Universidad de Alcalá

Práctica 1: Expresiones Regulares

591000 - Compiladores

Prof. Marçal Mora Cantallops, Universidad de Alcalá 23/09/2022

Objetivos de la práctica

- Aprender a usar y transformar expresiones regulares.
- Programar un sistema para evaluarlas.
- Programar un sistema para generar cadenas válidas.

Enunciado

Dada una expresión natural, en esta práctica se realizarán las siguientes tareas:

- Traducción de la ER a formato JFLAP
- Transformación de la ER a un Autómata Finito No Determinista
- Análisis del Autómata Finito No Determinista para comprobar su validez
- Transformación del AFND a un Autómata Finito Determinista
- Minimización del AFD.
- Comprobación de validez con cadenas de entrada.
- Transformación del AFD a un matriz de transición de estados para su implementación en una máquina de estados.
- Implementación de un programa que, dada una matriz de transición de estados, implemente una máquina que permita realizar las siguientes dos operaciones:

1. Dada una cadena de texto de entrada, analizarla para determinar si esa cadena de texto cumple con la ER original.
2. 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).

Defensa

En la defensa de la práctica se deberá exponer y explicar los distintos componentes que formen el sistema programado por el alumno, respondiendo a las preguntas del profesor, si corresponde. Adicionalmente, se planteará una Expresión Regular, que el alumno deberá tratar e integrar en su programa, junto con un conjunto de cadenas que el alumno debe ser capaz de probar para la funcionalidad 1 del programa, indicando si cumplen o no con la ER establecida, y además debe aportar las cadenas de la opción 2 hasta el límite establecido en cada ejercicio.

Ejemplo

Sea el alfabeto de la ER el formado por las letras a,b,c,d,e, y la ER $((ac)*(bb)*b)|(bc(d|e)*)$, compruebe si las cadenas de entrada "bcdedededdde" y "acacacbbbb" son válidas, y genere 100 cadenas distintas válidas para la ER indicada con una longitud máxima por cadena de 10 caracteres. Dispone de 10 minutos.

Práctica

Realice las siguientes tareas, que deberá reflejar adecuadamente en una memoria con todos los pasos seguidos:

- Seleccione un alfabeto formado por, al menos, tres caracteres distintos.
- Con los tres caracteres y los símbolos disponibles, proponga dos ERs que correspondan a alguna expresión con interés. Es obligatorio el uso de, al menos, ocho símbolos en total, sin contar paréntesis. Tanto los cierres ($*$,$+$) como la OR deben usarse al menos una vez en alguna de las ERs.
- Realice los pasos descritos para pasar de la ER hasta el AFD simplificado y su correspondiente matriz de estados.
- Implemente el programa descrito en el enunciado, dejando abierta la posibilidad de intercambiar las matrices de estados.

Contenido

1. Seleccione un alfabeto formado por, al menos, tres caracteres distintos.

Se ha elegido como alfabeto de las expresiones regulares los caracteres: a, b, c

2. Con los tres caracteres y los símbolos disponibles, proponga dos ERs que correspondan a alguna expresión con interés. Es obligatorio el uso de, al menos, ocho símbolos en total, sin contar paréntesis. Tanto los cierres (∗,+) como la OR deben usarse al menos una vez en alguna de las ERs.

Se han elegido como expresiones regulares las siguientes:

(a?bc(a)*)|ba

En esta expresión regular se utilizan 3 caracteres distintos, con 9 símbolos (contando los caracteres utilizados).

Explicación:

- (a?bc(a)*)
  - La interrogación indica que la “a” puede usarse o no al empezar la expresión.
  - Obligatoriamente tiene que ir seguido de los caracteres bc, por lo que puede empezar la expresión por abc, o bc
    - Por ejemplo:
      - abc
      - bc
  - Posteriormente se pueden poner tantas “a” como se deseen o no poner ninguna.
    - Por ejemplo:
      - abc
      - abcaaaaaaaaaaaa
- |ba
  - Se puede poner la expresión anterior o utilizar ba. No es válido las dos a la vez, es una u otra.
    - Por ejemplo:
      - ba
      - abc
      - abcaaaa
      - bc
      - bca

(b((a)+|(cb)+))?ca

En esta expresión regular se utilizan 3 caracteres distintos, con 10 símbolos (contando los caracteres utilizados).

