Procesadores del lenguaje

Práctica 1: Expresiones Regulares

Vega Martín Mellado - 05955285X

Septiembre 2024

Índice

1. Introducción

El objetivo principal de esta práctica es aprender a usar y transformar Expresiones Regulares (ER). Con este objetivo en mente, la práctica propone programar un sistema para evaluarlas y otro para generar cadenas válidas.
Para ello se plantean varios pasos a seguir:

1. **Seleccionar un alfabeto.** Este alfabeto deberá estar formado por, al menos, tres caracteres distintos.
2. **Proponer dos ERs.** Para estas ERs es necesario usar el alfabeto anterior y explicar a qué tipo de expresiones responden cada ER. A la hora de plantear cada ER hay que tener en cuenta de que no pueden tener menos de 8 símbolos cada una sin contar paréntesis y que tanto los cierres + y * y la OR deben estar presentes al menos una vez.
3. **Pasar de la ER hasta la matriz de estados.** Partiendo de la ER, primero se creara un Autómata Finito No Determinista (AFND). A partir de este se generará el Autómata Finito Determinista (AFD), que se simplificará. Por último, se obtendrá la matriz de estado correspondiente a cada ER.
4. **Implementación del programa.** Dada la matriz de estados, hay que implementar una máquina que permita realizar las siguientes dos operaciones:

   a) Dada una cadena de texto de entrada, analizarla para determinar si esa cadena de texto cumple con la ER original.

   b) Dar todas las posibles cadenas de texto de entradas válidas, hasta un número máximo determinado configurable (p.ej.100), que no sobrepasen una longitud máxima configurable (p.ej.10 caracteres).

   Este programa hay que implementarlo dejando abierta la posibilidad de modificar las matrices de estados de las ER.

En esta memoria se explicarán los pasos tomados para desarrollar la práctica, desde la selección y explicación del alfabeto y las ERs, pasando por la creación de los AFND y AFD para obtener las matrices de estado, hasta llegar a la implementación del código. Por último, se hará un breve resumen de cómo funciona el programa.

2. Tareas

Las tareas o pasos necesarios para resolver esta práctica se pueden agrupar en tres tareas generales que se desglosarán poco a poco. La primera tarea engloba la preparación de la práctica. Por otro lado, la segunda tarea engloba la creación de las matrices de estados de ambas ERs. Y, por último, la tercera tarea explica la implementación del programa.

2.1. Selección de alfabeto y de expresiones regulares

Como se ha dicho en la introducción, es imperativo, que el alfabeto tenga al menos 3 elementos, por tanto, y para no complicar mucho la práctica, se utilizará el alfabeto más simple {a, b, c}.
En cuanto a las Expresiones Regulares, se utilizarán las siguientes dos:

- Que haya al menos tres a seguidas en el lexema:

$$(a|b|c) * aaa + (a|b|c)*$$

- Que el lexema esté formado únicamente por secuencias 'abc' y 'cba':

$$((abc) * |(cba)*)+$$

2.2. Creación de el AFD y de la matriz de estados

El primer paso para crear el Autómata Finito Determinista es traducir las Expresiones Regulares. Esto se debe a que vamos a utilizar el programa JFLAP7.1 para crearlo y este no usa los mismos símbolos. A continuación, se introducirá la ER traducida en el programa para obtener el AFND, a partir del cuál se obtendrá el AFD y se minimizará para que nosotros podamos generar la matriz de estados.

2.2.1. Primera Expresión Regular

Como se ha comentado anteriormente, el primer paso es traducir la primera ER (que haya al menos tres a seguidas en el lexema $\rightarrow (a|b|c) * aaa + (a|b|c)*$). Una vez traducida, queda de la siguiente forma:

$$(a + b + c) * aaa(! + a) * (a + b + c)*$$

Una vez introducimos esta ER en el programa nos genera un AFND con 35 estados distintos. De estos 35 estados, solo hay 1 estado inicial y 1 estado final, como se muestra en la figura 1. Esto tiene mucho sentido ya que, como se muestra en la figura, hay muchos saltos que se pueden dan sin que se introduzca ningún carácter. También podemos var cómo se forman caminos alternativos por los que se podría pasar según el carácter recibido.

