UNIVERSIDAD DE GRANADA

Modelos de Computación
grado en ingeniería informática

Memoria de prácticas

Autora
Nazaret Román Guerrero

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2018-2019

Contents

1 Práctica 1: Ejercicios prácticos sobre Lenguajes y Gramáticas

1.1 Ejercicio 1.

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

La gramática que he creado utiliza 3 variables, que son las siguientes:

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

Los símbolos terminales vienen dados en el alfabeto del lenguaje, y son:

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

Las producciones necesarias para poder crear un lenguaje con la estructura anterior son las siguintes:

$$P = \left\{ \begin{array}{c} S \to aXdd \mid Y \mid \epsilon, \\ Y \to bYc \mid \epsilon,\ X \to aXdd \mid Y \mid \epsilon \end{array} \right\}$$

Las explicaré una a una.

- La producción $S \to aXdd$ comienza las palabras, introduciendo una a y el doble de d, y llama a la producción X donde se pueden seguir introduciendo a y d, introducir b y c o terminar.
- La producción $S \to Y$ comienza palabras donde las a y las d aparecen un total de 0 veces.
- La producción $S \to \epsilon$ da lugar a la palabra vacía, ya que el lenguaje permite las palabras con 0 a, 0 b y c y 0 d.
- La producción $Y \to bYc$ empieza a introducir los caracteres b y c. Esta producción se asegura que el número de ambos terminales sea el mismo.
- La producción $Y \to \epsilon$ se encarga de terminar la palabra.
- La producción $X \to aXdd$ introduce los caracteres a y d. Se encarga de asegurar que el número de d es el doble de a, debido a que introduce 2 d por cada a. Se pueden introducir tantas a y d como se desee, porque la producción se puede llamar a sí misma el número de veces que se quiera.
- La producción $X \to Y$ permite introducir las b y c en la palabra creada hasta ese momento. Una vez que se han empezado a introducir estos dos últimos caracteres, ya no se pueden introducir más a y d.
- Por último, la producción $X \to \epsilon$ permite finalizar la palabra que se ha estado creando.

Por último, el símbolo inicial es:

$$S = \{S\}$$

1.2 Ejercicio 2.

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

El lenguaje que debe generar la gramática es del tipo $\mathcal{L} = \{\pm(n) + .(n) + \text{ tal que } n \in \mathbb{N}\}$. Por tanto, la gramática que se ha creado es la siguiente:

- $V = \{X, Y, S\}$
- $T = \{+, -, 1, 2...\}$
- $P = \{\ S \to YX.X,\ Y \to + \mid -,\ X \to XX \mid 0 \mid 1 \mid ...\ \}$, explicadas a continuación.
  - La primera producción, $S \to YX.X$, comienza las palabras del lenguaje, obligando a colocar un signo al inicio (Y), un punto entre los números, y cifras (delante y detrás del punto).
  - La segunda, $Y \to + \mid -$, establece el signo que se puede poner: o positivo o negativo.
  - La tercera, $X \to XX$, permite introducir tantas cifras como se desee.
  - La última, $X \to 0 \mid 1 \mid ...$, permite colocar un número entre 0 y 9.
- $S = \{S\}$.

1.3 Ejercicio 3.

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

La gramática libre de contexto generada tiene la siguiente tupla:

- Las variables son $V = \{X, Y, Z, A, B, C, D, E, F, G, S\}$.
- Los símbolos terminales son $T = \{0, 1, 2\}$.
- $P = \left\{ \begin{array}{c} S \to 0XY1Z \mid 0A1BC2,\ X \to 0XY1 \mid 0E1 \mid \epsilon\ Y \to 1Y \mid 1,\ E \to DE1 \mid \epsilon \\ D \to 0D \mid 0,\ Z \to 2Z \mid \epsilon,\ A \to 0A \mid \epsilon,\ B \to 1BC2 \mid 1G2 \mid \epsilon\ C \to C2 \mid 2, \\ G \to 1GF \mid 1,\ F \to F2 \mid \epsilon \end{array} \right\}$,
  son las producciones, explicadas a continuación.
  - La producción $S \to 0XY1Z \mid 0A1BC2$ da lugar a una de las dos opciones del lenguaje, es decir, $i \neq j$ (la primera parte) o $j \neq k$ (la segunda parte).
  - Con las producciones X, Y, E y D nos encargamos de que los 0 y los 1 sean distintos, y hay tantas variables porque hay dos posibilidades: o hay más 0 que 1 o viceversa. Las producciones X e Y se encargan de que haya más 1 que 0. Las producciones D y E, de que haya más 0.

  - La producción Z se encarga de añadir todos los 2 deseados en el primer conjunto de palabras producidas por la gramática, es decir, añade los 2 cuando lo que se está teniendo en cuenta es que $i \neq j$.
  - La producción A se encarga de añadir todos los 0 que se deseen pero en el otro conjunto posible de palabras generadas por la gramática, es decir, cuando se está contando $j \neq k$. Es decir, esta producción es similar a la Z pero para la vertiente contraria.
  - Las producciones B, C, G y F contamos los 1 y 2 que estamos añadiendo, teniendo en cuenta, que al igual que en el caso anterior, hay dos posibilidades: que haya más 1 que 2, de lo que se encargan las producciones G y F, o que haya más 2 que 1, de lo que se encargan las producciones B y C.

- El símbolo inicial $S = \{S\}$.

1.4 Ejercicio 4.

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

- Hay 2 grupos de enemigos: grupos grandes (g) y grupos pequeños (p).
- Hay 2 tipos de monstruos: fuertes (f) y débiles (d).
- Los grupos grasndes 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 *"gfddddffffpdddfx"* 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?
Para cumplir con las restricciones, he creado la siguiente gramática:

- $V = \{G, P, X, Y, A, B, S\}$, donde
  - G sirve para crear grupos grandes,

  - P sirve para crear grupos pequeños,
  - X y B sirven para insertar los monstruos débiles e
  - Y y A sirven para insertar los monstruos fuertes.

- $T = \{g, p, f, d, x\}$, donde
  - g indica grupo grande,
  - p indica grupo pequeño,
  - f indica que es un monstruo fuerte,
  - d que es un monstruo débil y
  - x es la sala final de recompensas.

- $P = \left\{ \begin{array}{c} S \to Gx \mid Px,\ G \to gfdXP \mid gdfXP\ X \to dX \mid Y \mid \epsilon,\ \ Y \to fY \mid \epsilon, \\ P \to pA \mid \epsilon,\ A \to f \mid B,\ B \to dB \mid \epsilon \end{array} \right\}$, donde
  - La primera producción, $S \to Gx$, da lugar a niveles que tienen un grupo grande y añade al final la sala de recompensas.
  - La segunda, $S \to Px$, que crea niveles en los que no hay grupo grande, ya que tal y como yo lo entiendo, puede haber niveles en los que solo haya grupos grandes o grupos pequeños. Al igual que la anterior, añade al final la sala de recompensas.
  - Las tercera y cuarta son $G \to gfdXP \mid gdfXP$, que crea un grupo grande con los los dos primeros monstruos en cualquier orden, pero obligando a que haya uno débil y otro fuerte. Tras esto se pueden añadir más monstruos débiles y fuertes, o añadir un grupo pequeño al nivel.
  - La quinta producción, con la forma $X \to dX$, introduce todos los monstrueos débiles que se deseen.
  - La sexta y séptima, $X \to Y \mid \epsilon$, introducen monstruos fuertes o finalizan el nivel respectivamente.
  - Las dos que siguen, $Y \to fY \mid \epsilon$, insertan todos los monstruos fuertes que se deseen o finaliza el nivel, respectivamente.
  - La siguiente, $P \to pA \mid \epsilon$, crea un grupo pequeño donde se introduce 0 o 1 monstruo fuerte y monstruos débiles o finaliza el nivel sin introducir un grupo pequeño.
  - La producción $A \to f \mid B$ introduce un monstruo fuerte y los restantes débiles. Esta producción es llamada desde la producción que crea grupos pequeños.
  - Las dos últimas, $B \to dB \mid \epsilon$, introducen todos los monstruos débiles que se deseen o se finaliza el nivel, respectivamente.

