Grado en Ingeniería Informática.

Portafolio de Prácticas

Nombre de la asignatura:
Modelos de Computación.

Realizado por:
Raúl Ruano Narváez

Este portafolio contiene las practicas del curso 2018/19.

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA
Y DE TELECOMUNICACIÓN.

Granada, 9 de enero de 2019.

Índice general

1. Práctica 1: Lenguajes y Gramáticas **2**
1.1. Ejercicio 1 2
1.2. Ejercicio 2 2
1.3. Ejercicio 3 3
1.4. Ejercicio 4 3

2. Práctica 2: Lex como localizador de expresiones regulares con acciones asociadas **5**
2.1. Explicación del problema 5
2.2. Código LEX en C++ 5
2.3. Prueba del programa 10

3. Práctica 3: Autómatas y expresiones regulares **11**
3.1. Ejercicio 1 11
3.2. Ejercicio 2 14
3.3. Ejercicio 3 18
3.4. Ejercicio 4 18

4. Práctica 4: Simplificación de Autómatas, FNC, Ambigüedad y Autómatas con Pila **20**
4.1. Ejercicio 1 20
4.2. Ejercicio 2 23
4.3. Ejercicio 3 25

Capítulo 1

Práctica 1: Lenguajes y Gramáticas

1.1. Ejercicio 1

Calcula una gramática libre de contexto que genere el lenguaje $L = \{a^n b^m c^m d^{2n}\}$ tal que $n, m \geq 0$

Para realizar este ejercicio he tenido en cuenta que como el numero de a's es el doble que el de d's al final, lo que he hecho ha sido crear una primera regla de producción en la que se van a ir introduciendo primero las a's y las d's de la forma que cuando se introduzca una a, se introducen luego dos d's al final. Una vez que hemos terminado de insertar a's y d's lo siguiente es otra regla de producción que inserte b's y c's, para ello uso otra regla de producción la cual cada vez que se inserta una b se inserta una c y así cumplimos con la condición de que haya el mismo número de b's que de c's. Destacar que primero se insertan a's y d's una vez que se han introducido todas las necesarias se empiezan a insertar las b's y las c's, no pudiendo volver a insertar a's y d's después de las b's y las c's. Una vez explicado teóricamente lo que he realizado, adjunto la gramática a la que he llegado:

- $T = \{a, b, \varepsilon\}$
- $V = \{S, B\}$
- $S = \{S\}$
- $P = \{\{S \rightarrow aSdd \mid bBc \mid \varepsilon\} \ , \ \{B \rightarrow bBc \mid \varepsilon\}\}$

1.2. Ejercicio 2

Describir una gramática que genere los números decimales escritos con el formato [signo] [cifra][punto][cifra]. Por ejemplo, +3.45433, -453.23344, ...

Para realizar este ejercicio lo primero que hago es insertar el signo del número y la primera cifra, para ello tengo una regla de producción, concretamente la primera, que obliga a que suceda eso. Una vez insertado el signo y la primera cifra lo que hago es con otra regla de producción insertar tantos números como se quieran, pero hasta que se llegue al punto, es decir, primero obligo a que se inserte la parte entera del número al que queremos llegar y cuando ya hemos terminado de insertar la parte entera, tengo otra regla que añade

una cifra más, pero esta vez con un punto la cual lleva a otra regla de producción que se dedica a insertar solo la parte decimal. En resumen, primero obligo a insertar el signo y una cifra, luego se insertará la parte entera del numero hasta que queramos insertar la parte decimal, la cual se insertará seguidamente con otra regla de producción. Teniendo en cuenta esta explicación adjunto la gramática obtenida:

- $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \varepsilon\}$
- $V = \{S, A, B\}$
- $S = \{S\}$
- $P = \{\{S \to -A \mid +A\}\ ,\ \{A \to (0-9)A \mid (0-9).B\}\ ,\ \{B \to (0-9)B \mid \varepsilon\}\}$

1.3. Ejercicio 3

Calcula una gramática libre de contexto que genere el lenguaje $L = \{0^i 1^j 2^k\}$ tal que $i \neq j$ o $j \neq k$

Para abordar este problema lo primero que hago es ver que el problema puede tener solución de dos formas distintas en las que estaría correcto y es lo que nos dice la condición de que o i!=j o j!=k entonces lo primero que hago es crear una primera regla de producción que o se dedique a cumplir la primera o la segunda condición es decir una regla que inserte primero 00 y 1 y otra regla que inserte 11 y 2, para ello uso la regla de producción S para empezar cumpliendo una condición u otra. Una vez hemos decidido el camino por donde vamos a ir simplemente se van a insertar 00 y 1 o 11 y 2 dependiendo de lo que se quiera insertar, pero siempre se inserta primero dos 00 o dos 11 y luego un 1 o solo un 2 para así cumplir la condición que se nos pide. Para verlo mejor adjunto la gramática a la que he llegado:

- $T = \{0, 1, 2, \varepsilon\}$
- $V = \{S, A, B\}$
- $S = \{S\}$
- $P = \{\{S \to 00A1B \mid A11B2\}\ ,\ \{A \to 00A1 \mid \varepsilon\}\ ,\ \{B \to 11B2 \mid \varepsilon\}\}$

1.4. Ejercicio 4

Una empresa de videojuegos "The fantastic platform" están planteando diseñar una gramática capaz de generar niveles de un juego de plataformas, cada uno de los niveles siguiendo las siguientes restricciones:

- **Hay 2 grupos de enemigos: grupos grandes (g) y grupos pequeños (p).**
- **Hay 2 tipos de monstruos: fuertes (f) y débiles (d).**
- **Los grupos grandes de enemigos tienen, al menos, 1 monstruo fuerte y 1 débil.**

- **Y los 2 primeros monstruos pueden ir en cualquier orden. A partir del tercer monstruo, irán primero los débiles y después los fuertes.**
- **Los grupos pequeños tienen como mucho 1 monstruo fuerte.**
- **Al final de cada nivel habrá una sala de recompensas (x).**

Elaborar una gramática que genere estos niveles con sus restricciones. Cada palabra del lenguaje es un solo nivel. ¿A qué tipo de gramática dentro de la jerarquía de Chomsky pertenece la gramática diseñada?
¿Sería posible diseñar una gramática de tipo 3 para dicho problema?

Lo primero que he realizado es una regla de producción inicial que obligue a que las cadenas sean del tipo g**p*x o del tipo p**g**x tal y como nos dice el ejercicio es decir, "g" de grupo grande y "p" de grupo pequeño, las cadenas pueden ser de esos dos tipos ya que el grupo pequeño o grande puede ser el primero o segundo en definirlo, con la "x" representamos la recompensa que como bien dice el enunciado tiene que ir al final de cada nivel, luego al final de la cadena. Para representar el grupo grande (g) nos dice que tiene al menos un monstruo fuerte (f) o débil (d), los cuales los dos primeros pueden tener cualquier orden, pero a partir del tercer monstruo deben ir primero los débiles y luego los fuertes, para ello he usado la producción A, la cual obliga primero a introducir un monstruo débil o fuerte y viceversa y luego se van insertando monstruos débiles con la producción C hasta que queramos introducir monstruos fuertes que para ello usamos la producción D y con esta vamos a introducir los monstruos débiles hasta que finalicemos. Para construir el grupo pequeño como nos dicen que como mucho tienen un monstruo fuerte, después de haber insertado "p" que representa al grupo pequeño tenemos una regla de producción B la cual puedes insertar tantas "d" como se quieran y si se inserta una "f" automáticamente ya solo luego te dejaría insertar "d" ya que te llevaría a la producción E en la que solo te deja finalizar o insertar "d", cumpliendo con el requisito que nos propone el ejercicio. Finalmente la gramática que nos queda es la siguiente:

- $T = \{g, p, f, d, x, \varepsilon\}$
- $V = \{S, A, B, C, D, E\}$
- $S = \{S\}$
- $P = \{\{S \to gApBx \mid pBgAx\}\ ,\ \{A \to fdC \mid dfC\}\ ,\ \{B \to fE \mid dB\}\ ,$
  $\{C \to dC \mid dD\}\ ,\ \{D \to fD \mid \varepsilon\}\ ,\ \{E \to dE \mid \varepsilon\}\}$

Capítulo 2

Práctica 2: Lex como localizador de expresiones regulares con acciones asociadas

2.1. Explicación del problema

El problema que hemos propuesto para realizar un parsing con la herramienta LEX es el de realizar una búsqueda en Amazon.es y presentar los productos de la búsqueda en una tabla que sea visual. Para ello nuestro programa puede funcionar de dos formas diferentes, la primera forma es el poder realizar la búsqueda directamente desde el programa, este realizará una petición a la web de Amazon.es, se descargará el archivo HTML de la web y lo parseará con las expresiones regulares que hemos creado.

