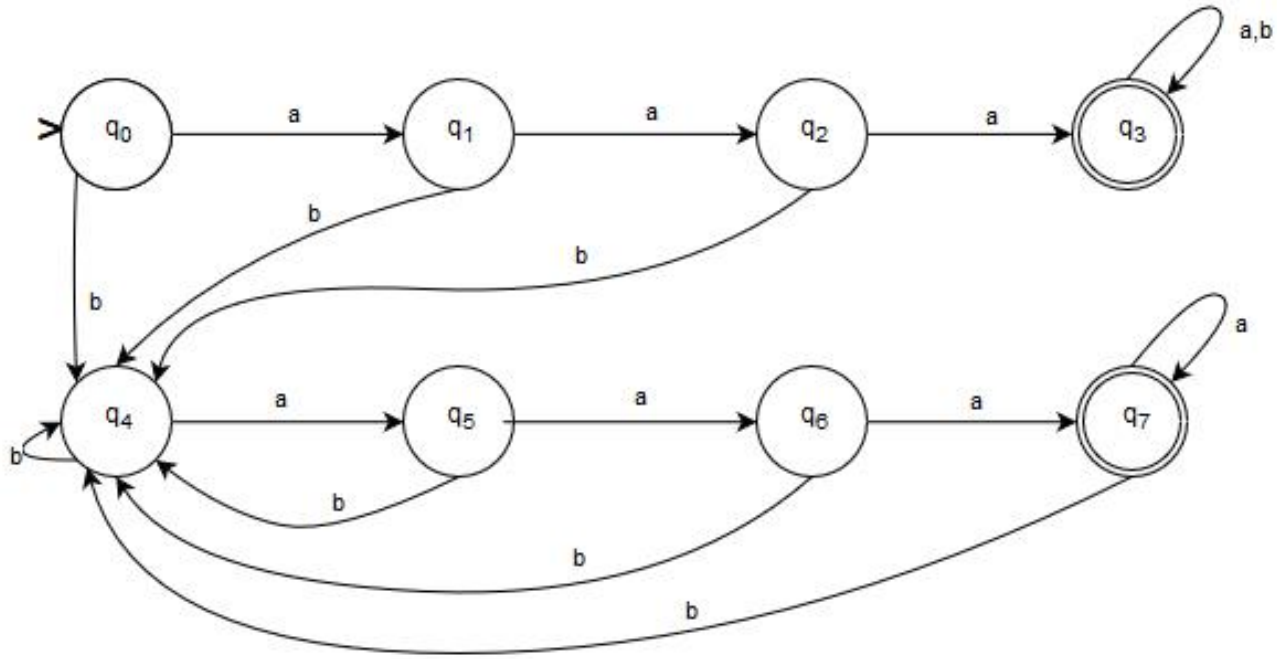
Práctica 2. Ejercicios Prácticos sobre Lenguajes y Gramáticas.

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.

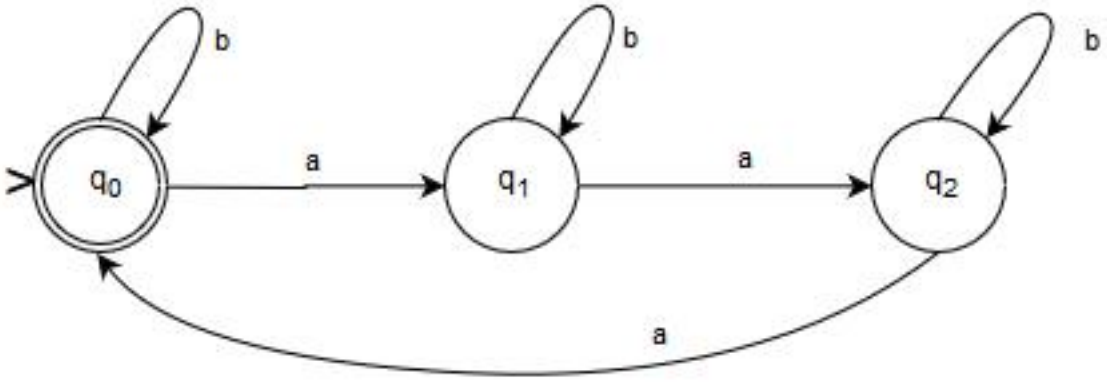


b. El lenguaje de las palabras que empiezan o terminan (o ambas cosas) en aaa.

