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

1 Prácticas

1.1 Práctica 1

1.1.1 Ejercicio 1

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

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:

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

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

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

$$S = \{S\}$$

Ésta es una gramática de **tipo 2**, ya que a la izquierda aparce 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:

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

$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -\}$$

$$P = \left\{ \begin{array}{c} S \to +X.X \mid -X.X \\ X \to 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \mid 7X \mid 8X \mid 9X \mid \\ 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array} \right\}$$

$$S = \{S\}$$

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

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:

$$V = \{S, X, Y, Z, P, R, A, B, M, K, U, D, N, L, C, Q\}$$

$$T = \{0, 1, 2\}$$

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

$$S = \{S\}$$

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, $S \to 0X1$, 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 $X \to 0X \mid X1 \mid \varepsilon$.

Después tenemos la producción $S \to 0Y2$, 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 $Y \to 0Y \mid Y2 \mid \varepsilon$.

La regla $S \to 1Z2$ 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 $Z \to 1Z \mid Z2 \mid \varepsilon$.

La regla $S \to 0A1P2$ 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 $A \to 0M \mid N1$, escogiendo si se quieren más 0 que 1 (se escogería OM) 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 sustituída por $M \to OMU \mid \varepsilon$, permitiendo poner tantos 0 como se deseen y poniendo por cada uno una variable U, la cuál puede ser sustituida luego por $U \to 1 \mid \varepsilon$, 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 $A \to 0M$ se puso un 0 extra, y como la regla $M \to OMU$ 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 $A \to N1$, 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 $N \to CN1 \mid \varepsilon$, 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 $C \to 0 \mid \varepsilon$). 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 $S \to 0R1B2$, 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 sustituída por $R \to 0R \mid \varepsilon$, es decir, por tantos 0 como se desee. La variable B puede ser sustituída por $B \to 1K \mid L2$, 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 $K \to 1KD \mid \varepsilon$, poniendo un símbolo 1, después otra variable K y finalmente una variable D, la cuál puede ser sustituida mediante la regla $D \to 2 \mid \varepsilon$, 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 $B \to L2$, sustituyendo luego la variable L por $L \to QL2 \mid \varepsilon$, poniendo una variable Q, una variable L y un 2. La variable Q sería sustituida luego mediante la regla $Q \to 1 \mid \varepsilon$, 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 "*gfddddfffpdddfx*" representa que el nivel tiene (*gfddddfff*) 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:

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

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

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

$$S = \{S\}$$

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 $S \to gGx \mid pPx$ 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 $G \to ddX \mid dfY \mid fdY \mid ffZ$. Si con la regla anterior se han producido dos enemigos débiles, se sustituye la variable X mediante la regla X $\to dX \mid fR$, 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 $R \to fR \mid gG \mid pP \mid \varepsilon$, 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 $Z \to dX \mid dV$, 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 $V \to dV \mid gG \mid pP \mid \varepsilon$ 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 $Y \to X \mid V \mid R \mid gG \mid pP \mid \varepsilon$, 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 $P \to dP \mid dB \mid fB$, 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 $B \to dB \mid gG \mid pP \mid \varepsilon$, 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:

