**PL1 PROCESADORES**

**Participantes**

* Yacine Admou Admou 09149242ª

**Caso a) Identificador**

Para el identificar tras un análisis del caso se puede concluir que la expresión regular debería proceder con la existencia de una letra sea mayúscula o minúscula primero para ser valida y posteriormente pueden aparecer o letras o números. La expresión regular es: **([a-zA-Z])([a-zA-Z0-9]\*)** que a la hora de colocar en el Jflap se usará (a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z)(a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z+0+1+2+3+4+5+6+7+8+9)\*

AFND

Diagrama

El contenido generado por IA puede ser incorrecto.

Casos con Jflap

Tabla

El contenido generado por IA puede ser incorrecto.

AFD

Texto

El contenido generado por IA puede ser incorrecto.

Matriz de estados

**Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.**

**Casos de practica**

Los casos de practica en este caso deben ser la cadena vacía que debe ser rechazada, así como cualquier expresión que comienza con un valor numérico de 0 a 9. Los únicos casos aceptados son aquellos que comienzan con cualquier letra sea mayúscula o minúscula. Ejemplos: 094na, sada22484.

Caso B) Paridad de As

Para el identificar tras un análisis del caso se puede concluir que la expresión regular debería proceder con la existencia de un conjunto C = {a,b,ε}. La expresión regular es:  **b\*(ab\*ab\*)\*.**

Una vez realizada la transformación a un autómata finito no determinista tenemos:

Diagrama

El contenido generado por IA puede ser incorrecto.

Casos con Jflap

Tabla

El contenido generado por IA puede ser incorrecto.

Autómata Finito Determinista

Diagrama

El contenido generado por IA puede ser incorrecto.

Matriz de Estados

Imagen que contiene transporte, mapa, esquiando, nieve

El contenido generado por IA puede ser incorrecto.

Casos de Prueba

Algunos de los casos de prueba con los que debemos comenzar son aquellos que contienen un numero par de as, empezando por el caso vacío, así como un numero n de bs y 0 as, y ya posteriormente el uso de un número impar de as para comprobar que se rechaza.

Casos C)

Caso d)

Código de la aplicación

**AFD**

**Explicación general del funcionamiento**

1. **Matriz de transiciones (int[][] transiciones)**
   * Cada fila representa un estado del autómata.
   * Cada columna representa un símbolo del alfabeto (según el mapa sigma).
   * El valor de cada celda indica a qué estado se transita al leer ese símbolo desde ese estado.
   * Por convención, si no hay transición, se puede usar -1.
2. **Estados finales (Set<Integer> estadosFinales)**
   * Son los estados en los que, si el autómata termina la lectura de una cadena, se considera que la cadena es aceptada.
3. **Alfabeto (Map<Character,Integer> sigma)**
   * Convierte un carácter del alfabeto en un índice de columna de la matriz.
   * Esto permite que la matriz de transiciones sea más eficiente y fácil de indexar.
4. **Constructores y getters**
   * Permiten inicializar el AFD y acceder a sus componentes de manera segura.
   * La clase no permite modificar directamente la matriz, los estados finales ni el alfabeto desde fuera (por ser final).

Simuladr

 **Estado inicial**

* Se asume que el AFD siempre comienza en el estado 0.

 **Recorrer la cadena**

* Cada carácter de la cadena se procesa uno por uno.

 **Comprobación del alfabeto**

* Si el carácter no existe en sigma, la cadena contiene símbolos inválidos → rechazo inmediato.

 **Transición**

* Se obtiene la columna correspondiente al símbolo en sigma.
* Se actualiza el estado según transiciones[estadoActual][columna].
* Si la transición no existe (-1), se rechaza la cadena.

 **Aceptación**

* Al terminar de leer la cadena, si el estado final actual pertenece a estadosFinales, la cadena es aceptada.

Main

 **Definición de DFA**

* Cada DFA se define mediante:
  + int[][] transiciones → matriz de estados y transiciones.
  + Set<Integer> estadosFinales → estados de aceptación.
  + Map<Character,Integer> sigma → alfabeto que asigna columnas a cada símbolo.

 **Simulación**

* Simulador.simular() recorre la cadena carácter por carácter:
  + Traduce el carácter a una columna con sigma.
  + Actualiza el estado según transiciones.
  + Rechaza la cadena si:
    - El símbolo no está en el alfabeto.
    - La transición es inválida (-1).
  + Acepta si el estado final tras leer toda la cadena pertenece a estadosFinales.

 **Interfaz con el usuario**

* Permite seleccionar el tipo de DFA a simular.
* Pide la cadena a evaluar.
* Muestra si la cadena es ACEPTADA o RECHAZADA.
* Maneja errores de entrada de manera segura.