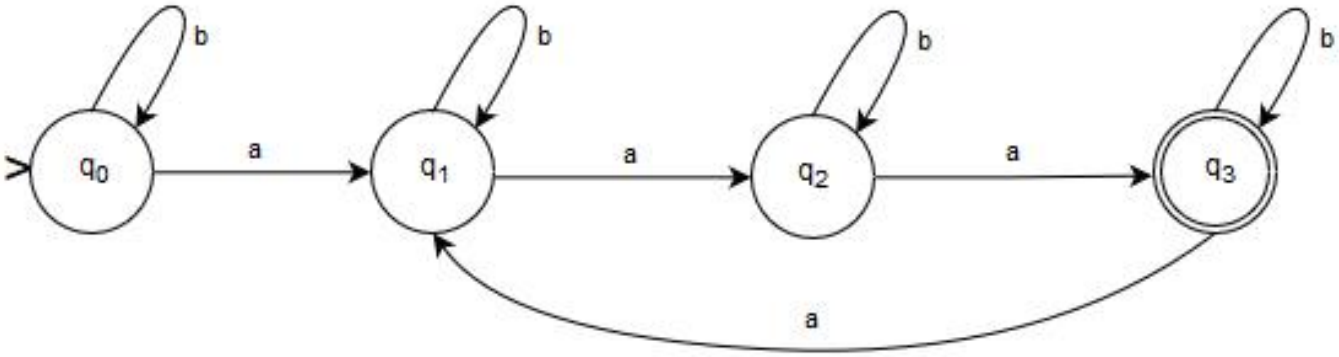


c. El lenguaje formado por las cadenas donde el número de a’s es divisible por 3.

- El número 0 es divisible por 3:

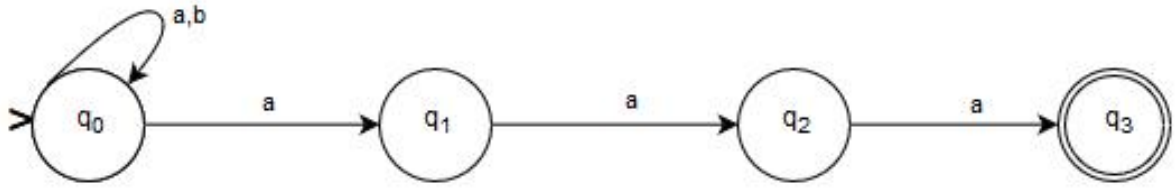


- El número 0 no es divisible por 3:

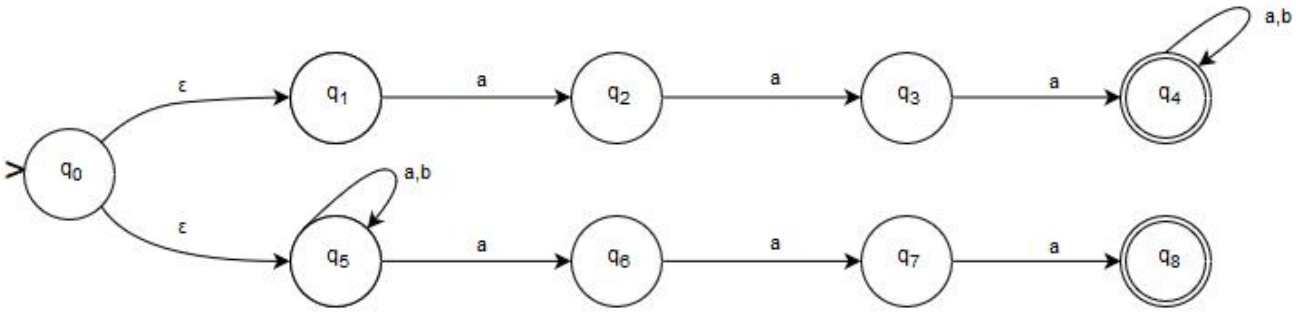


2. Construir un AFND que acepte cada uno de los siguientes lenguajes con alfabeto {a,b}:

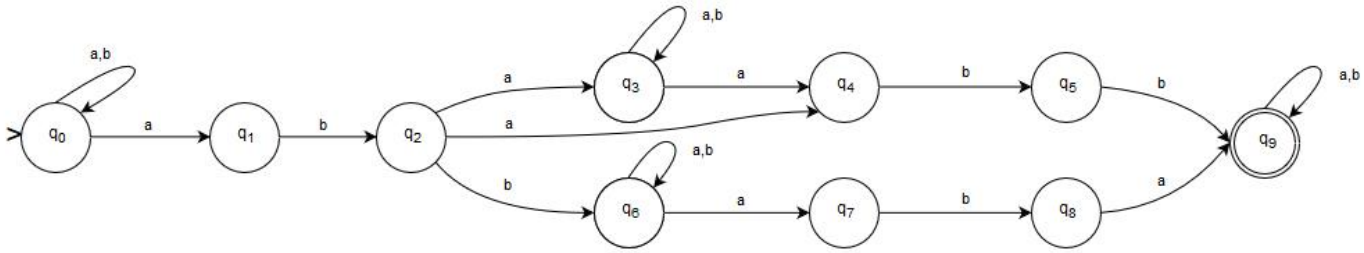
a. El lenguaje de las palabras que terminan en aaa.