Otra forma de ejecutar el programa es descargarnos directamente nosotros el HTML de una búsqueda que hayamos hecho en Amazon.es y se lo pasamos por linea de comandos la localización de nuestro fichero HTML en el sistema. Para ejecutar el programa basta con realizar "*make*" y seguidamente la ejecución del programa de las siguientes formas:

- **./bin/P2 -i**
- **./bin/P2**
- **./bin/P2 [localización_fichero]**

En cuanto a las expresiones regulares que hemos utilizado han sido aquellas que nos buscaban en el fichero HTML el número del producto, numero de paginas totales (pero en nuestro caso solo hemos hecho una búsqueda de la primera pagina de Amazon, no sería muy complicado adaptarlo para cualquier pagina, pero creemos que al ser simplemente programación en C++ y no usar la herramienta LEX lo hemos dejado pasar), el nombre del producto, su precio, y las valoraciones de este.

2.2. Código LEX en C++

Archivo: P2.l

```
        //----- Sección de Declaraciones --------------//

%{
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string>
#include <vector>
#include <sys/ioctl.h>
#include <unistd.h>
#include <iomanip>

using namespace std;

int nc, np, nl;
vector<vector<string>> datosObtenidos;
int indice=0;

void escribir_datos (string paginasTotales, vector<vector<string>> datosObtenidos,
 ↪  ostream& os);
string paginasTotales;
string palabra_buscada;
string busqueda_final;
string amazon = "https://www.amazon.es/s/ref=nb_sb_noss_2?__mk_es
                    _ES=ÅMÅŽÕÑ&url=search-alias%3Daps&field-keywords=";
%}

numero_producto <li[ ]id=\"result_[0-9]*
num_paginas_totales <span[ ]class=\"pagnDisabled\">[0-9]*<\/span>
nombre_producto <h2[ ]data-attribute=\"([^\"]*)
precio_producto <span[ ]class=\"a-size-base[ ]a-color-price[ ]s-price[
 ↪  ]a-text-bold\">EUR[ ][0-9]+([,][0-9]+)?<\/span>
estrellas_producto  <span[ ]class=\"a-icon-alt\">[0-9]+([,][0-9]+)?[ ]de[ ]un[
 ↪  ]máximo[ ]de[ ]5[ ]estrellas<\/span>
link_producto <div[ ]aria-hidden=\"true\"[ ]class=\"a-column[ ]a-span12[
 ↪  ]a-text-left\"><div[ ]class=\"a-section[ ]a-spacing-none[ ]a-inline-block[
 ↪  ]s-position-relative\"><a[ ]class=\"a-link-normal[ ]a-text-normal\"[
 ↪  ]href=\"([^\"]*)
%%
        //----- Sección de Reglas ---------------- //
.|\ {}

{numero_producto} {
                    string num_producto = yytext;
                    indice =
                     ↪  stoi(num_producto.substr(num_producto.find("_")+1,num_producto.length()));
                    datosObtenidos.resize(indice+1);
```

```
41                      datosObtenidos[datosObtenidos.size()-1].assign(4,"X");
42                    }
43
44  {num_paginas_totales} {
45                          string pag_totales = yytext;
46                          string final = pag_totales.substr(27, pag_totales.find("</"));
47                          paginasTotales = final.substr(0,final.find("<"));
48                        }
49
50  {nombre_producto} {
51                      string nombre_producto = yytext;
52                      datosObtenidos[indice][0] = nombre_producto.substr(20,
                         ↪  nombre_producto.length());
53                    }
54
55  {precio_producto} {
56                      string precio_producto = yytext;
57                      precio_producto =
                         ↪  precio_producto.substr(precio_producto.find("EUR"));
58                      precio_producto =
                         ↪  precio_producto.substr(0,precio_producto.length()-7);
59                      datosObtenidos[indice][1] = precio_producto;
60                    }
61
62  {estrellas_producto}  {
63                          string valoracion = yytext;
64                          valoracion = valoracion.substr(25,valoracion.length());
65                          valoracion = valoracion.substr(0,valoracion.find(" "));
66                          datosObtenidos[indice][2] = valoracion + "/5";
67                        }
68
69  {link_producto} {
70                    string link_producto = yytext;
71                    datosObtenidos[indice][3] = link_producto.substr(180,
                       ↪  link_producto.length());
72                  }
73  %%
74
75  using namespace std;
76
77  int main(int argc, char** argv){
78    int out;
79    string opcion;
80    FILE* file;
81    bool ejecucion = false;
82    nc = np = nl = 0;
83
84    int saved_stdout = dup(1);
```

```
85    dup2(NULL, 1);
86
87    if(argc==2)
88      opcion=argv[1];
89
90    //MODO 1: Se realiza petición con las palabras claves que se han buscado
91    if(argc == 1){
92      cout << "¿Que deseas buscar en Amazon.es" << endl;
93      cin >> palabra_buscada;
94
95      busqueda_final="\""+amazon+palabra_buscada+"\"";
96      string wgetFinal="wget -U \"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
        ↪  (KHTML, like Gecko) Chrome/61.0.3163.79 Safari/537.36\"
        ↪  --output-document=./datos/index.html "+busqueda_final;
97
98      out = system((const char*)wgetFinal.c_str());
99      cerr << setw(20) << "wget" << "-->" << strerror(out)  << left << endl;
100
101     file = fopen("./datos/index.html", "r");
102     if (file == NULL) {
103       cout << strerror(ferror(file)) << endl;
104       exit (-1);
105       yyin = file;
106     }else
107       yyin = file;
108
109     ejecucion=true;
110   }else if(argc == 2 && opcion.compare("-i")==0){
111     cout << "USO: ./P2 [archivo] o ./P2" << endl << "La primera opción es para pasarle
        ↪  un HTML de Amazon y la segunda opción realiza una petición a Amazon." <<
        ↪  endl;
112     return 0;
113   }else{
114     file = fopen(opcion.c_str(), "r");
115     if (file == NULL) {
116       cout << "El fichero no existe o no has dado la ruta correcta." << endl;
117       exit (-1);
118     }else{
119       yyin = file;
120       ejecucion=true;
121     }
122   }
123
124   if(ejecucion){
125     yylex();
126     dup2(saved_stdout, 1);
127     close(saved_stdout);
128     escribir_datos(paginasTotales, datosObtenidos, cout);
```

```
129   }
130 }
131
132 void escribir_datos (string paginasTotales, vector<vector<string>> datosObtenidos,
    ↪  ostream& os){
133   string separator = string(79,'-');
134   string flag = "\033[0m";
135   cout << endl;
136   cout << "SI SE ENCUENTRA UNA \"X\" EN LOS RESULTADOS ES QUE EL PRODUCTO NO TIENE ESE
      ↪  DATO." << endl << "LOS TITULOS NO ESTAN COMPLETOS DEBIDO A QUE SON DEMASIADO
      ↪  EXTENSO, SE HAN ACORTADO A 30 CARACTERES." << endl;
137   cout << flag << separator << endl;
138   cout <<"| " << "#" << setw(4) << " | " << "Nombre producto" << setw(25) << " | " <<
      ↪  "Precio " << setw(11) << " | " << setw(10) << "Valoraciones" << " | "  << endl;
139
140   cout << flag << separator  << "\033[0m" << endl;
141   for(int i=0;i<datosObtenidos.size();i++){
142     if(i<10){
143       cout << flag << separator << endl;
144       cout <<"| " << i << setw(4) << " | " << datosObtenidos[i][0].substr(0,30) <<
        ↪  setw(10) << " | ";
145       cout.width(10);
146       cout << datosObtenidos[i][1] << setw(8) << " | ";
147       cout.width(5);
148       cout << datosObtenidos[i][2] << setw(10) << " | "  << endl;
149       cout << separator << endl;
150     }else{
151       cout << flag << separator << endl;
152       cout <<"| " << i << setw(3) << " | " << datosObtenidos[i][0].substr(0,30) <<
        ↪  setw(10) << " | ";
153       cout.width(10);
154       cout << datosObtenidos[i][1] << setw(8) << " | ";
155       cout.width(5);
156       cout << datosObtenidos[i][2] << setw(10) << " | "  << endl;
157       cout << separator << endl;
158     }
159   }
160 }
```

