

MODELOS DE COMPUTACIÓN  
GRADO EN INGENIERÍA INFORMÁTICA

---

# Memoria de prácticas

---

**Autor**

Vladislav Nikolov Vasilev

**Grupo**

3º A1

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

CURSO 2018-2019

---

## Contents

<b>1</b>	<b>Prácticas</b>	<b>2</b>
1.1	Práctica 1 . . . . .	2
1.1.1	Ejercicio 1 . . . . .	2
1.1.2	Ejercicio 2 . . . . .	3
1.1.3	Ejercicio 3 . . . . .	3
1.1.4	Ejercicio 4 . . . . .	5
1.2	Práctica 2 . . . . .	9
1.2.1	Descripción del problema . . . . .	9
1.2.2	Resolución del problema . . . . .	10
1.2.3	Pruebas . . . . .	14
1.3	Práctica 3 . . . . .	16
1.3.1	Ejercicio 1 . . . . .	16
1.3.2	Ejercicio 2 . . . . .	20
1.3.3	Ejercicio 3 . . . . .	25
1.3.4	Ejercicio 4 . . . . .	26
1.4	Práctica 4 . . . . .	29
1.4.1	Ejercicio 1 . . . . .	29
1.4.2	Ejercicio 2 . . . . .	29
1.4.3	Ejercicio 3 . . . . .	30
<b>2</b>	<b>Ejercicios voluntarios</b>	<b>31</b>

---

# 1 Prácticas

## 1.1 Práctica 1

### 1.1.1 Ejercicio 1

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

#### Solución

Se define la gramática como una cuádrupla con la forma  $G = (V, T, P, S)$ , siendo  $V$  el conjunto de variables,  $T$  el conjunto de elementos terminales,  $P$  las reglas de producción y  $S$  el símbolo inicial. Se puede definir cada uno de los conjuntos de la siguiente forma:

$$V = \{S, X, Y\}$$

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

$$P = \{S \rightarrow aXdd \mid bYc \mid \varepsilon, X \rightarrow aXdd \mid bYc \mid add \mid \varepsilon, Y \rightarrow bYc \mid bc \mid \varepsilon\}$$

$$S = \{S\}$$

Ésta es una gramática de **tipo 2**, ya que a la izquierda aparece una variable sola, sin símbolos terminales, y a la derecha aparece la variable con símbolos terminales tanto por la derecha como por la izquierda, impidiendo por tanto que sea regular por la izquierda o por la derecha.

Con ésta gramática, primero se escoge si se van a empezar a generar una  $a$  con la secuencia  $dd$  al final, o si directamente se comenzará a generar la secuencia  $b$  seguida de  $c$ . Si se escoge la primera opción, se ponen tantas  $a$  al principio y  $dd$  al final como sea necesario, y después se puede escoger si se sigue con las  $b$  y  $c$ , o si se termina sin poner ninguno de los símbolos anteriores. Si se decide comenzar a poner  $b$  y  $c$  desde el principio o después de poner todas las  $a$  y  $dd$  que se quieran, se ponen todas las  $b$  y  $c$  que se quieran, hasta que se decida terminar la secuencia.

Gracias a las reglas de producción se pueden satisfacer todas las restricciones del lenguaje, ya que por cada  $a$  se genera  $dd$ , y por cada  $b$  se genera  $c$ . Además, se puede aceptar la cadena vacía o que alguna de las partes no esté.

---

### 1.1.2 Ejercicio 2

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

#### Solución

La solución más sencilla que se puede ofrecer a este problema consiste en utilizar una gramática libre de contexto, como se mostrará a continuación. No obstante, el problema también es resoluble mediante una gramática regular, aumentando sin embargo el número de producciones y de variables necesarias.

Definimos la gramática como una cuádrupla con la forma  $G = (V, T, P, S)$ , siendo  $V$  el conjunto de variables,  $T$  el conjunto de elementos terminales,  $P$  las reglas de producción y  $S$  el símbolo inicial. Se puede definir cada uno de los conjuntos de la siguiente forma:

$$\begin{aligned} V &= \{S, X\} \\ T &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -\} \\ P &= \left\{ \begin{array}{c} S \rightarrow +X.X \mid -X.X \\ X \rightarrow 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \mid 7X \mid 8X \mid 9X \mid \\ \quad 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array} \right\} \\ S &= \{S\} \end{aligned}$$

Primero se genera el signo y el punto, pudiendo escoger si el número es positivo o negativo. Después, en la parte entera y decimal se van generando números en el rango  $[0, 9]$ , pudiendo escoger cuál es el siguiente número o cuando terminar de insertar números.

### 1.1.3 Ejercicio 3

*Enunciado.* Calcular una gramática libre de contexto que genere el lenguaje  $L = \{0^i 1^j 2^k \mid \text{tal que } i \neq j \text{ o } j \neq k\}$ .

#### Solución

Definimos la gramática como una cuádrupla con la forma  $G = (V, T, P, S)$ , siendo  $V$  el conjunto de variables,  $T$  el conjunto de elementos terminales,  $P$  las reglas de producción y  $S$  el símbolo inicial. Se puede definir cada uno de los conjuntos de la siguiente forma:

---


$$\begin{aligned}
V &= \{S, X, Y, Z, P, R, A, B, M, K, U, D, N, L, C, Q\} \\
T &= \{0, 1, 2\} \\
P &= \left\{ \begin{array}{l} S \rightarrow 0X1 \mid 0Y2 \mid 1Z2 \mid 0A1P2 \mid 0R1B2, \ X \rightarrow 0X \mid X1 \mid \varepsilon \\ Y \rightarrow 0Y \mid Y2 \mid \varepsilon, \ Z \rightarrow 1Z \mid Z2 \mid \varepsilon, \ P \rightarrow P2 \mid \varepsilon, \ R \rightarrow 0R \mid \varepsilon, \\ A \rightarrow 0M \mid N1, \ M \rightarrow OMU \mid \varepsilon, \ U \rightarrow 1 \mid \varepsilon, \ N \rightarrow CN1 \mid \varepsilon, \ C \rightarrow 0 \mid \varepsilon, \\ B \rightarrow 1K \mid L2, \ K \rightarrow 1KD \mid \varepsilon, \ D \rightarrow 2 \mid \varepsilon, \ L \rightarrow QL2 \mid \varepsilon, \ Q \rightarrow 1 \mid \varepsilon \end{array} \right\} \\
S &= \{S\}
\end{aligned}$$

Ya que hay muchas reglas y puede no llegar a quedar claro para qué es cada una, vamos a ir comentándolas para que no queden dudas sobre el por qué de cada una de ellas.

La primera de ellas,  $S \rightarrow 0X1$ , indica que solo se van a producir los símbolos 0 y 1, dándose por tanto la condición  $j \neq k$ , ya que no hay ningún símbolo 2.  $X$  puede ser sustituido por tantos 0 o 1 como se desee, lo cuál corresponde a la producción  $X \rightarrow 0X \mid X1 \mid \varepsilon$ .

Después tenemos la producción  $S \rightarrow 0Y2$ , la cuál es parecida a la anterior, solo que esta vez se producen solo los símbolos 0 y 2, satisfaciendo por tanto las condiciones  $i \neq j$  y  $j \neq k$  simultáneamente. La variable  $Y$  puede ser sustituida por tantos 0 o 2 como se desee, lo cuál se corresponde con la producción  $Y \rightarrow 0Y \mid Y2 \mid \varepsilon$ .

La regla  $S \rightarrow 1Z2$  permite producir los símbolos 1 y 2. En este caso, esta regla permite satisfacer la restricción  $i \neq j$ , ya que no se produce ningún símbolo 0. La variable  $Z$  puede ser sustituida por tantos 1 y 2 como se desee. Ésto se corresponde con la producción  $Z \rightarrow 1Z \mid Z2 \mid \varepsilon$ .

La regla  $S \rightarrow 0A1P2$  permite producir los símbolos 0, 1 y 2, cumpliendo sin embargo la restricción  $i \neq j$ , introduciendo la desigualdad por tanto en la parte de los 0 y los 1, es decir, obligando que el número de 0 y de 1 sea diferente y permitiendo producir tantos 2 como se desee. La variable  $A$  se puede sustituir con la regla  $A \rightarrow 0M \mid N1$ , escogiendo si se quieren más 0 que 1 (se escogería  $0M$ ) o más 1 que 0 (se escogería en este caso  $N1$ ). Al haber escogido estas reglas, se asegura que como mínimo hay un símbolo más de ese tipo. La variable  $M$  puede ser sustituida por  $M \rightarrow OMU \mid \varepsilon$ , permitiendo poner tantos 0 como se deseen y poniendo por cada uno una variable  $U$ , la cuál puede ser sustituida luego por  $U \rightarrow 1 \mid \varepsilon$ , poniendo o no tantos 1 como  $U$  haya. Hay que tener en cuenta que el número de 1 será siempre menor que el número de 0, ya que al principio, con  $A \rightarrow 0M$  se puso un 0 extra, y como la regla  $M \rightarrow OMU$  produce una variable que pueda ser sustituida por 1 por cada 0 nuevo que se coloca, se asegura que se cumplirá la desigualdad  $i \neq j$  como se mencionó anteriormente, siendo en este caso

---

$i > j$  ya que  $num(1) \leq num(0) - 1$ . Algo similar ocurre con la regla  $A \rightarrow N1$ , ya que permite producir más 1 que 0 de la misma forma que antes. Primero se introduce un 1 extra y después se sustituye la variable  $N$  por  $N \rightarrow CN1 \mid \varepsilon$ , permitiendo poner tantos 1 como se deseen y permitiendo poner un 0 por cada nuevo 1 que se añade (lo cuál se corresponde a la regla  $C \rightarrow 0 \mid \varepsilon$ ). Aquí ocurre lo mismo que en el caso anterior, ya que cumplirá la restricción  $i \neq j$  verificando que  $i < j$ , debido a que  $num(0) \leq num(1) - 1$ .

Finalmente tenemos la regla  $S \rightarrow 0R1B2$ , la cuál es similar a la anterior mencionada debido a que permite producir los símbolos 0, 1 y 2, satisfaciendo sin embargo la desigualdad  $j \neq k$ , lo cuál significa que se producen tantos 0 como se deseen, pero el número de 1 y de 2 tiene que ser distinto. La variable  $R$  puede ser sustituida por  $R \rightarrow 0R \mid \varepsilon$ , es decir, por tantos 0 como se desee. La variable  $B$  puede ser sustituida por  $B \rightarrow 1K \mid L2$ , permitiendo en el primer caso que haya más 1 que 2, y en el segundo caso que haya más 2 que 1. La variable  $K$  puede ser sustituida por  $K \rightarrow 1KD \mid \varepsilon$ , poniendo un símbolo 1, después otra variable  $K$  y finalmente una variable  $D$ , la cuál puede ser sustituida mediante la regla  $D \rightarrow 2 \mid \varepsilon$ , poniendo como mucho tantos 2 como variables  $D$  haya. Con esto se cumple la desigualdad  $j \neq k$  ya que se ha puesto un 1 extra al principio, de forma que  $j > k$  y  $num(2) \leq num(1) - 1$ . En el caso de querer más 2 que 1, se escogería  $B \rightarrow L2$ , sustituyendo luego la variable  $L$  por  $L \rightarrow QL2 \mid \varepsilon$ , poniendo una variable  $Q$ , una variable  $L$  y un 2. La variable  $Q$  sería sustituida luego mediante la regla  $Q \rightarrow 1 \mid \varepsilon$ , permitiendo poner un o ningún 1 por cada variable  $Q$ . En este caso se cumpliría que  $j \neq k$  ya que  $j < k$  porque se cumple que  $num(1) \leq num(2) - 1$ .

#### 1.1.4 Ejercicio 4

*Enunciado.* 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 ( $x$ ).

Por ejemplo, la cadena terminal “*gfdddddfffpdddfx*” representa que el nivel tiene (*gfdddddfff*) un grupo grande con un monstruo fuerte, 4 débiles y otros 3 fuertes; después tiene (*pddddf*) un grupo pequeño con 3 débiles y uno fuerte.

---

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?

### Solución

Definimos la gramática como una cuádrupla con la forma  $G = (V, T, P, S)$ , siendo  $V$  el conjunto de variables,  $T$  el conjunto de elementos terminales,  $P$  las reglas de producción y  $S$  el símbolo inicial. Se puede definir cada uno de los conjuntos de la siguiente forma:

$$V = \{S, G, X, Y, Z, V, R, P, B\}$$

$$T = \{g, p, f, d, x\}$$

$$P = \left\{ \begin{array}{l} S \rightarrow gGx \mid pPx, G \rightarrow ddX \mid dfY \mid fdY \mid ffZ, X \rightarrow dX \mid fR, \\ Y \rightarrow X \mid V \mid R \mid gG \mid pP \mid \varepsilon, Z \rightarrow dX \mid dV, V \rightarrow dV \mid gG \mid pP \mid \varepsilon, \\ R \rightarrow fR \mid gG \mid pP \mid \varepsilon, P \rightarrow dP \mid dB \mid fB, B \rightarrow dB \mid gG \mid pP \mid \varepsilon \end{array} \right\}$$

$$S = \{S\}$$

La gramática es de **tipo 2**, ya que a la izquierda aparecen solo variables y a la derecha aparecen variables con símbolos terminales tanto por la derecha como por la izquierda o sin símbolos terminales, impidiendo por tanto que sea regular, tanto por la derecha o regular por la izquierda.

Una vez dicho esto, se va a proceder a explicar cómo funcionan las reglas de producción. Con  $S \rightarrow gGx \mid pPx$  se escoge con qué grupo empezar primero: si uno grande ( $gGx$ ) o uno pequeño ( $pPx$ ).

Si se escoge empezar con un grupo grande, como da igual en qué orden están los dos primeros enemigos, se puede sustituir la variable  $G$  mediante la regla  $G \rightarrow ddX \mid dfY \mid fdY \mid ffZ$ . Si con la regla anterior se han producido dos enemigos débiles, se sustituye la variable  $X$  mediante la regla  $X \rightarrow dX \mid fR$ , que permite poner primero todos los enemigos débiles que se quieran y poner uno fuerte al final. Después de poner el fuerte, la variable  $R$  se puede sustituir mediante la regla  $R \rightarrow fR \mid gG \mid pP \mid \varepsilon$ , que permite poner tantos enemigos fuertes como se quieran, y después un grupo grande, uno pequeño o terminar de poner grupos de enemigos. Si por el contrario, al principio del grupo grande se escogen poner dos enemigos fuertes, entonces se tiene que poner como mínimo un enemigo débil. Esto se hace a través de la variable  $Z$ , que se sustituye mediante la regla  $Z \rightarrow dX \mid dV$ , que pone un enemigo débil y permite escoger si seguir con la variable  $X$  para poner enemigos débiles o fuertes, o seguir con la variable  $V$ , que se sustituye mediante

---

la regla  $V \rightarrow dV \mid gG \mid pP \mid \varepsilon$  y permite poner tantos enemigos débiles como se quieran o poner un nuevo grupo grande, pequeño o terminar de poner grupos. Si por el contrario se escoge poner un enemigo fuerte y uno débil, como ya se cumple la restricción del grupo grande, se puede sustituir la variable  $Y$  mediante la regla  $Y \rightarrow X \mid V \mid R \mid gG \mid pP \mid \varepsilon$ , que permite poner enemigos fuertes y débiles, solo fuertes, solo débiles, poner un grupo grande, uno pequeño o terminar de insertar grupos.

Si en cambio se escoge empezar con un grupo pequeño, la variable  $P$  puede ser sustituida mediante la regla  $P \rightarrow dP \mid dB \mid fB$ , que permite poner tantos enemigos débiles como se quieran y se puede escoger si se quiere un enemigo fuerte, o si por el contrario solo se van a producir débiles. En todo caso, si se produce un enemigo fuerte o si no se decide producir se escoge el camino de la variable  $B$ , la cuál puede ser sustituida con la regla  $B \rightarrow dB \mid gG \mid pP \mid \varepsilon$ , permitiendo de nuevo poner cuantos enemigos débiles se desee, y después poner un grupo grande, uno pequeño o terminar de insertar grupos de enemigos.

Como se puede comprobar, estas reglas permiten crear niveles de forma flexible, ya que se pueden combinar los grupos grandes y los pequeños en el orden que se quiera. Además, también permite generar los enemigos de una forma versátil, permitiendo muchas combinaciones posibles.

Respecto a la segunda pregunta, es posible diseñar una gramática de tipo 3 para este problema. Esto se debe al hecho de que, aunque la gramática obtenida inicialmente sea de tipo de 2, no se garantiza que el lenguaje sea de tipo 2, si no que también puede ser de tipo 3. Para ello, definimos la gramática como una cuádrupla  $G = (V, T, P, S)$ , donde  $V$  son las variables,  $T$  los símbolos terminales,  $P$  las reglas de producción y  $S$  el símbolo inicial. Cada uno de los conjuntos tendría la siguiente forma:

$$\begin{aligned}
 V &= \{S, G, X, Y, Z, V, R, P, B\} \\
 T &= \{g, p, f, d, x\} \\
 P &= \left\{ \begin{array}{l} S \rightarrow gG \mid pP, G \rightarrow ddX \mid dfY \mid fdY \mid ffZ, X \rightarrow dX \mid fR, \\ Y \rightarrow dX \mid dV \mid fR \mid gG \mid pP \mid x, Z \rightarrow dX \mid dV, V \rightarrow dV \mid gG \mid pP \mid x, \\ R \rightarrow fR \mid gG \mid pP \mid x, P \rightarrow dP \mid dB \mid fB, B \rightarrow dB \mid gG \mid pP \mid x \end{array} \right\} \\
 S &= \{S\}
 \end{aligned}$$

Como se puede comprobar fácilmente, esta gramática es de **tipo 3**, ya que a la izquierda aparece la variable sola y a la derecha aparece, o bien un símbolo terminal, o bien uno o más símbolos terminales acompañados de una variable a la derecha. Por tanto, se trata de una gramática regular por la derecha.



---

Se puede comprobar fácilmente que esta gramática produce las mismas palabras que la anterior. Las diferencias son que el símbolo de recompensa de nivel  $x$  se genera cuando no se quieren generar más grupos de enemigos en vez de al principio como se hacía antes. Esto también implica que todos los  $\varepsilon$  se han sustituido por el símbolo  $x$ . Adicionalmente, para que el lenguaje fuese regular por la derecha, a las producciones de  $Y$  que anteriormente solo implicaban un cambio de variable se les ha añadido un símbolo terminal que además cumple las restricciones impuestas por el problema (una  $d$  para las variables  $X$  y  $V$  y una  $f$  para la variable  $R$ ).

---

## 1.2 Práctica 2

Esta práctica ha sido realizada con mi compañera Nazaret Román Guerrero. Aquí se incluye la descripción y solución del problema para ambos alumnos.

### 1.2.1 Descripción del problema

El problema que se ha decidido abordar consiste en la creación de un programa capaz de traducir el lenguaje natural en código ejecutable en Python. Ya que de por sí el problema sería demasiado grande y complejo, lo hemos restringido a crear un traductor que permite convertir expresiones simples relacionadas con el manejo de listas en lenguaje natural a Python.

Las funcionalidades que hemos decidido implementar son:

- Creación de listas, tanto vacías como con elementos.
- Inserción de elementos en las listas.
- Borrado de elementos de una lista.
- Obtención de un elemento en una posición de una lista.
- Imprimir una lista.
- Recorrer una lista, usando el delimitador por defecto o escogiendo uno que se desee.
- Obtener la longitud de una lista.
- Ordenar una lista, permitiendo que se ordene al revés.
- Copiar una lista en otra.
- Concatenar dos listas en una nueva o una ya existente.
- Comparar listas, permitiendo poner como expresión sumas de  $n$  listas, la longitud de una lista, obtener el elemento de una lista, ordenar listas o comparar directamente listas entre sí. Los operadores que soporta la traducción son igual ( $==$ ), diferente ( $!=$ ), menor ( $<$ ), mayor ( $>$ ), menor o igual ( $\leq$ ) y mayor o igual ( $\geq$ ).
- Mostrar el resultado de la comparación.

Los elementos que se pueden insertar en listas son números enteros y reales (los reales tienen una parte entera separada por el símbolo gráfico . de la parte decimal) y cadenas de caracteres (encerradas entre comillas simples, con espacios y sin restricciones de longitud).

### 1.2.2 Resolución del problema

Para resolver el problema hemos utilizado la herramienta *Flex*. Hemos creado un programa escrito en C que permite procesar símbolos de entrada y obtener la traducción en Python correspondiente.

Se procede a mostrar ahora el código:

```

1  /* ----- Declaraciones ----- */
2
3  %option noyywrap
4  %{
5  #include <stdio.h>
6
7  char * procesado;
8  char * elem;
9  int i = 0;
10 int j = 0;
11 int comparar = 0;
12 char * comparacion;
13 %}
14
15 letra      [a-zA-Z]
16 digito     [0-9]
17 espacio    [ ]
18 entero     \-?{digito}+
19 numero     {entero}(\.{digito}+)?
20 delimitador  ""[^\t\n]+""
21 cadena     ""({letra}|{digito}|{espacio})*""
22 variable    ({letra}|{digito}|_)+
23 crear      "crear" {variable}(" ({numero}|{cadena}|{espacio})+)?
24 longitud   "longitud" {variable}
25 imprimir   "imprimir" {variable}
26 recorrer   "recorrer" {variable}(" {delimitador})?
27 insertar    "insertar" {variable}" "({cadena}|{numero})
28 borrar      "borrar" {variable}" "({cadena}|{numero})
29 obtener     "obtener" {variable}" " {entero}
30 copiar      "copiar" {variable}" " {variable}
31 concatenar   "concatenar" {variable}" " {variable}" " {variable}
32 ordenar     "ordenar" {variable}" reves"?
33 suma        {variable}" mas " {variable}(" mas " {variable})*
34 operador    "igual"|"diferente"|"menor"|"menor igual"|"mayor"|"mayor
              igual"
35 expresion    {longitud}|{suma}|{obtener}|{variable}|{ordenar}
36 comparar     "comparar" {expresion}" " {operador}" " {expresion}

```

---

```

37
38 %%
39
40 {crear} {procesado = yytext + 6;
41         char * lista = malloc(strlen(procesado));
42         elem = malloc(strlen(procesado) * 2);
43         for (i=0; i<strlen(procesado) && procesado[i] != ' '; i++)
44             lista[i] = procesado[i];
45         if (i < strlen(procesado)) {
46             procesado += i + 1;
47             j = 0;
48             int cambiar_espacio_coma = 0;
49             for (i = 0; i < strlen(procesado); i++) {
50                 if (procesado[i] == ',') {
51                     if (cambiar_espacio_coma != 0) {
52                         elem[j] = ',';
53                         j++;
54                         elem[j] = ' ';
55                         j++;
56                         cambiar_espacio_coma = 0;
57                     }
58                 } else {
59                     elem[j] = procesado[i];
60                     j++;
61                     cambiar_espacio_coma = 1;
62                 }
63             }
64         }
65         printf("%s = [%s]", lista, elem);}
66 {longitud} {procesado = yytext + 9;
67             if (comparar > 0)
68                 comparar--;
69             printf("len(%s)", procesado);}
70 {imprimir} {procesado=yytext + 9; printf("print(%s)", procesado);}
71 {recorrer} {procesado = yytext + 9;
72             char * lista = malloc(strlen(procesado));
73             char * delim;
74             int usar_delim = 0;
75             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
76                 lista[i] = procesado[i];
77             if (strlen(procesado) >= i + 1)
78                 usar_delim = 1;
79             procesado += i + 1;
80             delim = procesado;
81             printf("for item in %s:\n    print(item", lista);
82             if (usar_delim)
83                 printf(", end = %s", delim);
84             printf(")");}
85 {insertar} {procesado = yytext + 9;
86             char * lista = malloc(strlen(procesado));
87             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
88                 lista[i] = procesado[i];
89             procesado += i + 1;

```

---

---

```

90         elem = procesado;
91         printf("%s.append(%s)", lista, elem);}
92 {borrar}    {procesado = yytext + 7;
93             char *lista = malloc(strlen(procesado));
94             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
95                 lista[i] = procesado[i];
96             procesado += i + 1;
97             elem = procesado;
98             printf("%s.remove(%s)", lista, elem);}
99 {obtener}   {procesado = yytext; procesado += 8;
100             char *lista = malloc(strlen(procesado));
101             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
102                 lista[i] = procesado[i];
103             procesado += i + 1;
104             elem = procesado;
105             if(comparar > 0) {
106                 printf("%s[%s]", lista, elem);
107                 comparar--;
108             }
109             else
110                 printf("print(%s[%s])", lista, elem);}
111 {copiar}    {procesado = yytext + 7;
112             char *orig = malloc(strlen(procesado));
113             char *dest = malloc(strlen(procesado));
114             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
115                 orig[i] = procesado[i];
116             procesado += i + 1;
117             dest = procesado;
118             printf("%s = %s", dest, orig);}
119 {concatenar} {procesado = yytext + 11;
120             char *l1 = malloc(strlen(procesado));
121             char *l2 = malloc(strlen(procesado));
122             char *dest = malloc(strlen(procesado));
123             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
124                 l1[i] = procesado[i];
125             procesado += i + 1;
126             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
127                 l2[i] = procesado[i];
128             procesado += i + 1;
129             dest = procesado;
130             printf("%s = %s + %s", dest, l1, l2);}
131 {ordenar}   {procesado = yytext + 8;
132             char *lista = malloc(strlen(procesado));
133             if (comparar > 0)
134                 comparar--;
135             for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
136                 lista[i] = procesado[i];
137             if (strlen(procesado) >= i + 1)
138                 printf("%s.sort(reverse=True)", lista);
139             else
140                 printf("%s.sort()", lista);}
141 {suma}      {procesado = yytext;
142             char * suma = malloc(strlen(procesado));

```

---

---

```

143         int salta_palabra = 0;
144         j = 0;
145         if (comparar > 0)
146             comparar--;
147         for(i=0; i<strlen(procesado); i++) {
148             if (procesado[i] == ' ') {
149                 salta_palabra = !salta_palabra & 0x1;
150                 if(salta_palabra) {
151                     suma[j] = ' ';
152                     j++;
153                     suma[j] = '+';
154                     j++;
155                     suma[j] = ' ';
156                     j++;
157                 }
158             } else {
159                 if (!salta_palabra) {
160                     suma[j] = procesado[i];
161                     j++;
162                 }
163             }
164         }
165         printf("%s", suma);
166     {comparar} {comparar = 2; printf("comparacion = "); yyles(9);}
167     {operador} {procesado = yytext;
168                 i = 0;
169                 if(procesado[i] == 'i')
170                     printf(" == ");
171                 else if(procesado[i] == 'd')
172                     printf(" != ");
173                 else if(procesado[i] == 'm') {
174                     if(procesado[i+1] == 'e' && yyleng == 5)
175                         printf(" < ");
176                     else if(procesado[i+1] == 'e' && yyleng > 5)
177                         printf(" <= ");
178                     else if(procesado[i+1] == 'a' && yyleng == 5)
179                         printf(" > ");
180                     else
181                         printf(" >= ");}
182     {variable} {if (comparar > 0)
183                 comparar--;
184                 printf("%s", yytext);}
185     "mostrar comparacion" {printf("print(comparacion)");}
186     . {}
187
188 %%
189
190 int main(int argc, char *argv[]) {
191
192     if (argc == 2) {
193         yyin = fopen(argv[1], "rt");
194
195         if (yyin == NULL) {

```

---

```

196     printf("El fichero %s no se pudo abrir\n", argv[1]);
197     exit(-1);
198 }
199 } else
200     yyin = stdin;
201
202 yylex();
203
204 return 0;
205 }
206

```

Cabe mencionar que a la hora de crear una lista, si se deciden insertar cadenas de caracteres, por la forma en la que está hecho el código, éstas no pueden contener espacios, ya que si no serían sustituidas por espacios. Cuando se inserten de otra forma sí que pueden contenerlos.

### 1.2.3 Pruebas

Para probar el funcionamiento del programa, vamos a pasarle un fichero de texto que contiene expresiones en lenguaje natural que deberán ser convertidas a Python. Aquí se puede ver un ejemplo de la salida obtenida:

```

nazar@nazar-G663-780:~/Escritorio/ETSIIIT_comp/3/NC/P2/ugr_modelos_computacion/src/P2$ cat ejemplo_completo.txt
crear lista 'esto' 'es' 'la' 'primera' 'lista'
copiar lista lista2
imprimir lista2
recorrer lista2
comparar lista igual lista2
mostrar comparacion
crear lista 1 2 3 4
crear lista 4 3 2 1
concatenar lista lista2 lista_larga
ordenar lista_larga revers
recorrer lista_larga
comparar lista menor lista4
mostrar comparacion
comparar lista diferente lista4
mostrar comparacion
comparar lista igual ordenar lista
mostrar comparacion
longitud lista_larga
comparar lista mas lista5 igual lista5 mas lista_larga
mostrar comparacion
comparar longitud lista igual longitud lista2
mostrar comparacion
comparar obtener lista 0 diferente obtener lista 2
mostrar comparacion
obtener lista_larga -1
crear lista_nueva
insertar lista_nueva 1.1
insertar lista_nueva -123.45
insertar lista_nueva 15
imprimir lista_nueva
borrar lista_nueva 15
imprimir lista_nueva
nazar@nazar-G663-780:~/Escritorio/ETSIIIT_comp/3/NC/P2/ugr_modelos_computacion/src/P2$
nazar@nazar-G663-780:~/Escritorio/ETSIIIT_comp/3/NC/P2/ugr_modelos_computacion/src/P2$ ./compylex.py ejemplo_completo.txt
lista = ['esto', 'es', 'la', 'primera', 'lista']
lista2 = []
print(lista2)
for item in lista:
    print(item)
for item in lista2:
    print(item, end = ', ')
comparacion = lista == lista2
print(comparacion)
lista = [1, 2, 3, 4]
lista2 = [4, 3, 2, 1]
lista_larga = lista + lista4
lista_larga.sort(reverse=True)
for item in lista_larga:
    print(item, end = ', ')
comparacion = lista == lista4
print(comparacion)
comparacion = lista3 == lista4
print(comparacion)
comparacion = lista3 == lista4.sort()
print(comparacion)
len(lista_larga)
lista5 = [5, 6]
comparacion = lista3 + lista5 == lista4 + lista5 + lista_larga
print(comparacion)
comparacion = len(lista1) == len(lista2)
print(comparacion)
comparacion = lista[0] != lista[2]
print(comparacion)
print(lista_larga[-1])
lista_nueva = []
lista_nueva.append(1.1)
lista_nueva.append(-123.45)
lista_nueva.append(15)
print(lista_nueva)
lista_nueva.remove(15)
print(lista_nueva)
nazar@nazar-G663-780:~/Escritorio/ETSIIIT_comp/3/NC/P2/ugr_modelos_computacion/src/P2$

```

Para comprobar que funciona, vamos a redirigir la salida a un fichero con extensión .py y vamos a probar a ejecutarlo con Python. Se puede ver el resultado en la siguiente imagen:





---

## 1.3 Práctica 3

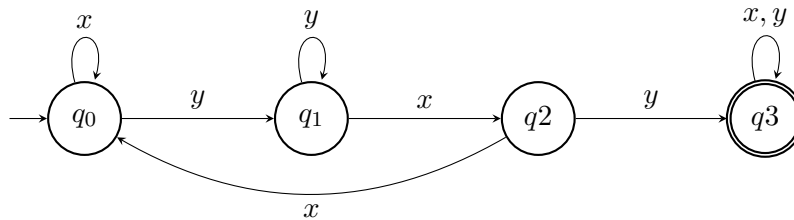
### 1.3.1 Ejercicio 1

*Enunciado.* En el alfabeto  $\{x, y\}$ , construir un AFD que acepte cada uno de los siguientes lenguajes:

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

#### Solución

a) Definimos como Autómata Finito Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final. El autómata quedaría de la siguiente forma:

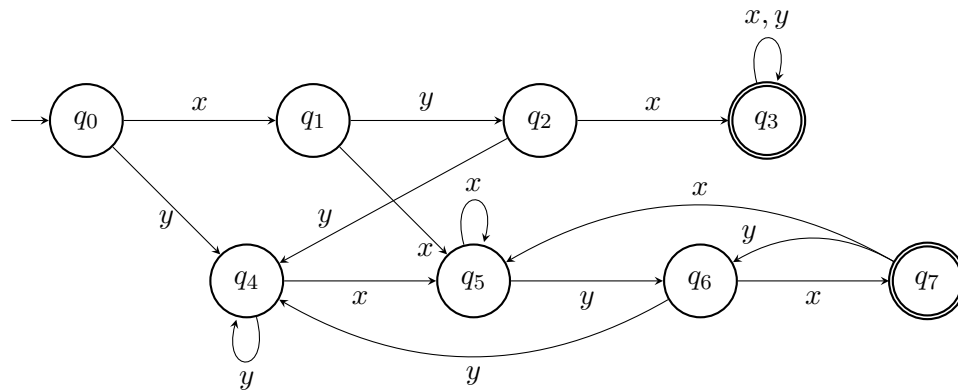


Cada estado representa lo siguiente:

- $q_0$  representa el estado inicial, y se continúa en él mientras le lleguen  $x$ , lo cual significa que todavía no le ha llegado el primer elemento de la subcadena que debe contener.
- $q_1$  representa que ha llegado la primera  $y$ , y se mantiene en ese estado mientras sigan llegando  $y$ , ya que simboliza que al menos habrá una que forme parte de la cadena  $xyx$ . Se pasará al estado  $q_2$  cuando le llegue una  $x$ , que representa el elemento de en medio.

- $q_2$  representa que ha llegado la  $x$  que forma la parte central de la cadena a aceptar. Si le llega una  $x$  vuelve a  $q_0$ , ya que se ha interrumpido la cadena. Si le llega una  $y$  significa que le ha llegado el último elemento de la cadena a aceptar y pasa al estado  $q_3$ .
- $q_3$  representa el estado final, al que se llega una vez que se han encontrado todos los símbolos consecutivos de la cadena. Se permanece en este estado le llegue lo que le llegue, ya que se ha encontrado la cadena que se buscaba. Si se llega a este estado, se acepta la cadena de entrada.

b) Definimos como Autómata Finito Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final. El autómata quedaría de la siguiente forma:



Antes de explicar lo que representa cada estado, hace falta clarificar algunas ideas sobre el autómata. La parte de arriba del autómata representa el autómata capaz de aceptar las palabras que contienen la cadena  $xyx$  al principio, mientras que la parte de abajo del autómata es la que es capaz de aceptar las palabras que contienen la cadena anterior al final de la palabra. Se aceptará una palabra si el autómata se encuentra en el estado  $q_3$  o en el  $q_7$  tras leer todos los símbolos de entrada.

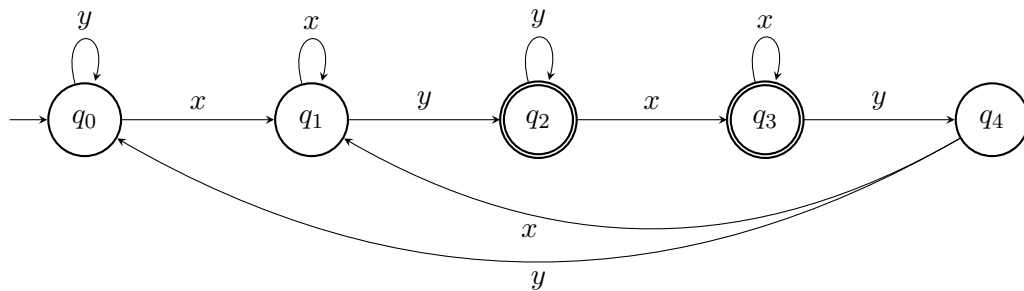
Una vez dicho esto, se va a proceder con la explicación:

- $q_0$  es el estado inicial. De aquí se pueden dar dos situaciones. La primera es que le llegue una  $x$ , lo cuál podría representar que la cadena  $xyx$  se encuentra al principio de la palabra, y entonces se pasaría al estado  $q_1$ . En caso de que llegue una  $y$ , se descarta que la cadena se encuentre al principio de la palabra y se pasa al estado  $q_4$  para intentar comprobar que se encuentra al final de la palabra.

- 
- $q_1$  representa que ha llegado la primera  $x$  y que se está comprobando que la cadena esté al principio. En caso de que le llegue una  $y$  se pasará al estado  $q_2$ , esperando por tanto a que le llegue el último elemento. En caso de que le llegue una  $x$ , significa que la cadena a buscar no se pudo encontrar al principio de la palabra y por tanto se intentará buscar al final de ésta. Por tanto, se pasará al estado  $q_5$ , simbolizando que ha llegado la primera  $x$  que puede estar al final de la palabra.
  - $q_2$  representa que han llegado los símbolos  $xy$  al principio de la cadena y está esperando a que llegue la última  $x$ . En caso de que esto ocurra, se pasará al estado  $q_3$ . En caso de que llegue una  $y$  significaría que se está interrumpiendo la cadena al inicio y por tanto se pasaría al estado  $q_4$ , indicando que se intentará comprobar que la cadena está al final de la palabra, y que todavía no ha llegado el primer elemento de ésta.
  - $q_3$  es un estado final que indica que se ha encontrado la cadena  $xyx$  al principio de la palabra, y que por tanto se acepta la palabra. Ya que se ha cumplido la restricción, el autómata permanecerá en este estado sin importar que símbolo le venga de entrada.
  - $q_4$  indica que, o bien el primer símbolo no ha sido una  $x$  o que la cadena  $xyx$  que se buscaba al principio o al final ha sido interrumpida con una  $y$  (por ejemplo con la cadena  $xyy$ ), y que por tanto aun no ha llegado el primer elemento de la cadena a buscar. El autómata permanecerá en este estado mientras le siga llegando el símbolo  $y$ , y pasará al estado  $q_5$  en cuanto le llegue una  $x$ , que simboliza el primer elemento de la cadena.
  - $q_5$  representa un estado en el que se está buscando la cadena  $xyx$  al final de la palabra y que ha llegado el primer elemento, bien porque la cadena había sido interrumpida por una  $y$ , como se ha mencionado en el estado anterior, o bien porque al principio ha llegado una cadena del tipo  $xx$ . Mientras le sigan llegando  $x$ , el autómata permanecerá en este estado, ya que este es el primer elemento de la cadena. Pasará a  $q_6$  en cuánto le llegue el símbolo  $y$ .
  - $q_6$  simboliza que se está buscando la cadena al final de la palabra y que de momento han llegado los símbolos  $xy$ . Si le llega el símbolo  $y$ , entonces se interrumpe la cadena, y se pasa al estado  $q_4$ . En caso de que llegue una  $x$ , el autómata pasará al estado  $q_7$ .
  - $q_7$  representa es un estado final que indica que se ha encontrado la cadena  $xyx$  al final de una palabra, probablemente. El autómata se quedará en este estado, y por tanto aceptará la palabra en el caso de que no le llegue ningún símbolo más. En caso de que le llegue una  $x$ , pasará al estado  $q_5$ , indicando que ha llegado el primer elemento de la cadena de nuevo. En caso de que le llegue una  $y$ , pasará al estado  $q_6$ , indicando que de momento lleva la cadena

$xy$ . Esto se debe a que anteriormente le llegó el símbolo  $x$ , con lo cuál, ahora con una  $y$ , lleva dos elementos de la cadena.

c) Definimos como Autómata Finito Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final. El autómata quedaría de la siguiente forma:



Se va ahora a comentar cada estado:

- $q_0$  representa el estado inicial. El autómata se mantendrá en este estado mientras le sigan llegando símbolos  $y$ . Pasará al siguiente estado,  $q_1$  cuando le llegue el símbolo  $x$ , el cuál forma parte de la cadena  $xy$  a buscar.
- $q_1$  representa ha llegado el símbolo  $x$  de la cadena  $xy$ . Se mantendrá en este estado mientras le sigan llegando símbolos  $x$ , ya que este sigue siendo el primer elemento de la cadena. Pasará al siguiente estado,  $q_2$ , en cuanto le llegue el símbolo  $y$ .
- $q_2$  es el primer estado final del autómata. Indica que se ha procesado la palabra y que se han encontrado un número impar de cadenas  $xy$ . Mientras le lleguen símbolos  $y$ , permanecerá en este estado. Por tanto, si el autómata está en este estado y no le llegan nuevos símbolos, o le siguen llegando  $y$ , aceptará la palabra.
- $q_3$  es el segundo estado final. Representa la situación en la que ha llegado un número de cadenas  $xy$  impares y que ha llegado una nueva  $x$ , la cuál puede llegar a convertirse en una nueva cadena  $xy$ , haciendo que el número de éstas sea par. No obstante, como todavía no ha llegado un símbolo  $y$ , es un estado final, ya que como se mencionó anteriormente han llegado un número impar de cadenas  $xy$ . Por tanto, si el autómata se encuentra en este estado al finalizar de procesar la palabra, la aceptará. Se mantendrá en este

---

estado mientras le sigan llegando símbolos  $x$ , y pasará al estado  $q_4$  si le llega un símbolo  $y$ .

- $q_4$  es un estado que representa que han llegado un número par de cadenas  $xy$ . Si le llega un símbolo  $y$  pasará al estado  $q_0$ . Si le llega una  $x$ , pasará al estado  $q_1$ , ya que se ha obtenido el primer símbolo de una potencial cadena  $xy$ .

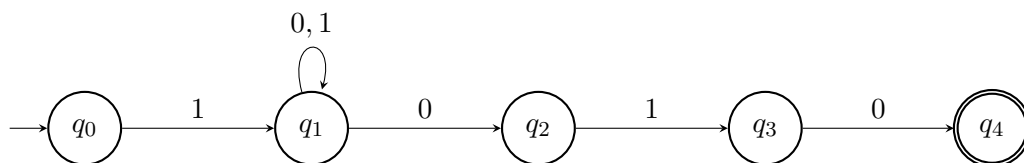
### 1.3.2 Ejercicio 2

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

- a) El lenguaje de las palabras que empiezan en 1 y terminan en 010.
- b) El lenguaje de las palabras que empiezan o terminan (o ambas cosas) en 101.
- c) El lenguaje de las palabras que contienen, simultáneamente, las subcadenas 0101 y 100. El AFND también acepta las cadenas en las que las subcadenas están solapadas (por ejemplo, “10100” y “100101” serían palabras aceptadas).

### Solución

a) Definimos como Autómata Finito No Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final. El autómata quedaría de la siguiente forma:



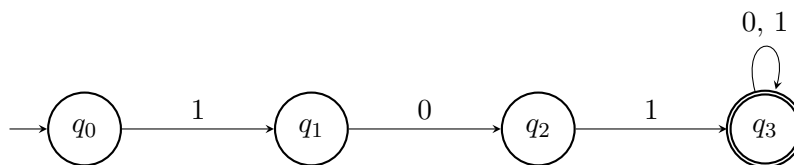
A continuación, se va a explicar lo que representa cada estado:

- $q_0$  es el estado inicial. Desde él se pasa al estado  $q_1$  en caso de que le llegue un 1 al principio de la palabra a procesar. En caso contrario, el autómata rechaza la palabra.

- $q_1$  representa que ya ha llegado un símbolo 1 como primer elemento de la palabra. Como se está trabajando con un AFND, el autómata estará en este estado mientras le sigan llegando 0 y 1, pero además transicionará al estado  $q_2$  en cuanto le llega un 0.
- $q_2$  representa que ha llegado un 0 que potencialmente forma parte de la cadena 010 que se encuentra al final de la palabra. En caso de que le llegue un 1, el autómata pasaría al estado  $q_3$ . En caso de llegarle un 0, el autómata no sabría qué hacer y no pasaría a ningún estado. No obstante, ya que se mantiene al mismo tiempo en el estado  $q_1$ , el autómata permanecería en este estado y pasaría de nuevo al estado  $q_2$ .
- $q_3$  representa que ha llegado un 1 que potencialmente forma parte de la cadena 010 que se encuentra al final de una palabra. En caso de que le llegue un 0, el autómata transicionará al estado  $q_4$ . En caso de llegarle un símbolo 0, como no hay ninguna transición, el autómata no sabría que hacer y no haría nada. Sin embargo, como todavía se encuentra en el estado  $q_1$ , permanecería en este.
- $q_4$  es el estado final del autómata, y representa que le ha llegado una palabra con un 1 al principio y la cadena 010 al final. Por tanto, si al terminar de procesar una palabra el autómata se encuentra en este estado, se acepta la palabra. Esto es con la condición de que no le lleguen nuevos símbolos de entrada. Sin embargo, si le llegan, el autómata no sabrá que hacer, y por tanto dejará de aceptar la palabra. Sin embargo, como aún se mantiene en el estado  $q_1$ , hará alguna de las acciones especificadas para ese estado en función de la entrada.

b) Definimos como Autómata Finito No Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final.

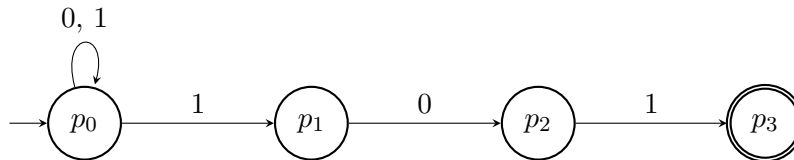
El autómata tiene dos partes: una para aceptar las palabras que contienen la cadena 101 al principio y una para aceptar las palabras que contienen 101 al final. Primero se procederá a mostrar el autómata que acepta las palabras que contienen la cadena al principio, el cuál quedaría de la siguiente forma:



Como se puede ver, el autómata rechazará cualquier palabra que no empiece por 101, ya que en las transiciones no se especifica que hacer cuando le llegue un

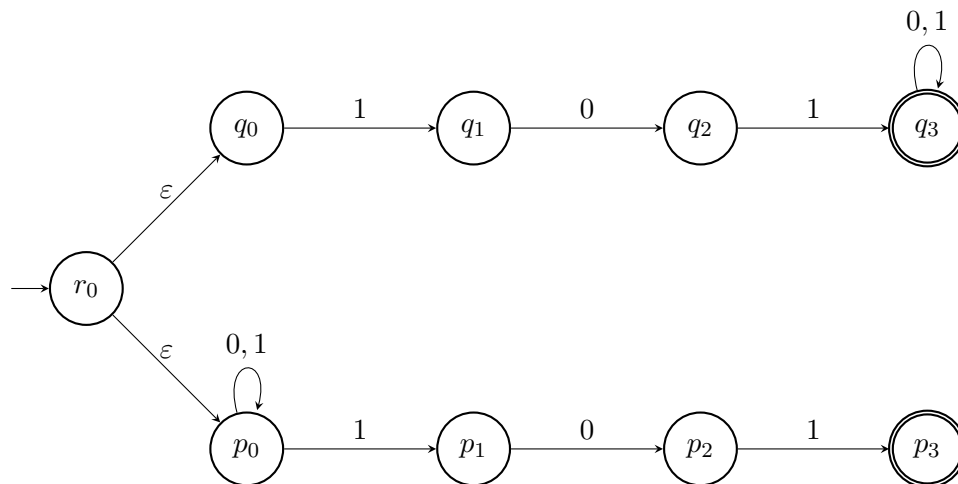
símbolo que no se espere. Una vez encontrada la cadena, da igual qué símbolo le llegue, ya que habrá cumplido el objetivo inicial.

A continuación se va a construir el autómata que acepte las palabras que terminen en la secuencia 101:



Como se puede comprobar fácilmente, este autómata procesa las palabras y las acepta si contienen la cadena 101 al final, es decir, si llega al estado  $p_3$  y se mantiene ahí, sin que le llegue ningún símbolo nuevo.

Para crear el autómata que acepta a la palabras que comiencen en 101 o acaben en 101, o ambas cosas, lo único que se tiene que hacer es unir los dos autómatas en uno mediante transiciones nulas. El resultado sería el siguiente:



Para que el autómata acepte una palabra de entrada, al terminar de procesarla debe encontrarse o bien en  $q_3$ , o bien en  $p_3$ , o bien en ambos estados.

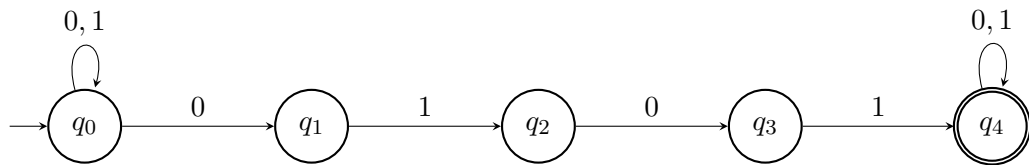
c) Definimos como Autómata Finito No Determinista a la quintupla  $M = (Q, A, q_0, \delta, F)$ , donde  $Q$  es el conjunto de estados,  $A$  el alfabeto de entrada,  $q_0$  el estado inicial,  $\delta$  el conjunto de funciones de transición y  $F$  el estado final.

El autómata tiene 2 partes: una que se encarga de comprobar que la palabra contenga las dos cadenas en el orden 0101 y 100 y la otra que se encarga de

---

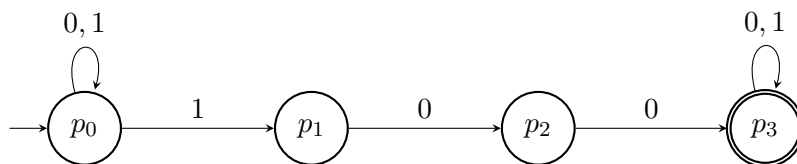
comprobar que contiene las cadenas en el orden 100 y 0101. Ambas aceptan cadenas solapadas.

Antes de empezar, vamos a construir los autómatas que aceptan cada cadena por separado, empezando por el que acepta las palabras que contengan la cadena 0101:



Se puede comprobar fácilmente que este autómata procesa la palabra y busca en ella la cadena 0101, y la acaba aceptando si llega al estado  $q_4$ , una vez encontrada ésta. Al llegar a este estado, como se ha cumplido la restricción, da igual qué símbolo le llegue, ya que se mantendrá en el estado final.

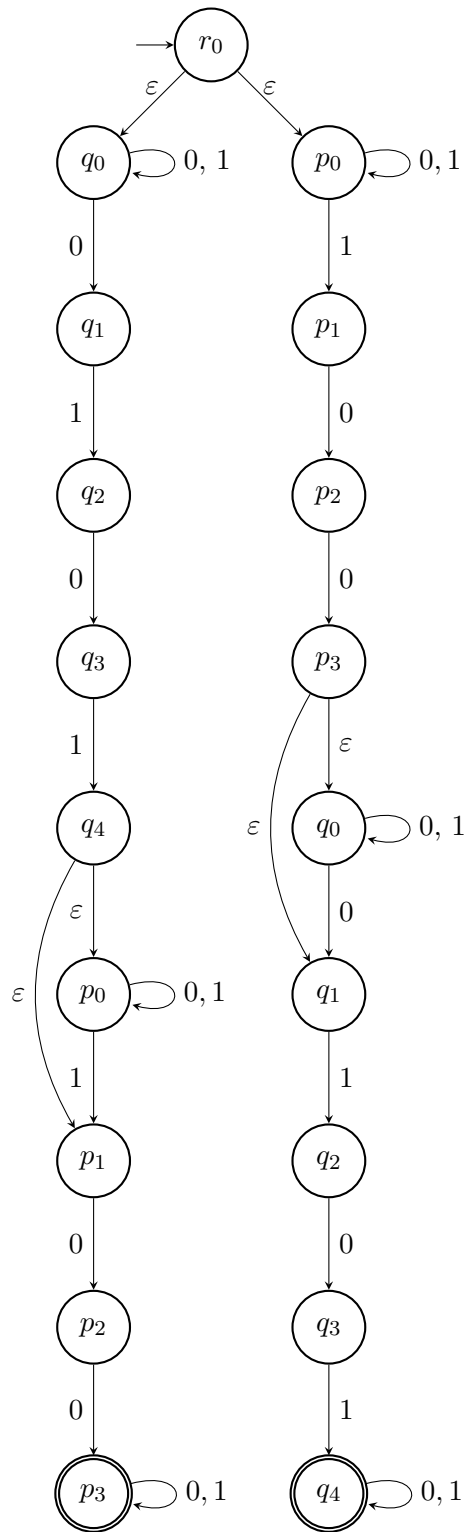
A continuación se muestra el autómata que busca la cadena 100:



De nuevo, es fácil comprobar que el autómata procesa la palabra y determina si la cadena 100 forma parte de ella, aceptando la palabra de entrada si llega al estado  $p_3$ . Como da igual que símbolo de entrada le llegue una vez encontrada la cadena se mantendrá en ese estado final.

Una vez dicho esto, se va a construir el autómata que acepte las palabras que contengan las cadenas 0101 y 100 en cualquier orden, y de ser el caso, solapadas. El autómata sería el siguiente:





---

Para que se solapen las cadenas, se han añadido transiciones nulas que permiten pasar de la parte final de un autómata al estado siguiente del inicial del otro. Adicionalmente, entre el final de un autómata y el principio del otro se han añadido transiciones nulas para conectarlos.

El autómata aceptará las palabras si llega a uno de los dos estados finales del autómata. Como son mutuamente excluyentes, solo puede llegar a uno.

### 1.3.3 Ejercicio 3

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

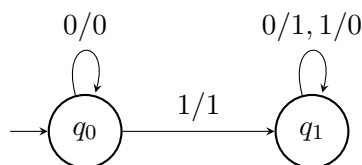
**Nota:** *El complemento a dos se realiza cambiando ceros por unos y unos por ceros, y luego, al resultado, sumándole uno en binario.*

**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++.*

#### Solución

Se va a construir una máquina de Mealy que permita calcular el complemento a dos de un número binario. En esta máquina, las salidas correspondientes a las entradas se indican en las transiciones en vez de en los estados en sí.

El resultado sería el siguiente:



El comportamiento del autómata sería el siguiente:

- Si el primer elemento de entrada es un 0, y mientras le sigan llegando símbolos 0, el autómata imprimirá el símbolo 0. Esto se debe a que al dígito más a la derecha se le suma un 1 en el complemento a 2. Por tanto, si el complemento a 1 de 0 es 1, y al sumarle el 1 del complemento a 2, su valor pasa a ser 0, y se tiene un 1 de acarreo, el cuál se sumará al siguiente símbolo de entrada. Si vuelve a entrar un 0, se obtiene un 1 del complemento a 1 y se le suma el acarreo, produciendo un 0 de nuevo con acarreo, y así sucesivamente. En el

---

autómata esto se representa con el estado  $q_0$  y la transición que sale y entra hacia él.

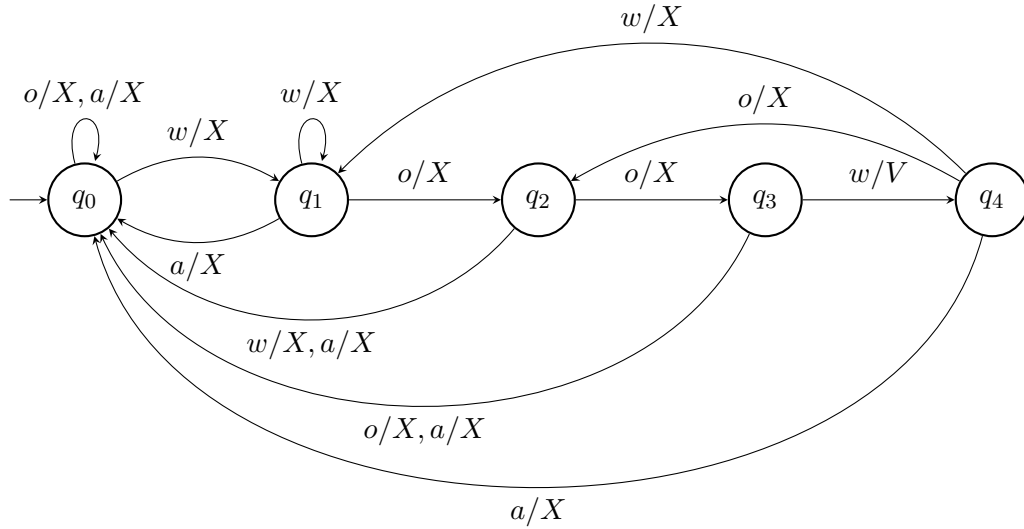
- Si el primer símbolo es 1, o llega el primer 1 de la cadena, se le hace su complemento a 1, el cuál es 0, y se le suma un 1 si es el primer elemento de entrada o si han llegado anteriormente símbolos 0 y se ha producido acarreo, resultando por tanto en 1. En este caso, no se produciría acarreo, y los símbolos que vengan a continuación tendrán su valor inverso. Esto se representa con la transición de  $q_0$  a  $q_1$ .
- Una vez que ya no hay acarreo y sigan llegando símbolos de entrada, se imprimirá el valor contrario al de entrada. Por ejemplo, si llega un 0 se imprimirá un 1, y si llega un 1, un 0. Esto se representa con la transición que entra y sale de  $q_1$ .

#### 1.3.4 Ejercicio 4

*Enunciado.* 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

#### Solución

Se plantea una máquina de Mealy como solución al problema dado, la cuál tiene la siguiente forma:



A continuación se va a explicar brevemente el funcionamiento de la máquina:

- $q_0$  es el estado inicial y al que transicionan todos los estados cuando les llega un símbolo que no esperaban. Representa además un estado en el que todavía no ha llegado ningún elemento de la cadena “woow”.
- $q_1$  representa el estado siguiente de  $q_0$ , en el cuál le llega la primera  $w$ . Mientras le sigan llegando  $w$  permanecerá en este estado, ya que sigue siendo el primer elemento de la cadena. Cuando le llegue una  $o$  pasará al siguiente estado.
- $q_2$  representa que ha llegado la primera  $o$ , y si le llega cualquier otro elemento que no sea otra  $o$ , vuelve al estado  $q_0$ . En caso de que le llegue, pasa al siguiente estado.
- $q_3$  representa un estado en el que han llegado las dos  $o$  y está esperando una  $w$  para pasar al siguiente estado y encender finalmente el led de color verde. En caso de llegarle otro símbolo, volverá al estado  $q_0$ .
- $q_4$  representa que han llegado todos los elementos de la cadena “woow” y que se ha encendido el led verde (mediante la transición  $w/X$  para pasar de  $q_3$  a  $q_4$ ). Desde aquí, como se pueden solapar las cadenas, pueden darse distintas situaciones. La primera es que le llegue una  $a$ , pasando por tanto al estado  $q_0$ . Si le llega una  $w$ , significa que puede llegar una nueva cadena “woow” sin solapar, y por tanto pasará al estado  $q_1$ , simbolizando que ha llegado la primera  $w$ . En caso de llegarle una  $o$ , significa que posiblemente la cadena se

---

solape, pasando por tanto al estado  $q_2$ , indicando que ha llegado la primera  $o$ .

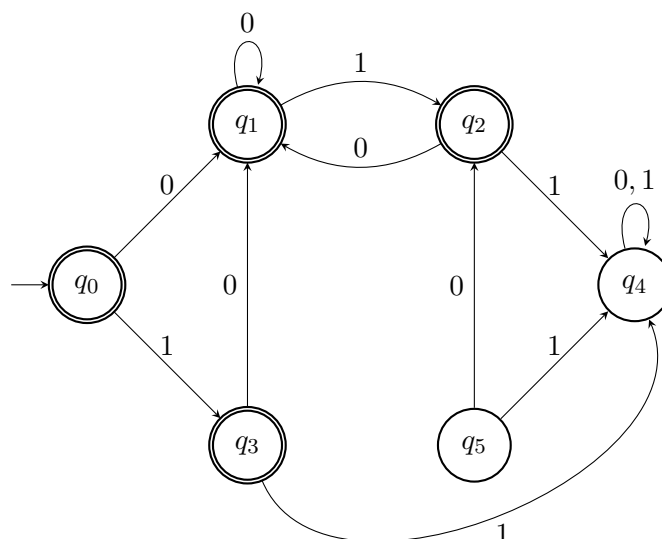
---

## 1.4 Práctica 4

### 1.4.1 Ejercicio 1

*Enunciado.* Dado el siguiente autómata responde a las siguientes cuestiones razonadamente:

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

#### 1.4.2 Ejercicio 2

*Enunciado.* 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)  $S \rightarrow AbB, A \rightarrow aA \mid \varepsilon, B \rightarrow aB \mid bB \mid \varepsilon$

b)  $S \rightarrow abaS \mid babS \mid baS \mid \varepsilon$

c)  $S \rightarrow aSA \mid \varepsilon, A \rightarrow bA \mid \varepsilon$

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

**Solución**

### 1.4.3 Ejercicio 3

*Enunciado.* 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.

---

## 2 Ejercicios voluntarios