- $S = \{S\}$, el símbolo inicial.

Esta gramática es independiente del contexto en la jerarquía de Chomsky. No puede ser de tipo 3 debido a que hay que añadir al final la sala de recompensas, y eso obliga a añadir un terminal al final de la regla, o bien, tener dos variables en las dos primeras reglas, lo que hace que no se cumplan las restricciones de forma de las gramáticas regulares.

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

Esta práctica la he realizado con mi compañero de clase, Vladislav Nikolov Vasilev. La memoria de esta práctica, por tanto, está en el documento de prácticas que él ha entregado.

3 Práctica 3: Ejercicios prácticos sobre autómatas y expresiones regulares

3.1 Ejercicio 1.

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.

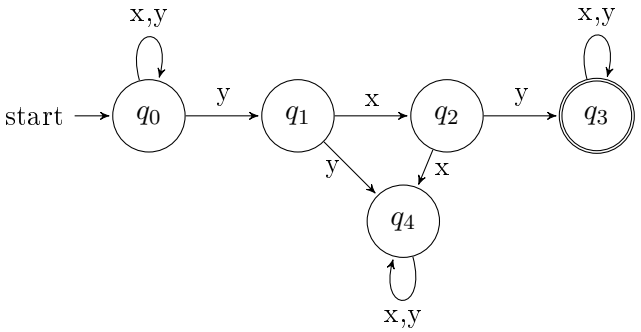
b) El lenguaje de las palabras que comienzan o terminan en xyx (o ambas cosas).

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

Paras que los autómatas sean deterministas, deben estar definidas todas las transiciones y no debe haber transiciones nulas. La resolución de los apartados es la que sigue:

a) El significado de los estados es el siguiente:

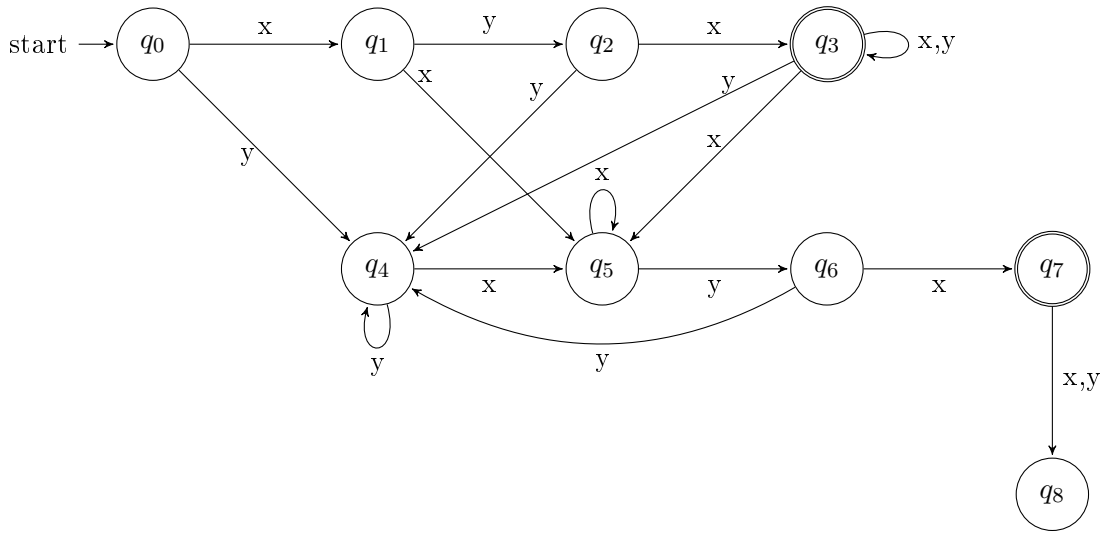
- q_0: cuando aún no se ha recibido ningún elemento de la subcadena.
- q_1: la primera y ya ha sido recibida.
- q_2: se ha recibido la x. Por ahora la subcadena formada es yx.
- q_3: la segunda y se ha recibido. La subcadena ya está completa y se pueden seguir recibiendo x e y si se desea.
- q_4: es un estado de error. Se llega a éste cuando se recibe una y en q_1 o una x en q_2.



b) En este caso, primero he construido un AFND y después he generado su AFD asociado, eliminando las transiciones nulas. El significado de los estados en el AFND son los siguientes:

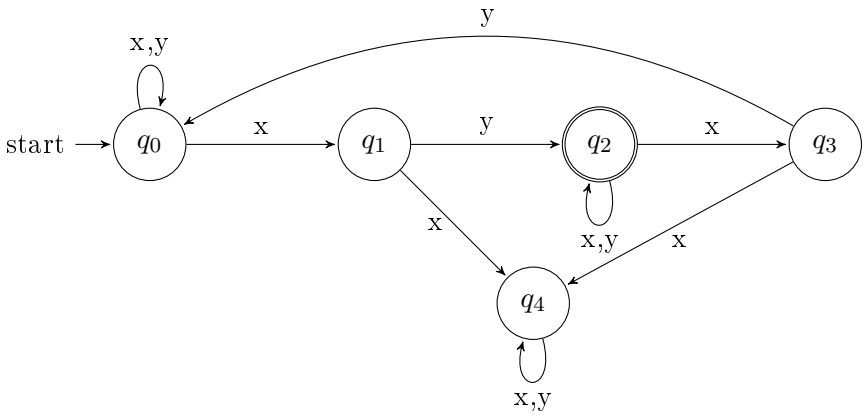
- q_0: es el estado inicial, aún no se ha recibido nada.

- q_1: es el estado para leer palabras que ***empiezan*** por la subcadena. En este estado se ha leído la primera x.
- q_2: se ha recibido la y.
- q_3: se ha recibido la segunda x. Una vez en este estado, la palabra ya puede ser aceptada puesto que comienza por la subcadena. Se pueden seguir introduciendo símbolos si se desea.
- q_4: este estado sirve como "transición" entre las cadenas que empiezan por la subcadena y las que acaban. Cuando entramos en este estado, o bien ya tenemos una cadena que empieza por la subcadena y estamos buscando si también acaba por dicha subcadena, o si, por el contrario, no tenemos una cadena que empiece por la subcadena y por tanto hay que buscar que acabe en xyx para poder aceptarla. En este estado se pueden recibir tantos símbolos como se desee.
- q_5: en este estado se ha recibido la primera x de la subcadena. Si estando en este estado llega una nueva x en lugar de una y, se vuelve al estado q_4.
- q_6: en este estado ya se ha recibido la y de la subcadena. Si estando en este estado recibimos una y en lugar de una x para poder seguir adelante, deberemos volver a empezar para buscar la cadena.
- q_7: en este estado se ha recibido la última x de la subcadena. La palabra se acepta pues, al menos, acaba por la subcadena xyx.
- q_8: es un estado de error. Se llega a este estado cuando en el estado q_7, que simboliza el final de la palabra recibe nuevos símbolos y por tanto la palabra deja de ser válida.



c) El significado de los estados es el siguiente:

- q_0: es el estado inicial, cuando aún no se ha recibido ningún elemento de la subcadena, o bien, actua como aceptor de la y cuando el número de apariciones de la subcadena es par.
- q_1: la x ya ha sido recibida.
- q_2: se ha recibido la y. En este estado se contabiliza que el número de veces que aparece la subcadena es impar. Es el estado en el que se aceptan las palabras. La palabra puede seguir recibiendo símbolos.
- q_3: comienza de nuevo la subcadena, por lo que el número es par. En este estado se recibe la x.
- q_4: es un estado de error. Se llega a éste cuando se recibe una x en q_1 o una x en q_3.



3.2 Ejercicio 2.

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

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

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

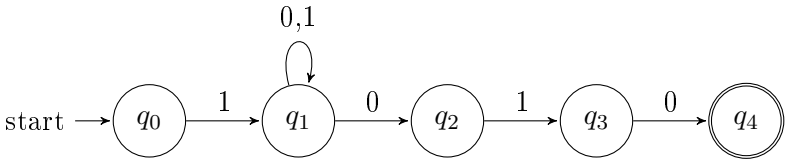
c) El lenguaje de las palabras que contienen, simultáneamente, las subcadenas 0101 y 100. El AFDN también acepta las cadenas en las que las subcadenas están solapadas (por ejemplo, "010100" y "100101" serían palabras aceptadas).

