PL1 PROCESADORES

**Yacine Admou Admou 09149242A**

**Índice**

[Identificadores 3](#_Toc210910932)

[**Autómata Finito No Determinista** 3](#_Toc210910933)

[**Casos de practica** 3](#_Toc210910934)

[**Autómata Finito Determinista** 4](#_Toc210910935)

[**Autómata Finito Determinista Minimizado** 4](#_Toc210910936)

[**Caso sin simplificar** 5](#_Toc210910938)

[**Autómata Finito No Determinista** 5](#_Toc210910939)

[**Casos de practica** 5](#_Toc210910940)

[**Autómata Finito Determinista** 6](#_Toc210910941)

[**Autómata Finito Determinista Minimizado** 6](#_Toc210910942)

[**Matriz de estados** 6](#_Toc210910943)

[Paridad de As 7](#_Toc210910944)

[**Autómata Finito No Determinista** 7](#_Toc210910945)

[**Casos de Prueba** 7](#_Toc210910946)

[**Autómata Finito Determinista** 8](#_Toc210910947)

[**Autómata Finito Determinista Minimizado** 8](#_Toc210910948)

[**Matriz de Estados** 9](#_Toc210910949)

[Casos C) 9](#_Toc210910950)

[**Autómata Finito No Determinista** 9](#_Toc210910951)

[**Casos de Prueba** 10](#_Toc210910952)

[**Autómata Finito Determinista** 10](#_Toc210910953)

[**Autómata Finito Determinista Minimizado** 11](#_Toc210910954)

[**Matriz de estados** 11](#_Toc210910955)

[Suma de Variables y Floats 12](#_Toc210910956)

[**Autómata Finito No Determinista** 12](#_Toc210910957)

[**Casos de Prueba** 12](#_Toc210910958)

[**Autómata Finito Determinista** 13](#_Toc210910959)

[**Autómata Finito Determinista Minimizado** 13](#_Toc210910960)

[**Matriz de estados** 14](#_Toc210910961)

[Código de la aplicación 14](#_Toc210910962)

[**Clase AFD** 14](#_Toc210910963)

[**Simulador** 15](#_Toc210910964)

[**PL1\_Procesadores** 16](#_Toc210910965)

[**Casos Prueba Identificadores** 18](#_Toc210910966)

[**Casos Prueba Número Par de as** 18](#_Toc210910967)

[**Casos Prueba Coma Flotante** 19](#_Toc210910968)

[**Casos Prueba Suma Variables y Floats** 19](#_Toc210910969)

# **Identificadores**

Para el identificador 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. A su vez se puede usar una versión simplificada para resolver el tiempo de espera. La versión es Er = (l+L)(l+L+d)\* donde l es el conjunto de letras minúsculas [a-z], L es el conjunto de letras mayúsculas [A-Z] y finalmente d es el conjunto de números [0-9]. 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)\*

## **Autómata Finito No Determinista**

Una vez se accede al JFlap y se introduce la expresión regular simplificada obtenemos el siguiente autómata finito no determinista. Este es un **AFND con transiciones λ**, donde el estado inicial es **q0** y el de aceptación es **q1**. El autómata usa ε para ramificarse en distintos caminos que aceptan combinaciones de símbolos d, l y L. Cada rama representa una forma válida de construir cadenas con letras o dígitos, y al final todas confluyen en el estado de aceptación.

Diagrama

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 (d). Los únicos casos aceptados son aquellos que comienzan con cualquier letra sea mayúscula o minúscula (l o L). Ejemplos: ldlld, ddllLLLL.

Tabla

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista**

Después del AFND iremos a transformarlo en un Autómata Finito Determinista usando el JFlap como ya antes mencionado. Este AFD reconoce cadenas compuestas por letras minúsculas, mayúsculas y dígitos, distinguiendo los posibles comienzos de la cadena y usando bucles para validar que se repitan mientras la entrada sea correcta.

Imagen que contiene mapa, texto, colgando, esquiando

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista Minimizado**

Con el AFD anterior buscamos minimizarlo obteniendo así este AFD minimizado que acepta cualquier cadena formada por letras minúsculas, mayúsculas y dígitos, reduciendo toda la complejidad original a un solo estado de aceptación con bucle.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Caso sin simplificar**

Como comparación se han añadido los AFND, AFD, AFD Minimizado y los casos de prueba en JFlap del caso completo para comprobar que el resultado final de ambos es el mismo y que por tanto la matriz de transición de estados de ambos es la misma.

## **Autómata Finito No Determinista**

Cumple con las mismas condiciones del AFND simplificado solo que en este caso contiene todos los elementos.

Diagrama

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.