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

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

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

$$S = \{S\}$$

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 ε 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 codigo:

```
    /* ------------------------ Declaraciones ------------------------------- */

%option noyywrap
%{
#include <stdio.h>

char * procesado;
char * elem;
int i = 0;
int j = 0;
int comparar = 0;
char * comparacion;
%}

letra           [a-zA-Z]
digito          [0-9]
espacio         [ ]
entero          \-?{digito}+
numero          {entero}(\.{digito}+)?
delimitador     "'"[^\t\n]+"'"
cadena          "'"({letra}|{digito}|{espacio})*"'"
variable        ({letra}|{digito}|_)+
crear           "crear "{variable}(" "({numero}|{cadena}|{espacio})+)?
longitud        "longitud "{variable}
imprimir        "imprimir "{variable}
recorrer        "recorrer "{variable}(" "{delimitador})?
insertar        "insertar "{variable}" "({cadena}|{numero})
borrar          "borrar "{variable}" "({cadena}|{numero})
obtener         "obtener "{variable}" "{entero}
copiar          "copiar "{variable}" "{variable}
concatenar      "concatenar "{variable}" "{variable}" "{variable}
ordenar         "ordenar "{variable}" reves"?
suma            {variable}" mas "{variable}(" mas "{variable})*
operador        "igual"|"diferente"|"menor"|"menor igual"|"mayor"|"mayor
      igual"
expresion       {longitud}|{suma}|{obtener}|{variable}|{ordenar}
comparar        "comparar "{expresion}" "{operador}" "{expresion}
```

```
%%

{crear}    {procesado = yytext + 6;
           char * lista = malloc(strlen(procesado));
           elem = malloc(strlen(procesado) * 2);
           for (i=0; i<strlen(procesado) && procesado[i] != ' '; i++)
                lista[i] = procesado[i];
                if (i < strlen(procesado)) {
                    procesado += i + 1;
                    j = 0;
                    int cambiar_espacio_coma = 0;
                    for (i = 0; i < strlen(procesado); i++) {
                        if (procesado[i] == ' ') {
                          if (cambiar_espacio_coma != 0) {
                            elem[j] = ',';
                            j++;
                            elem[j] = ' ';
                            j++;
                            cambiar_espacio_coma = 0;
                          }
                        } else {
                          elem[j] = procesado[i];
                          j++;
                          cambiar_espacio_coma = 1;
                        }
                    }
                }
                printf("%s = [%s]", lista, elem);}
{longitud}      {procesado = yytext + 9;
                if (comparar > 0)
                  comparar--;
                printf("len(%s)", procesado);}
{imprimir}      {procesado=yytext + 9; printf("print(%s)", procesado);}
{recorrer}      {procesado = yytext + 9;
                char * lista = malloc(strlen(procesado));
                char * delim;
                int usar_delim = 0;
                for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                  lista[i] = procesado[i];
                if (strlen(procesado) >= i + 1)
                  usar_delim = 1;
                procesado += i + 1;
                delim = procesado;
                printf("for item in %s:\n    print(item", lista);
                if (usar_delim)
                  printf(", end = %s", delim);
                printf(")");}
{insertar}      {procesado = yytext + 9;
                char * lista = malloc(strlen(procesado));
                for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                  lista[i] = procesado[i];
                procesado += i + 1;
```

```
                  elem = procesado;
                  printf("%s.append(%s)", lista, elem);}
{borrar}          {procesado = yytext + 7;
                  char *lista = malloc(strlen(procesado));
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    lista[i] = procesado[i];
                  procesado += i + 1;
                  elem = procesado;
                  printf("%s.remove(%s)", lista, elem);}
{obtener}         {procesado = yytext; procesado += 8;
                  char *lista = malloc(strlen(procesado));
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    lista[i] = procesado[i];
                  procesado += i + 1;
                  elem = procesado;
                  if(comparar > 0) {
                    printf("%s[%s]", lista, elem);
                    comparar--;
                  }
                  else
                    printf("print(%s[%s])", lista, elem);}
{copiar}          {procesado = yytext + 7;
                  char *orig = malloc(strlen(procesado));
                  char *dest = malloc(strlen(procesado));
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    orig[i] = procesado[i];
                  procesado += i + 1;
                  dest = procesado;
                  printf("%s = %s", dest, orig);}
{concatenar}      {procesado = yytext + 11;
                  char *l1 = malloc(strlen(procesado));
                  char *l2 = malloc(strlen(procesado));
                  char *dest = malloc(strlen(procesado));
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    l1[i] = procesado[i];
                  procesado += i + 1;
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    l2[i] = procesado[i];
                  procesado += i + 1;
                  dest = procesado;
                  printf("%s = %s + %s", dest, l1, l2);}
{ordenar}         {procesado = yytext + 8;
                  char *lista = malloc(strlen(procesado));
                  if (comparar > 0)
                    comparar--;
                  for(i=0; i<strlen(procesado) && procesado[i]!=' '; i++)
                    lista[i] = procesado[i];
                  if (strlen(procesado) >= i + 1)
                    printf("%s.sort(reverse=True)", lista);
                  else
                    printf("%s.sort()", lista);}
{suma}            {procesado = yytext;
                  char * suma = malloc(strlen(procesado));
```