En este caso, los autómatas no tienen que tener todas las transiciones definidas, solo las que no interesan, y pueden contener trasnsiciones nulas. El diseño de cada autómata es el siguiente:

a) Los estados de este autómata se definen como sigue:

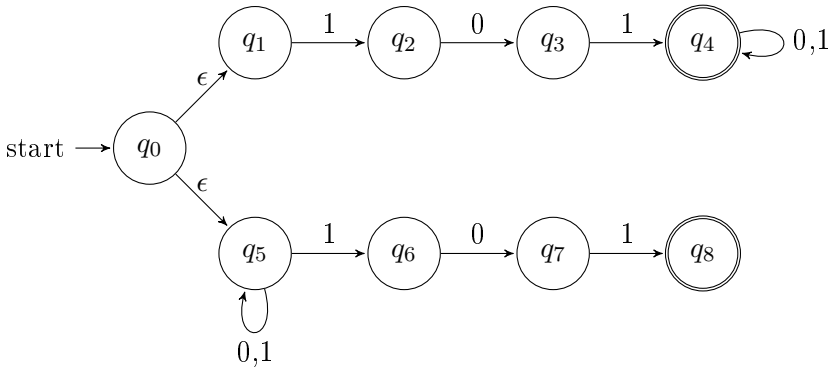
- q_0: cuando aún no se ha recibido ningún elemento de la subcadena.

- q_1: el primer 1 ya ha sido recibido. A partir de aquí, se pueden recibir tantos símbolos comose quiera, teniendo en cuenta que debe acabar por la subcadena 010.
- q_2: se ha recibido el primer 0 de la subcadena.
- q_3: el 1 de la subcadena ya ha sido recibido. La subcadena hasta el momento es 01.
- q_4: Se ha recibido el último símbolo de la subcadena y de la palabra, el 0. Este estado finaliza y acepta las palabras que han llegado hasta ahí.



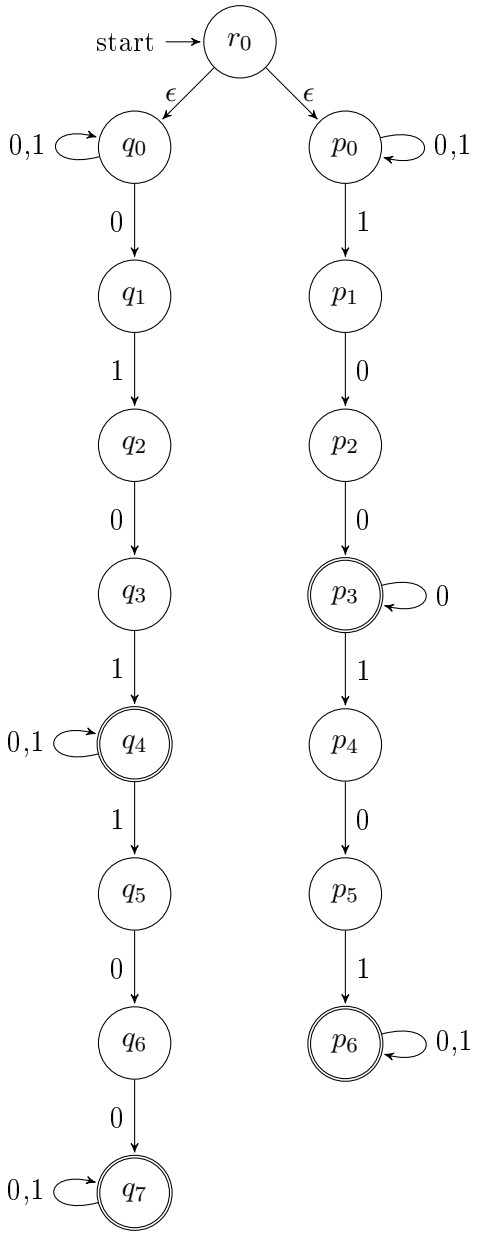
b) Los estados de este autómata son los siguientes:

- q_0: es el estado inicial, aún no se ha recibido nada.
- q_1: es el estado "pseudo-inicial" para leer palabras que ***empiezan*** por la subcadena.
- q_2: se ha recibido el primer 1.
- q_3: se ha recibido el 0. La subcadena formada es 10.
- q_4: se ha recibido el segundo 1, por lo que la subcadena está completa; se pueden seguir recibiendo símbolos si se desea.
- q_5: es el estado "pseudo-inicial" para leer palabras que ***acaban*** por la subcadena. En este estado se leen todos los símbolos que se desee antes de comenzar a leer la subcadena.
- q_6: al igual que ocurre en el estado q_2, en este estado se recibe el primer 1.
- q_7: se ha recibido el 0.
- q_8: se ha recibido el último 1. La subcadena ya ha sido recibida completa.



c) En este autómata, los estados se dividen en 2 grandes subconjuntos: los estados q_i y los estados p_i. Los q_i se encargan de buscar primero la cadena 0101 y después la subcadena 100, teniendo en cuenta que ambas pueden estar solapadas. Los estados p_i se encargan de encontrar primero la subcadena 100 y después la 0101, que, al igual que en el caso anterior, pueden estar solapadas. Los estados de este autómata son los siguientes:

- r_0: es el estado inicial, no se ha recibido nada aún.
- q_0: en este estado se leen todos los símbolos que se desee antes de comenzar a buscar la subcadena 0101.
- q_1: se ha recibido el primer 0 de la subcadena 0101.
- q_2: se ha recibido el primer 1 de la subcadena 0101.
- q_3: se ha recibido el segundo 0 de la subcadena 0101.
- q_4: se ha recibido el segundo 1, por lo que la subcadena 0101 está completa; se pueden seguir recibiendo símbolos si se desea. En este estado la palabra ya es aceptada puesto que se ha encontrado al menos una de las dos subcadenas (la 0101).
- q_5: se ha recibido el 1 de la subcadena 100.
- q_6: se ha recibido el primer 0 de la subcadena 100.
- q_7: se ha recibido el último 0 de la subcadena 100 por lo que ya está completa. En este estado se acepta la palabra porque contiene ambas subcadenas. Además, se pueden seguir recibiendo símbolos si se considera necesario.
- p_0: se pueden recibir tantos símbolos como se quiera. Aún no se ha leído ningún símbolo de ninguna de las cadenas.
- p_1: se ha recibido el 1 de la cadena 100.
- p_2: se ha recibido el primer 0 de la cadena 100.
- p_3: se ha recibido el segundo 0 de la cadena 100. Este estado es aceptor puesto que ya se ha encontrado una de las subcadenas en la palabra. Además, se pueden seguir recibiendo símbolos (0) si se desea. En el caso de que las subcadenas 100 y 0101 estén solapadas, este estado también es el receptor del primer 0 de la subcadena 0101.
- p_4: se ha recibido el primer 1 de la subcadena 0101.
- p_5: se ha recibido el segundo 0 de la subcadena 0101.
- p_6: se ha recibido el segundo 1 de las subcadena 0101, por lo que ya está completa. En este estado, se pueden aceptar las palabras ya que, en este caso, aparecen ambas subcadenas en la palabra. Se pueden seguir recibiendo símbolos si se desea.



3.3 Ejercicio 3.

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

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

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

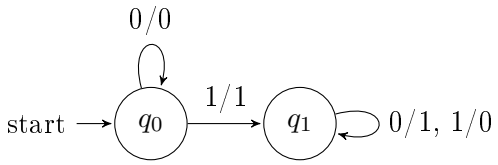
El diseño que he llevado a cabo es una máquina de Mealy con dos estados. El complemento a dos mantiene todos los bits iguales hasta el primer 1 (incluido) y a partir de éste, invierte los bits restantes. Por ejemplo:

$$Complemento_2(101110100) = 010001100$$

Por tanto, teniendo esto en cuenta, los significados de cada estado son:

- q_0: se reciben todos los 0 que se siguen codificando como 0. Cuando llega el primer 1, se codifica como 1 y se pasa al estado q_1.
- q_1: se reciben los restantes símbolos de la palabra: los 0 se codifican como 1 y los 1 como 0.