Tabla

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista**

Texto

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista Minimizado**

**Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.**

## **Matriz de estados**

A raíz de ambos AFD minimizados podemos ver cómo se mueve entre estados y cuáles son los casos que dan error permitiéndonos tener las 2 siguientes tablas, donde la segunda es la tabla que se usara para la programación de este AFD en Java.

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Letras | Dígitos | Otros caracteres |
| Q0 | Q1 | Error | Error |
| Q1 | Q1 | Q1 | Error |

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Letras | Dígitos | Otros caracteres |
| Q0 | 1 | -1 | -1 |
| Q1 | 1 | 1 | -1 |

# **Paridad de As**

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\*)\*.**

## **Autómata Finito No Determinista**

El autómata puede seguir distintos recorridos gracias a las transiciones λ, y en cada uno se leen símbolos como a o b. De esta forma admite diferentes combinaciones posibles de esas letras. Finalmente, todos los recorridos válidos desembocan en el estado final, lo que implica que el lenguaje aceptado está formado por la unión de varias cadenas distintas.

Diagrama

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.

Tabla

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista**

El AFD reconoce cadenas de a y b. Desde el estado inicial **q0**, las transiciones llevan a distintos estados, con bucles en algunos para repetir símbolos y conexiones que permiten alternar entre a y b. En conjunto, el autómata valida diversas combinaciones de estos caracteres de forma simplificada.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista Minimizado**

Es un autómata con dos estados que reconoce combinaciones de las letras a y b. El símbolo b puede repetirse indefinidamente en ambos estados, mientras que cada a provoca un cambio de estado, alternando entre q0 y q1.

Imagen que contiene transporte, mapa, esquiando, nieve

El contenido generado por IA puede ser incorrecto.

## **Matriz de Estados**

A raíz del AFD minimizado podemos ver cómo se mueve entre estados y cuáles son los casos que dan error permitiéndonos tener las 2 siguientes tablas, donde la segunda es la tabla que se usara para la programación de este AFD en Java.

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | a | b | Otro carácter |
| Q1 | Q0 | Q1 | Error |
| Q0 | Q1 | Q0 | Error |

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | a | b | Otro carácter |
| Q1 | 1 | 0 | -1 |
| Q0 | 0 | 1 | -1 |

# **Casos C)**

Tras analizar las posibles formas de proceder para encontrar la expresión regular se llega a la conclusión de que la expresión regular es ER = ([0-9]+(\.[0-9]+)?). Que al traducirla a JFlap da que la ER = ((0+1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)\*((!+((.))(0+1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)\*)))

## **Autómata Finito No Determinista**

El AFND inicia en q0 y permite distintos recorridos para validar cadenas con diversas combinaciones de símbolos. Gracias a las transiciones ε se enlazan los distintos bloques de estados, y todos los caminos correctos terminan llegando al estado final q1, lo que le permite aceptar varias secuencias dentro del mismo lenguaje.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Casos de Prueba**

Para este caso se buscan los casos en los que el vacío, cualquier valor fuera del conjunto de los números de coma flotante debe ser rechazado y si la entrada no es completa con parte entera y decimal si así se ha decidió debe fallar. Cabe recalcar que el único divisor valido es el punto {.} y no la coma {,}. El resto de entradas numéricas han de ser aceptadas.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista**

Este autómata muestra una red de estados interconectados que permiten recorrer distintas rutas según los símbolos de entrada. Cada transición refleja posibles secuencias que forman parte del lenguaje aceptado. En conjunto, garantiza que cualquier cadena válida pueda reconocerse siguiendo alguno de los caminos disponibles.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista Minimizado**

Este diagrama muestra un **AFD minimizado** con tres estados principales: **q1** como inicial, **q3** como estado intermedio y **q2** como estado de aceptación. El autómata avanza desde el inicio procesando dígitos, pasando primero por q3, luego por q0 y finalmente llegando a q2. Los bucles en q3 y q2 permiten repetir símbolos, lo que garantiza que se puedan aceptar cadenas más largas de números. En resumen, este autómata reconoce secuencias de dígitos y las valida siguiendo un recorrido ordenado hacia el estado final.

Imagen que contiene báscula, balancearse, medallón, reloj

El contenido generado por IA puede ser incorrecto.

## **Matriz de estados**

A raíz del AFD minimizado podemos ver cómo se mueve entre estados y cuáles son los casos que dan error permitiéndonos tener las 2 siguientes tablas, donde la segunda es la tabla que se usara para la programación de este AFD en Java.