2.3. Prueba del programa

```
raul@raul-UX305LA:~/Dropbox/UNIVERSIDAD/MC/PRACTICA 2$ ./bin/P2
¿Que deseas buscar en Amazon.es
portatil
--2018-12-30 13:08:35--  https://www.amazon.es/s/ref=nb_sb_noss_2?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&url=search-alias%3Daps&field-keywords=portatil
Resolviendo www.amazon.es (www.amazon.es)... 72.247.214.159
Conectando con www.amazon.es (www.amazon.es)[72.247.214.159]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: no especificado [text/html]
Guardando como: “./datos/index.html”

./datos/index.html                                      [       <=>

2018-12-30 13:08:37 (461 KB/s) - “./datos/index.html” guardado [728187]

               wget-->Success

SI SE ENCUENTRA UNA "X" EN LOS RESULTADOS ES QUE EL PRODUCTO NO TIENE ESE DATO.
LOS TITULOS NO ESTAN COMPLETOS DEBIDO A QUE SON DEMASIADO EXTENSO, SE HAN ACORTADO A 30 CARACTERES.
-------------------------------------------------------------------------------
| #  | Nombre producto                      | Precio          | Valoraciones |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 0  | Lenovo Ideapad 330-15IGM - Ord       | EUR 229,00      | 4,1/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 1  | HP Notebook 15-da1013ns - Orde       | EUR 499,99      | 4,5/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 2  | Acer Aspire 3 A315-41-R8ZC Ord       | EUR 479,99      | 3,9/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 3  | Lenovo Ideapad 320-15AST - Ord       | EUR 599,00      | 2,8/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 4  | Lenovo IdeaPad 320-15IKB - Ord       | EUR 533,29      | 3,8/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 5  | Acer Aspire 5 A515-51G - Orden       | EUR 693,03      | 4,1/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 6  | HP Notebook 15-da0014ns - Orde       | EUR 399,99      | 3,4/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 7  | ASUS K540LA-XX1313T - Ordenado       |          X      | 3,9/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 8  | HP 15-bw068ns - Ordenador port       |          X      | 3,2/5        |
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 9  | HP Notebook 15-BS533NS - Port&       | EUR 660,98      |   5/5        |
-------------------------------------------------------------------------------
```

Capítulo 3

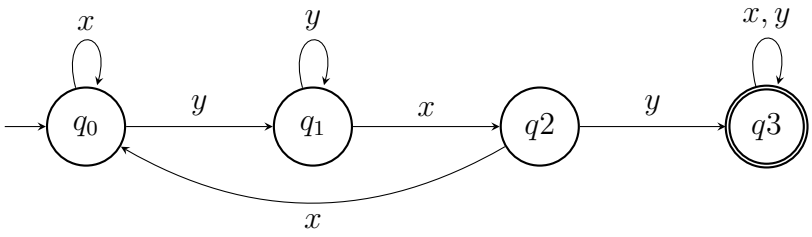
Práctica 3: Autómatas y expresiones regulares

3.1. Ejercicio 1

En el alfabeto {x,y}, construir un AFD que acepta cada uno de los siguientes lenguajes:

1. **El lenguaje de las palabras que contienen la subcadena yxy.**
2. **El lenguaje de las palabras que comienzan o terminan en xyx (o ambas cosas).**
3. **El lenguaje $L \subseteq \{x, y\}^*$ que acepta aquellas palabras con un número impar de ocurrencias de la subcadena xy.**

Solución apartado 1:
He definido este Autómata Finito Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El autómata que he generado ha sido el siguiente, el cual luego será explicado en profundidad:



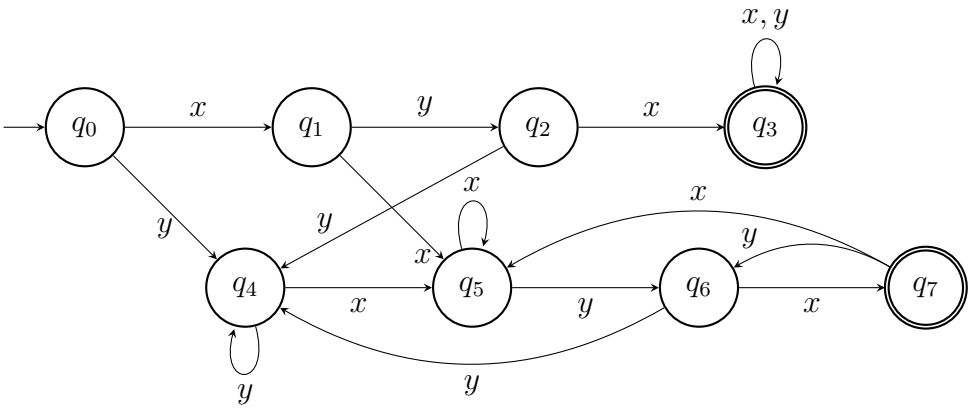
Ahora explicaré cada uno de los estados:

- **q0**, estado inicial en el cual se permanece mientras solo le lleguen "x", ya que este no cambiará hasta que no le llegue una "y".
- **q1**, estado en el que ya se ha introducido una "x" y va a permanecer mientras le estén llegando "y", el cual cambiará cuando le llegue una nueva "x".

- **q2**, estado que representa que se ha introducido una "y" seguido de una "x", en este estado si se le introduce una "y" volverá al estado inicial, debido a que se perdería el orden que buscamos de cadenas que es "yxy", como estamos en el segundo carácter "x", si introducimos una "x" ya no se correspondería con la cadena que buscamos.

- **q3**, estado final el cual ya se ha introducido "yxy" por lo tanto se acepta la cadena y ya se pueden introducir cualquier carácter ya sea "x" o "y" en cualquier orden, ya que la cadena ya ha sido aceptada.

Solución apartado 2:
He definido este Autómata Finito Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El autómata que he generado ha sido el siguiente, el cual luego será explicado en profundidad:



Ahora explicaré cada uno de los estados:

- **q0**, estado inicial por el que se van a producir dos situaciones, si le llega una "x" quiere decir que tiene posibilidad de que la cadena comience por "x" pudiendo llegar a la cadena que buscamos al principio de los caracteres "xyx", en cambio, si de primera en este estado le llega un "y" se deduce que la cadena que se va a leer no va a comenzar por la tripleta "xyx", luego ya obviamos la posibilidad de que ocurra al principio y solo nos centraremos en que pueda ocurrir al final.

- **q1**, estado el cual ya ha aceptado una "x" al principio de la cadena y ahora tiene dos posibilidades insertar una "y" y buscar llegar a tener la tripleta "xyx" al principio de la cadena o introducir una "x" y saltarnos la búsqueda de la tripleta "xyx" al principio de la cadena.

- **q2**, estado en el que se ha aceptado la pareja de caracteres "xy" al principio de la cadena, luego tiene dos posibilidades, llegar a un estado final y aceptar la cadena introduciendo una "x" o introducir una "y" y olvidarnos de buscar la tripleta en el inicio de la cadena de "xyx", básicamente en los estados "q_1" y "q_2" lo que se hace es ir aceptando los caracteres que conforman la tripleta "xyx" y si en algún momento se introduce un carácter "x" o "y" en la posición que se busca, saltamos a buscar la tripleta "xyx" al final de la cadena.

- **q3**, estado final en el que ya se ha aceptado la tripleta "xyx" y que esta se encuentra al principio de la cadena, una vez aceptada esta, como nos dice que acepta una u otra o ambas, con que asegure una condición la otra no hace falta comprobarla, ya que si por casualidad termina en "xyx" ya estaría aceptada por empezar en "xyx".