El diseño del autómata es el siguiente:



3.4 Ejercicio 4.

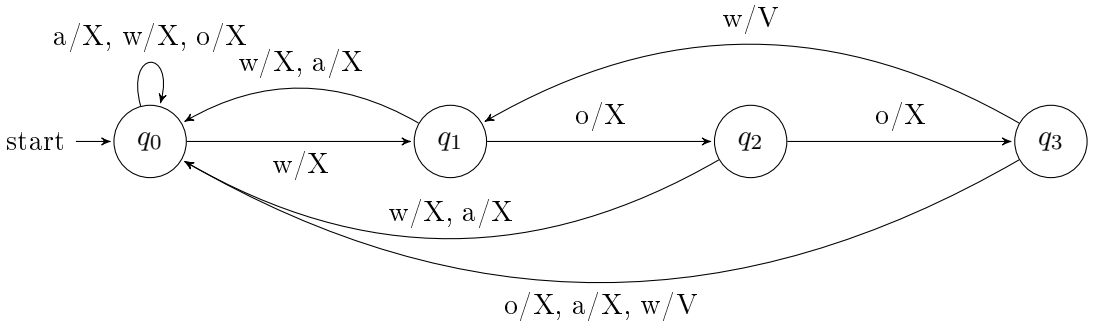
Diseñar una Máquina de Mealy o de Moore que, dada una cadena usando el alfabeto $A = a, w, o$, encienda un led verde (salida V) cada vez que se detecte la cadena "$woow$" en la entrada, apagándolo cuando lea cualquier otro símbolo después de esta cadena (representamos el led apagado con la salida "X"). El autómata tiene que encender el led verde (salida V), tantas veces como aparezca en la secuencia "$woow$" en la entrada, y esta secuencia puede estar solapada.

He diseñado una máquina de Mealy con 4 estados para controlar cuando llega la cadena y poder encender el led. El significado de los estados es:

- q_0: es el estado inicial. Se reciben todos los símbolos que se deseen. Cuando en los restantes estados se han leído los tres primeros símbolos de la subcadena (woo), este estado actúa como receptor de la última w, por lo que el led se encien y la salida pasa a ser V. Sea lo que sea que se reciba tras esta w, el led se apaga y pasa a ser X en la salida.
- q_1: en este estado se ha recibido la primera w de la subcadena. El led sigue apagado. Este estado también recibe la w final de la cadena, para contemplar en caso en el que las cadenas estén solapadas. Si se recibe una w porque están solapadas, el led se enciende y el símbolo siguiente que se recibe (la o), apaga el led y se continuan aceptando los símbolos de la subcadena.
- q_2: se ha recibido la primera o.

- q_3: se ha recibido la segunda *o*.

El autómata por tanto es el que sigue:



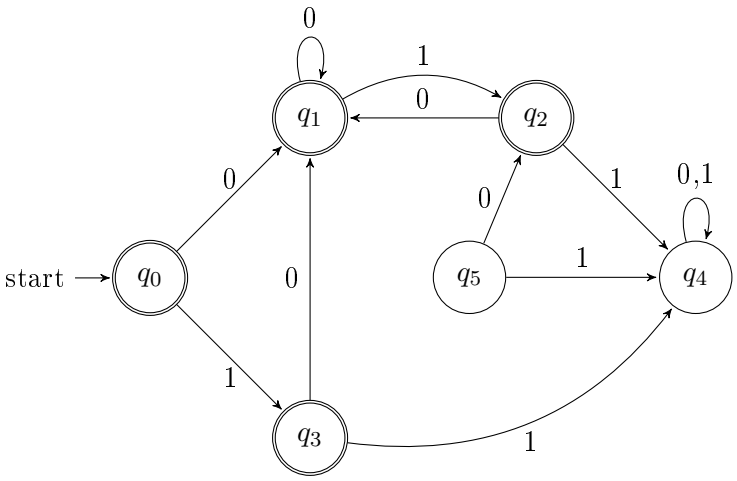
4 Práctica 4: Ejercicios prácticos sobre minimización

4.1 Ejercicio 1.

Dado el siguiente autómata responde a las siguientes cuestiones razonadamente:

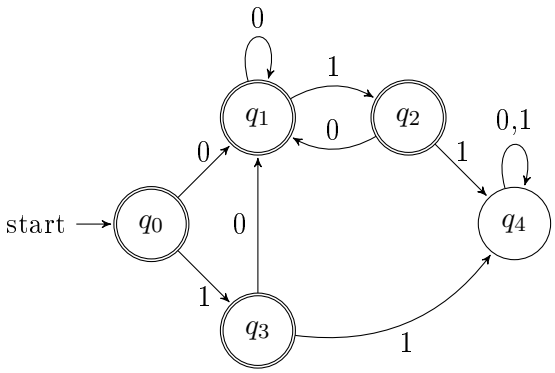
a) ¿Habría alguna forma de optimizar el autómata para que reduciendo su complejidad siga aceptando exactamente el mismo lenguaje?

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



a) El autómata se puede minimizar mediante el algoritmo de minimización. Los pasos son los siguientes:

1. Se eliminan los estados inaccesibles. En este autómata, como podemos observar, el estado q_5 es inaccesible, ya que a partir del estado inicial no se puede llegar a él de ninguna manera. Por tanto, lo eliminamos.



2. Se comparan todas las parejas de estados para encontrar los estados indistinguibles (en la tabla, las casillas con un guión no son válidas).

   I. Marcamos con una X todas las casillas de la tabla en las que sabemos que los estados son distinguibles, como es el caso del estado 4 con el resto de estados. Esto se debe a que los demás estados son finales pero el 4 no lo es; por tanto, son distinguibles obligatoriamente.

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

   II. Se comprueban todas las combinaciones de los demás estados con el estado 3. Tal y como se muestra en la segunda tabla, la pareja de estados (0,3) es distinguible, puesto que la pareja (3,4) es distinguible y dependían de ésta. La pareja (1,3) también es distinguible, puesto que dependía de la pareja (2,4), que hemos comprobado también que es distinguible. Por eso, tanto (0,3) como (1,3) se marcan con una X en la tabla. No obstante, la pareja (2,3) no sabemos si es distinguible o no, y por tanto, dejamos la casilla sin marcar (con un signo de interrogación rojo).

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

1		-	-	-
2			-	-
3	X	X	?	-
4	X	X	X	X
	0	1	2	3

   III. Comprobamos ahora las combinaciones de todos los estados con el estado 2. La pareja (0,2) es distinguible puesto que depende de la pareja (3,4) y ésta, como hemos comprobado antes, es distinguible; por tanto, se marca la tabla con una X. De forma análoga, la pareja (1,2) también es distinguible porque depende de la pareja (2,4), anteriormente comprobada.

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

1		-	-	-
2	X	X	-	-
3	X	X	?	-
4	X	X	X	X
	0	1	2	3

IV. Por último, comprobamos la pareja que queda, la (0,1). Esta depende de la pareja (2,3), la cual no hemos podido marcar como distinguible. Por tanto, ambas parejas son indistinguibles.

	0	1
0	1	3
1	1	2

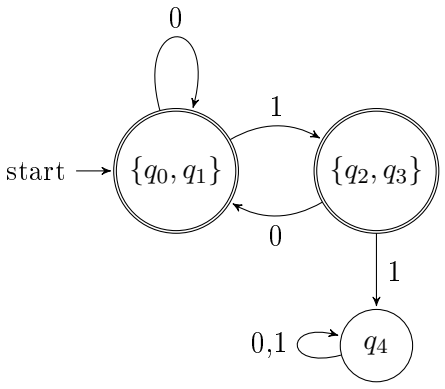
1	?	-	-	-
2	X	X	-	-
3	X	X	?	-
4	X	X	X	X
	0	1	2	3

Lo que significa que cada pareja se puede fusionar en un estado, ya que:

$$q_0 \equiv q_1$$

$$q_2 \equiv q_3$$

3. Se lleva a cabo la minimización.



b) La gramática que da lugar al lenguaje que acepta este autómata es la siguiente:

$$V = \{S, A, B\}$$

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

$$P = \{S \to A \mid B \mid \epsilon, A \to AA \mid B \mid 0 \mid \epsilon, B \to 1A \mid 1\}$$

$$S = \{S\}$$

Esta gramática puede ponerse en forma normal de Chomsky, eliminando las producciones nulas que hay y transformando las reglas para que esté de la forma $R_1 \to R_2R_3$ ó $R_1 \to c$, donde R_1, R_2 y R_3 son producciones y c es un símbolo terminal.