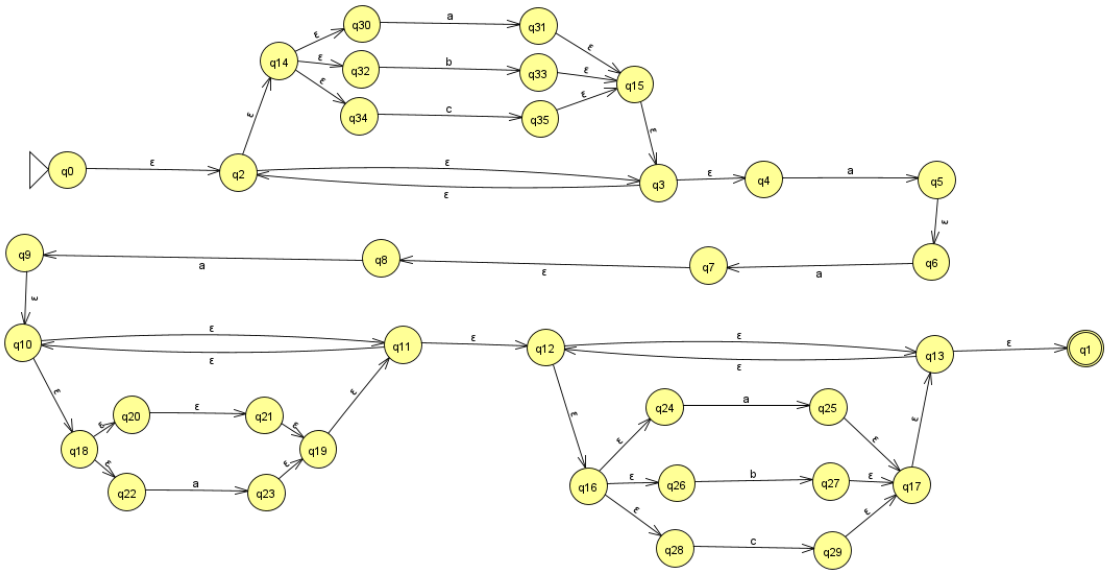
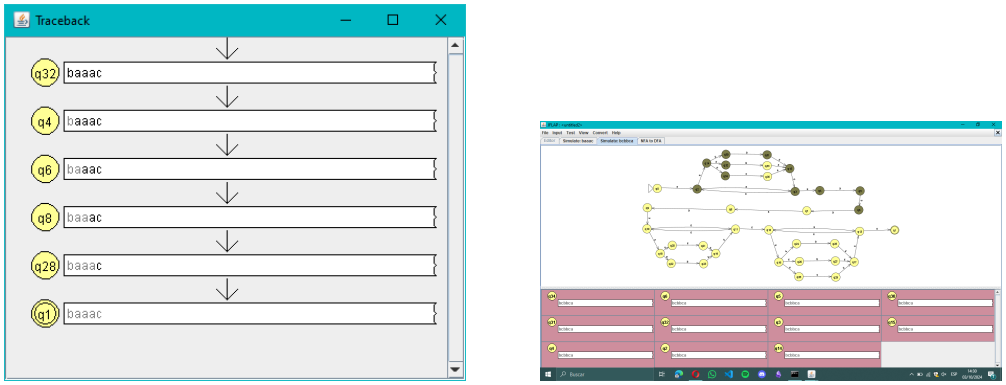


Figura 1: AFND de la primera ER.

Para comprobar que funciona como es debido, vamos a realizar dos comprobaciones rápidas. Para ello vamos a introducir dos lexemas, en uno de ellos el AFND tiene que acaba en q1 (el estado final), mientras que con el segundo lexema no tendría que llegan nunca al estado final. Para ello, usamos los lexemas 'baaac' y 'bcbbca'. Como se observa en las figuras 2a y 2b, el AFND funciona como se espera.