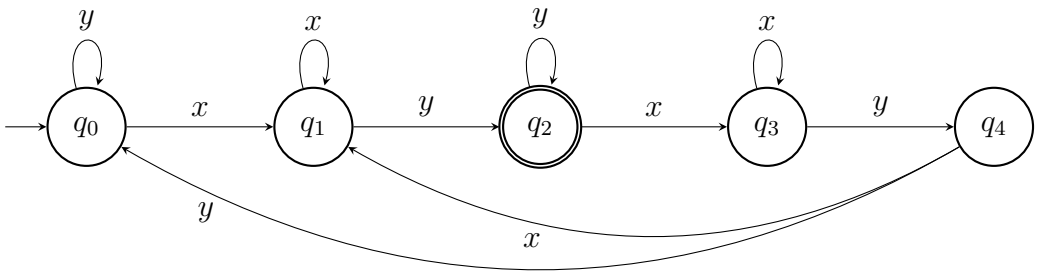
- **q4**, en este punto en el que estamos el autómata ya ha detectado que la cadena no empieza por "xyx" luego se va a proceder a buscar la posibilidad de aceptar al final "xyx" en la cadena, para ello en este estado se pueden introducir una "y" que produciría un bucle en este mismo estado o una "x" la cual te va a llevar al siguiente estado, esto es debido a que como queremos buscar la tripleta "xyx" y si metemos una "x" quiere decir que la tripleta puede ser que termine con "xyx".

- **q5**, en este estado ya se ha introducido una "x" y lo que va a esperar ahora es que si le siguen introduciendo "x" se va a quedar en el mismo estado ya que lo que buscamos es "xyx" y si le introducen una "y" pasa al siguiente estado el cual esta a un paso de aceptar "xyx" al final de la cadena, básicamente es un paso entre la "x" primera de la tripleta "xyx" y la "y".

- **q6**, en este estado ya estamos en el momento en el que se ha aceptado la pareja de caracteres "xy" y esta cerca de aceptar la tripleta "xyx", pero puede o llegar a ese estado, o que se le introduzca una "y" la cual va a producir un reinicio de la lectura de la búsqueda de esa tripleta ya que volveríamos al inicio debido a que se rompería el orden que buscamos de dicha tripleta, en cambio si se introduce una "x" podremos decir que pasamos al siguiente estado, estado que es final ya que hemos llegado a lo que buscábamos que es "xyx", pero ocurre una situación que explicare en el estado correspondiente.

- **q7**, estado final en el que ya hemos llegado a encontrar "xyx" al final de una cadena, que pasa que si ya no se introducen más caracteres hemos finalizado, pero si se introduce una "x" o una "y" vamos a regresar a los estados "q_5" y "q_6" debido a que ya la cadena no termina en "xyx", luego se tendría que realizar otra vez el procedimiento desde los estados anteriores, se podría decir que se entra en un bucle en el que no se acepta la cadena hasta que esta no termine totalmente en "xyx".

Solución apartado 3:
He definido este Autómata Finito Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El autómata que he generado ha sido el siguiente, el cual luego será explicado en profundidad:



Ahora explicaré cada uno de los estados:

- **q0**, estado inicial en el que si le llega una "y" se queda en bucle en este mismo estado y si le llega una "x" pasa al siguiente estado, si le entra una "y" se queda en el mismo estado debido a que lo que buscamos es una "x" para pasar al siguiente estado, ya que lo que buscamos es que nuestra cadena tenga un numero impar de veces la pareja de caracteres "xy".

- **q1**, estado en el que ya se ha aceptado una "x" y en el que si le llega una "x" se queda en el mismo estado y si le llega una "y" pasa al siguiente estado en el que ya tendríamos un número impar de veces "xy", cuando antes digo que se queda en el mismo estado si introduce una "x" es debido a que como ya hemos aceptado una "x" y al introducirla pasábamos a este estado no tendría sentido volver al estado inicial por esto mismo, por que ya está aceptada la "x".

- **q2**, estado final, en este momento se ha llegado a una cantidad de parejas de "xy" impar por eso aceptamos la cadena siempre y cuando terminemos aquí ya que si volvemos a recoger caracteres pueden ocurrir dos cosas que se introduzca una "y" y nos quedemos en el mismo estado debido a que sigue habiendo un numero impar de "xy" o introducir una "x" la cual va a producir un cambio de estado a "q_3" ya que no tendríamos un numero impar de "xy" y tendríamos que buscar otra vez otra "xy" para poder volver de nuevo a buscar otra "xy" más que nos lleve de nuevo a este estado final, es decir, buscar dos estados mas de "xy" para llegar de nuevo a una cadena final.

- **q3**, estado en el que se va a permanecer cuando llegan "x" y se va a pasar al siguiente estado cuando le llegue una "y", es un estado intermediario que va a llegar a un estado en el que va haber parejas de "xy" pares cuando se pase "q_4", luego es un estado simplemente de intermediario.

- **q4**, estado en el que hay un número par de parejas de "xy" como ya hemos aceptado una "xy" impar pero al introducir más caracteres la cadena deja de ser aceptada ya que no hay un numero impar de parejas de "xy" pues este estado lo que hace es que le llegan parejas de "xy" pares, en el cual si se le introduce una "x" se volvería al estado "q_1" y se realiza el procedimiento anteriormente explicado o se pasa a "q_0" cuando se lea una "y" debido a que habría que introducir otra "x" para poder llegar a la pareja "xy" y poder aceptarla luego en "q_2".

3.2. Ejercicio 2

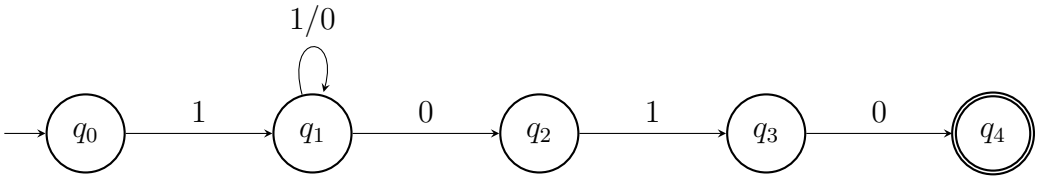
En el alfabeto {0, 1}, construir un AFND que acepte cada uno de los siguientes lenguajes:

1. **El lenguaje de las palabras que empiezan en 1 y terminan en 010.**

2. **El lenguaje de las palabras que empiezan o terminan (o ambas cosas) en 101.**

3. **El lenguaje de las palabras que contienen, simultáneamente, las subcadenas 0101 y 100. El AFDN también acepta las cadenas en las que las subcadenas están solapadas (por ejemplo, "10100" y "100101" serían palabras aceptadas).**

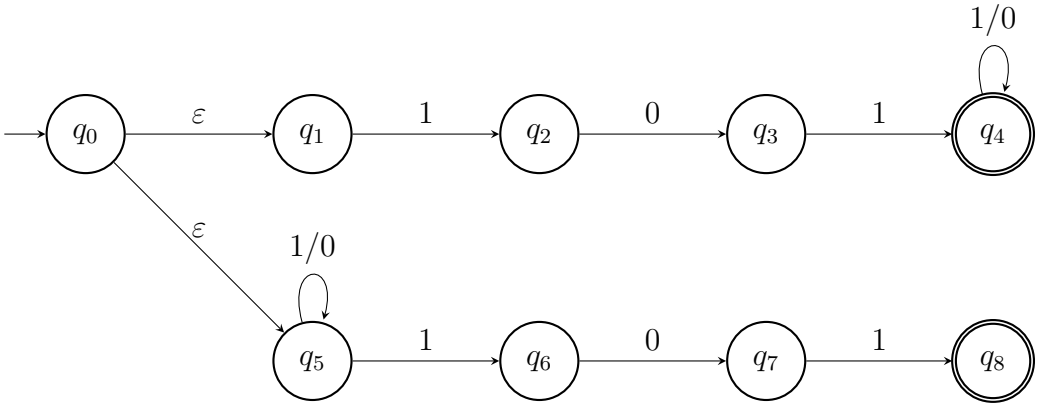
Solución apartado 1:
He definido este Autómata Finito NO Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El autómata que he generado ha sido el siguiente, el cual luego será explicado en profundidad:



Ahora explicaré cada uno de los estados:

- **q0**, estado inicial del autómata el cual si le llega un "1" podrá aceptar la cadena para ello pasará al segundo estado, en caso contrario denegará la cadena, ya que esta debería empezar en "1".
- **q1**, estado en el cual ya se ha aceptado un primer "1" del estado anterior, y va a permanecer en este estado mientras se introduzcan "0" o "1", pero puede ocurrir que se introduzca un "0" el cual hace que se cambie de estado para ver si se produce la segunda condición de que termine la cadena en "010".
- **q2**, en este estado ya se ha aceptado un "1" inicial de la cadena, una serie de "0" y de "1" y ha sido aceptado el primer "0" que puede producir que estemos al final de la cadena, llegando a tener "010".
- **q3**, en este punto hemos recibido ya parte de la la tripleta "010" la cual buscamos que este al final, la parte recibida es "01", si por el contrario estando en "q_2" no recibe un "1" como seguimos estando en "q_1" seguiríamos en ese estado, es decir, que mientras no introduzcamos expresamente "010" vamos a seguir estando en "q_1".
- **q4**, estado final en el cual ya ha sido aceptada la cadena que buscamos, es decir, que al principio de la cadena haya un "1", después una serie de "0" y de "1" y al final se termina la cadena con "010".

Solución apartado 2:
He definido este Autómata Finito NO Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El autómata que he generado ha sido el siguiente, el cual luego será explicado en profundidad:



Lo primero que necesito explicar es que cada una de las alternativas del autómata tanto la de la que empieza en "q_1" y la que empieza en "q_5" son dos autómatas diferentes uno que acepta al principio la cadena "101" y luego la segunda que acepta la cadena "101" al final, lo he dividido en dos por que así pueden ser aceptadas ambas, una o ninguna.

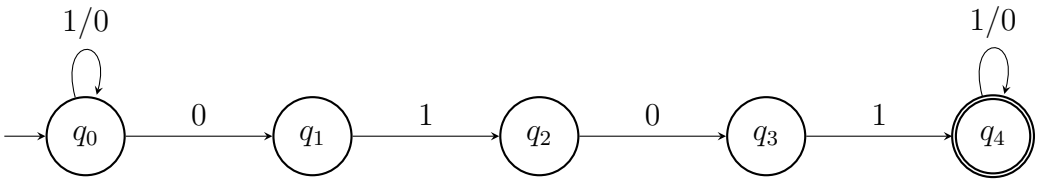
El funcionamiento de la primera parte del autómata es simplemente la de aceptar de inicio la cadena "101" y finalmente quedar en un bucle en el que se van a estar aceptando tanto "0" como "1" en el orden que se desee, pero eso si, ya habrá sido aceptada la cadena principal que buscamos.

La segunda parte del autómata en cambio realiza lo contrario, se van a leer una serie de "0" y de "1" hasta que se encuentre la cadena "101" al final de la cadena. Destacar que si el autómata pasa del estado "q_5" al estado "q_6" pero estando en "q_6" se inserta un "1" de nuevo seguirá estando en el bucle de "q_5" leyendo "0" y "1".

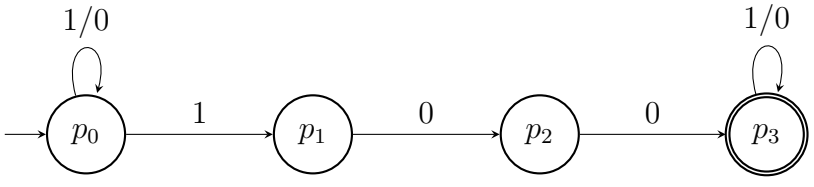
Básicamente lo único que he realizado es los dos autómatas por separado y los he unido con las transiciones de epsilon (nulas).

Solución apartado 3:
He definido este Autómata Finito NO Determinista el cual esta constituido por los diferentes estados Q, el alfabeto de entrada A, el estado inicial q_0, el conjunto de funciones de transición δ y el estado final F. El proceso de generación del autómata final ha sido el siguiente:

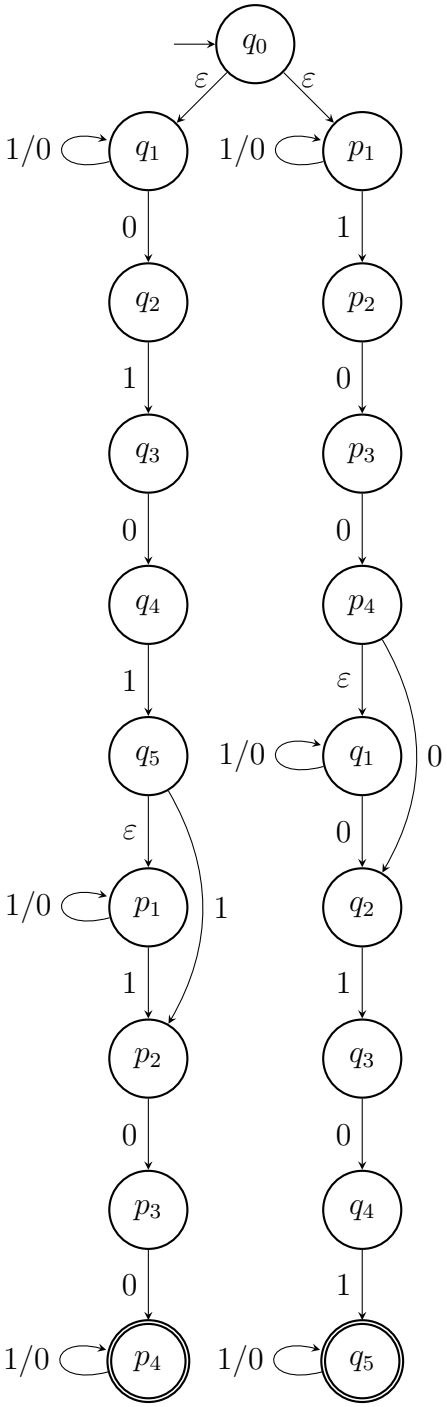
Lo primero que he realizado es el autómata que se encarga de aceptar cadenas que contengan la subcadena "0101". Este autómata es sencillo ya que simplemente en el primer estado se aceptar "0" y "1" hasta que se introduce un "1" y se empieza a comprobar si se ha llegado a la cadena "0101" y si es así se llega a un estado final "q_4" en el que ya se van a leer "0" y "1" pero ya siendo aceptada la cadena, ya que incluye la subcadena "0101".



Lo segundo que he realizado es el autómata que se encarga de aceptar cadenas que contengan la subcadena "100". Sigue el mismo procedimiento que el anterior autómata pero simplemente en vez de aceptar la cadena "0101" acepta la cadena "100".



Una vez tenemos los dos autómatas por separado, lo que nos quedaría es, juntar a ambos para que puedan aceptar cadenas en las que se puedan encontrar ambas subcadenas solapadas, para ello se ha realizado el siguiente autómata:



El autómata final que he generado no es mas que la unión de ambos autómatas antes generados pero teniendo en cuenta que ambos pueden aceptar las cadenas antes o después que la otra (de ahí las dos copias de ambos autómatas en las dos ramas), es decir, aceptar "1000101" o "0101100". Para suplir el problema del solapamiento lo único que he realizado es poner transiciones que pasan de "p_4" a "q_2" que sería el caso de que ocurriera el solapamiento de "100" con "0101" quedando "100101". En cambio, para aceptar la cadena solapada al revés lo que he realizado es una transición desde "q_5" a "p_2" para poder aceptar la cadena "010100".

3.3. Ejercicio 3

Calcular una máquina de Mealy o Moore que codifique el complemento a dos de un número en binario.

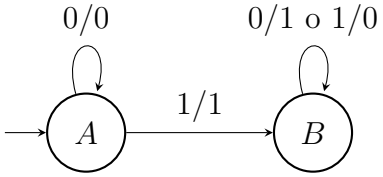
Nota: El complemento a dos se realiza cambiando ceros por unos y unos por ceros, y luego, al resultado, sumándole uno en binario.