La gramática en Forma Normal de Chomsky quedaría de la siguiente manera:

$$P = \{S \to AA, A \to AA \mid YA \mid BA \mid 0, B \to YX, X \to 0, Y \to 1\}$$

4.2 Ejercicio 2.

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

a) $S \to AbB, A \to aA \mid \epsilon, B \to aB \mid bB \mid \epsilon$

b) $S \to abaS \mid babS \mid baS \mid \epsilon$

c) $S \to aSA \mid \epsilon, A \to bA \mid \epsilon$

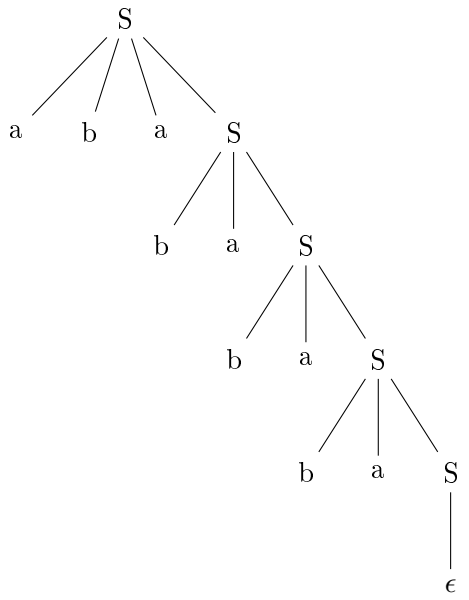
Nota: Explicar y demostrar cuidadosamente si la gramática no es ambigua (con lenguaje natural).

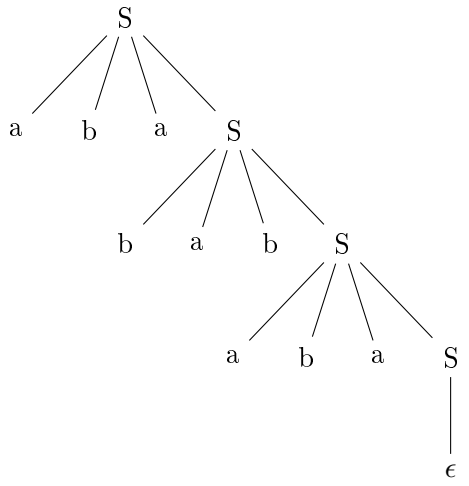
a) En este caso, la gramática no es ambigua. En las producciones dadas, el símbolo inicial genera la producción A, el terminal b y la producción B. Podría parecer ambigua debido a que tanto la producción A como la B generan un terminal a seguido de otra producción.

No obstante, no lo es por el siguiente motivo: las producciones A y B no tienen ninguna relación entre sí; esto es, cuando se utiliza la producción A, no hay forma de llegar a la producción B y viceversa. Esto hace que los árboles que se generan para cada palabra, automáticamente, sean únicos, ya que en el símbolo inicial ambas producciones son convocadas y cada una puede solo puede llamarse recursivamente a ella misma.

Comprobado que la gramática no es ambigua, el lenguaje por tanto tampoco es ambiguo.

b) Esta grámática sí es ambigua, ya que, por ejemplo, la palabra *abababa* puede ser generada con dos árboles distintos:





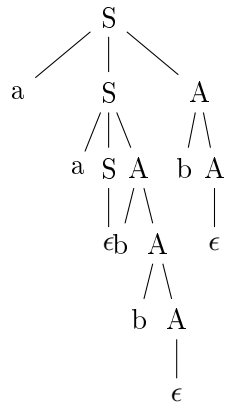
No obstante, el lenguaje no es ambiguo debido a que existe una gramática no ambigua que genera el mismo lenguaje. Esta nueva gramática se basa en "no poder volver hacia atrás", es decir, en el caso original, en el segundo árbol vemos como primero se utiliza la producción $abaS$ y en el tercer paso se vuelve a utilizar la misma, habiendo cambiado de producción entre el primer y el tercer paso. Con esta nueva gramática, una vez que se utiliza una producción, o bien se sigue utilizando esa hasta que se desee o bien no se utiliza, pero una vez que se cambia de producción

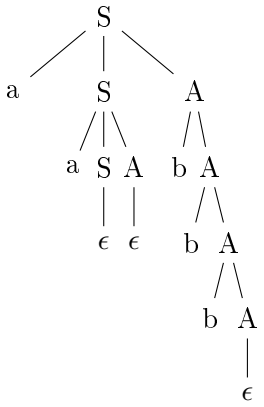
no se puede volver atrás. De esta forma evitamos que la gramática sea ambigua.

Las producciones de la nueva gramática son:

$$P = \left\{ \begin{array}{c} S \to abaX \mid babY \mid baZ, \\ X \to abaX \mid Y \mid Z, \\ Y \to babY \mid Z, \\ Z \to baZ \mid \epsilon \end{array} \right\}$$

c) Esta grámática también es ambigua, ya que la palabra *aabbb* puede ser generada con dos árboles distintos:





Pero, al igual que en el caso anterior, el lenguaje tampoco es ambiguo. Para crear una gramática no ambigua que genere el mismo lenguaje solo hay que ocuparse de que los árboles crezcan exclusivamente por la derecha (y no por el centro también, como es el caso) y obligar a que primero se introduzcan todas las a y después todas las b.

Las producciones de esta nueva gramática son las que siguen:

$$P = \left\{ \begin{array}{c} S \to aXA, \\ X \to aX \mid A \mid \epsilon, \\ A \to bA \mid \epsilon \end{array} \right\}$$

4.3 Ejercicio 3.

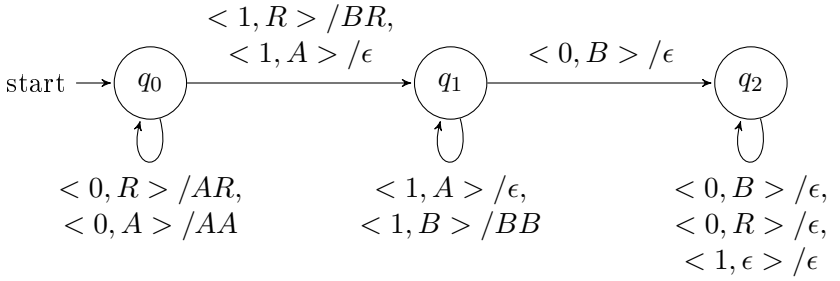
Encontrar el autómata que acepte el siguiente lenguaje L.

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

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

Para poder utilizar un autómata que genere este lenguaje, es necesario que sea un autómata con pila para poder contar el número de 0 y de 1 que estamos introduciendo.

Siguiendo el criterio de pila vacía, el autómata es el siguiente:



El significado de los estados de autómata es el siguiente:

- q_0: Se está recibiendo el primer bloque de 0.
- q_1: Se están recibiendo los 1, según el número de 0 que ha habido insertados antes. Cuando se han insertado todos los 1 según los 0 previos, se inserta en la pila el número de 1 extra que se están añadiendo.
- q_2: teniendo en cuenta el número de 1 extra que hemos añadido anteriormente, se añaden los 0 necesarios. Al final, cuando la pila ya está vacía, se añade el último 1.