(a) Lexema correcto

(b) Lexema incorrecto

Figura 2: Pruebas realizadas al AFND 1

A continuación vamos a generar el AFD correspondiente. Con la aplicación JFLAP esto se puede conseguir con unos poco clicks del ratón, dándonos el resultado de la figura 3. Esto unificará o creará nuevos estados dependiendo de los saltos que den en función de la entrada que se reciba. Por tanto es posible que 5 estados del AFND se condensen en dos estados del AFD. Al igual que es posible que aparezcan más estados iniciales de los que había en el AFND, como es el caso.

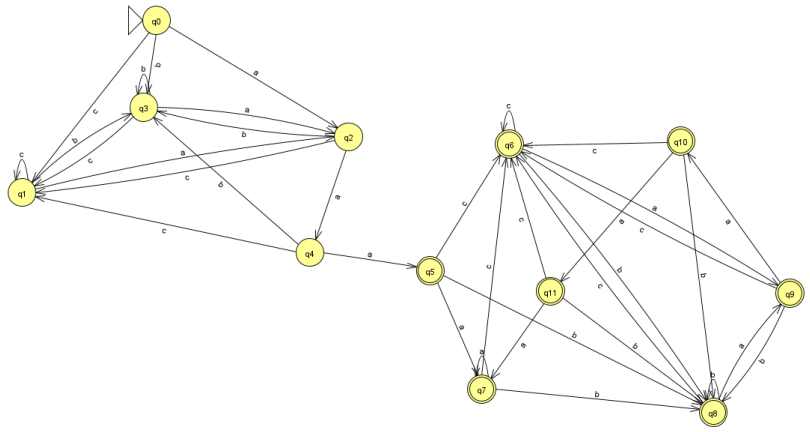


Figura 3: AFD de la primera ER.

El grafo obtenido es bastante más pequeño que el AFND inicial, ya que cuenta con únicamente con 11 estados, de los cuales 1 es inicial y 7 son finales. Por desgracia para nosotros, este AFD sigue siendo muy lioso como para obtener la matriz de estados a partir de él. Es por este motivo que necesitamos minimizarlo, para así juntar/simplificar estados repetidos y obtener el grafo de la figura 4. Esta operación sigue siendo muy fácil con la ayuda de JFLAP.

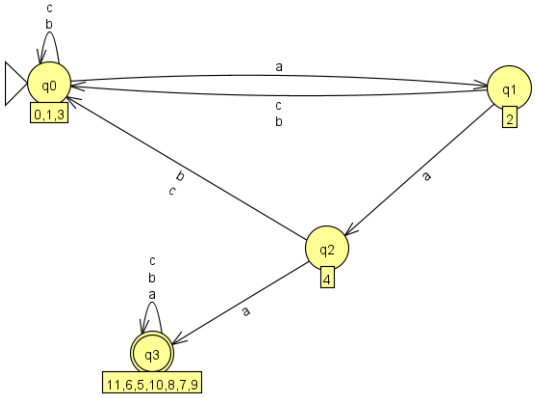


Figura 4: AFD minimizado de la primera ER.

Como se observa en la figura 4, hemos pasado de un grafo de 35 estados, a un grafo de solo 4 gracias a condensar unos estados en otros y simplificar estados que llevan al mismo punto del grafo. A partir de este AFD minimizado, ya podemos generar la matriz de estados más fácilmente. Para ello se recorrerá cada estado anotando en una tabla a qué estado se debe de saltar dependiendo del carácter de entrada, quedando la tabla 1.

x	a	b	c
q0	q1	q0	q0
q1	q2	q0	q0
q2	q3	q0	q0
q3	q3	q3	q3

Tabla 1: Matriz de estados de la primera ER.