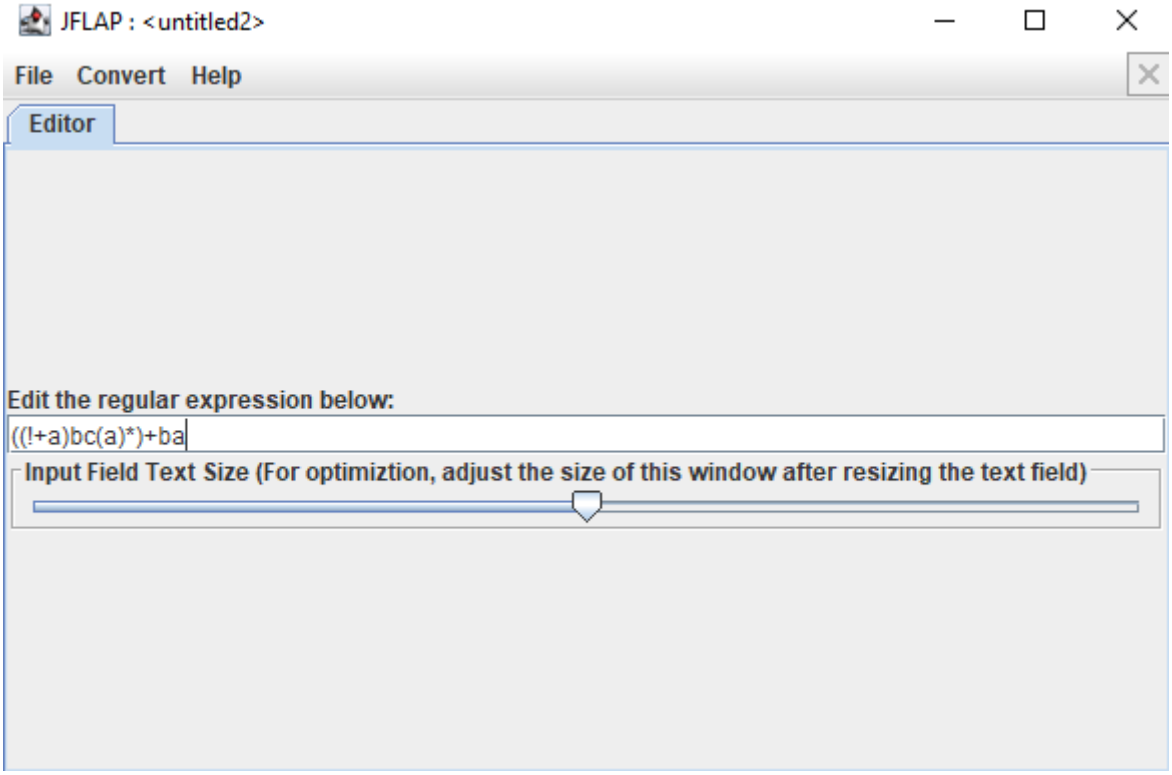
Explicación:

- (b((a)+|(cb)+))?
  - La expresión tiene que empezar por b, y se tiene que elegir si insertar a continuación la “a” o “cb”, las veces que se desee, pero no se pueden juntar.
  - La interrogación al final indica que puede darse la expresión o no, pero no puede repetirse.
  - Por ejemplo:
    - ba
    - baaaaaaaa
    - bcb
    - bcbcbcbcb
    - bbc – Estaría mal, y no la aceptaría.
    - bcbcbbc – También estaría mal y no lo aceptaría.
  - Importante los ejemplos anteriores sólo sirven para explicar esta parte, ya que al usar la expresión regular completa, estarían mal porque debe acabar en “ca”.
- ()?ca
  - Indica que la expresión explicada anteriormente es opcional, y que obligatoriamente tiene que acabar en ca.
    - Por ejemplo:
      - bcbcbcbcbca
      - bcbca
      - ca
      - bca – No lo aceptaría
      - bcbbca – No lo aceptaría

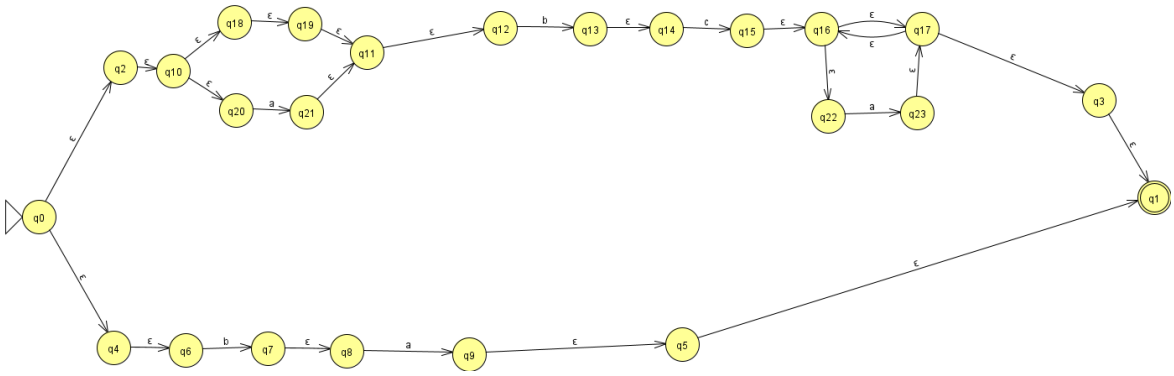
3. Realice los pasos descritos para pasar de la ER hasta el AFD simplificado y su correspondiente matriz de estados.