Nota 2: El complemento a dos es la forma en que se calcula el entero opuesto a uno dado para la representación binaria de los enteros con signo en C++.

Para solucionar este problema lo primero que he tenido en cuenta es como calcular el complemento a dos de un numero que esta en binario, para ello he usado la forma en la que lo calculábamos en otras asignaturas la cual lo que se hacia era coger el número en binario y se van cogiendo bit a bit cada número de derecha a izquierda. Lo primero que hacemos es buscar el primer 1 que nos encontremos, como ya he dicho, de derecha a izquierda, cuando lo encontremos este bit lo dejamos con ese 1 y a partir del siguiente vamos cambiando 0's por 1's y viceversa hasta llegar al ultimo bit de la izquierda.

Teniendo en cuenta esto he realizado el siguiente autómata el cual hay que tener en cuenta que cada estado tiene una entrada y una salida (E/S), en el caso del estado A lo que realiza es que mientras lee 0's se queda en ese mismo estado y escribe un 0, cuando se recibe un 1 por primera vez se escribe un 1 y se cambia de estado al B y aquí lo que se realiza es el cambio de los 0's por 1's y viceversa.

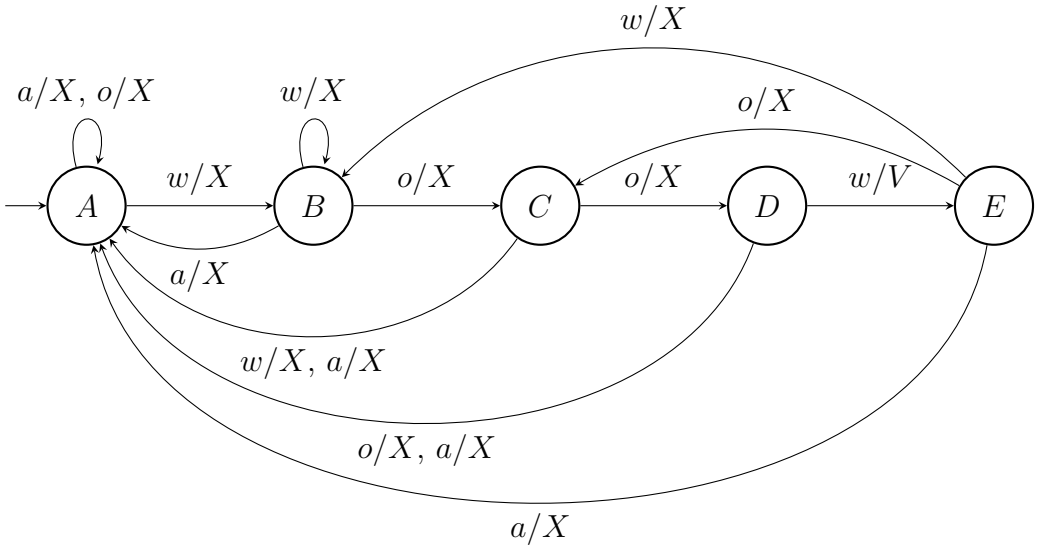
Hay que entender que este autómata lee los bits de la cadena que le llega de derecha a izquierdas.



3.4. Ejercicio 4

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.

La siguiente máquina de Mealy que he generado tiene cinco estado los cuales serán explicados con mas detalles luego.



Para abordar este ejercicio lo que he necesitado han sido cinco estados los cuales tienen su entrada y su salida y en función de esta se quedaran en el mismo estado o cambiaran, estos estados los voy a explicar uno a uno:

- **A**, estado inicial en el que van a ir todas las transiciones cuando están en un estado y se produce la lectura de un carácter que no es el que se buscaba. De este estado se saldrá cuando se introduzca una "w", el primer carácter de la subcadena "$woow$".

- **B**, estado siguiente en el que ya hemos leído una "w" y en el que se va a permanecer hasta que se lea una "o" la cual te va a llevar al siguiente estado o en cambio puede volver al estado anterior "A" si se introduce una "a".

- **C**, estado en el que ya se ha leído la subcadena "wo" y en el que si se lee un carácter diferente a "o" se volverá al estado inicial "A", si se lee el carácter "o" se pasará al estado siguiente "D".

- **D**, en este estado ya hemos casi leído la subcadena que buscamos y sucede como en el caso anterior, si se lee un carácter diferente a "w" se vuelve al estado inicial ya que romperíamos con la cadena que buscamos que es "$woow$", en cambio, si leemos el carácter "w" pasamos al estado final en el que activaríamos la luz poniendo en la salida una "V".

- **E**, estado final, en este estado simplemente te va a llevar a los estados "A", "B" o "C" dependiendo de la lectura que haga ya que si lee una "a" lo llevaría al estado inicial por que para volver a encender el leed necesitamos la palabra "$woow$" y el carácter "a" no forma parte de este haciendo que tengamos que buscarlo desde el principio. En cambio, si se introduce una "w" volveríamos al estado "B" que era el estado en el que ya teníamos introducida una "w". Por ultimo, puede ser que se introduzca una "o", que nos llevaría al estado "C" ya que en este caso ya tendríamos introducido una "w" y una "o" lo que nos lleva a saltarnos el estado "A" y "B".

Capítulo 4

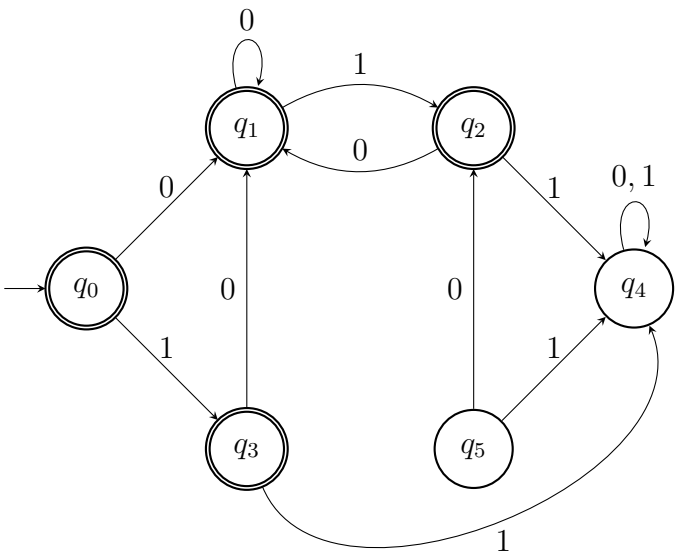
Práctica 4: Simplificación de Autómatas, FNC, Ambigüedad y Autómatas con Pila

4.1. Ejercicio 1

Dado el siguiente autómata responde a las siguientes cuestiones razonadamente:

a. **¿Habría alguna forma de optimizar el autómata para que reduciendo su complejidad siga aceptado exactamente el mismo lenguaje?**

b. **¿Se podría obtener la gramática que genera este lenguaje en la Forma Normal de Chomsky? En caso afirmativo, proporcionar dicha gramática en FNC. En caso contrario, justificar.**

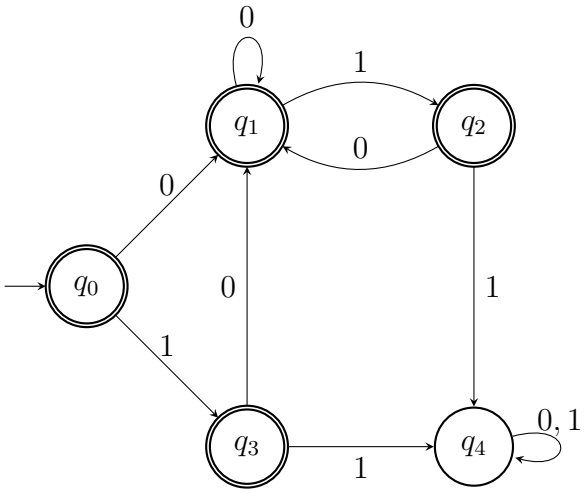


Solución apartado a:
Para comprobar si el autómata se puede optimizar o no lo que voy a realizar es aplicar

el algoritmo de optimización y si obtengo el mismo autómata no se puede optimizar y si conseguimos optimizarlo nos quedará un autómata más sencillo y óptimo.