2.2.2. Segunda Expresión Regular

Para la segunda Expresión Regular se lleva a cabo el mismo proceso que con la primera. Por tanto, el primer paso es traducir la ER original $(((abc) * |(cba)*)+)$, quedando:

$$((abc) * +(cba)*)(! + ((abc) * +(cba)*))*$$

Una vez tenemos la ER traducida, generamos el AFND con JFLAP que nos devolverá el grafo de la figura 5. En este caso también se observan caminos alternativos que dependiendo la entrada, pero lo que más llama la atención es que si entra en uno de los caminos, tiene que cumplir muchas otras condiciones para llegar al final de la rama.

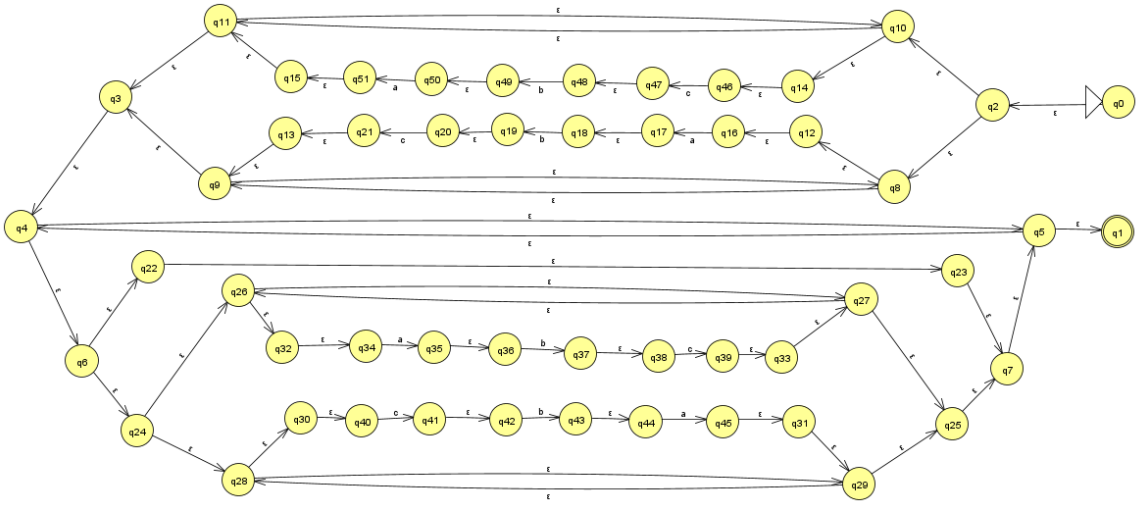
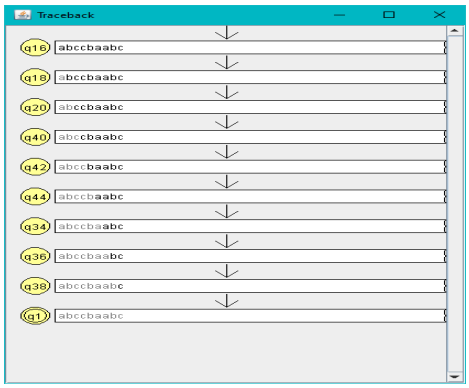


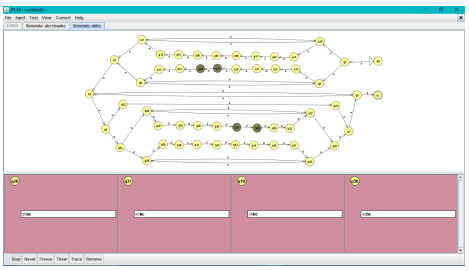
Figura 5: AFND de la segunda ER.

Para comprobar que hemos creado el AFND correcto vamos a introducir los lexemas 'abccbaabc' y 'abbc' que deberían ser correcto e incorrecto, respectivamente. Como vemos en las figuras 6a y 6b, el autómata funciona como se espera, por tanto, el siguiente paso es crear el AFD.

Como se puede observar en la figura 7 el grafo se ha simplificado bastante al crear el AFD. Los siguientes pasos son minimizar este AFD para eliminar los estados repetidos y crear la tabla con la matriz de estados.