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

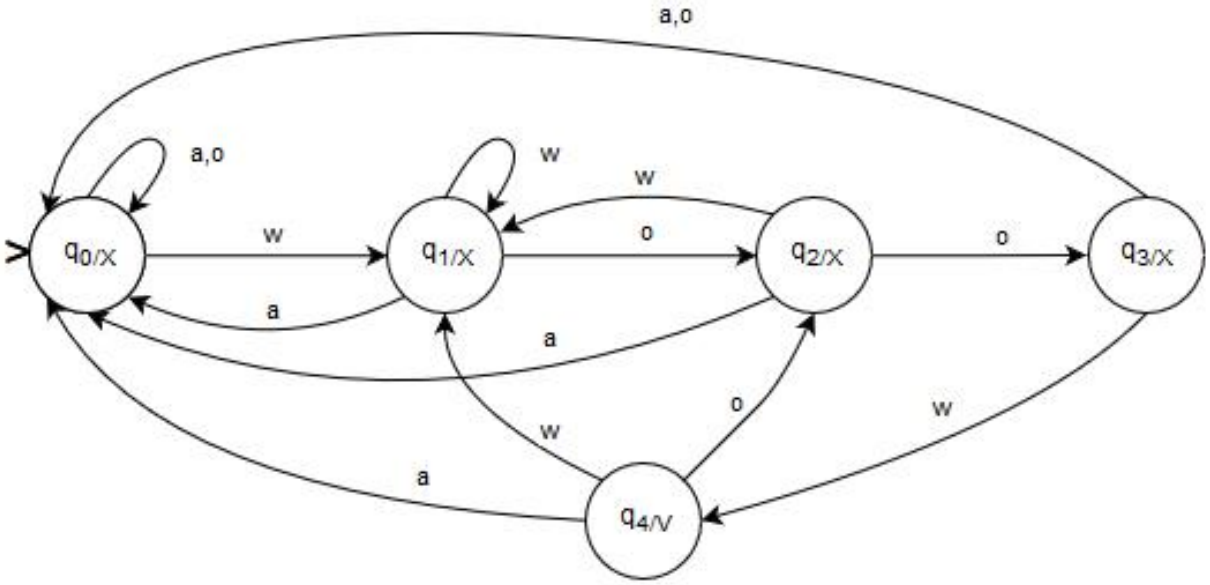


c. El lenguaje de las palabras que contengan, simultáneamente, las subcadenas aba y abb. Este AFND también acepta cadenas en la que estas subcadenas están solapadas (por ejemplo, las palabras “ababb” y “aaabbbaba” serían aceptadas).

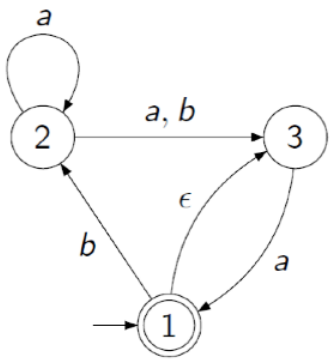


3. Diseñar una Máquina de Mealy o de Moore que, dada una cadena usando el alfabeto A={'a', 'w', 'o'}, encienda un led verde (salida 'V') cada vez que se detecte la cadena "woow" en la entrada, apagándolo cuando lea cualquier otro símbolo después de esta cadena (representamos el led apagado con la salida"X").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

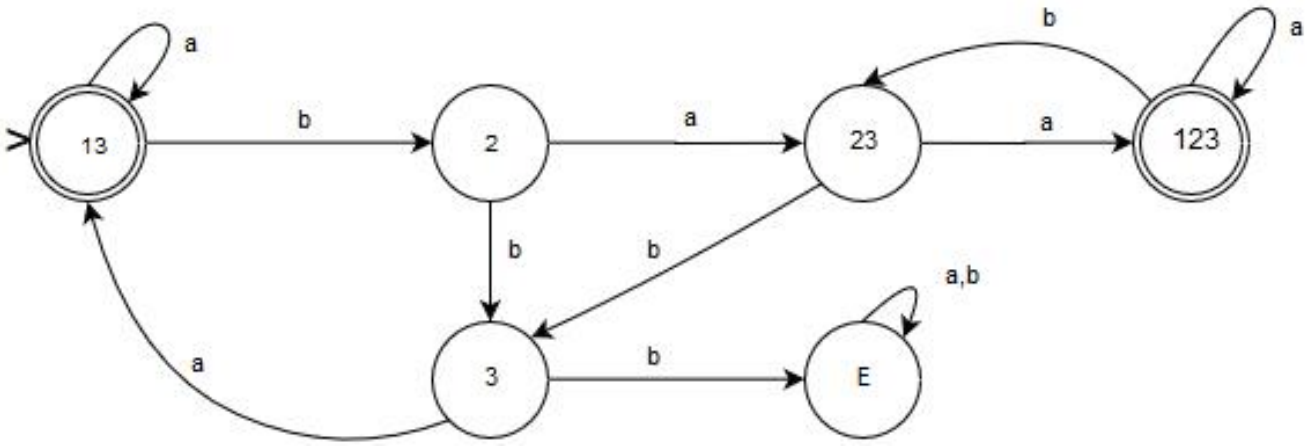
entrada	aaawoawoowoowwoowa
salida	XXXXXXXXXVXXVXXXVX



4. Obtener un AFD equivalente al AFND siguiente:



E/A	a	b
<u>13</u>	13	2
2	23	3
23	123	3
3	13	∅
<u>123</u>	123	23



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

1. Se propone hacer un programa que sea capaz de leer la web blablacar.es y pueda ofrecer un listado con toda la información de los 10 próximos viajes que están por salir. Además, se recomendará un viaje en base al mejor precio, mayor puntuación y mayores opiniones.

2. Requisitos:
   a. Curl: https://curl.haxx.se/libcurl/c/libcurl-tutorial.html
   b. Comandos terminal:
      i. lex p3.l
      ii. gcc lex.yy.c -o p3 -ll -L/usr/lib/x86_64-linux-gnu –lcurl
      iii. ./p3 ciudadOrigen ciudadDestino

3. Solución:
   a. Código fuente:

```
/*
//////////////////////////////////////////////////////////////////////////////
//
// Francisco Javier Caracuel Beltrán
//
// Práctica 3 - Modelos de Computación
//
// Grado en Ingeniería Informática - Curso 2016/2017
//
//////////////////////////////////////////////////////////////////////////////
/*

/*
//////////////////////////////////////////////////////////////////////////////
// Sección de Declaraciones
//
*/

%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>

#define nTrip 10
```

```
int iPrice;
int iName;
int iDate;
int iAge;
int iScore;
int iRatings;

typedef int (*compfn)(const void*, const void*);

struct MemoryStruct {
  char *memory;
  size_t size;
};

struct Trip{
    char name[50];
    double price;
    double score;
    int ratings;
    char date[50];
    int age;
};

struct Trip trip[nTrip];

static size_t WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp);

void setPrice();
void setName();
void setDate();
void setAge();
void setScore();
void setRatings();

int compare (struct Trip *a, struct Trip *b);
void showData();

%}
```

```
all        .*
enter      \n{all}
digit      [0-9]
price1     {digit}{1,2}
price2     {digit}{2}
price3     "<span class=\"size20\">,"
price      ({price1}{price3}{price2})
name1       "<h2 class=\"ProfileCard-info ProfileCard-info--name u-truncate\">"
name2      "</h2>"
name       ({name1}{enter}{2}{name2})
date1      "<h3 class=\"time light-gray\" itemprop=\"startDate\" content=\""[0-9]{4}-[0-9]{2}-[0-
9]{2}"\">"
date2      "</h3>"
date       ({date1}{enter}{2}{date2})
age1       "<div class=\"ProfileCard-info\">"
age2       " años<br />"
age        ({age1}{enter}{age2})
score1     "<span class=\"u-textBold u-darkGray\">"
score2     "/"{digit}{1}
score      ({score1}{all}{score2})
ratings1    "<span class=\"u-gray\"> - "
ratings     ({ratings1}{digit}+)

/*
//
////////////////////////////////////////////////////////////////////////////
*/

%%

{price}     {setPrice();}
{name}      {setName();}
{date}      {setDate();}
{age}       {setAge();}
{score}     {setScore();}
{ratings}   {setRatings();}
\n          {}
.           {}

%%
```

```
int main (int argc, char *argv[]) {

   if(argc != 3){
      printf("\nDebes introducir la ciudad de salida y de destino.\nUso: "
            "%s salida destino\n\n", &argv[0][0]);
      return -1;
   }

   char start[50], end[50];

   memcpy(start, &argv[1][0], 50);
   start[50] = '\0';

   memcpy(end, &argv[2][0], 50);
   end[50] = '\0';

   // Struct donde se guardará el código fuente de la página
   struct MemoryStruct chunk;

   // Reserva de memoria e inicialización de la longitud del código fuente
   chunk.memory = malloc(1);
   chunk.size = 0;

   // A través de CURL se obtiene el código fuente de blablacar
   CURL *curl;
   CURLcode res;

   curl = curl_easy_init();

   // Si no ha habido ningún error al iniciar curl
   if (curl) {

      char url[500] = "https://www.blablacar.es/coche-compartido/";

      strcat(url, &start[0]);
      strcat(url, "/");
      strcat(url, &end[0]);
      strcat(url, "/");

      // Se envía la url de la que se quiere obtener respuesta
      curl_easy_setopt(curl, CURLOPT_URL, &url[0]);

      // Se permite la redirección que tenga la url
      curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);

      // Función que se encargará de guardar el código fuente
      curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
```

```
        // Se le envía la variable donde se va a escribir el código fuente
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&chunk);

        // Para que el servidor de la url detecte un agente de navegador
        curl_easy_setopt(curl, CURLOPT_USERAGENT, "libcurl-agent/1.0");

        // Se realiza la petición
        res = curl_easy_perform(curl);

        // Se detecta si ha habido errores
        if (res != CURLE_OK){
           fprintf(stderr, "\nError al conectar con el servidor de blablacar.com.\n\nError: %s\n\n",
              curl_easy_strerror(res));
           return -2;
        }

        // Se limpia el objeto curl
        curl_easy_cleanup(curl);

        // Se le indica a lex el texto que debe escanear
        yy_scan_string(chunk.memory);

        // Se libera la memoria del objeto que tiene el código fuente
        free(chunk.memory);

        // Se vacía completamente el objeto curl
        curl_global_cleanup();

    }

    // Se inicializan las variables que contarán
    iPrice = 0;
    iName = 0;
    iDate = 0;
    iAge = 0;
    iScore = 0;

    // Se inicia el escaneo del código fuente
    yylex();

    // Se muestran los resultados obtendidos
    showData();

    return 0;

}
```

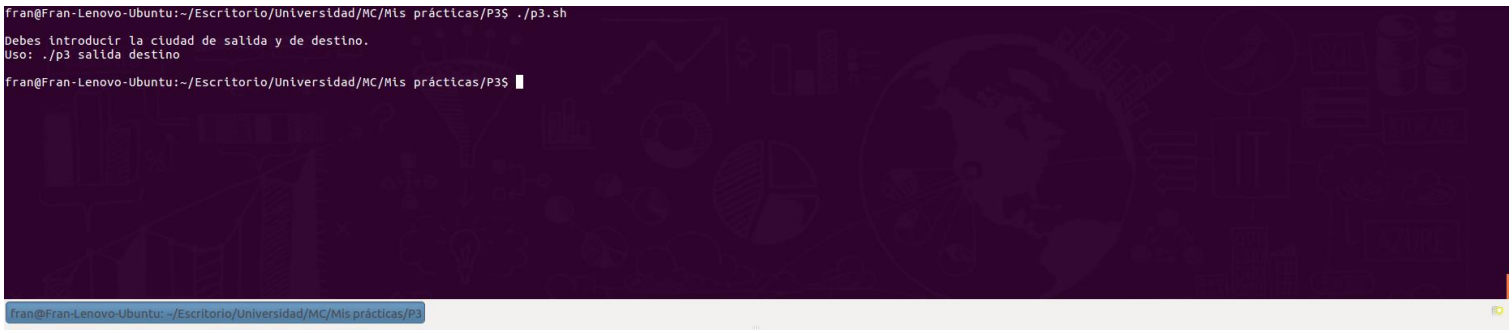
```
static size_t WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp){

   size_t realsize = size * nmemb;
   struct MemoryStruct *mem = (struct MemoryStruct *) userp;

   mem->memory = realloc(mem->memory, mem->size + realsize + 1);

   if (mem->memory == NULL) {
      /* out of memory! */
      printf("not enough memory (realloc returned NULL)\n");
      return 0;
   }

   memcpy(&(mem->memory[mem->size]), contents, realsize);
   mem->size += realsize;
   mem->memory[mem->size] = 0;

   return realsize;

}

void setPrice(){

   trip[iPrice].price = atof(&yytext[0]) + atof(&yytext[yyleng-2])/100;

   iPrice++;

}

void setName(){

   // Desde el principio hasta que empieza el nombre hay 92 caracteres.
   // El total de caracteres estáticos son 122.
   // La cadena ocupara el tamaño de yytext menos 122.
   char n[50];
   memcpy(n, &yytext[92], yyleng-122);
   n[yyleng-122] = '\0';

   strncpy(trip[iName].name, n, sizeof trip[iName].name - 1);

   iName++;

}
```

```
void setDate(){

   // Desde el principio hasta que empieza el nombre hay 91 caracteres.
   // El total de caracteres estáticos son 113.
   // La cadena ocupara el tamaño de yytext menos 113.
   char d[50];
   memcpy(d, &yytext[91], yyleng-113);
   d[yyleng-113] = '\0';

   strncpy(trip[iDate].date, d, sizeof trip[iDate].date - 1);

   iDate++;

}

void setAge(){

   trip[iAge].age = atoi(&yytext[87]);

   iAge++;

}

void setScore(){

   // Se averigua si la cadena tiene "," o no
   char *c;
   int index;

   c = strchr(&yytext[37], ',');
   index = (int)(c-&yytext[37]);

   // Valor de la puntuación
   double s = atof(&yytext[36]);

   // Si tenía "," se suma el decimal
   if(index == 0){
      s += atof(&yytext[38])*0.1;
   }

   trip[iScore].score = s;

   iScore++;

}
```

```
void setRatings(){

   trip[iRatings].ratings = atoi(&yytext[24]);

   iRatings++;

}

int compare (struct Trip *trip1, struct Trip *trip2){

   int result = 0;

   // Se ordena por el precio
   if(trip1->price < trip2->price)
      result = -1;
   else if(trip1->price > trip2->price)
      result = 1;
   else{

      // En caso de tener el mismo precio se ordena por puntuación
      if(trip1->score > trip2->score)
         result = -1;
      else if(trip1->score < trip2->score)
         result = 1;
      else{

         // En caso de tener la misma puntuación se ordena por el número
         // de opiniones
         if(trip1->ratings > trip2->ratings)
            result = -1;
         else if(trip1->ratings < trip2->ratings)
            result = 1;

      }

   }

   return result;

}
```

```
        void showData(){

            printf("\nResultados (por orden de salida):\n");

            for(int i=0; i<nTrip; i++){
                printf ("Viaje %d: Nombre-> %s, Precio-> %.2f€, Fecha-> %s, Edad: %d años, Puntuación: %.1f,
        Opiniones: %d\n",
                        i+1, trip[i].name, trip[i].price, trip[i].date, trip[i].age, trip[i].score, trip[i].ratings);
            }

            printf("\n\n");

            qsort((void *)&trip, nTrip, sizeof(struct Trip), (compfn)compare);

            printf("Te recomiendo que cojas el blablacar:\nNombre-> %s, Precio-> %.2f€, Fecha-> %s, Edad:
        %d años, Puntuación: %.1f, Opiniones: %d\n",
                        trip[0].name, trip[0].price, trip[0].date, trip[0].age, trip[0].score, trip[0].ratings);

            printf("\n\n");

}
```

b. Capturas:



c. Referencias:
   i. Librería curl para C: https://curl.haxx.se/libcurl/c/libcurl-tutorial.html
   ii. Ejemplo básico para empezar a trabajar: https://curl.haxx.se/libcurl/c/simple.html
   iii. Otro ejemplo: https://curl.haxx.se/libcurl/c/getinmemory.html
   iv. Analizar string directamente: http://stackoverflow.com/questions/780676/string-input-to-flex-lexer

Práctica 4.

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 (a) | 010 S (b) | 101 S (c) | ε (d)

S → (a) 01S → (a) 0101S → (a) 010101S → (a) 01010101S
→ (b) 01010S → (a) 0101001S
→ (b) 01010010S
→ (c) 01010101S

* Gramática ambigua

- Cuatro veces la primera instrucción y ε para finalizar.
- Primera instrucción, segunda instrucción, tercera instrucción y ε para finalizar.

Un lenguaje no puede ser ambiguo si es regular, por lo que este lenguaje no es inherentemente ambiguo.

Esta gramática se pasa a autómata no determinista, se convierte a autómata determinista y se consigue una gramática no ambigua, que hace que el lenguaje no sea inherentemente ambiguo.

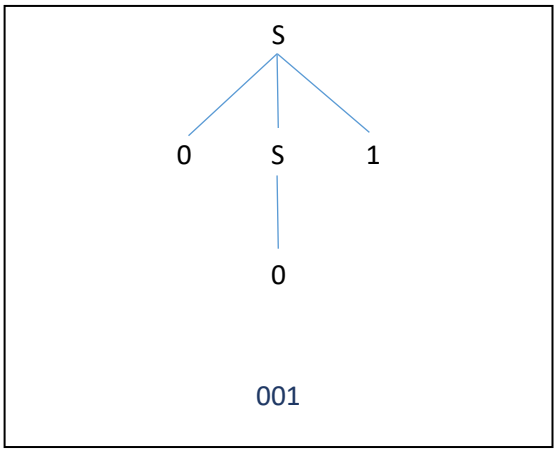
E/A	0	1
>q_0	q_1	q_4
q_1	Ø	q_0q_2
q_4	q_5	Ø
q_0q_2	$q_0q_1q_3$	q_4
q_5	Ø	q_0q_6
$q_0q_1q_3$	q_1	$q_0q_2q_4$
q_0q_6	q_1	q_4
$q_0q_2q_4$	$q_0q_1q_3q_5$	q_4
$q_0q_1q_3q_5$	q_1	$q_0q_2q_4q_6$
$q_0q_2q_4q_6$	$q_0q_1q_3q_5$	q_4

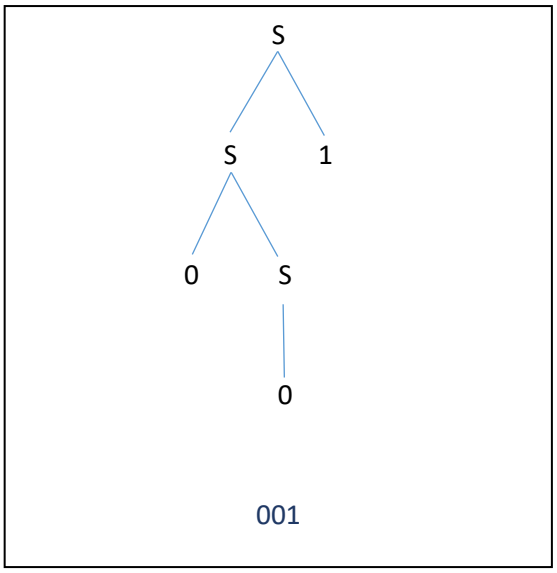
Gramática generada no ambigua:

$S \rightarrow 0q_1 \mid 1q_4$
$q_4 \rightarrow 0q_5$
$q_5 \rightarrow 1q_0q_6$
$q_0q_6 \rightarrow 0q_1 \mid 1q_4$
$q_0q_1q_3q_5 \rightarrow 0q_1 \mid 1q_0q_2q_4q_6$
$q_1 \rightarrow 1q_0q_2$
$q_0q_2 \rightarrow 0q_0q_1q_3 \mid 1q_4$
$q_0q_1q_3 \rightarrow 0q_1 \mid 1q_0q_2q_4$
$q_0q_2q_4 \rightarrow 0q_0q_1q_3q_5 \mid 1q_4$
$q_0q_2q_4q_6 \rightarrow 0q_0q_1q_3q_5 \mid 1q_4$

b) S → 0 S 1 | S 1 | 0 S | 0

L = { 0^i 1^j | i>0, j>= 0}





* Gramática ambigua.

- Gramática no ambigua:

  $S \rightarrow 0S \mid 0 \mid 0S_1$

  $S_1 \rightarrow 1S_1 \mid 1$

  El lenguaje no es ambiguo porque primero se calculan los 0 y no existe manera de calcular con dos producciones diferentes el mismo número de 0, excepto cuando se use "0S", que se soluciona porque seguro habrá, al menos, un 1. Tampoco se pueden conseguir el mismo número de 1 de manera diferente.

c) S → A 1 B A → 0 A | ε B → 0 B | 1 B | ε

La gramática no es ambigua ya que A→0A no genera ambigüedad, B→0B|1B tampoco y al tener 1 en la primera producción S→A1B, permite que sea imposible generar dos palabras iguales con diferentes producciones. El 1 hace de separación y al asegurar que existe, al menos, uno, no permite la ambigüedad. Si no estuviera ese 1, sí sería ambiguo.

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 → ola;		

Primero se buscan producciones con términos independientes solo o con variables que estén en VT. Si se introduce en VT una variable, se cancelan todas las producciones que comiencen con dicha variable.

VT = {B, D, C, O}

En la primera pasada se eliminan B→bb, B→D, D→de, C→c, O→o.

VT = {B, D, C, O, E, J}

En la segunda pasada se eliminan E→elO, E→Perl, J→Kc.

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

En la tercera pasada se eliminan A→dEs, A→Jbi.

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

En la cuarta pasada se eliminan S→moA, S→cI.

V-VT = {I, P}

Se consiguen las producciones que son necesarias. Serán todas aquellas cuya variable esté en VT y no tenga ninguna en V-VT

B→bb	
B→D	
D→de	
C→c	
O→o	
E→elO	*No se añade E→Perl porque P está en V-VT.
J→kC	
A→dEs	*No se añade A→jBI porque P está en V-VT.
S→moA	*No se añade S→cI porque P está en V-VT.

V_s = {S, A, E, O}
J = {~~A~~, ~~E~~, ~~O~~}

T_s = {m, o, d, s, e, l}

a. Se añade S a V_s.
b. Se añade m, o en T_s; A en J; A en V_s.
c. Se elimina A de J.
d. Se busca la variable A y se mete d, s en T_s y E en J y V_s.
e. Se busca la variable E y se mete en e, l en T_s, se saca E de J y se mete O. Se mete O en V_s.
f. Se busca la variable O y se mete o en T_s, se saca O de J y se añade en V_s.

Como no quedan más variables en J, ha terminado el algoritmo.

El resto de producciones no se tienen en cuenta.

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

   a. Se eliminan producciones ε y se añaden a H.
   b. En la siguiente pasada se añaden las que tengan todas las variables en H.

H = {X, Y, Z, S}

S → YZ | XZ | XY | X | Y | Z | Yz | Xz | z
X → xx
Y → yy
Z → yx

y se añaden el resto de las producciones menos Y→ ε, X→ ε, con lo que nos queda:

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

Se añaden las producciones que tengan variables unitarias a H.

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

Se añaden las producciones haciendo los intercambios del conjunto de variables que aparecen en H:

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

4. Pasar la siguiente gramática a forma normal de Greibach:
   S → a | CD | CS
   A → a | b | SS
   C → a
   D → AS

   Se recomienda que esté en F.N. de Chomsky pero ya lo está.

   Se renombran las variables de manera que se evite hacer el paso 2 y no hay que resolver la recursión por la izquierda:

   $X_3 = S$ $X_2 = A$ $X_4 = C$ $X_1 = D$

   Las producciones con el cambio de variable serían (están en F.N. de Greibach):

   $X_3 \rightarrow a \mid X_4X_1 \mid X_4X_3$
   $X_2 \rightarrow a \mid b \mid X_3X_3$
   $X_4 \rightarrow a$
   $X_1 \rightarrow X_2X_3$

   Las que ya están en F.N.G. no es necesario modificarlas. Poco a poco se va realizando sustitución con las producciones que no lo cumplen:

   - Se aplica sustitución a $X_3 \rightarrow X_4X_1$.
     Se elimina esa producción y se añade:
     $X_3 \rightarrow aX_1$.

   $X_3 \rightarrow a \mid aX_1 \mid X_4X_3$
   $X_2 \rightarrow a \mid b \mid X_3X_3$
   $X_4 \rightarrow a$
   $X_1 \rightarrow X_2X_3$

   - Se aplica sustitución a $X_3 \rightarrow X_4X_3$.
     Se elimina esa producción y se añade:
     $X_3 \rightarrow aX_3$.

   $X_3 \rightarrow a \mid aX_1 \mid aX_3$
   $X_2 \rightarrow a \mid b \mid X_3X_3$
   $X_4 \rightarrow a$
   $X_1 \rightarrow X_2X_3$

- Se aplica sustitución a $X_2 \rightarrow X_3X_3$.
  Se elimina esa producción y se añade:
  $X_2 \rightarrow aX_3$ $X_2 \rightarrow aX_1X_3$ $X_2 \rightarrow aX_3X_3$

$X_3 \rightarrow \underline{a} \mid \underline{aX_1} \mid \underline{aX_3}$
$X_2 \rightarrow \underline{a} \mid \underline{b} \mid \underline{aX_3} \mid \underline{aX_1X_3} \mid \underline{aX_3X_3}$
$X_4 \rightarrow \underline{a}$
$X_1 \rightarrow X_2X_3$

- Se aplica sustitución a $X_1 \rightarrow X_2X_3$.
  Se elimina esa producción y se añade:
  $X_1 \rightarrow aX_3$ $X_1 \rightarrow bX_3$ $X_1 \rightarrow aX_3X_3$ $X_1 \rightarrow aX_1X_3X_3$ $X_1 \rightarrow aX_3X_3X_3$

$X_3 \rightarrow \underline{a} \mid \underline{aX_1} \mid \underline{aX_3}$
$X_2 \rightarrow \underline{a} \mid \underline{b} \mid \underline{aX_3} \mid \underline{aX_1X_3} \mid \underline{aX_3X_3}$
$X_4 \rightarrow \underline{a}$
$X_1 \rightarrow \underline{aX_3} \mid \underline{bX_3} \mid \underline{aX_3X_3} \mid \underline{aX_1X_3X_3} \mid \underline{aX_3X_3X_3}$