(a?bc(a)*)|ba

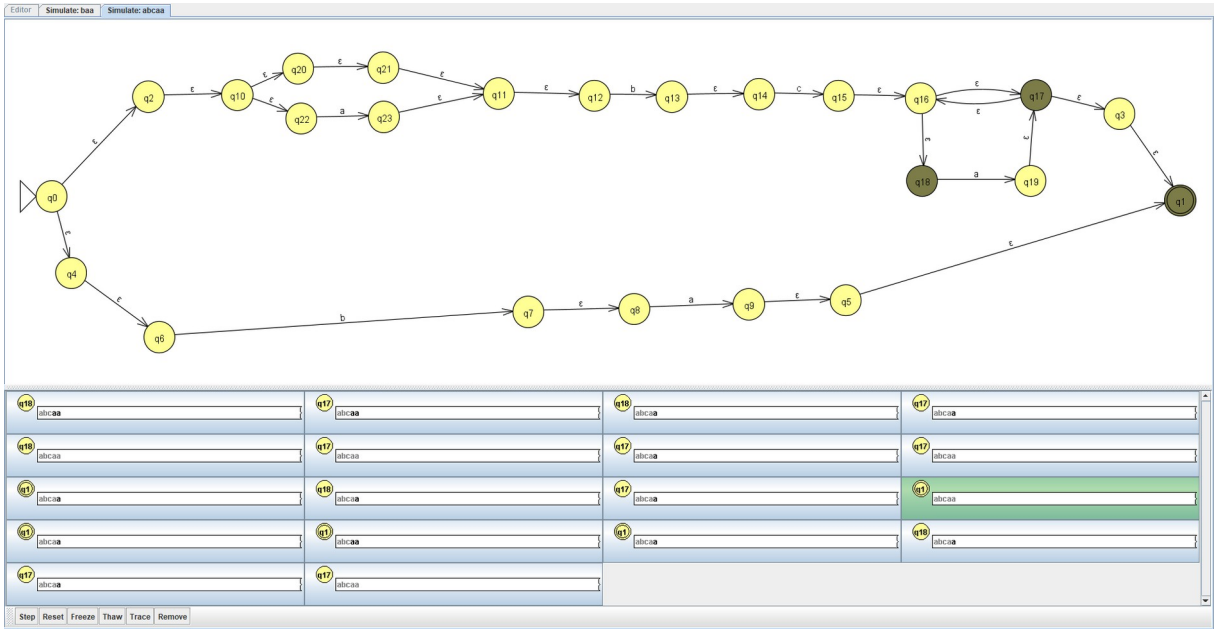
1. Transformación de la expresión anterior al formato aceptado por JFLAP quedando:
   - ((!+a)bc(a)*)+ba
2. Metemos la expresión adaptada en JFLAP:



3. Y convertimos la expresión a un autómata finito no determinista quedando lo siguiente:

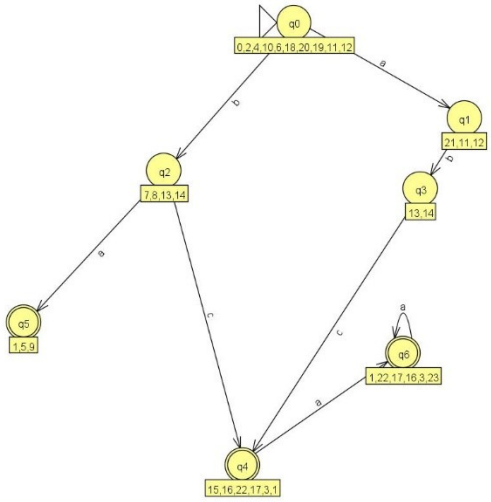


- Ejemplo de ejecución:

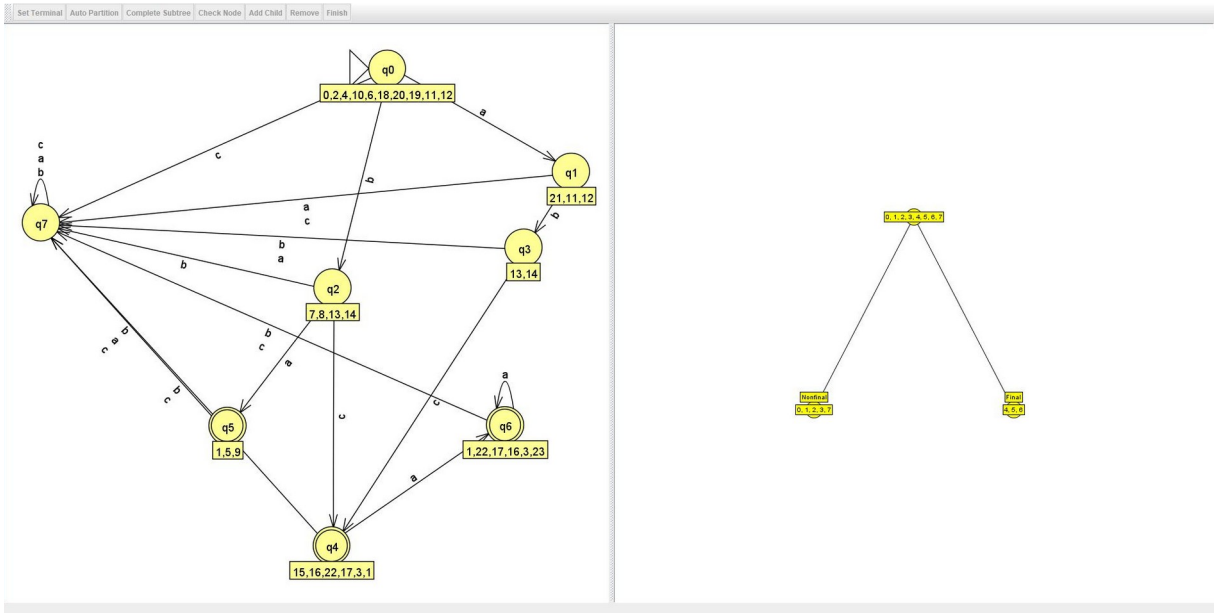


Como se puede observar en el ejemplo, se ha conseguido llegar al final utilizando “abcaa”.