Tras dibujar el autómata, extraeremos la gramática. El autómata queda tal que:

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{A, B, R, \epsilon\}, \delta, q_0, R, \varnothing)$$

donde

$$\begin{array}{ll} \delta(q_0, 0, R) = \{(q_0, AR)\} & \delta(q_1, 1, B) = \{(q_1, BB)\} \\ \delta(q_0, 0, A) = \{(q_0, AA)\} & \delta(q_1, 0, B) = \{(q_2, \quad \epsilon)\} \\ \delta(q_0, 1, R) = \{(q_1, BR)\} & \delta(q_2, 0, B) = \{(q_2, \quad \epsilon)\} \\ \delta(q_0, 1, A) = \{(q_1, \quad \epsilon)\} & \delta(q_2, 0, R) = \{(q_2, \quad \epsilon)\} \\ \delta(q_1, 1, A) = \{(q_1, AA)\} & \delta(q_2, 1, \quad \epsilon) = \{(q_2, \quad \epsilon)\} \end{array}$$

5 Anexo

5.1 Semana del 24 de septiembre de 2018

5.1.1 Ejercicio 1.

Describir el lenguaje generado por la siguiente gramática,

$$S \to XYX$$
$$X \to aX \mid bX \mid \varepsilon$$
$$Y \to bbb$$

Esta gramática tiene los siguientes elementos definidos como:

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

$$T = \{a, b\}$$
$$S = \{S\}$$

Con los elementos anteriores y las producciones dadas, el lenguaje genera palabras que constan de una serie de símbolos a ó b, seguidas por 3 símbolos b consecutivos y por último, otra serie de símbolos a ó b. Por ejemplo, se producen palabras como:

- $S \to XYX \to aXYaX \to aaXYabX \to aabXYXaba$

$$Palabra_{producida} = aabbbbaba$$

- $S \to Y$ (utilizando $X \to \epsilon$)

$$Palabra_{producida} = bbb$$

Como hemos comprobado en los ejemplos anteriores, la expresión regular del lenguaje generado es:

$$\mathcal{L} = (a + b)^* \cdot bbb \cdot (a + b)^*$$

5.1.2 Ejercicio 2.

Describir el lenguaje generado por la siguiente gramática,

$$S \to SS \mid XaXaX \mid \varepsilon$$
$$X \to bX \mid \varepsilon$$

La gramática anterior tiene los elementos S, V y T definidos como:

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

$$T = \{a, b\}$$
$$S = \{S\}$$

Con los elementos descritos y las producciones que se dan, el lenguaje genera palabras que constan de una serie (0 o más) de símbolos b, seguidos de un símbolo a, seguido de 0 o más símbolos b, sucedidos de otra a y por último seguida por 0 o más b. Esta secuencia se puede repetir 0 o más veces. Por ejemplo, se producen palabras como:

- $S \to XaXaX \to bXabXa \to bbXaba \to bbbXaba$

$$Palabra_{producida} = bbbaba$$

- $S \to \epsilon$

$$Palabra_{producida} = \epsilon$$

- $S \to SS \to XaXaXS \to XaXaX\epsilon$ (sustituyendo las X y la segunda S por ϵ)

$$Palabra_{producida} = aa$$

Por tanto, la expresión regular del lenguaje generado es:

$$\mathcal{L} = (b^* \cdot a \cdot b^* \cdot a \cdot b^*)^*$$

5.1.3 Ejercicio 3.

Piensa en un problema de la vida real en el que se pudiese aplicar los conocimientos vistos hoy en clase. Al igual que el ejemplo de la oficina de hoy, imagina y describe una gramática que se pueda utilizar para resolver dicho problema real.

Se puede hacer para un problema de logística donde se contronlen los caminos y las rutas por las que viajan los camiones/furgonetas. Para simplificar el problema, supondremos que hay, por ejemplo, cuatro rutas (R) posibles: a, b, c y d, y que están conetadas entre ellas, es decir, que un camión puede comenzar desde la ruta inicial i hacia, por ejemplo, la ruta b y de ésta pasar a la ruta d, y que el camión puede repetir las rutas todas las veces que desee.

La gramática se define como sigue:

$$V = \{A, B, C, D, S\}$$
$$T = \{a, b, c, d, i\}$$
$$P = \{S \to iR \mid \epsilon, R \to aR \mid bR \mid cR \mid dR \mid \epsilon\}$$
$$S = \{S\}$$

5.2 Semana del 1 de octubre de 2018

5.2.1 Ejercicio 4.

Encontrar si es posible una gramática lineal por la derecha o una gramática libre de contexto que genere el lenguaje L, en cada uno de los casos, supuesto que $L \subseteq a, b, c^*$ y verifica que:

- $u \in L$ si y solamente si verifica que u no contiene dos símbolos b consecutivos.

  La tupla (V, T, P, S) se define así:

  $$V = \{S, X, Y\}$$
  $$T = \{a, b, c\}$$
  $$P = \{S \to X \mid bX \mid c \mid \epsilon, X \to aX \mid aY \mid \epsilon, Y \to bX\}$$
  $$S = \{S\}$$

  Con una grámatica con las producciones anteriores es imposible generar palabras con dos b consecutivas, debido a que cada vez que se inserta una b, se obliga a utilizar inmediamente después la producción $X \to aX$ o la producción $X \to aY$, de forma que siempre se inserta una a tras cada b. En este caso no he complicado más la gramática y practimente no he tenido en cuenta el símbolo c exceptuando una cadena formada por una sola c que surge de la producción S.

- $u \in L$ si y solamente si verifica que u contiene dos símbolos b consecutivos.

  En este caso, la tupla (V, T, P, S) que se forma es:

  $$V = \{S, X, Y, Z\}$$
  $$T = \{a, b, c\}$$
  $$P = \{S \to X \mid Y, X \to bXa \mid bXc \mid XY \mid YX \mid bZ \mid bZa \mid bZc,$$
  $$Y \to aYb \mid cYb \mid XY \mid YX, Z \to \epsilon\}$$
  $$S = \{S\}$$

  En este apartado todas las palabras que se crean están obligadas a tener dos o más b consecutivas debido a las producciones de la gramática. En este caso, la producción Y no tiene ϵ por lo que se ve obligada a volver a introducir una producción X. Ésta a su vez tiene que recurrir a una producción Z que obliga a introducir 2 b consecutivas.

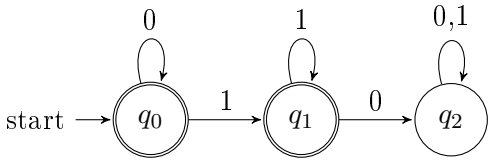
- $u \in L$ si y solamente si verifica que contiene un número impar de símbolos c.
  La tupla (V, T, P, S) que se forma es:

$$V = \{S, X, Y, Z\}$$
$$T = \{a, b, c\}$$
$$P = \{S \to X \mid Y, X \to bXa \mid bXc \mid XY \mid YX \mid bZ \mid bZa \mid bZc,$$
$$Y \to aYb \mid cYb \mid XY \mid YX, Z \to \epsilon\}$$
$$S = \{S\}$$

5.3 Semana del 8 de octubre de 2018

5.3.1 Ejercicio 5.

¿Qué lenguaje genera el siguiente autómata?



El anterior autómata produce cadenas que solo contienen 0 o que contienen 0 y 1 y acaban en 1. En el caso de que la cadena contenga 0 y 1 contenga 0 después del último 1, no se acepta. La expresión regular del lenguaje es:

$$\mathcal{L} = (0 + 1)^* \cdot 1$$

5.3.2 Ejercicio 6.

Sea $M = (Q, 0, 1, q_0, \delta, F)$, diseñar el autómata M en los siguientes casos:

1. que reconozca todas las palabras que **no** empiezan por 00.
2. que reconozca todas las palabras que **no** empiezan por 00, pero **sí** contengan 00.
3. diseñar el autómata que acepte las palabras que rechaza el AFD anterior.

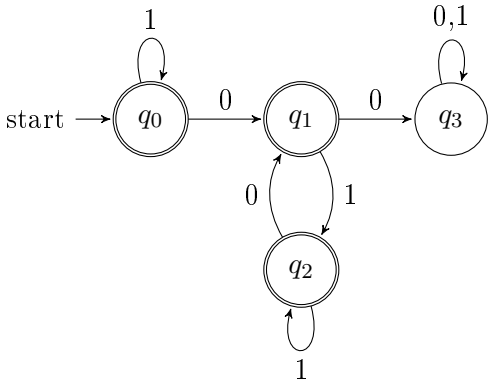
Definir para dichos autómata Q, q_0, δ, F.