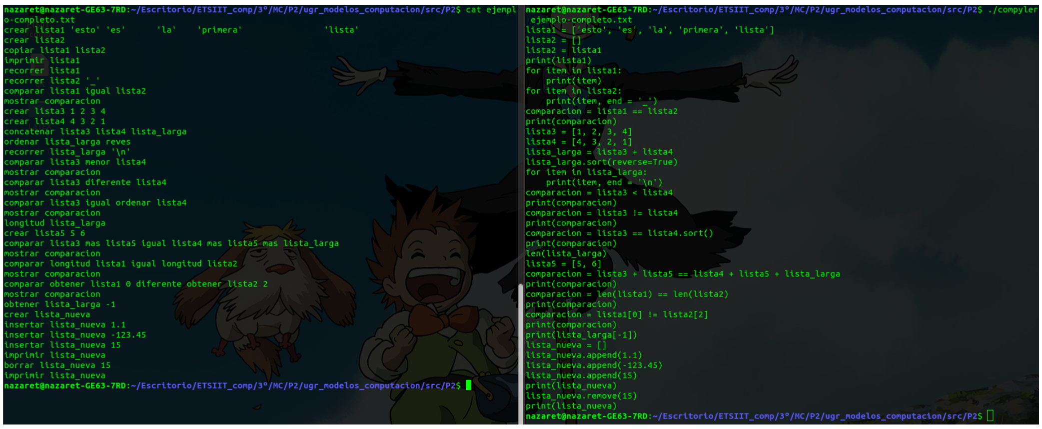
```
                int salta_palabra = 0;
                j = 0;
                if (comparar > 0)
                  comparar--;
                for(i=0; i<strlen(procesado); i++) {
                  if (procesado[i] == ' ') {
                    salta_palabra = !salta_palabra & 0x1;
                      if(salta_palabra) {
                        suma[j] = ' ';
                        j++;
                        suma[j] = '+';
                        j++;
                        suma[j] = ' ';
                        j++;
                      }
                  } else {
                    if (!salta_palabra) {
                      suma[j] = procesado[i];
                      j++;
                    }
                  }
                }
                printf("%s", suma);}
{comparar}      {comparar = 2; printf("comparacion = "); yyless(9);}
{operador}      {procesado = yytext;
                i = 0;
                if(procesado[i] == 'i')
                  printf(" == ");
                else if(procesado[i] == 'd')
                  printf(" != ");
                else if(procesado[i] == 'm') {
                  if(procesado[i+1] == 'e' && yyleng == 5)
                    printf(" < ");
                  else if(procesado[i+1] == 'e' && yyleng > 5)
                    printf(" <= ");
                  else if(procesado[i+1] == 'a' && yyleng == 5)
                    printf(" > ");
                  else
                    printf(" >= ");}}
{variable}      {if (comparar > 0)
                  comparar--;
                printf("%s", yytext);}
"mostrar comparacion" {printf("print(comparacion)");}
.                                 {}

%%

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

  if (argc == 2) {
    yyin = fopen(argv[1], "rt");

    if (yyin == NULL) {
```

```
        printf("El fichero %s no se pudo abrir\n", argv[1]);
        exit(-1);
      }
    } else
      yyin = stdin;

    yylex();

    return 0;
}

```

Cabe mencionar que a la hora de crear una lista, si se deciden insertart 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:



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:


nazaret@nazaret-GE63-7RD:~/Escritorio/ETSIIT_comp/3º/MC/P2/ugr_modelos_computacion/src/P2$ ./compyler ejemplo-completo.txt > ejemplo_salida.py
nazaret@nazaret-GE63-7RD:~/Escritorio/ETSIIT_comp/3º/MC/P2/ugr_modelos_computacion/src/P2$ python3 ejemplo_salida.py
['esto', 'es', 'la', 'primera', 'lista']
esto
es
la
primera
lista
esto_es_la_primera_lista_True
4
4
3
3
2
2
1
1
True
True
False
False
True
True
1
[1.1, -123.45, 15]
[1.1, -123.45]
nazaret@nazaret-GE63-7RD:~/Escritorio/ETSIIT_comp/3º/MC/P2/ugr_modelos_computacion/src/P2$