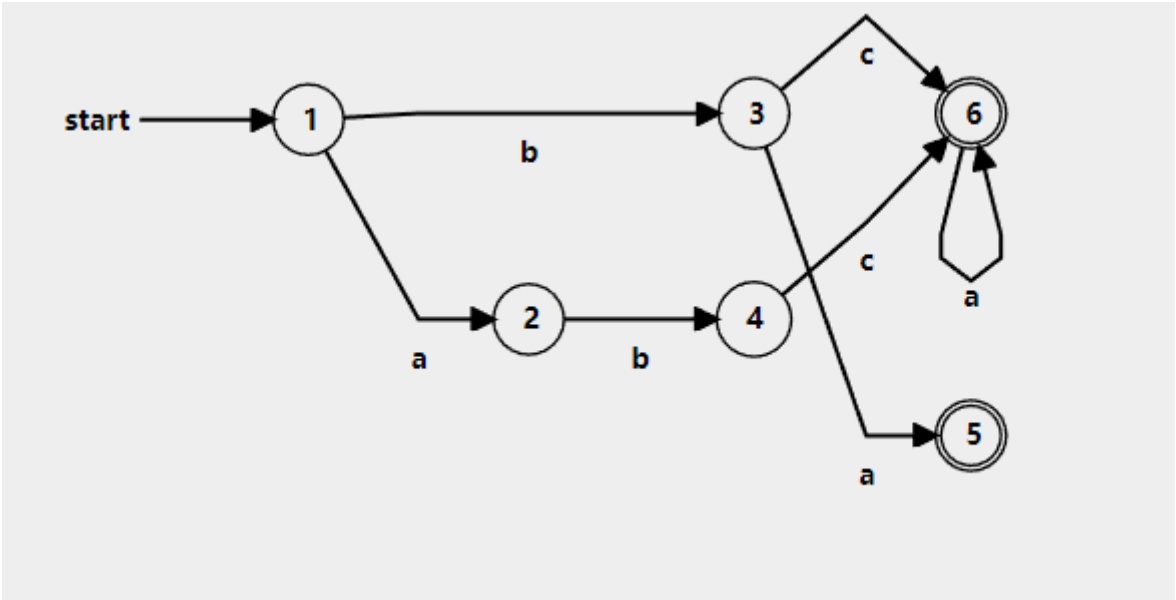
4. Ahora convertimos el AFND a AFD quedando lo siguiente:



5. Minimizamos el AFD:
    - Quedando a la izquierda el AFD con un estado adicional que es el estado de “caja de sastre” donde se irá cuando se meta un carácter no válido.
    - A la derecha nos muestra tres nodos, nos indican cuáles son nodos finales o no finales.



- A partir de una calculadora online, obtenemos el AFD mínimo (Enlace).



6. Obtenemos la matriz de estados a partir del AFD mínimo:

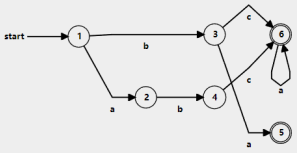
- A partir de una calculadora online, obtenemos la matriz de estados (Enlace).
- Los nodos con TYPE “accept” son nodos finales.

Input: (a?bc(a)*)|ba

CONVERT

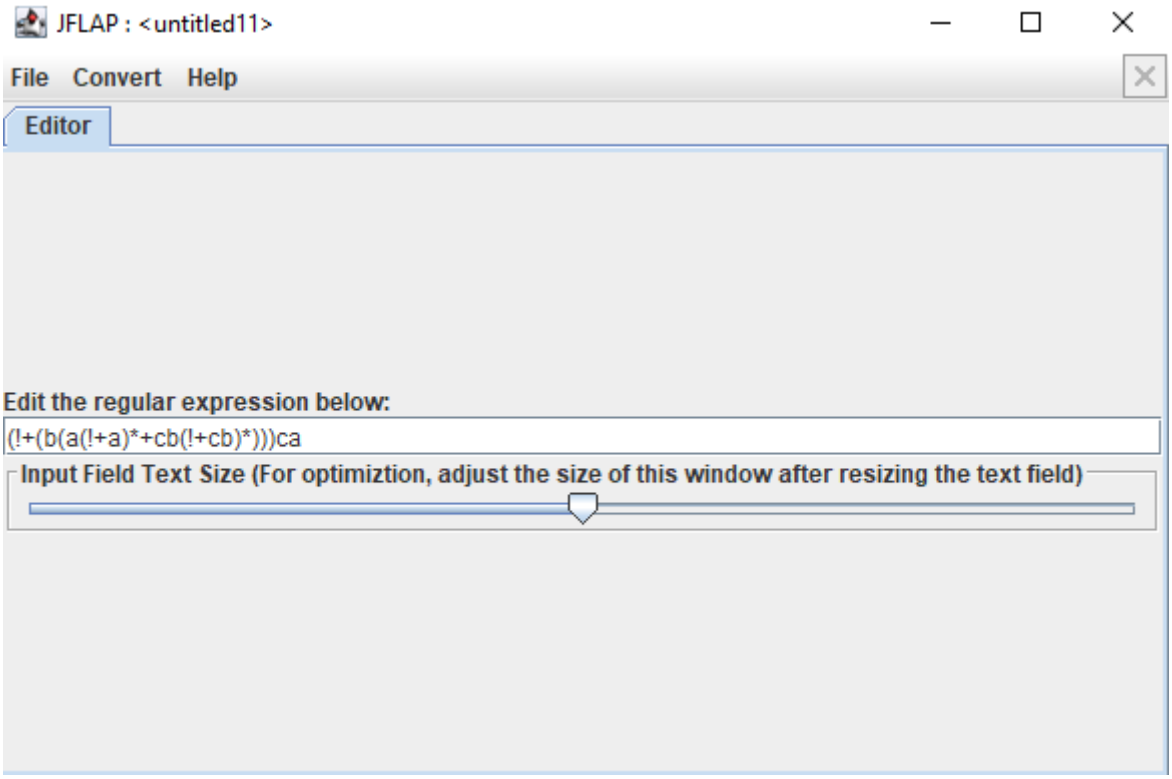
DFA: https://cyberzhg.github.io/toolbox/nfa2dfa?regex=KGE/YmMoYSkqKXxiYQ==