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Dígitos | Punto | Otro carácter |
| Q1 | Q3 | Error | Error |
| Q3 | Q3 | Q0 | Error |
| Q0 | Q2 | Error | Error |
| Q2 | Q2 | Error | Error |

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Dígitos | Punto | Otro carácter |
| Q1 | Q3 | -1 | -1 |
| Q3 | Q3 | Q0 | -1 |
| Q0 | Q2 | -1 | -1 |
| Q2 | Q2 | -1 | -1 |

# **Suma de Variables y Floats**

Tras un análisis de la petición vemos que es la unión de 2 de los casos anteriores siendo esos mismos los identificadores y los números de coma flotante. Teniendo en cuenta las expresiones regulares que ya se yan mencionado anteriormente llegamos a que la expresión regular es ER = (NUM|ID)([+\-](NUM|ID))\*. El problema con esta expresión es el gran tamaño que tiene por lo que al querer pasarla a JFlap se han tenido en cuenta varias cosas para su funcionamiento correcto. Los símbolos + y – para que la aplicación no los tome como operadores se han usado la s = suma y r = resta, a su vez la n proviene de número y la i proviene de identificador. Con estos cambios se ha llegado a que la expresión regular a usar es ER = (n+i)((r+s)(n+i))\*. Se ha tenido en cuenta que se puede solo colocar un único número o identificador sin necesidad de añadir ningún otro valor después.

## **Autómata Finito No Determinista**

Este AFND tiene como estado inicial q0 y q1el de aceptación. El autómata presenta múltiples caminos que combinan símbolos como i, n, r y s, conectados mediante transiciones ε que permiten moverse sin consumir entrada. En conjunto, reconoce distintas secuencias formadas por esos símbolos, convergiendo todas finalmente hacia el estado de aceptación.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Casos de Prueba**

Para este caso se buscan los casos en los que el vacío, y cualquier caso que cumpla con tener al menos 1 número o identificador y al añadir un operador tenga un segundo valor que añadir.

Tabla

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista**

Los estados intermedios (**q1, q2, q5, q6**) conectan con **q3** y **q4** mediante símbolos i, n, r y s. Según la combinación de estos caracteres se recorren distintos caminos, que al final confluyen en los estados donde las cadenas resultan aceptadas.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Autómata Finito Determinista Minimizado**

Este autómata tiene solo dos estados: **q0** como inicial y **q1** como final. La transición de q0 a q1 puede hacerse con los símbolos s o r, lo que significa que basta con leer uno de esos caracteres para llegar al estado de aceptación. En resumen, reconoce cadenas formadas por un único símbolo, ya sea s o r.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Matriz de estados**

A raíz del AFD minimizado podemos ver cómo se mueve entre estados y cuáles son los casos que dan error permitiéndonos tener las 2 siguientes tablas, donde la segunda es la tabla será una referencia de uso para Java.

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Num/Iden | r/s | Otro carácter |
| Q0 | Q1 | Error | Error |
| Q1 | Q1 | Q0 | Error |

|  |  |  |  |
| --- | --- | --- | --- |
| Estado |  |  |  |
| Q0 | 1 | -1 | -1 |
| Q1 | 1 | 0 | -1 |

# **Código de la aplicación**

## **Clase 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.

1. **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.

1. **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.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. **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).

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

## **Simulador**

1. **Estado inicial**

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

1. **Recorrer la cadena**

Cada carácter de la cadena se procesa uno por uno.

1. **Comprobación del alfabeto**

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

1. **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.

1. **Aceptación**

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

## **PL1\_Procesadores**

1. **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.

En la definición del cuarto caso se ha procedido a usar una matriz de estados más grande que la necesaria por el autómata para asegurar así el funcionamiento correcto de la aplicación.

Texto

El contenido generado por IA puede ser incorrecto.

1. **Simulación**

* Simulador.simular() recorre la cadena carácter por carácter:
* Traduce el carácter a una columna con sigma.
* Actualiza el estado según transiciones.

Acepta si el estado final tras leer toda la cadena pertenece a estadosFinales. Rechaza la cadena si: El símbolo no está en el alfabeto. La transición es inválida (-1).

1. **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.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

## **Casos Prueba Identificadores**

Aquí hay unos casos de prueba de la aplicación según lo ya explicado en los casos de prueba anteriores.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

## **Casos Prueba Número Par de as**

Aquí hay unos casos de prueba de la aplicación según lo ya explicado en los casos de prueba anteriores.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto.

## **Casos Prueba Coma Flotante**

Aquí hay unos casos de prueba de la aplicación según lo ya explicado en los casos de prueba anteriores.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto.

## **Casos Prueba Suma Variables y Floats**

Aquí hay unos casos de prueba de la aplicación según lo ya explicado en los casos de prueba anteriores.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto. Texto, Carta

El contenido generado por IA puede ser incorrecto.