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:

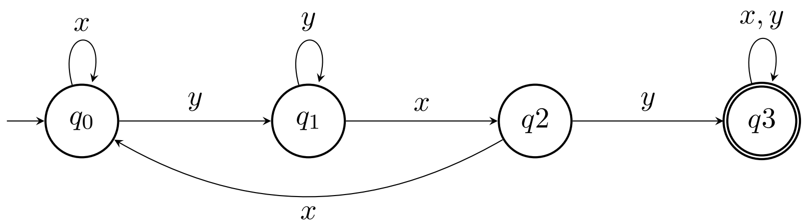
a) El lenguaje de las palabras que contienen la subcadena yxy.

b) El lenguaje de las palabras que comienzan o terminan en xyx (o ambas cosas).

c) 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, δ el conjunto de funciones de transición y F el estado final. El autómata quedaría de la siguiente forma:



Cada estado representa lo sguiente:

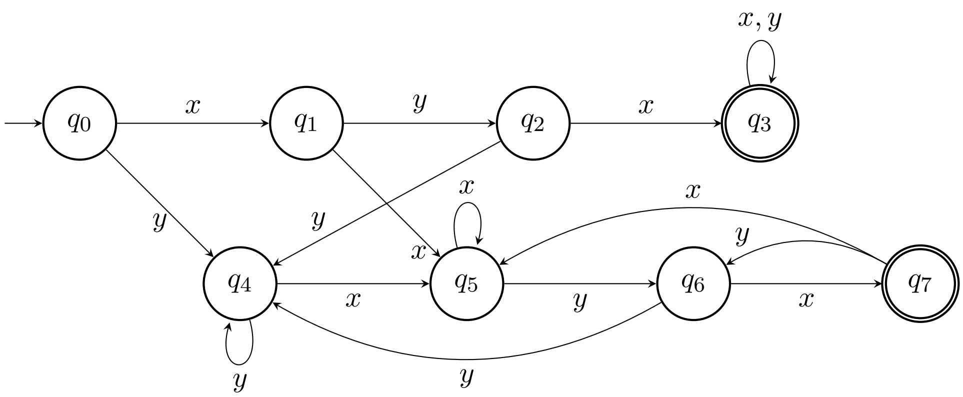
- q_0 representa el estado inicial, y se continúa en él mientras le lleguen x, lo cuál significa que todavía no le ha llegado el primer elemento de la subcadena que debe contener.

- 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 yxy. 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, δ 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.

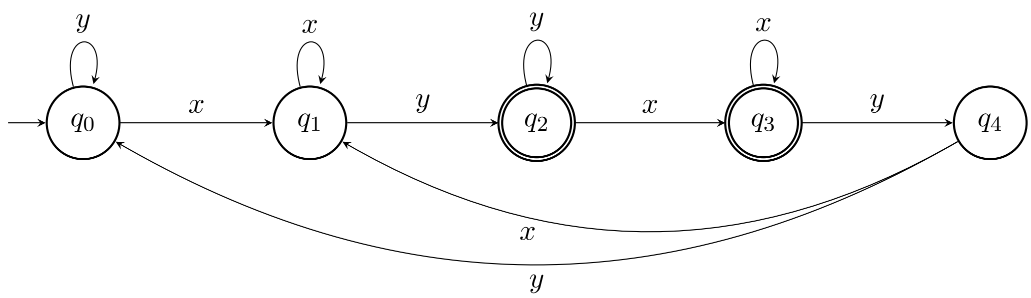
- q_5 representa un estado en el que se está buscando la cadena xyx al final de la palabra y que ha llegado el primer elemento, bien porque la cadena había sido interrumpida por una y, como se ha mencionado en el estado anterior, o bien porque al principio ha llegado una cadena del tipo xx. Mientras 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, δ 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 síbolos 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:

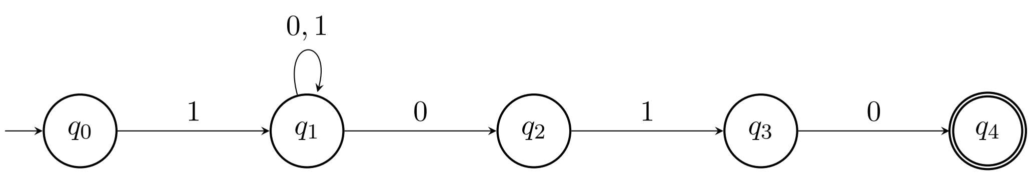
a) El lenguaje de las palabras que empiezan en 1 y terminan en 010.