DFA STATE	Min-DFA STATE	TYPE	a	b	c
{A}	1		2	3	
{B}	2			4	
{C}	3		5		6
{D}	4				6
{E}	5	accept			
{F,G}	6	accept	6		

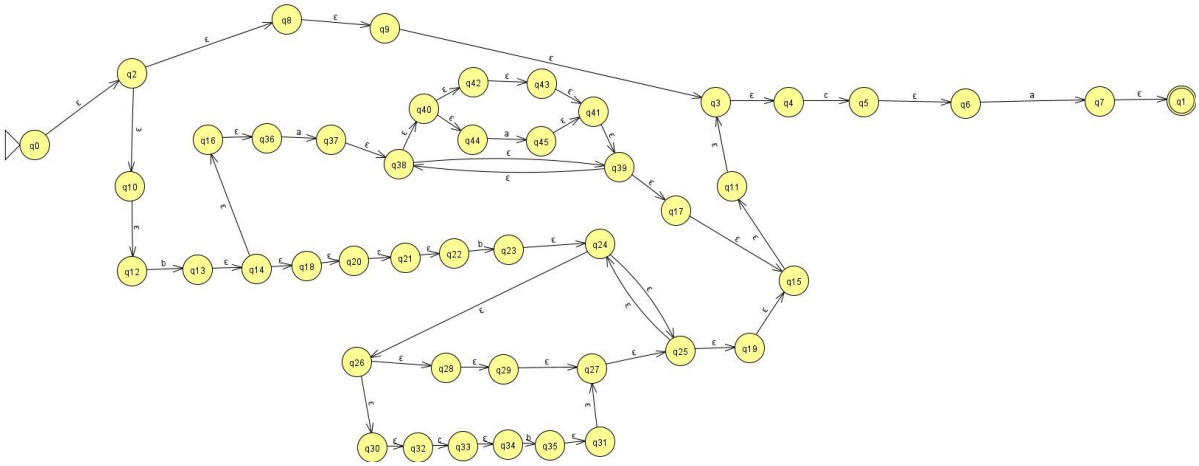


(b((a)+|(cb)+))?ca

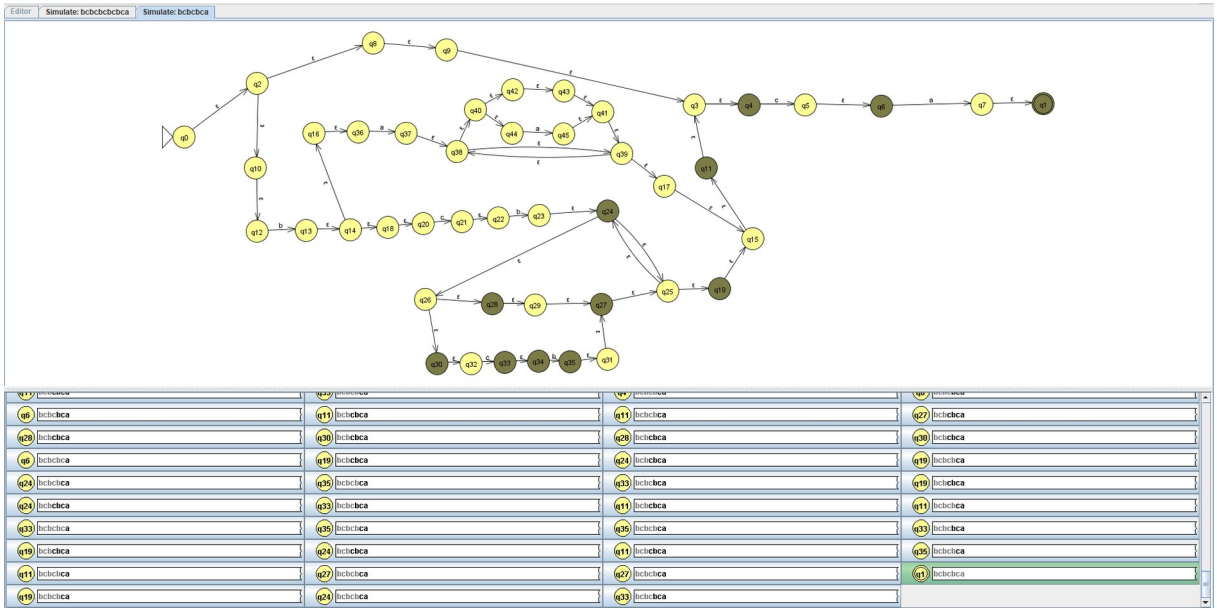
1. Transformación de la expresión anterior al formato aceptado por JFLAP quedando:
   - (!+(b(a(!+a)*+cb(!+cb)*)))ca
2. Metemos la expresión adaptada en JFLAP:



3. Y convertimos la expresión a un autómata finito no determinista quedando lo siguiente:

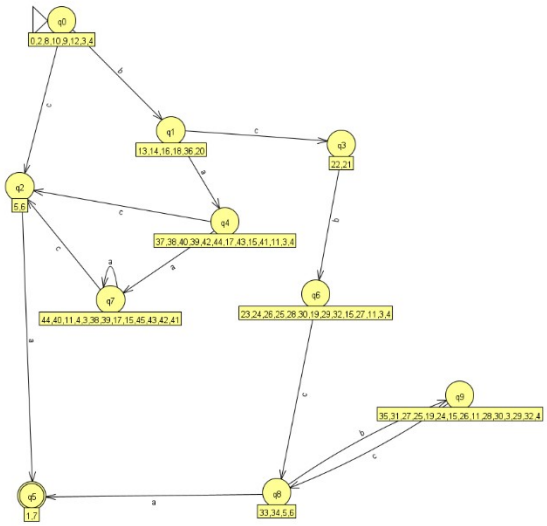


- Ejemplo de ejecución:

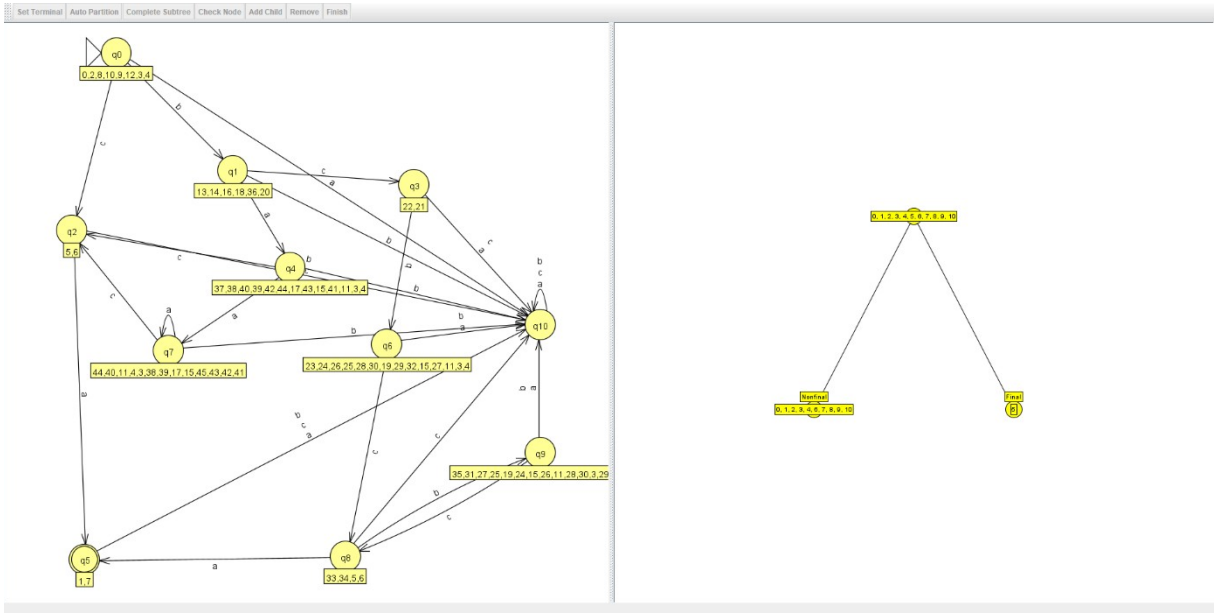


Como se puede observar en el ejemplo, se ha conseguido llegar al final utilizando “bcbcbca”.

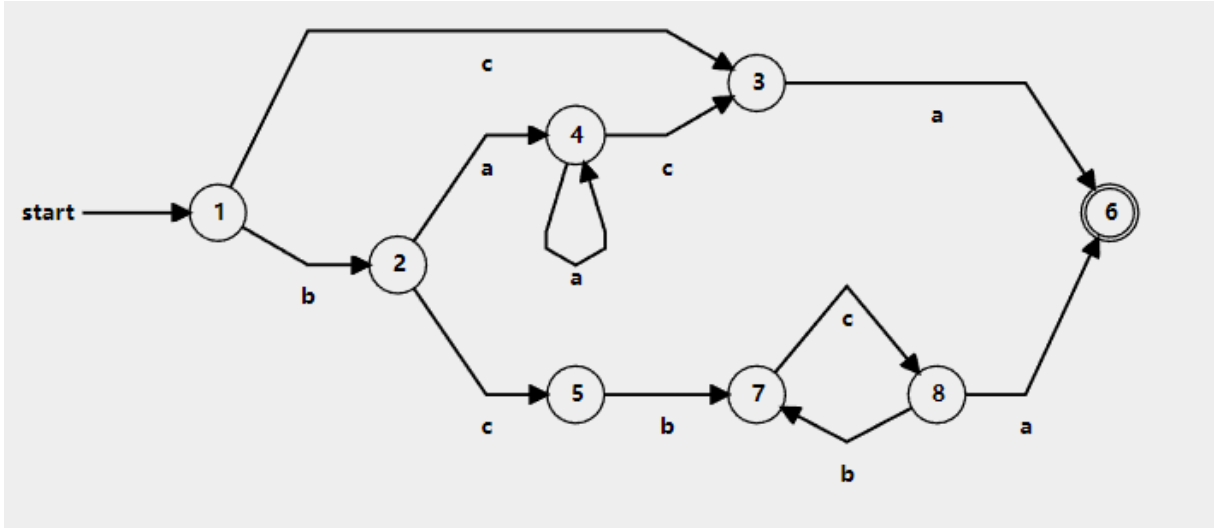
4. Ahora convertimos el AFND a AFD quedando lo siguiente:



5. Minimizamos el AFD:
   - Quedando a la izquierda el AFD con un estado adicional que es el estado de “caja de sastre” donde se irá cuando se meta un carácter no válido.
   - A la derecha nos muestra tres nodos, nos indican cuáles son nodos finales o no finales.