(a) Lexema correcto



(b) Lexema incorrecto

Figura 6: Pruebas realizadas al AFND 2

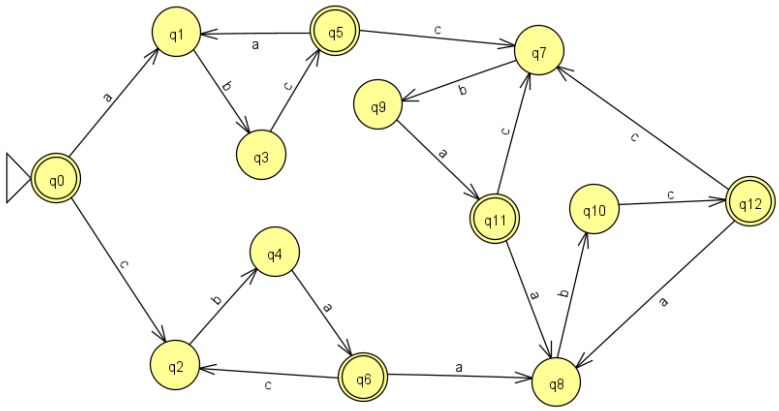


Figura 7: AFND de la segunda ER.

A partir del AFD minimizado (figura 8), generamos la tabla con la matriz de estados. Esta matriz representa a qué estado debe saltar el autómata dependiendo de en qué estado se encuentre y qué entrada reciba. En este caso esto se ve representado en la tabla 2. Los huecos en la tabla son saltos no válidos, es decir, que llevarían a un error (en el código se representaran con el estado -1 para marcar el error y simplificarlo).

x	a	b	c
q0		q1	
q1			q4
q2		q3	
q3	q4		
q4	q0		q2

Tabla 2: Matriz de estados de la segunda ER.

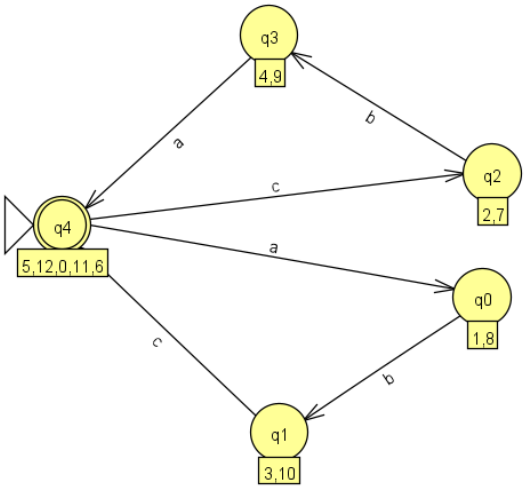


Figura 8: AFND de la segunda ER.

2.3. Implementación del código

El lenguaje de programación que se ha usado para resolver esta práctica es Java. Como es un lenguaje orientado a objetos, se hace uso de las clases para las diferentes partes del programa. Por último, se utiliza Java Swing para la creación de una pequeña interfaz que facilita el uso del programa.

2.3.1. Núcleo

El núcleo de la práctica está divido en tres clases distintas 'Main', 'MaquinaDeEstados' y 'Automata'. A continuación se explicará en detalle las partes más importantes de cada una de las clases.

Clase Automata

La clase Automata está formada por los atributos 'alfabeto', 'estados', 'estadoInicial', 'estadosFinales', 'estadosSalto', y 'matriz'. Por otro lado, está formada por otros muchos métodos, de los cuales los más importantes son 'getSiguienteEstado', 'getEstadoInicial', 'esFinal', 'inicializarAtributos', 'cargarAtributos' y 'cargarMatriz'.

Los tres primeros métodos devuelven algún objeto, ya sea devolver el siguiente estado a partir de la matriz, devolver el estado inicial o devolver si un estado es final o no. Por contra, los tres últimos métodos no devuelven ningún tipo de objeto. Estos métodos sirven para inicializar y cargar datos en los atributos de la manera correcta y en el formato correcto. Por ejemplo, para la matriz se hace uso de un HashMap cuya clave es el estado en el que se encuentra el autómata y su valor es otro HashMap. ELa clave de este segundo HashMap es el carácter de entrada mientras que el valor continene el estado al que se va a saltar.