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 AFDN 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, δ 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.

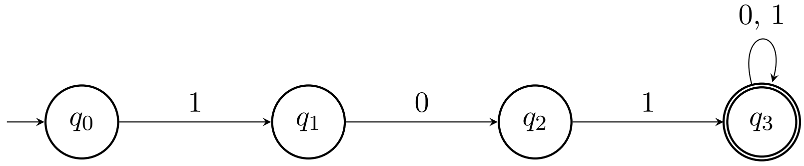
- q_2 representa que ha llegado un 0 que potencialmente forma parte de la cadena 010 que se encuentra al final de la palabra. En caso de que le llegue un 1, el autómata pasaría al estado q_3. En caso de llegarle un 0, el autómata no sabría qué hacer y no pasaría a ningún estado. No obstante, ya que se 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.

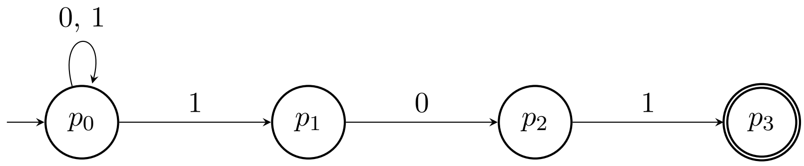
b) Definimos como Autómata Finito No Determinista a la quintupla $M = (Q, A, q_0, \delta, F)$, donde Q es el conjunto de estados, A el alfabeto de entrada, q_0 el estado inicial, δ el conjunto de funciones de transición y F el estado final.

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:



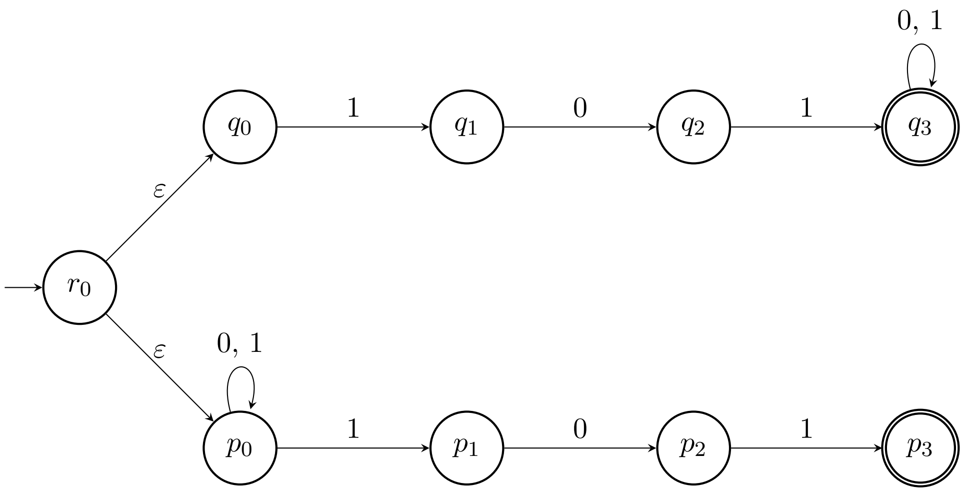
Como se puede ver, el autómata rechazará cualquier palabra que no empiece por 101, ya que en las transiciones no se especifica que hacer cuando le llegue un símbolo que no se espere. Una vez encontrada la cadena, da igual qué símbolo le llegue, ya que habrá cumplido el objetivo inicial.