1. Primer autómata

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$A = \{0, 1\}$$
$$estado_{inicial} = q_0$$

$$\delta = \begin{bmatrix} (q_0, 0) = q_1 & (q_1, 0) = q_3 & (q_2, 0) = q_1 & (q_3, 0) = q_3 \\ (q_0, 1) = q_0 & (q_1, 1) = q_2 & (q_2, 1) = q_2 & (q_3, 1) = q_3 \end{bmatrix}$$

$$F = \{q_0, q_1, q_2\}$$



2. Segundo autómata

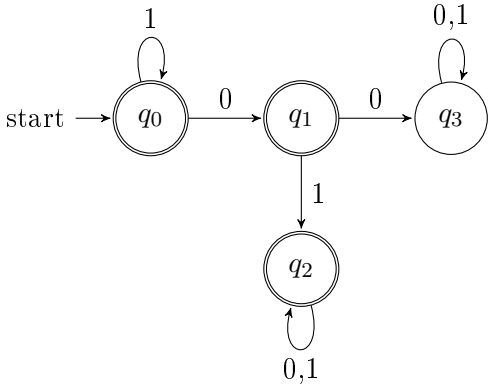
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$A = \{0, 1\}$$

$$estado_{inicial} = q_0$$

$$\delta = \begin{bmatrix} (q_0, 0) = q_1 & (q_1, 0) = q_3 & (q_2, 0) = q_2 & (q_3, 0) = q_3 \\ (q_0, 1) = q_0 & (q_1, 1) = q_2 & (q_2, 1) = q_2 & (q_3, 1) = q_3 \end{bmatrix}$$

$$F = \{q_0, q_1, q_2\}$$



3. Tercer autómata

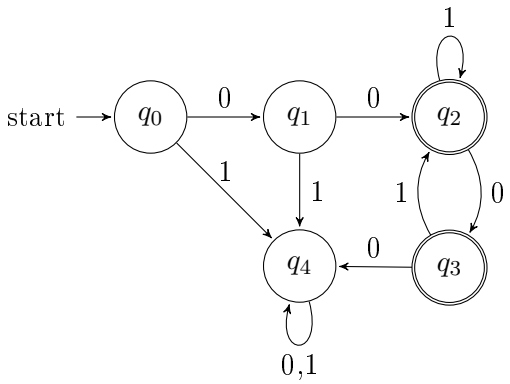
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$A = \{0, 1\}$$

$$estado_{inicial} = q_0$$

$$\delta = \begin{bmatrix} (q_0,0) = q_1 & (q_1,0) = q_3 & (q_2,0) = q_2 & (q_3,0) = q_3 \\ (q_0,1) = q_0 & (q_1,1) = q_2 & (q_2,1) = q_2 & (q_3,1) = q_3 \end{bmatrix}$$

$$F = \{q_0, q_1, q_2\}$$



5.4 Semana del 15 de octubre de 2018

5.4.1 Ejercicio 7.

Diseñar un autómata que acepte los siguientes lenguajes. En todos los casos, en el alfabeto 0,1.

1. Lenguaje que acepta las cadenas de mínima longitud 2, cuyos primeros símbolos son el mismo.

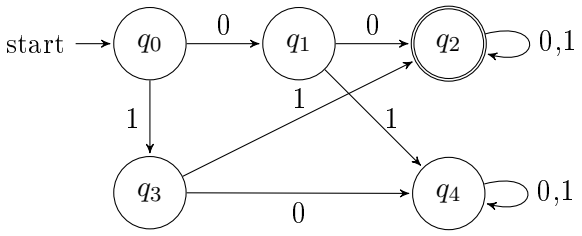
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$A = \{0, 1\}$$

$$estado_{inicial} = q_0$$

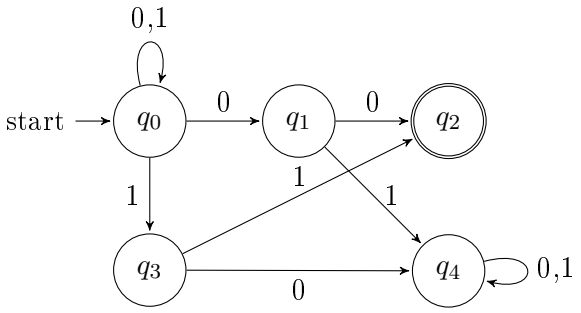
$$\delta = \begin{bmatrix} (q_0,0) = q_1 & (q_1,0) = q_2 & (q_2,0) = q_2 & (q_3,0) = q_4 & (q_4,0) = q_4 \\ (q_0,1) = q_3 & (q_1,1) = q_4 & (q_2,1) = q_2 & (q_3,1) = q_2 & (a_4,1) = q_4 \end{bmatrix}$$

$$F = \{q_2\}$$



2. Lenguaje cuyas cadenas tienen una longitud mínima de 2, y cuyo últimos símbolos son los mismos.

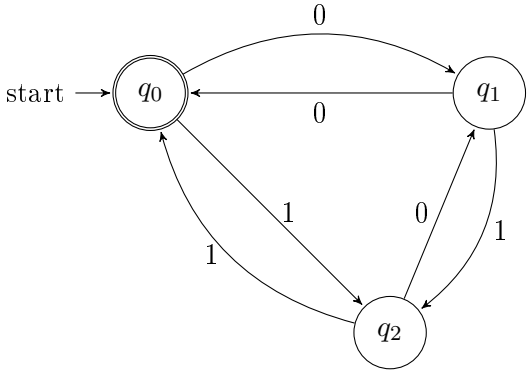
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$A = \{0, 1\}$$
$$estado_{inicial} = q_0$$
$$\delta = \begin{bmatrix} (q_0, 0) = \{q_0, q_1\} & (q_1, 0) = q_2 & (q_2, 0) = q_2 & (q_3, 0) = q_4 & (q_4, 0) = q_4 \\ (q_0, 1) = \{q_0, q_3\} & (q_1, 1) = q_4 & (q_2, 1) = q_2 & (q_3, 1) = q_2 & (a_4, 1) = q_4 \end{bmatrix}$$
$$F = \{q_2\}$$



5.4.2 Ejercicio 8.

Autómata con un número impar de 0 y con un número impar de 1.

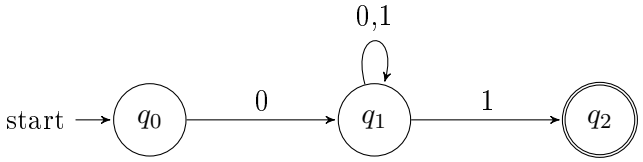
$$Q = \{q_0, q_1, q_2\}$$
$$A = \{0, 1\}$$
$$estado_{inicial} = q_0$$
$$\delta = \begin{bmatrix} (q_0, 0) = q_1 & (q_1, 0) = q_0 & (q_2, 0) = q_1 \\ (q_0, 1) = q_3 & (q_1, 1) = q_3 & (q_2, 1) = q_0 \end{bmatrix}$$
$$F = \{q_0\}$$



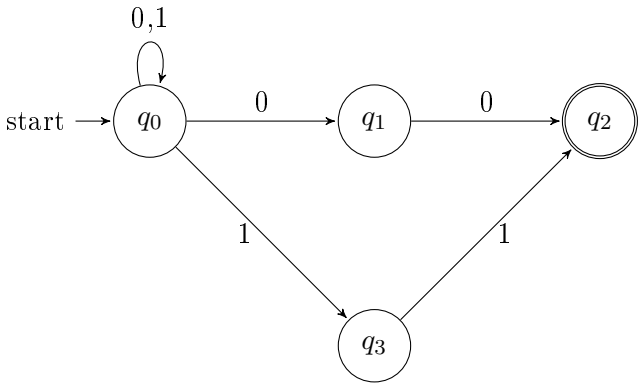
5.4.3 Ejercicio 9.