Los pasos que voy a realizar para aplicar dicho algoritmo son primero eliminar estados inaccesibles, aplicar el algoritmo de minimización y finalmente generar el autómata final con los estados que hemos calculado.

Lo primero que voy a realizar es el primer paso del algoritmo el cual consiste eliminar, como ya he dicho, estados inaccesibles, aquellos estados a los que no se puede llegar, en el caso del autómata del ejercicio, podemos observar que el único estado al que no se puede llegar es "$q5$" por lo tanto eliminamos ese estado y el autómata que nos quedaría es el siguiente:



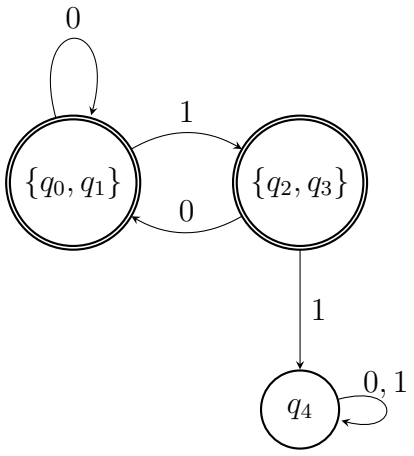
Una vez realizado la eliminación de estados inaccesibles lo siguiente es aplicar el algoritmo de minimización, este algoritmo lo que realiza es intentar disminuir el numero de estados, dejando solo aquellos que son distinguibles entre sí. Dos estados que no son equivalentes, se dice que son distinguibles. Voy a construir las diferentes tablas que he necesitado para aplicar este algoritmo:

	X={2,4}	Y={0,1,3}
0	Y X	Y Y Y
1	X X	Y X X

	X={2,1,3}	Y={4}	Z={0}
0	X X X	Y	X
1	Y X Y	Y	X

	X={2,3}	Y={4}	Z={0,1}
0	Z Z	Y	Z Z
1	Y Y	Y	X X

Cada tabla es cada una de las iteraciones que he tenido que dar para llegar a un autómata minimizado, lo que he realizado primero es dividir en dos conjuntos los estados finales y los no finales y a partir de ahí voy comprobando a que estado se dirige cada estado en función de si le llega un 0 o un 1, y lo que pongo en función de lo que le llega es a que conjunto de estado va, en el primer caso los conjuntos posibles son "X" o "Y", luego si estamos en "2", por ejemplo, y le llega un "0" se va al estado "q_1" el cual se encuentra en el conjunto de estados "Y" pues lo señalo y así con todos. Cuando se acaba lo primero que hacemos es como "q_2" y "q_4" son distinguibles entre si los separamos en dos conjuntos diferentes tal y como se ve en la segunda tabla. Aquellos estados que usen el mismo conjunto de estados los ponemos juntos, es el caso de "1" y "3" con "2" luego los juntamos en "X". Volvemos aplicar el procedimiento que hemos realizado en la primera tabla, pero ahora con un nuevo conjunto, el conjunto "Z", lo que nos da lugar a la segunda tabla, la cual nos hace ver que "2" y "3" no son distinguibles entre si, que "4" es distinguible de todos y que "0" y "1" no son distinguibles entre si, por lo tanto nos quedan los conjuntos "$X = 2, 3$", "$Y = 4$" y "$Z = 0, 1$", y ya hemos terminado de minimizar el autómata debido a que ya están agrupados cada uno de los estados con aquellos que son equivalentes. Por lo tanto el autómata minimizado que nos ha quedado es el siguiente:



Solución apartado b:
Para calcular (si existe) la forma normal de Chomsky lo que hay que realizar es pasar el Autómata Finito Determinista a gramática, la cual tiene renombrados los estados $S = \{q_0, q_1\}$, $A = \{q2, q3\}$ y $B = \{q4\}$. La gramática final que hemos deducido del AFD que hemos minimizado es la siguiente:

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \{S \to 0S \mid 1A \mid \varepsilon, A \to 0S \mid 1B \mid \varepsilon, B \to 0B \mid 1B\}$
- $S = \{S\}$

Una vez tenemos el autómata en forma de gramática, el primer paso que he realizado es el de eliminar producciones inútiles, aquellas que no producen símbolo terminal y de variables inalcanzables. En este caso "B" no produce símbolo terminal, la eliminamos y quitamos aquellas producciones en las que aparezca, quedando por lo tanto la gramática de la siguiente forma:

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \{S \to 0S \mid 1A \mid \varepsilon, A \to 0S \mid \varepsilon\}$
- $S = \{S\}$

Una vez eliminada producciones inútiles, el siguiente paso es eliminar las producciones nulas, dejando solo los terminales, para ello simplemente cambiamos ε por los terminales en el caso de la primera producción por "0" y "1" y en el caso de la segunda producción por "0", quedando la gramática de la siguiente forma:

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \{S \to 0S \mid 1A \mid 0 \mid 1,\ A \to 0S \mid 0\}$
- $S = \{S\}$

Finalmente, el ultimo paso nos llevará a la forma normal de Chomsky, la cual nos dice que las producciones tienen que tener forma de $A \to BC$ o $A \to a$, luego lo que tenemos que hacer es cambiar cada terminal por una variable y donde se use este terminal poner la variable que hace referencia al terminal, por ejemplo, si tenemos el terminal "0" podemos cambiar este terminal por $X \to 0$ y luego donde usemos el terminal junto con una variable lo cambiamos por la variable, quedando entonces nuestra gramática finalmente como:

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \{X \to 0,\ Y \to 1,\ S \to XS \mid YA \mid 0 \mid 1,\ A \to XS \mid 0\}$
- $S = \{S\}$

4.2. Ejercicio 2

Observando las siguientes gramáticas, determinar cuáles de ellas son ambiguas y, en su caso, comprobar si los lenguajes generados son inherentemente ambiguos. Justificar la respuesta.

a. $S \to AbB, A \to aA|\varepsilon, B \to aB|bB|\varepsilon$

b. $S \to abaS|babS|baS|\varepsilon$

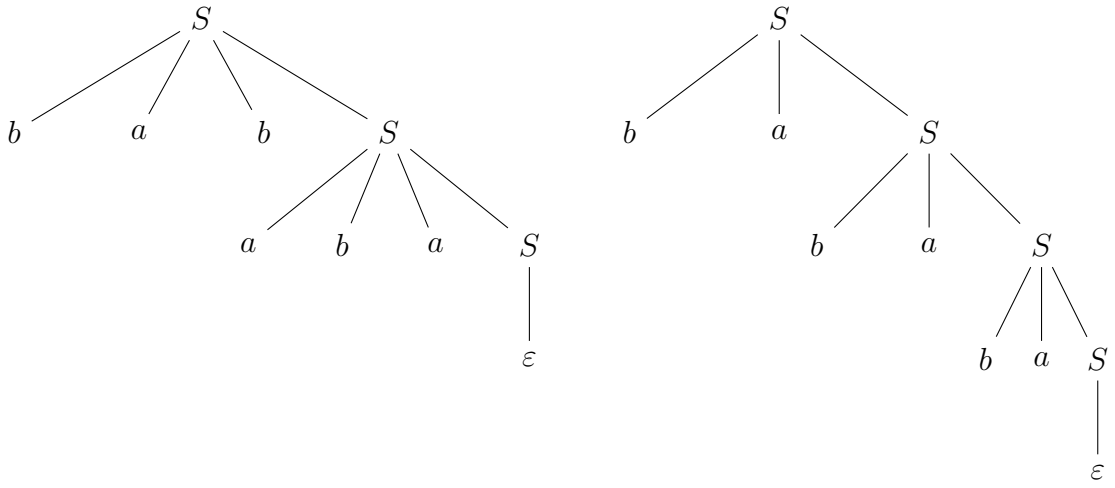
c. $S \to aSA|\varepsilon, A \to bA|\varepsilon$

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

Solución apartado a:
Para comprobar si una gramática es ambigua o no lo que voy a realizar es intentar buscar dos arboles diferentes que produzcan la misma palabra, para ello he intentado buscar para esta gramática que se cumpla esto que he citado, pero no he podido encontrar dos arboles distintos que formen la misma palabra, debido a que en la rama principal de la izquierda siempre se van a ir insertando "a" hasta que se inserta el símbolo vacío, en cambio, por la rama principal derecha se van a ir insertando "a" o "b" entrelazadas o no, es decir, ligadas, solo "a" o solo "b", luego de ahí que no podamos sacar dos arboles distintos que formen la misma palabra, esto es debido a que hay un claro procedimiento, siempre el mismo, para crear unos tipos de palabras muy concretas.

Debido a que hemos encontrado ya una gramática que no es ambigua podemos decir directamente que el lenguaje no es inherentemente ambiguo.

Solución apartado b:

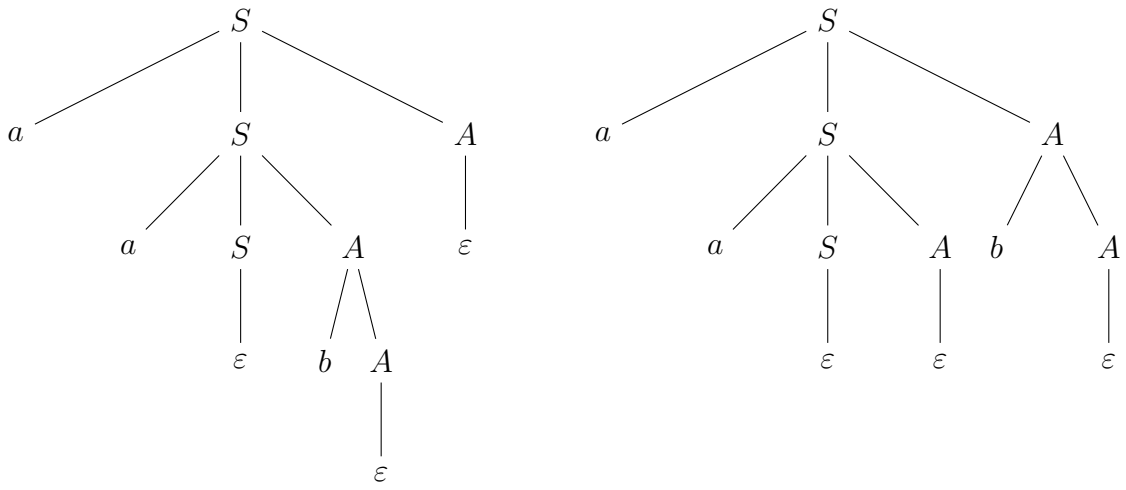


La gramática es ambigua debido a que con dos arboles totalmente distintos he podido llegar a una misma palabra, concretamente he conseguido llegar a la palabra "$bababa$" en ambos arboles. Para comprobar si el lenguaje es inherentemente ambiguo lo que realizado es la creación de otro lenguaje (modificación del que tenemos) que genere las mismas palabras pero buscando que no sea ambiguo, para así si encontramos un lenguaje que genere las mismas palabras pero que no sea ambiguo podremos decir que el lenguaje no será inherentemente ambiguo. He generado el siguiente lenguaje:

$$S \to abaS \mid baBS \mid \varepsilon,\ B \to b \mid \varepsilon$$

Con este lenguaje no he sido capaz de encontrar dos arboles diferentes para crear una misma palabra, debido a que siempre para generar "$bababa$" vamos a tener que usar la producción "B" ya que si solo usamos la primera producción no conseguimos llegar a la palabra que buscamos, luego podemos deducir que el lenguaje no es inherentemente ambiguo.

Solución apartado c:



La gramática es ambigua ya que he conseguido una misma palabra, concretamente "aab", con dos arboles completamente distintos. Para demostrar si el lenguaje es inherentemente ambiguo procedo a realizar lo mismo que en el apartado anterior, buscar una gramática que no sea ambigua para así poder saber si es o no inherentemente ambigua, dependiendo de la que genero. En mi caso he generado una gramática, la cual puede producir las mismas palabras que la gramática que nos da el problema, solo que concretamente esta no es ambigua, ya que no he encontrado dos arboles diferentes para una misma palabra. Destacar, que ha sido fácil generar dicha gramática, por que la inicial lo que hacia simplemente es insertar mínimo una "a", seguida de más "a" o no y luego de "b" o no. También podría generar palabras vacías. Como esta gramática es sencilla, he sido capaz de generar una nueva gramática que realice lo mismo pero no ambigua, la cual es la siguiente:

$$S \to aS \mid bA \mid \varepsilon,\ A \to bA \mid \varepsilon$$

Como esta gramática no es ambigua, directamente diremos que no es inherentemente ambigua.

4.3. Ejercicio 3

Encontrar el autómata que acepte el siguiente lenguaje L.

$$L = \{0^i 1^j 0^k 1 \mid i + k = j; i, j, k \in \mathbb{N}\}$$

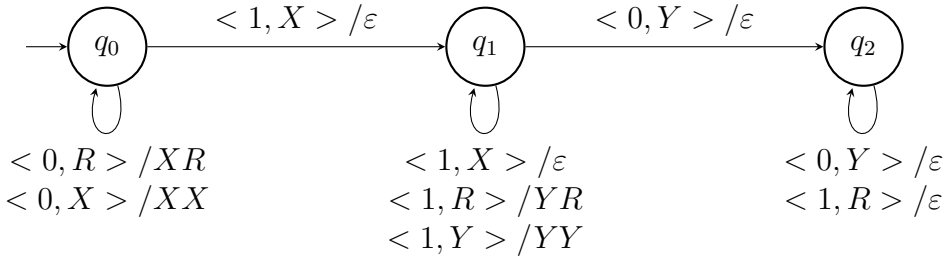
Una vez diseñado el autómata, calcular la gramática libre de contexto que acepta L eliminando posibles producciones inútiles que hayan ido apareciendo durante el proceso.

Para solucionar este problema he decidido realizar un Autómata con Pila ya que debemos recordar símbolos para poder comprobar la condición que se pide en la que $i + k = j$. Adelantar que i,j o k no puede ser 0 para que así siempre haya al menos un valor de cada del alfabeto de entrada. Este autómata está formado por la siguiente séxtupla:

- $Q = \{q_0, q_1, q_2\}$
- $A = \{0, 1\}$

- $B = \{R, X, Y\}$
- $q_0 = \{q_0\}$
- $Z_0 = \{R\}$
- $F = \{\emptyset\}$

El Autómata con Pila que he generado es el siguiente:



Lo primero que tengo que decir que la condición final de aceptación de las cadenas es la de la pila vacía. Lo que realiza este autómata es primero en el estado q_0 lo que realiza es que si llega un 0 y la pila está vacía, metemos una X en la pila y si ya había una X metemos otra, digamos que en q_0 vamos a ir metiendo tantos 0's como valor de "i" haya. Si estamos en q_0 y leemos un 1, pasamos al estado q_1 pero antes sacamos la ultima X que habrá en la pila. Estando en q_1 el autómata si le llega un 1 y en la pila hay X's las irá sacando de esta y en cuanto la pila quede vacía, es decir, en la pila solo está R se empiezan a meter tantas Y's como el valor de j que quede (el valor de j no es mas que el número de 1 que le llega por la cinta, pues en este momento leerá tantos 1's como 1's les llegue por la cinta). En cuanto, estando en q_1, nos llegue un 0, sacamos de la pila la ultima Y que estará en la pila y pasamos a q_2, en el cual se van a ir sacando las Y's de la pila si le van llegando 0's, en cuanto la pila se vacíe al completo y le llegue un 1 por la cinta se finalizará y se aceptará la cadena.

Lo que se realiza, en resumen, es primero cada 0 que llega se introduce una X en la pila, cuando se acabe de meter los 0's y se empiecen a meter 1's se van a ir sacando tantas X's de la pila como 1's lleguen hasta que la pila quede vacía y se empiecen a meter Y's en la pila. Cuando se acaben de leer los 1's de la cinta, se pasa a leer de nuevo 0's los cuales sacarán tantas Y's como 0's haya y en cuanto se lee el 1 del final y la pila esté vacía se aceptará la cadena.

La generación de la gramática y su simplificación no ha sido realizada por falta de tiempo.