

PRÁCTICA 1.- LENGUAJES Y GRAMÁTICAS

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

La gramática $G = \{V, T, P, S\}$, libre de contexto y que genera el lenguaje L , está definida por:

- $V = \{S, X\}$
- $T = \{a, b, c, d\}$
- $P = \begin{cases} S \rightarrow aSdd \mid X \mid \epsilon \\ X \rightarrow bXc \mid \epsilon \end{cases}$

Explicación: para encontrar una gramática que genere este lenguaje, debemos fijarnos que las palabras que pertenecen a dicho lenguaje presentan simetría. Esto quiere decir, que cuando la palabra tenga una 'a' al comienzo, acabará con dos 'd' y que cuando contenga m 'b's, a continuación tendrá que tener m 'c's. Además, tenemos que contemplar los casos en los que los índices n y/o m sean igual a cero; pudiendo generar palabras de la forma $a^n d^{2n} / n > 0$, de la forma $b^m c^m / m > 0$ o incluso generar la palabra vacía.

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

La gramática $G = \{V, T, P, S\}$, que genera las palabras anteriormente descritas, está definida por:

- $V = \{S, X, Y, Z\}$
- $T = \{+, -, ., n\}$
- $P = \begin{cases} S \rightarrow +X|-X \\ X \rightarrow nX|nY \\ Y \rightarrow .Z \\ Z \rightarrow nZ|n \end{cases}$

Nota: por una mayor simplicidad a la hora de la redacción del lenguaje, he utilizado el símbolo $n \in \mathbb{N} / 0 \leq n \leq 9$

Explicación: para encontrar esta gramática he seguido los siguientes pasos:

- Toda palabra de este lenguaje debe comenzar por o bien el símbolo '+' o bien el símbolo '-'.
- A continuación debe tener una cifra de al menos un dígito.
- Después es obligatorio que continve con '.'.
- Debe acabar con una cifra de al menos un dígito.

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

La gramática $G = \{V, T, P, S\}$, libre de contexto y que genera el lenguaje L , está definida por:

- $V = \{S, X, Y, M, N, A, B, C\}$

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

- $P = \begin{cases} S \rightarrow X \mid M & A \rightarrow 0A \mid 0 \\ X \rightarrow X_2 \mid Y & B \rightarrow 1B \mid 1 \\ Y \rightarrow 0X_1 \mid A \mid B & C \rightarrow 2C \mid 2 \\ M \rightarrow 0M \mid N \\ N \rightarrow 1N_2 \mid B \mid C \end{cases}$

Explicación: para encontrar una gramática que genere el lenguaje L , debemos primero entender la forma de las palabras de este.

- Aquellas en que $i \neq j$, sin tener en cuenta el valor que tome k . Estas a su vez se dividen en dos grupos, las que $i > j$ y las que $i < j$.
- Aquellas en que $j \neq k$, sin tener en cuenta el valor que tome i . Estas a su vez se dividen en dos grupos, las que $j > k$ y las que $j < k$.

Con esto en cuenta, las reglas que forman la gramática siguen el siguiente patrón:

- Primero, elegimos entre $i \neq j$ ($S \rightarrow X$) y $j \neq k$ ($S \rightarrow M$).
- A continuación, dependiendo del caso, introducimos el dígito cuyo índice no nos importa tantas veces como queramos con las reglas ' $X \rightarrow X_2$ ' o ' $M \rightarrow 0M$ '.
- Después, con las reglas ' $Y \rightarrow 0X_1$ ' o ' $N \rightarrow 1N_2$ ', introducimos dígitos para que $i=j$ o $j=k$.
- Finalmente, añadimos '0' o '1' en el primer caso o '1' o '2' en el segundo, tantos como queramos y siempre al menos uno para romper la igualdad $i=j$ o $j=k$, cumpliendo $i \neq j$ o $j \neq k$.

4. Una empresa necesita una gramática que genere niveles de un juego, siguiendo estas restricciones:

- Hay dos grupos de enemigos: grandes (g) y pequeños (p).
- Hay dos tipos de monstruos: fuertes (f) y débiles (d).
- Los grupos grandes tienen al menos 1 monstruo fuerte y 1 débil, pudiendo ir en cualquier orden.* A partir del tercero, irán primero los débiles y después los fuertes. *los dos primeros
- Los grupos pequeños tienen como mucho 1 monstruo fuerte.
- Al final de cada nivel, habrá una sala de recompensa (x).

Elabora una gramática que genere estos niveles. ¿A qué tipo de gramática dentro de la jerarquía de Chomsky pertenece? ¿Sería posible diseñar una gramática de tipo 3 para este problema?

La gramática $G = \{V, T, P, S\}$, de tipo 3 y que genera las palabras solicitadas, está definida:

- $V = \{S, G, X, Y, Z, W, P, U\}$
- $T = \{g, p, d, f, x\}$
- $P = \begin{cases} S \rightarrow gG \\ G \rightarrow d f X | f d X | f f Y | d d Z \\ X \rightarrow d X | f W | P P \\ Y \rightarrow d X \\ Z \rightarrow d Z | f W \\ W \rightarrow f W | P P \\ P \rightarrow d P | f U | x \\ U \rightarrow d U | x \end{cases}$

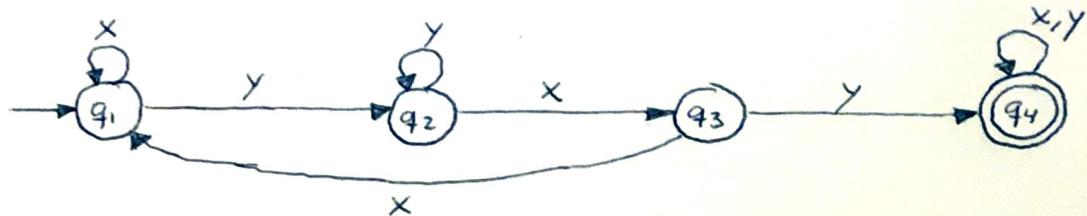
Explicación: inicialmente, introducimos una g para marcar el grupo grande. Metemos dos monstruos, débiles o fuertes, y en función de la combinación,

limitamos las siguientes acciones. Una vez introducimos una p, hemos pasado a los grupos pequeños, pudiendo añadir como mucho una f. Finalmente, cerramos con una x la palabra.

PRACTICA 3. - AUTÓMATAS Y EXPRESIONES REGULARES

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

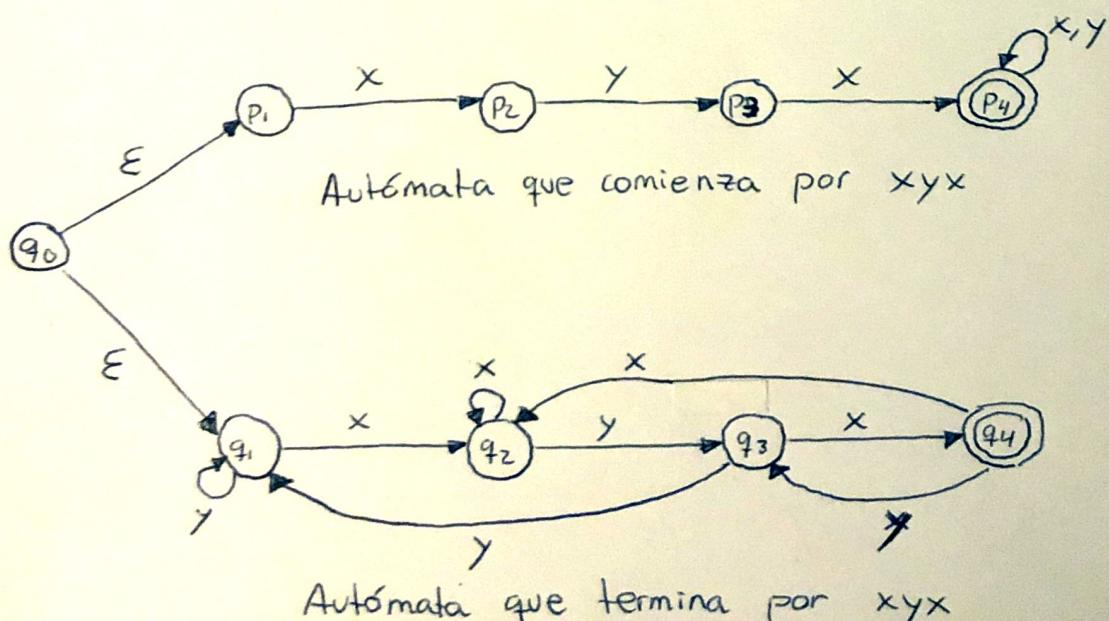
a) El lenguaje de las palabras que contengan la subcadena yxy .



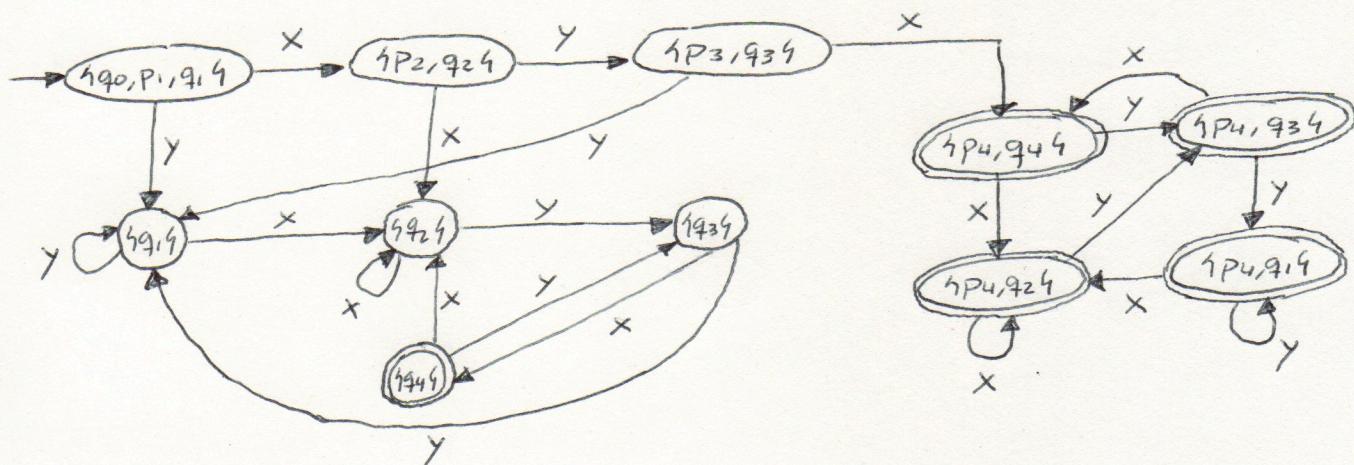
Explicación: para entender el autómata, describiremos sus estados:

- $q_1 \rightarrow$ no se ha introducido ninguna letra de la subcadena necesaria. No es final.
- $q_2 \rightarrow$ se ha introducido la primera letra de la subcadena (x). No es final.
- $q_3 \rightarrow$ se ha introducido las dos primeras letras de la subcadena (yx). No es final.
- $q_4 \rightarrow$ se ha introducido la subcadena (yxy). Es final y nos permite introducir cualquier letra del alfabeto en cualquier orden.

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



Como el autómata encontrado es un AFND y necesitamos un AFD, realizamos el proceso de paso de uno a otro, dando como solución:

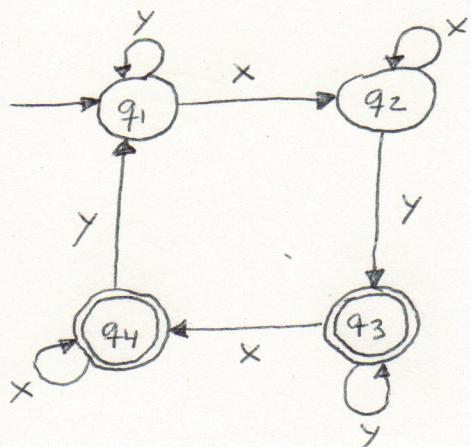


Explicación: el autómata resultante es un AFD y podemos ver representadas las tres tipos de palabras de nuestro lenguaje con los siguientes estados:

- Palabras que solo acaban en xyx : están representadas con los estados $\bar{1}q_1 \bar{q}$, $\bar{1}q_2 \bar{q}$ y $\bar{1}q_3 \bar{q}$ como estados intermedios y $\bar{1}q_4 \bar{q}$ como final.
- Palabras que solo empiezan en xyx : están representadas con los estados $\bar{1}p_2, q_2 \bar{q}$ y $\bar{1}p_3, q_3 \bar{q}$ como estados intermedios, y $\bar{1}p_4, q_3 \bar{q}$, $\bar{1}p_4, q_2 \bar{q}$ y $\bar{1}p_4, q_1 \bar{q}$ como estados finales.
- Palabras que empiezan y acaban por xyx : están representadas por $\bar{1}p_4, q_4 \bar{q}$ como estado final.

En el autómata vemos que o vamos hacia el camino de empezar por xyx y ya luego decidimos si también queremos que acabe en xyx , o bien elegimos que únicamente acabe en xyx .

c) El lenguaje $L \subseteq \{x, y\}^*$ que acepta aquellas palabras con un número impar de ocurrencias de la subcadena xy.

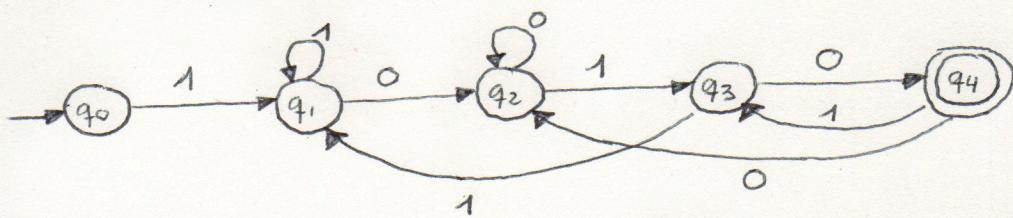


Explicación: el autómata es muy simple, tiene cuatro estados:

- q_1 : cadena xy introducida un nº par de veces
- q_2 : cadena xy introducida un nº par de veces y he introducido una x.

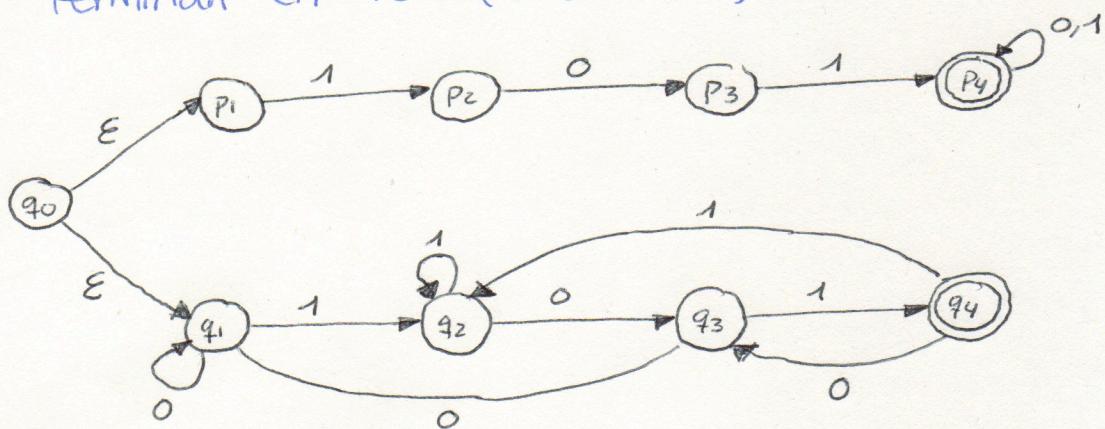
- q_3 : estado final, he introducido un nº impar de veces la cadena xy.
- q_4 : estado final, he introducido un nº impar de veces la cadena xy y he introducido una x.

2. En el alfabeto {0,1}, construir un AFND que acepte cada uno de los siguientes lenguajes:
al Palabras que empiezan en 1 y acaban en 010.



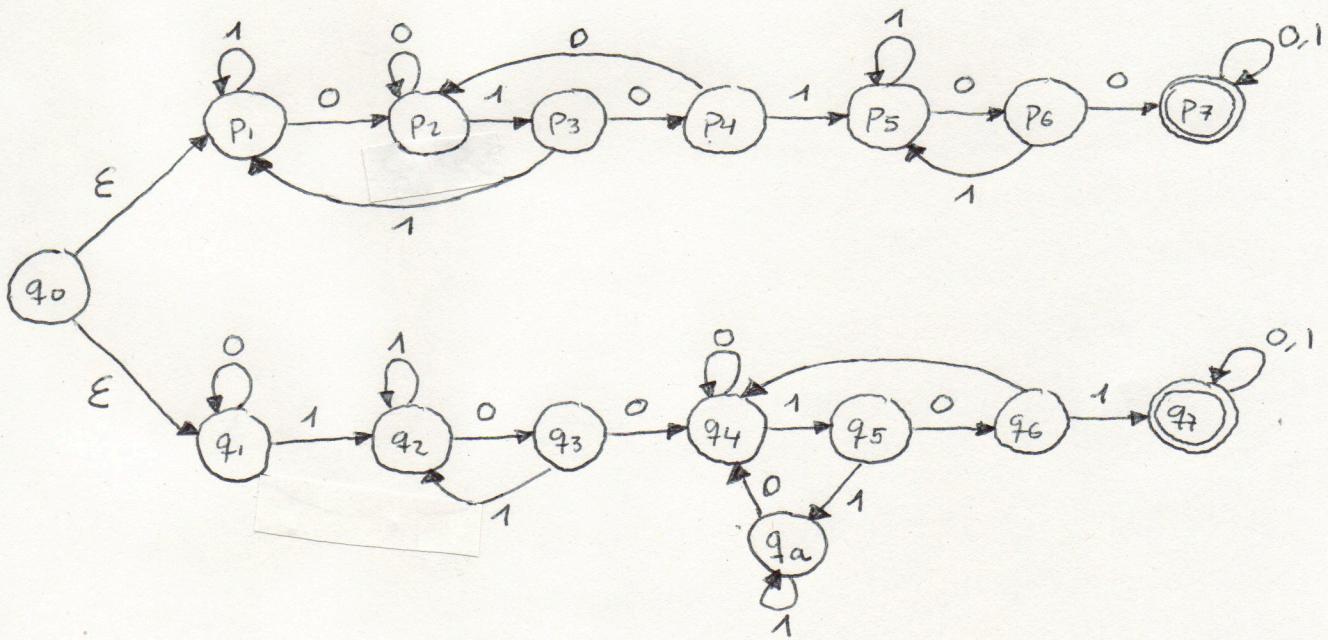
Explicación: al ser un AFND, es un autómata sencillo. Inicialmente obligamos a introducir un 1 para cumplir con la primera condición. Los estados q_1, q_2 y q_3 representan haber introducido cero letras, una letra (0) y dos letras (01) de la cadena que queremos que cierre la palabra. Finalmente, q_4 sería nuestro estado final, simbolizando haber introducido la cadena completa (010). Si en q_4 añadimos otra letra, volveríamos a un estado anterior.

b) El lenguaje de las palabras que empiezan o terminan en 101 (o ambas).



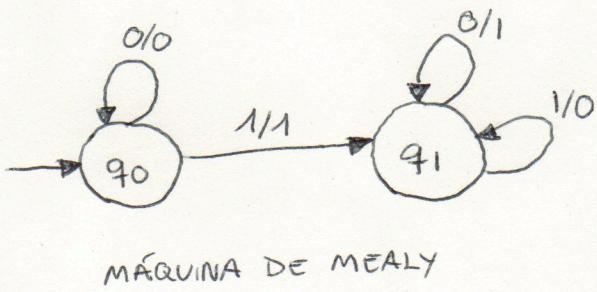
Explicación: en el ejercicio anterior, apartado b, nos pedían un AFD que generase el lenguaje de las palabras que empiezan o terminan en xyx (o ambas), por lo que este ejercicio es idéntico, sustituyendo ' x ' por '1' y ' y ' por '0'. Destacar que el caso en el que se cumple que empieza y acaba en 101 está contemplado en el estado q_4 .

c) El lenguaje de las palabras que contienen las subcadenas 0101 y 100. El AFND acepta cadenas en las que las subcadenas se solapen.



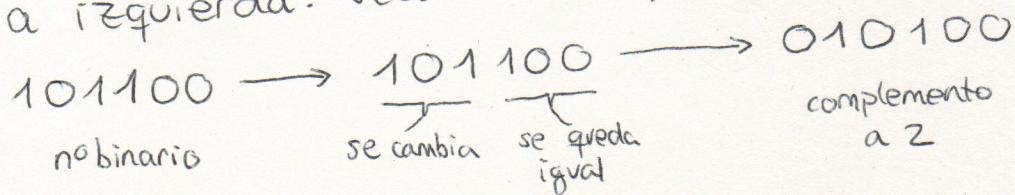
Explicación: para realizar un AFND con estas características debemos fijarnos en que para contener ambas cadenas solo pueden ir, en este caso, primero una subcadena y luego la otra. Además, si se solapan, yendo '0101' primera y luego '100' solo existe la subcadena '010100' como resultado; y yendo '100' primera y luego '0101' solo existe la subcadena '100101' como resultado. El problema se reduce a crear un AFND que acepte o una de estas cadenas o la otra, pero teniendo en cuenta que cada una de estas subcadenas nuevas se puede seguir dividiendo en las dos originales.

3. Calcular una máquina de Mealy o Moore que codifique el complemento a dos de un no binario.

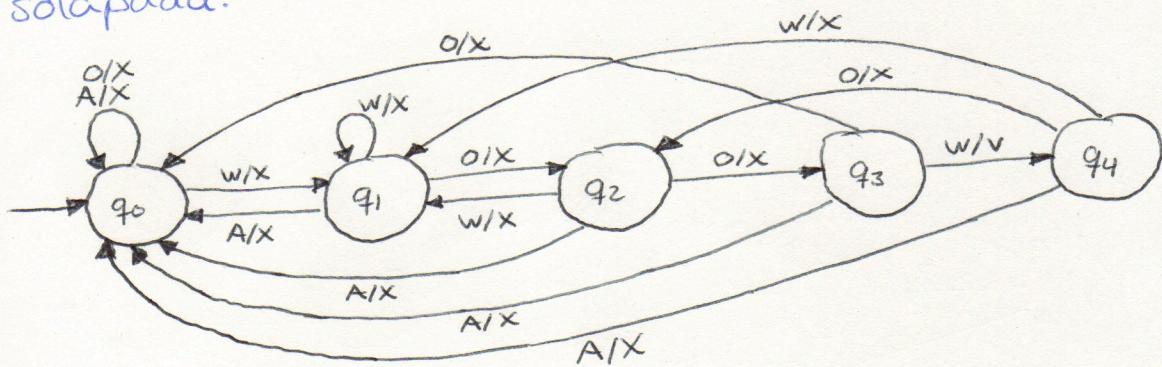


- q_0 : estado en el que no se acarrea
- q_1 : estado en el que sí se acarrea

Explicación: calcular una máquina de Mealy que codifique el complemento a dos de un no binario es simple si nos percatamos que eso implica dejar igual todos los '0's hasta que encontramos el primer '1', dejandolo igual, y a continuación, hacer el complemento a uno del resto del número. Cabe destacar que leeríamos el número de derecha a izquierda. Veamos un ejemplo:



4. Diseñar una máquina de Mealy o de Moore que dada una cadena usando el alfabeto $A = \{a, w, 0\}$ encienda un led verde (salida V) cada vez que se detecte la cadena 'woow', apagándolo cuando lea otro símbolo (salida X). La secuencia puede estar solapada.



Explicación: para entender esta máquina de Mealy analizaremos primero sus estados:

- q_0 : no se ha introducido ningún carácter de la cadena solicitada.
- q_1 : se ha introducido el primer carácter de la cadena solicitada (w).
- q_2 : se ha introducido los dos primeros caracteres de la cadena solicitada (wo).
- q_3 : se ha introducido los tres primeros caracteres de la cadena solicitada (woow).
- q_4 : se ha introducido la cadena solicitada.

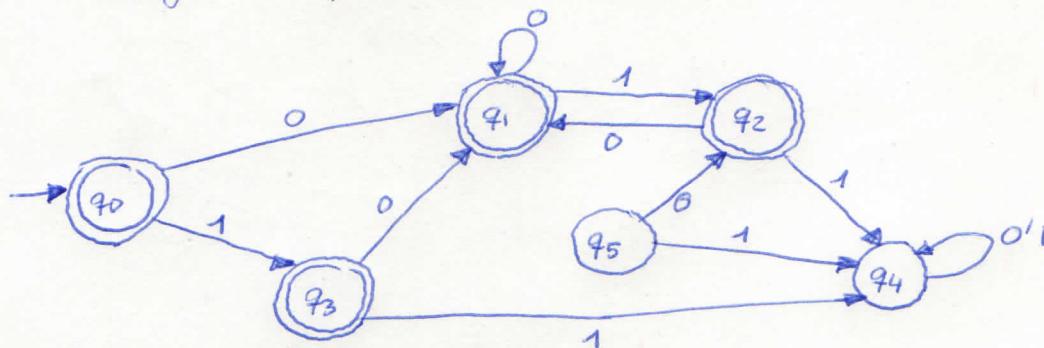
Además, destacaremos que la máquina solo devolverá 'V' cuando pasemos del estado q_3 al q_4 . También podemos apreciar que la cadena 'woowoow', se puede solapar, generando la cadena 'woowooow', lo cuál está contemplado en nuestra máquina.

Finalmente, podemos fijarnos en que el carácter 'A' nos sirve como una forma de reinicio, devolviéndonos la máquina al estado inicial q_0 en cualquier caso.

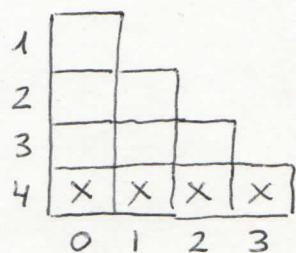
PRÁCTICA 4.- OPTIMIZACIÓN, FNC, AMBIGÜEDAD Y AP

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

a) ¿Habrá alguna forma de optimizar el autómata para que reduciendo su complejidad siga aceptando el mismo lenguaje?



Para optimizar el autómata debemos eliminar los estados inaccesibles, en este caso, q_5 . A continuación, realizamos el algoritmo de minimización de autómatas para ver si existen estados idénticos:



Paso 1: Estados distinguibles iniciales (uno es final y otro no).

$$0 \equiv 1 \\ 2 \equiv 3$$

1			
2	X	X	
3	X	X	
4	X	X	X
	0	1	2

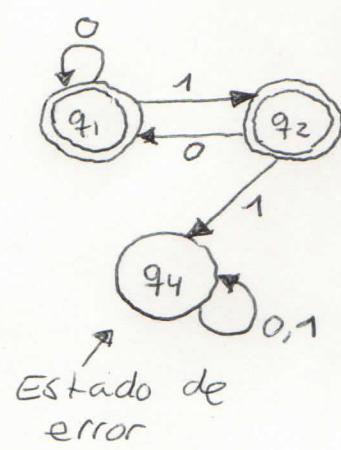
Paso 2: otros estados distinguibles.
(marcados en azul el motivo)

q_0	01	01
q_3	13	12
q_2	14	01
q_3	14	14
q_2	14	13
q_0	13	01
q_1	12	12

distingüibles
si 3 y 2
distingüibles

Con las equivalencias $0 \equiv 1$ y $2 \equiv 3$, tenemos como resultado el autómata que acepta cadenas compuestas de '0' y '1', siempre y cuando no contengan la subcadena '11'.

$$L = \{(0,1)^*\} / \text{no contiene subcadena '11'} \wedge \text{no contiene palabra vacía}$$



b) ¿Se podría obtener la gramática que genere este lenguaje en la Forma Normal de Chomsky?

$$\begin{array}{l} S \rightarrow 0S|1A|\epsilon \\ A \rightarrow 0S|\epsilon \end{array} \Rightarrow \begin{array}{l} S \rightarrow 0S|1A|1|0 \\ A \rightarrow 0S|0 \end{array}$$

Para poder pasar a la FNC, primero eliminamos las transiciones que generan producciones nulas y producciones unitarias. En este caso, tenemos producciones nulas en 'S' y 'A', siendo ambas anulables. $H = \{S, A\}$

$$\begin{array}{l} S \rightarrow 0S|1A|0|1 \\ A \rightarrow 0S|0 \end{array} \xrightarrow{\text{FNC}} \begin{array}{l} S \rightarrow C_0 S | C_1 A | 0 | 1 \\ A \rightarrow C_0 S | 0 \\ C_0 \rightarrow 0 \\ C_1 \rightarrow 1 \end{array}$$

La FNC deja las reglas con la forma $A \rightarrow BC$ ó $A \rightarrow a$. Para llegar a ella, primero sustituimos las letras minúsculas por un regla ($C \rightarrow c$), siempre y cuando estas no sean de la forma $A \rightarrow a$. En este caso, sustituimos '0' y '1' por las reglas $C_0 \rightarrow 0$ y $C_1 \rightarrow 1$. A continuación, si tenemos reglas de la forma $A \rightarrow A_0 A_1 \dots A_n$ con $n \in \mathbb{N}$, las separamos, añadiendo nuevas reglas de la forma $D_0 \rightarrow A_0 A_1$ y sustituyendo en A, quedando $A \rightarrow D_0 A_2 \dots A_n$ con $n \in \mathbb{N}$, hasta que todas las reglas queden con una forma $A \rightarrow BC$. En este caso, no es necesario aplicar este paso.

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

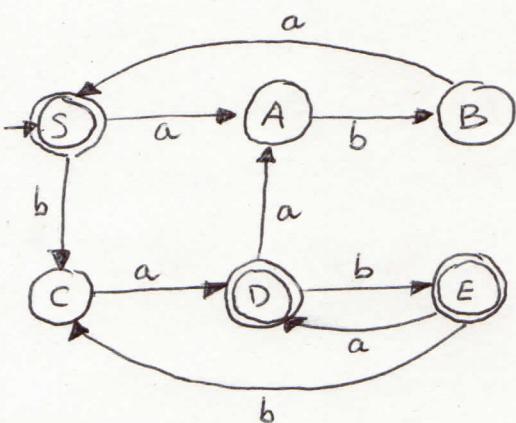
a) $S \rightarrow A\overset{(1)}{b}B$
 $A \rightarrow a\overset{(2)}{A}|\epsilon$
 $B \rightarrow a\overset{(4)}{B}|b\overset{(5)}{B}|\epsilon$

Nota: he etiquetado las reglas para facilitar la explicación.
NO AMBIGUA

Las palabras pertenecientes a este lenguaje tienen la forma $(a^* + b + (ab)^*)$. Si nos fijamos, cada palabra está subdividida en tres partes: la primera que se rellena con las reglas (2) y (3), permitiéndonos poner tantas 'a's como queramos; la segunda que utiliza la regla (1), obligandonos a poner una b; y finalmente, la tercera, con las reglas (4), (5) y (6), que introduce 'a's con la regla (4) y 'b's con la (5), de la manera que queramos. Teniendo esto en cuenta, podemos decir que cualquier palabra se genera de una única manera.

b) $S \rightarrow abaS \mid babS \mid baS \mid \epsilon$ AMBIGUA

Podemos demostrar que la gramática es ambigua ya que, por ejemplo, la palabra 'bababa', perteneciente al lenguaje generado por esta gramática, se puede generar de dos maneras posibles; una, aplicando las reglas (2), (1) y (4), y otra, aplicando (3), (3), (3) y (4). Si encontramos una gramática no ambigua que genere este lenguaje, podemos decir que no es inherentemente ambigua.

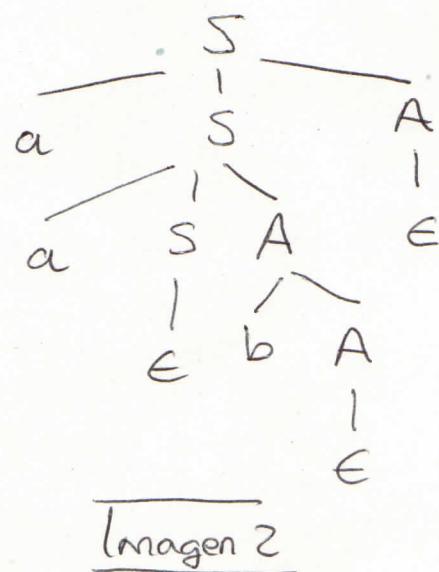
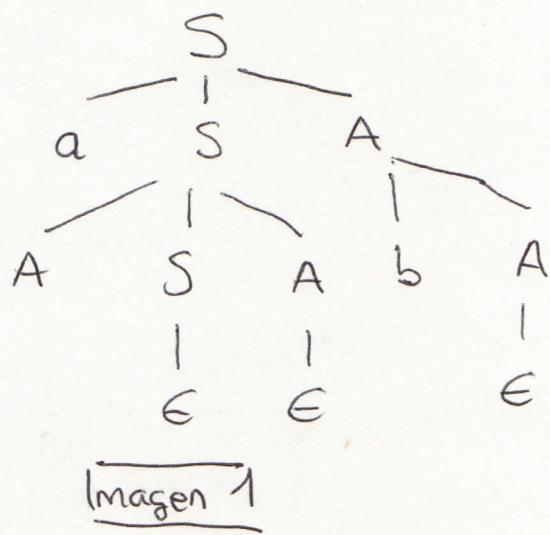


$$\begin{aligned}
 S &\rightarrow aA|bC|\epsilon \\
 A &\rightarrow bB \\
 B &\rightarrow aS \\
 C &\rightarrow aD \\
 D &\rightarrow aA|bE|\epsilon \\
 E &\rightarrow aD|bC|\epsilon
 \end{aligned}$$

Explicación: he construido un autómata que genera las mismas palabras del lenguaje generado por la gramática dada. De ahí, he pasado a una nueva gramática no ambigua, por lo que queda demostrado que el lenguaje generado no es inherentemente ambiguo.

c) $S \rightarrow a^{\text{(1)}} S A | \epsilon^{\text{(2)}}$ AMBIGUA
 $A \rightarrow b^{\text{(3)}} A | \epsilon^{\text{(4)}}$

Podemos demostrar que es una gramática ambigua, ya que con la palabra 'aab', podemos ver que se puede generar de dos maneras distintas; una (Imagen 1) y otra (Imagen 2).



$$L = \{a^n b^m / n \geq 0, m \geq 0, (n, m) \in \mathbb{N} \times \mathbb{N}\}$$

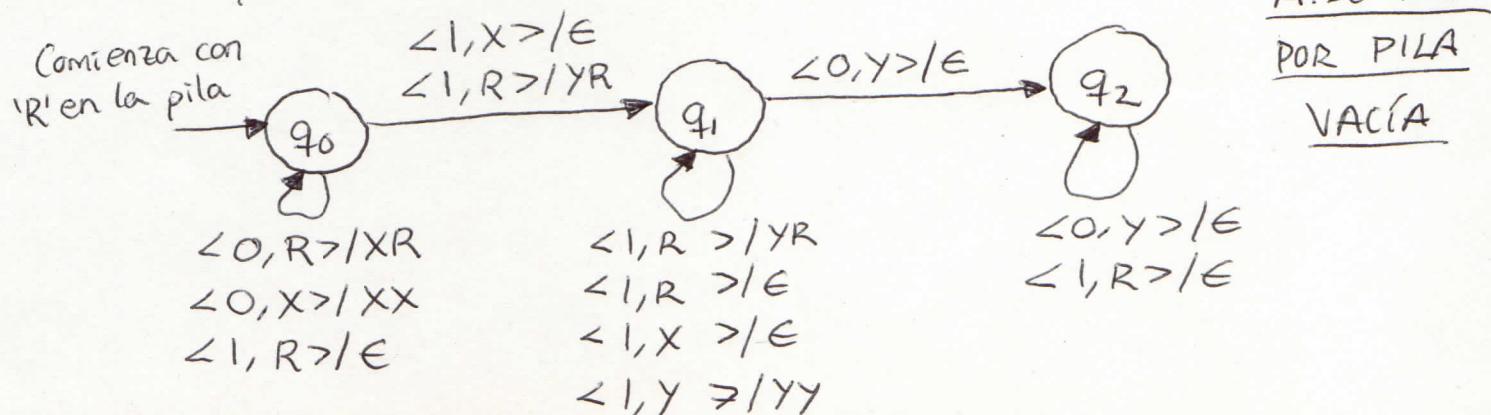
Una vez hemos reconocido el lenguaje, si encontramos una gramática no ambigua que lo genere, podremos decir que el lenguaje no es inherentemente ambiguo.

$S \rightarrow aX1\epsilon$
$X \rightarrow aX1bY1\epsilon$
$Y \rightarrow bY1\epsilon$

3. Encontrar el autómata que acepte el lenguaje $L = \{0^i 1^j 0^k 1^l \mid i+k=j; i, j, k, l \in \mathbb{N}\}$. Una vez diseñado el autómata, calcular la gramática libre de contexto que acepta L , eliminando posibles producciones inútiles.

Al depender unos índices de otros, buscaremos un autómata de pila. Debemos tener en cuenta los siguientes casos:

- $i=0, j=k$ ó $k=0, i=j$: introducimos primero n '1's, luego n '0's y un '1' & primero n '0's, y luego $n+1$ '1's.
- $i \geq k$ & $k \geq i$, con $i, k \geq 0$: primero metemos '0's, luego '1's, y luego metemos tantos '0's como diferencia haya entre estos dos, y acabamos con un '1'.
- La palabra '1'.



Para pasar a gramática, primero trataremos las reglas "sencillas":

- $(q_0, \epsilon) \in \delta(q_0, 1, R) \Rightarrow [q_0, R, q_0] \rightarrow 1$
- $(q_1, \epsilon) \in \delta(q_0, 1, X) \Rightarrow [q_0, X, q_1] \rightarrow 1$
- $(q_1, \epsilon) \in \delta(q_1, 1, R) \Rightarrow [q_1, R, q_1] \rightarrow 1$
- $(q_1, \epsilon) \in \delta(q_1, 1, \text{X}) \Rightarrow [q_1, X, q_1] \rightarrow 1$
- $(q_2, \epsilon) \in \delta(q_1, 0, Y) \Rightarrow [q_1, Y, q_2] \rightarrow 0$
- $(q_2, \epsilon) \in \delta(q_2, 0, Y) \Rightarrow [q_2, Y, q_2] \rightarrow 0$
- $(q_2, \epsilon) \in \delta(q_2, 1, R) \Rightarrow [q_2, R, q_2] \rightarrow 1$

Y a continuación, las reglas "complejas":

$$\rightarrow (q_0, XR) \in \delta(q_0, 0, R) \quad \boxed{R}_{(q_0)} \xrightarrow{0} \boxed{X}_{(q_0)} \xrightarrow{u_1} \boxed{R}_{(r_1)} \xrightarrow{u_2} \boxed{\sqcup}_{(r_2)}$$

$$\Rightarrow [q_0, R, r_2] \rightarrow 0 [q_0, X, r_1] [r_1, R, r_2] \quad \forall r_1, r_2 \in Q_n \\ (q_0, q_1, q_2)$$

$$\rightarrow (q_0, XX) \in \delta(q_0, 0, X) \quad \boxed{X}_{(q_0)} \xrightarrow{0} \boxed{X}_{(q_0)} \xrightarrow{u_1} \boxed{X}_{(r_1)} \xrightarrow{u_2} \boxed{\sqcup}_{(r_2)}$$

$$\Rightarrow [q_0, X, r_2] \rightarrow 0 [q_0, X, r_1] [r_1, X, r_2] \quad \forall r_1, r_2 \in Q_n \\ (q_0, q_1, q_2)$$

$$\rightarrow (q_1, YR) \in \delta(q_0, 1, R) \quad \boxed{R}_{(q_0)} \xrightarrow{1} \boxed{Y}_{(q_1)} \xrightarrow{u_1} \boxed{R}_{(r_1)} \xrightarrow{u_2} \boxed{\sqcup}_{(r_2)}$$

$$\Rightarrow [q_0, R, r_2] \rightarrow 1 [q_1, Y, r_1] [r_1, R, r_2] \quad \forall r_1, r_2 \in Q_n \\ (q_0, q_1, q_2)$$

$$\rightarrow (q_1, YR) \in \delta(q_1, 1, R) \quad \boxed{R}_{(q_1)} \xrightarrow{1} \boxed{Y}_{(q_1)} \xrightarrow{u_1} \boxed{R}_{(r_1)} \xrightarrow{u_2} \boxed{\sqcup}_{(r_2)}$$

$$\Rightarrow [q_1, R, r_2] \rightarrow 1 [q_1, Y, r_1] [r_1, R, r_2] \quad \forall r_1, r_2 \in Q_n \\ (q_0, q_1, q_2)$$

$$\rightarrow (q_1, YY) \in S(q_1, 1, Y) \quad \boxed{Y}_{(q_1)} \xrightarrow{1} \boxed{Y}_{(q_1)} \xrightarrow{u_1} \boxed{Y}_{(\text{f})} \xrightarrow{u_2} \boxed{L}_{(r_2)}$$

$$\Rightarrow [q_1, Y, r_2] \rightarrow 1 [q_1, Y, r_1] [q_1, Y, r_2] \quad \forall r_1, r_2 \in Q_n$$

(q_0, q_1, q_2)

Cuando tenemos todas las reglas desarrolladas, eliminamos aquellas que sean inútiles. Entendemos como reglas inútiles:

- Las que contengan la regla anterior.
- Aquellas que pasen de un estado q_x a otro q_y , con $x > y$.
- Aquellas con bucles infinitos.
- Aquellas que no estén definidas.

Por simplicidad, sustituiré las reglas por letras mayúsculas tal que:

$[q_0, R, q_0]$ por 'A', $[q_0, R, q_1]$ por 'B', $[q_0, R, q_2]$ por 'C',
 $[q_1, R, q_1]$ por 'D', $[q_2, R, q_2]$ por 'E', $[q_1, R, q_2]$ por 'F',
 $[q_0, X, q_1]$ por 'G', $[q_1, X, q_1]$ por 'H', $[q_1, Y, q_2]$ por 'I' y
 $[q_2, Y, q_2]$ por 'J'.

Tenemos la gramática $G = \{V, T, P, S\}$ definida por:

$$\bullet V = \{S, A, B, C, D, E, F, G, H, I, J\}$$

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

$$\bullet P = \begin{cases} S \rightarrow A \mid B \mid C \\ A \rightarrow 1 \\ B \rightarrow OGD \\ C \rightarrow OGF \mid 1IE \\ D \rightarrow 1 \\ E \rightarrow 1 \end{cases}$$

$$\begin{aligned} F &\rightarrow 1IE \\ G &\rightarrow 1 \mid OGH \\ H &\rightarrow 1 \\ I &\rightarrow 0 \mid 1IJ \\ J &\rightarrow 0 \end{aligned}$$