Diseñar un AFND para los siguientes lenguajes. **Cada AFND debería respetar el número de estados y de transiciones**. Cada transición con 2 símbolos cuenta como dos transciones. En todos los casos el alfabeto es 0,1.

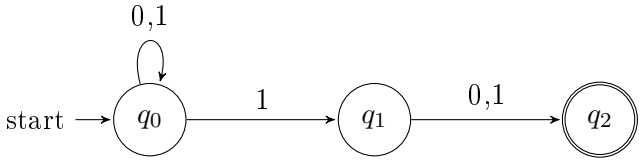
1. L acepta las cadenas de longitud mayor o igual que 2 que empiezan en 0 y terminan en 1. No más de 3 estados y 4 transiciones.



2. L es el lenguaje que contiene las adenas cuya longitud es al menos 2 y que sus dos últimos símbolos son el mismo. No más de 4 estados y 6 transiciones.



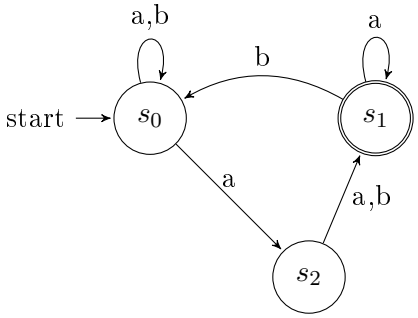
3. L es el lenguaje cuyas cadenas son al menos de longitud 2 y que tienen un 1 en la penúltima posición. No más de 3 estados y 5 transciones.



5.5 Semana del 22 de octubre de 2018

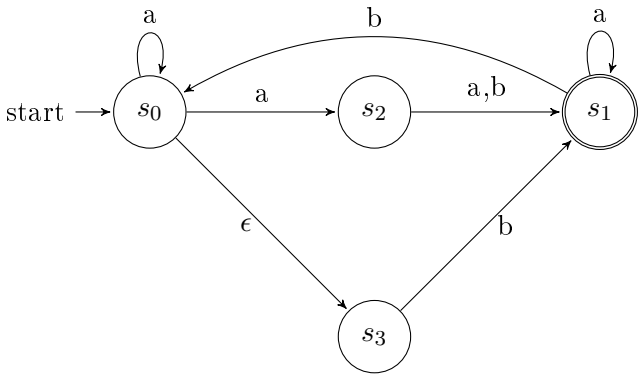
5.5.1 Ejercicio 10.

Pasar a un autómata finito determinista:



5.5.2 Ejercicio 11.

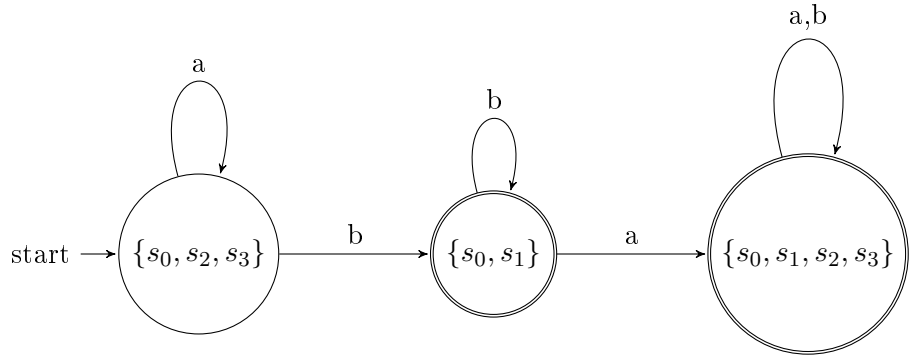
Pasar a AFD



Para poder hacer el AFD, rellenamos la tabla con los estados para saber qué nuevos estados se crearán en el AFD.

El procedimiento a seguir es el siguiente: se extraen por ejemplo los estados de s_0 a los que le entra una a y se comprueba a qué estados se desplazan dichos estados cuando les entra el mismo símbolo. De ahí se crea un estado con el conjunto de los estados anteriores. Si alguno de los estados de este conjunto es final, el estado resultante también lo es. Debajo de la tabla se muestra el diseño del AFD siguiendo este procedimiento.

	a	b
s_0	$\{s_0, s_2, s_3\}$	$\{s_0, s_1\}$
s_1	s_1	s_0
s_2	s_1	s_1
s_3	s_0	s_1



5.6 Semana del 29 de octubre de 2018

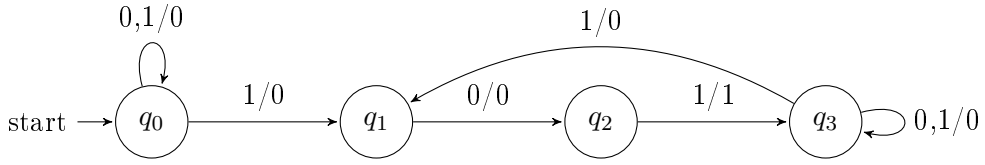
5.6.1 Ejercicio 12.

Diseñar un autómata finito de estados que dada una entrada en el alfabeto 0,1, escriba en la salida 1 si encuentra el patrón de entrada 101, en caso contrario que escriba 0.

1. Realizar el diseño del autómata en papel.
2. Implementar dicho autómata en el lenguaje de programación que se desee.

El desarrollo del ejercicio es el que sigue:

1. Para diseñar el autómata, se tiene en cuenta que se escribe 1 cada vez que encuentra la secuencia 101, y 0 en el resto de casos; por tanto, el diseño del autómata es el siguiente:



2. El código en C++ es el siguiente:

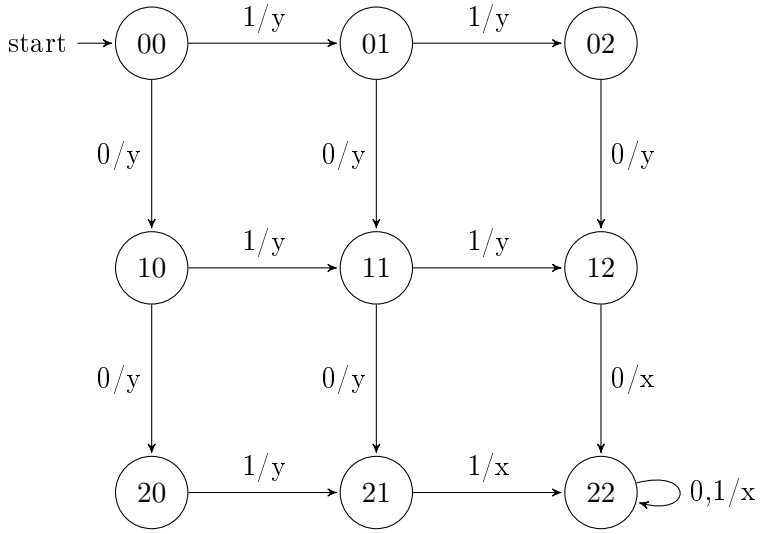
```
1   while(true)
2   {
3     // Se busca la secuencia
4     if(in[i] == '1' && in[i+1] == '0' && in[i+2] == '1')
5       out[j] = 1;
6
7     else
8       out[j] = 0;
9
10    // Se aumentan los indices de entrada y salida
11    i++;
12    j++;
```

```
13    }
14
```

5.6.2 Ejercicio 13.

Diseñar un autómata finito de estados con una salida y con una entrada. La salida se pone a 'x' cuando al menos ha encontrado dos '1' y dos '0' en la entrada, sin importar el orden de aparición.

Este autómata se va a diseñar con forma de matriz de manera que se vea claramente el funcionamiento. Cuando no se cumplen las condiciones, se escribe y en la salida. Una vez conseguidas las condiciones, se escribe x en la salida. El diseño es este:



5.7 Semana del 5 de noviembre de 2018

5.7.1 Ejercicio 14.

Indicar si los siguientes lenguajes son regulares o no. Justificando la respuesta.

1. $L_1 = \{0^i 1^j : j \leq i\}$

$$z \in (L) \implies A = \{0, 1, c\} \exists n \in \mathbb{N} / z = 0^n 1^m = uvwdonden = m$$