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:

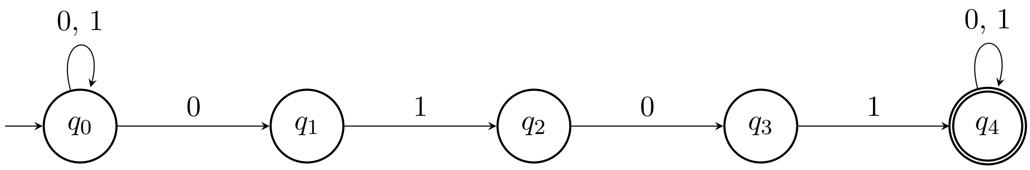


Para que el autómata acepte una palabra de entrada, al terminar de procesarla debe encontrarse o bien en q_3, o bien en p_3, o bien en ambos estados.

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, δ el conjunto de funciones de transición y F el estado final.

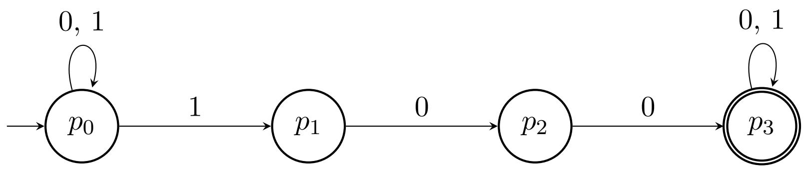
El autómata tiene 2 partes: una que se encarga de comprobar que la palabra contenga las dos cadenas en el orden 0101 y 100 y la otra que se encarga de comprobar que contiene las cadenas en el orden 100 y 0101. Ambas aceptan cadenas solapadas.

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



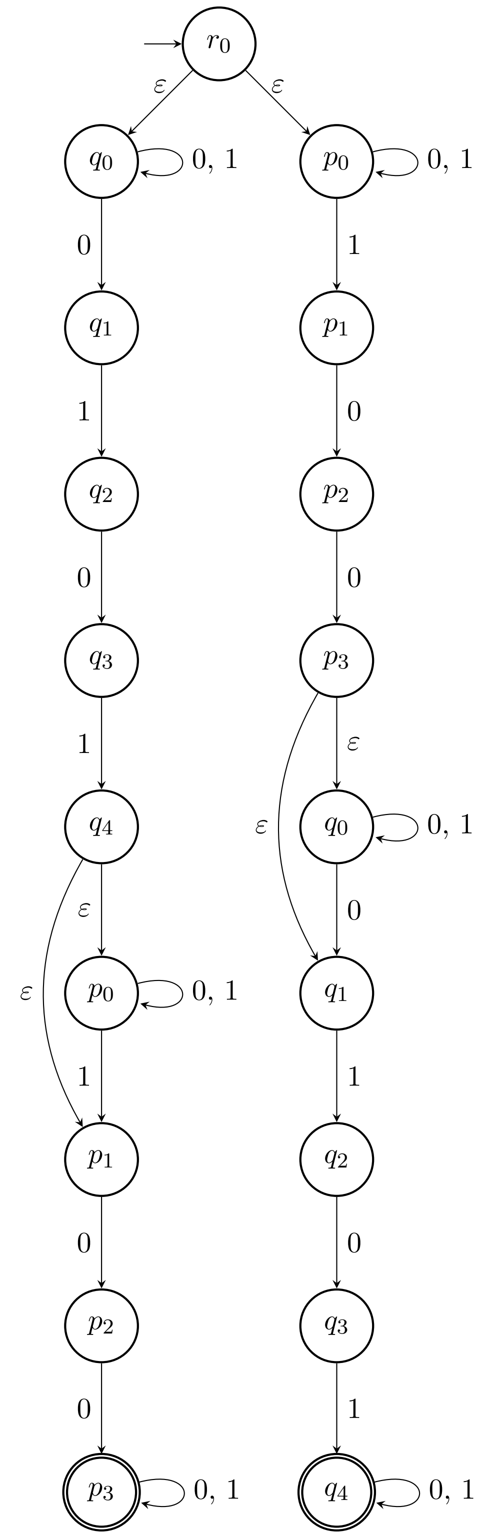
Se puede comprobar fácilmente que este autómata procesa la palabra y busca en ella la cadena 0101, y la acaba aceptando si llega al estado q_4, una vez encontrada ésta. Al llegar a este estado, como se ha cumplido la restricción, da igual qué símbolo le llegue, ya que se mantendrá en el estado final.

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.

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

<u>**Nota 2**</u>: *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

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

1.4 Práctica 4

2 Ejercicios voluntarios