Clase MaquinaDeEstados

La clase MaquinaDeEstados se usa principalmente para recorrer un objeto de la clase Automata. Es por eso que no precisa de muchos atributos. En este caso solo tres son necesarios: estadoActual y afd (automata finito determinista). Los métodos más importantes, que son 'compruebaCadena', 'generaCadenas' e 'intercambiarMatriz', se apoyan en estos atributos para funcionar.

El método 'compruebaCadena', recibe una cadena la cual recorre carácter a carácter. Por cada uno de los carácteres, se apoya en la función 'aceptaCaracter' para saber si ese carácter es admitido por el autómata. Por último, se comprueba si el carácter lleva a un estado final y si es el último de la cadena. Dependiendo del resultado de las diferentes operaciones, se determina si una cadena es válida o no.

El método 'intercambiarMatriz' recibe un objeto de tipo MaquinaDeEstados e intercambia el autómata asociado. Esto se hace de esta manera ya que al cambiar la matriz de estados, también tendrías que cambiar los estados, el estado inicial, los estados finales, etc. Y la clase Automata es la que contiene todos esos datos, por tanto es mucho más sencillo intercambiar el automata entero que cambiar los valores de cada variable una a una.

Por último, el método 'generaCadenas' se ha creado para generar una cantidad de cadena específicas con una longitud específica. Para ello se apoya en el método 'generarRamas'. Esto se hace de esta manera ya que se utiliza un árbol para generar las cadenas, de esta manera, en el árbol se generarán todas las combinaciones posibles con las letras del alfabeto especificado. Mientras se van generando las cadenas se va comprobando la cantidas de cadenas que ya han sido aceptadas, y una vez la cadena sea de la longitud deseada, se llama a 'compruebaCadena' y en caso de que cumpla los requisitos, se almacenará esa cadena, en caso contrario se descartará.

Clase Main

La clase Main es la más sencilla de todas. En el método main de esta clase, se crean los dos autómatas, las dos máquinas de estados y se inicializa la interfaz gráfica.

2.3.2. Interfaz gráfica

La interfaz gráfica no es muy complicada de entender. La aplicación cuenta con tres menús, por lo que se han creado tres clases.

La primera es el 'MenuPrincipal', en esta clase se crean tres botones, uno para ir al menú para comprobar cadenas, otro para ir al menú para generar cadenas y otro para intercambiar las matrices de estado.

La segunda clase es el 'MenuComprobar', en esta clase hay un campo de texto en el que se introduce una cadena para comprobarla y con los botones se puede elegir qué Expresión Regular quieres usar para comprobar la cadena.

La tercera clase es el 'MenuGenerar' donde se especificará la longitud de las cadenas, la cantidad máxima de cadenas a generar y la ER que se va a usar para generarlas. Al presionar un botón aparecerán las cadenas que han sido generadas.

3. Uso del programa

Al ejecutar el programa, aparecerá el primer menú (figura 9).

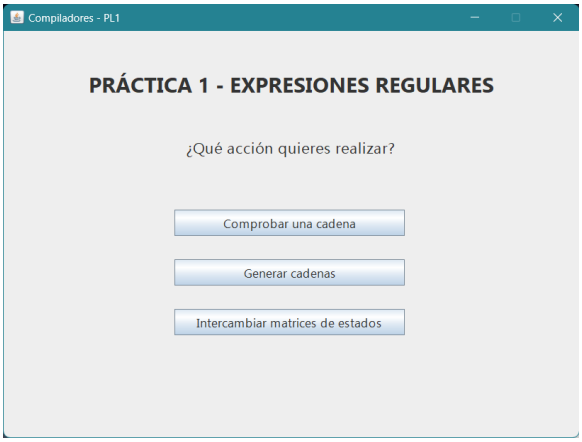


Figura 9: Menú principal de la aplicación.

Si se elige el primer botón del menú principal, se abrirá el menú para comprobar cadenas (figura 10).

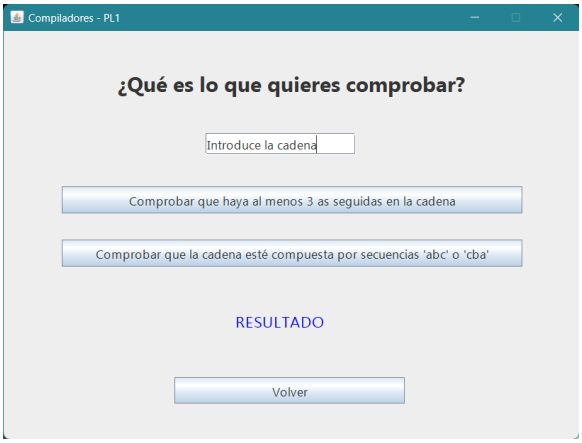
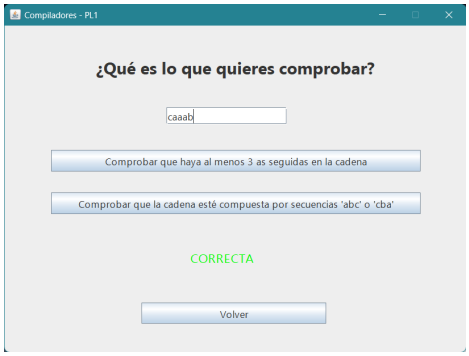


Figura 10: Menú para comprobar cadenas.

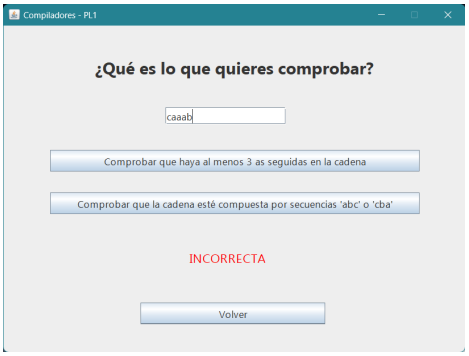
Dependiendo de la cadena que se introduzca y de la ER que se elija, el resultado será uno u otro, como se puede ver en la figura 11.

Por otro lado, si se elige el segundo botón del menú principal, se abrirá el menú para generar cadenas (figura 12).

Por último si se selecciona el último botón del menú principal se intercambiarán las matrices de las máquinas de estado y aparecerá una alerta para informar de que se ha hecho correctamente (figura 13).



(a) Cadena correcta (ER 1)



(b) Cadena incorrecta (ER 2)

Figura 11: Pruebas realizadas en la aplicación

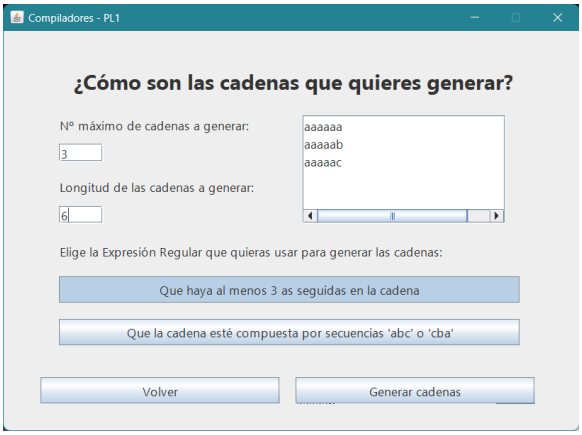


Figura 12: Menú para generar cadenas.

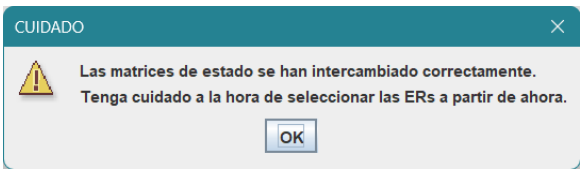


Figura 13: Alerta de cambio de matrices.