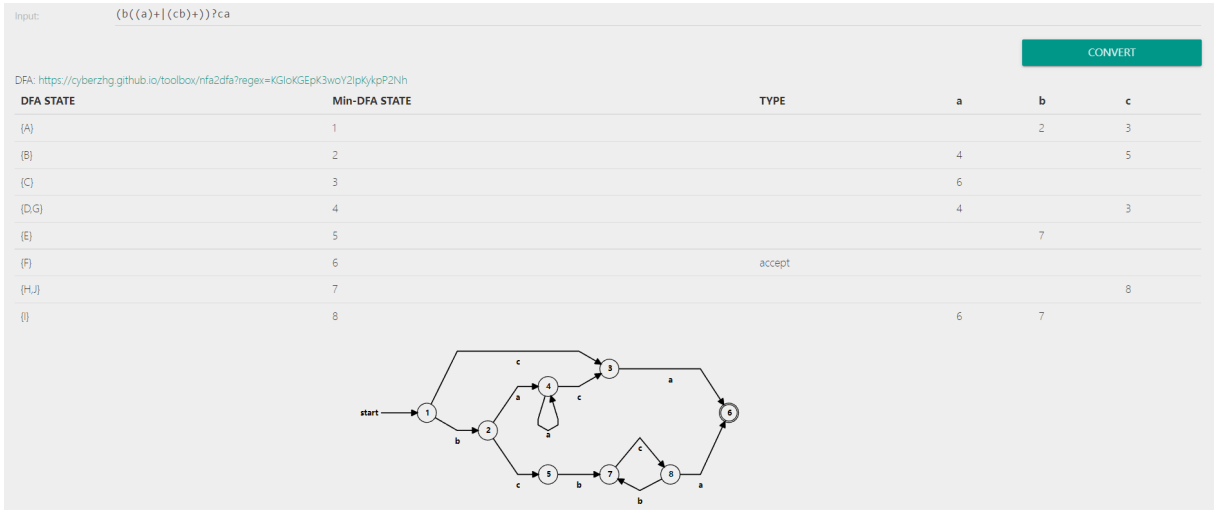


- A partir de una calculadora online, obtenemos el AFD mínimo (Enlace).



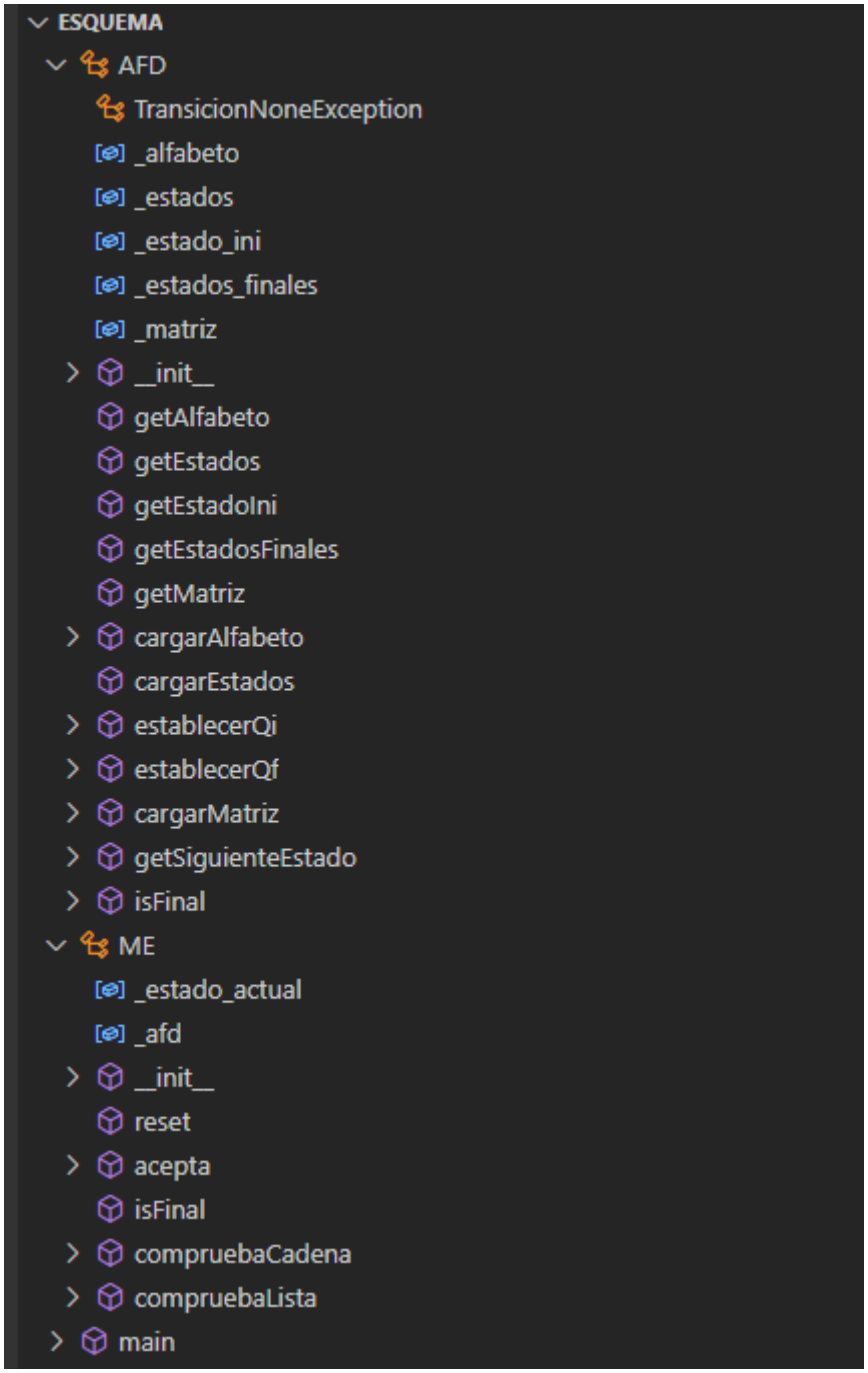
6. Obtenemos la matriz de estados a partir del AFD mínimo:

- A partir de una calculadora online, obtenemos la matriz de estados (Enlace).
- Los nodos con TYPE “accept” son nodos finales.



4. Implemente el programa descrito en el enunciado, dejando abierta la posibilidad de intercambiar las matrices de estados.

El programa consta de un fichero llamado “Programa.py” con la siguiente estructura:



Donde se utilizan las clases AFD (Autómata Finito Determinista) y ME (Máquina de Estados) para instanciar primero el autómata pasándole el alfabeto, los estados, el estado inicial, los estados finales y la matriz como argumentos de la siguiente forma:

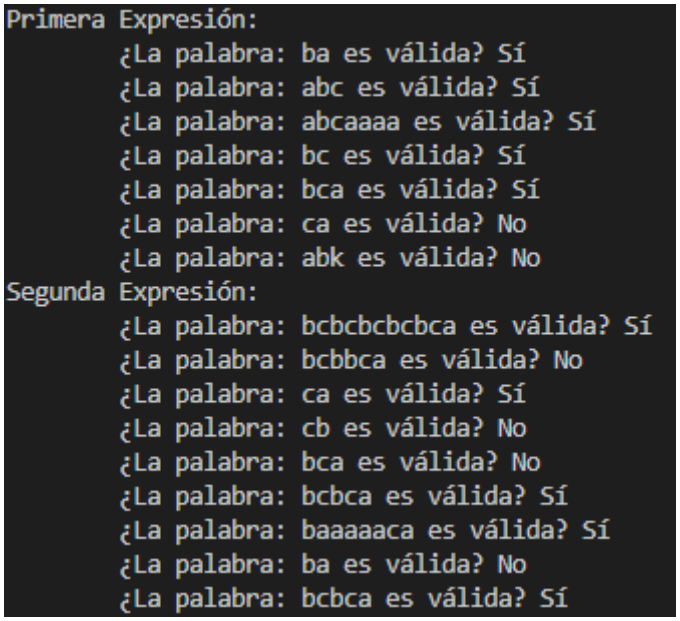
```
afd = AFD(estados = [*range(1,7)], estados_finales = [5,6], estado_ini = 1, alfabeto= ['a','b','c'], transiciones=[2,3,None,None,4,None,5,None,6,None,None,6,None,None,None,6,None,None])
me:ME = ME(afd = afd)
```

Las transiciones de la matriz a de pasarse en forma de lista. El anterior ejemplo es para la instanciación de la primera expresión regular.

Después se instancia ME para ir recorriendo la matriz del autómata finito determinista. Su utilidad reside en que le pasas una palabra (utilizando el método

‘compruebaCadena’) o una lista de palabras (utilizando el método ‘compruebaLista’) y te dice si es una palabra apropiada o no.

Ejemplo de ejecución del programa:

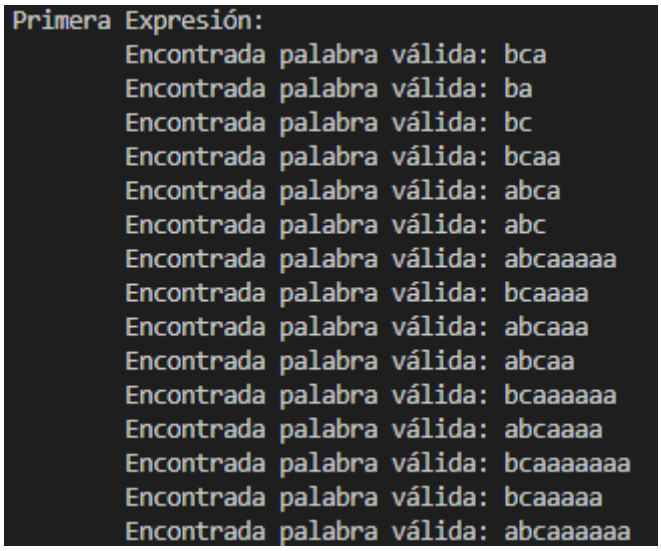


Cabe destacar que la clase ME tiene el método ‘reset’ para resetear la clase y reiniciarla a su momento inicial. Es necesario usarlo cuando se prueban varias palabras seguidas.

Se ha programa la última parte del programa, como desconocía una forma eficiente de hacerlo, he utilizado la librería random para generar de forma aleatoria una secuencia de caracteres válidos en la expresión regular. Para generar cadenas aleatoriamente, ha de llamarse al método ‘obtenerCadenas’.

Ejemplo de ejecución:

- Cadenas válidas generadas aleatoriamente para la primera expresión:



- Cadenas válidas generadas aleatoriamente para la segunda expresión: