Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación
Inga. Vivian Damaris Campos González
Tutor académico: Luisa María Ortíz Romero

Manual Técnico - AFDGraph

José Alexander López López
Carné: 202100305
Fecha de Elaboración: 24/03/2025

Índice

1. Introducción

1.1. Propósito

Este manual proporciona una guía detallada para la instalación, configuración y mantenimiento del programa **AFDGraph**, desarrollado en Java como parte del curso de Lenguajes Formales y de Programación.

1.2. Alcance

El sistema permite visualizar Autómatas Finitos Deterministas (AFD) a partir de una descripción textual en formato .lfp, aplicando conceptos de analizadores léxicos y programación orientada a objetos.

1.3. Público Objetivo

Dirigido a desarrolladores y personal técnico que requieran comprender la estructura y funcionamiento del sistema, así como a docentes y estudiantes del área de Ciencias de la Computación interesados en la visualización de autómatas.

2. Descripción General del Sistema

2.1. Arquitectura

El sistema está basado en una arquitectura de interfaz gráfica en Java Swing, con componentes de análisis léxico para procesar los archivos de entrada.

2.2. Tecnologías Utilizadas

- Java SE 8+
- Java Swing para la interfaz gráfica
- Manejo de archivos con `File` y `FileReader`
- Estructuras de datos dinámicas (`HashMap`, `ArrayList`)
- Autómatas Finitos Deterministas para el análisis léxico

2.3. Requisitos del Sistema

- Sistema Operativo: Windows, Linux o MacOS
- Java Development Kit (JDK) 8 o superior
- IDE recomendado: NetBeans, Eclipse o IntelliJ IDEA
- Memoria RAM: 2GB mínimo
- Espacio en disco: 100MB mínimo

3. Instalación y Configuración

3.1. Instalación de Java

Descargar e instalar JDK desde `https://www.oracle.com/java/technologies/javase-downloads.html`.

3.2. Clonación del Repositorio

Ejecutar en terminal:

```
git clone https://github.com/JoseArt777/-LFP-202100305-.git
cd -LFP-202100305-/Proyecto1
```

3.3. Ejecución del Proyecto

1. Abrir el proyecto en el IDE de su preferencia.
2. Compilar y ejecutar la clase principal `Main.java`.

4. Estructura del Código

El código se encuentra estructurado en los siguientes paquetes:

- **modelo:** Contiene las clases que representan los autómatas y sus componentes.
- **analizador:** Implementa el analizador léxico para procesar los archivos .lfp.
- **gui:** Contiene las clases de la interfaz gráfica.
- **reportes:** Genera los reportes de tokens y errores.
- **graficador:** Se encarga de generar la representación visual del autómata.

4.1. Clases Principales

- `AFD`: Representa un autómata finito determinista.
- `Estado`: Representa un estado del autómata con sus transiciones.
- `AnalizadorLexico`: Implementa el análisis léxico de los archivos de entrada.
- `Token`: Representa un token identificado durante el análisis léxico.
- `Error`: Representa un error léxico encontrado durante el análisis.
- `GraficadorAFD`: Genera la visualización gráfica del autómata.
- `VentanaPrincipal`: Implementa la interfaz gráfica principal.
- `Main`: Contiene el método principal `main()`.

5. Implementación del Analizador Léxico

5.1. Diagrama del Autómata del Analizador

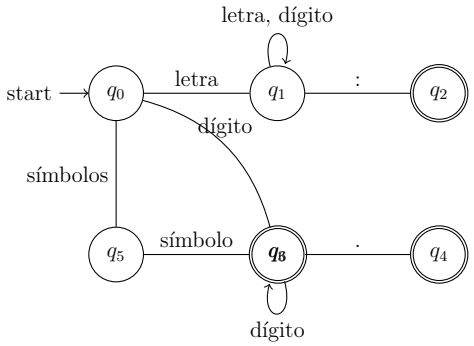


Figura 1: Autómata Finito Determinista para el analizador léxico.

5.2. Tokens Reconocidos

Token	**Patrón**	**Descripción**
IDENTIFICADOR	[a-zA-Z][a-zA-Z0-9]*	Nombres de variables
LLAVE_ABRE	{	Llave de apertura
LLAVE_CIERRA	}	Llave de cierre
DOS_PUNTOS	:	Dos puntos
COMA	,	Coma
COMILLAS	”	Comillas
PARENTESIS_ABRE	(	Paréntesis de apertura
PARENTESIS_CIERRA	)	Paréntesis de cierre
CORCHETE_ABRE	[	Corchete de apertura
CORCHETE_CIERRA	]	Corchete de cierre
FLECHA	-¿	Flecha
IGUAL	=	Signo igual
CADENA	”.*”	Cadena entre comillas

Cuadro 1: Tokens reconocidos por el analizador léxico.

5.3. Implementación del Autómata

Listing 1: Fragmento de código del analizador léxico

```
public class AnalizadorLexico {
    private String entrada;
    private int posicion;
    private char caracterActual;
    private ArrayList<Token> tokens;
    private ArrayList<Error> errores;

    public AnalizadorLexico(String entrada) {
```

```
        this.entrada = entrada;
        this.posicion = 0;
        this.tokens = new ArrayList<>();
        this.errores = new ArrayList<>();
        if (!entrada.isEmpty()) {
            this.caracterActual = entrada.charAt(0);
        }
    }

    public void analizar() {
        while (posicion < entrada.length()) {
            // Ignorar espacios en blanco
            if (Character.isWhitespace(caracterActual)) {
                siguienteCaracter();
                continue;
            }

            // Identificar tokens
            if (Character.isLetter(caracterActual)) {
                // Procesar identificador
                procesarIdentificador();
            } else if (Character.isDigit(caracterActual)) {
                // Procesar n mero
                procesarNumero();
            } else if (caracterActual == '{') {
                tokens.add(new Token(TipoToken.LLAVE_ABRE, "{", posicion));
                siguienteCaracter();
            }
            // ... resto de los casos para otros tokens
            else {
                // Error: car cter no reconocido
                errores.add(new Error("Car cter no reconocido: " + caracte
                siguienteCaracter();
            }
        }
    }

    private void procesarIdentificador() {
        // Implementaci n del reconocimiento de identificadores
    }

    private void siguienteCaracter() {
        posicion++;
        if (posicion < entrada.length()) {
            caracterActual = entrada.charAt(posicion);
        }
    }
```

```
    // Otros m todos del analizador
}
```

6. Estructura del Autómata

6.1. Clase AFD

Listing 2: Clase AFD

```
public class AFD {
    private String nombre;
    private String descripcion;
    private List<String> estados;
    private List<String> alfabeto;
    private String estadoInicial;
    private List<String> estadosFinales;
    private Map<String, Map<String, String>> transiciones;

    // Constructor, getters y setters

    public AFD(String nombre, String descripcion, List<String> estados,
               List<String> alfabeto, String estadoInicial,
               List<String> estadosFinales,
               Map<String, Map<String, String>> transiciones) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.estados = estados;
        this.alfabeto = alfabeto;
        this.estadoInicial = estadoInicial;
        this.estadosFinales = estadosFinales;
        this.transiciones = transiciones;
    }

    // M todos para acceder y manipular el aut mata
}
```

7. Generación de Gráficos

7.1. Implementación del Graficador

El sistema utiliza Java Swing para dibujar los autómatas, representando cada estado como un círculo y las transiciones como flechas. Los estados finales se representan con un doble círculo.

Listing 3: Fragmento de código del graficador

```
public class GraficadorAFD extends JPanel {
    private AFD automata;
```

```
private Map<String, Point> posicionesEstados;

public GraficadorAFD(AFD automata) {
    this.automata = automata;
    this.posicionesEstados = new HashMap<>();
    calcularPosiciones();
}

private void calcularPosiciones() {
    // Algoritmo para distribuir los estados en el panel
    // Se utiliza un algoritmo de distribuci n circular
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                         RenderingHints.VALUE_ANTIALIAS_ON);

    // Dibujar estados
    for (String estado : automata.getEstados()) {
        Point pos = posicionesEstados.get(estado);

        // Dibujar c rculo del estado
        g2d.setColor(Color.WHITE);
        g2d.fillOval(pos.x - 25, pos.y - 25, 50, 50);
        g2d.setColor(Color.BLACK);
        g2d.drawOval(pos.x - 25, pos.y - 25, 50, 50);

        // Si es estado final, dibujar doble c rculo
        if (automata.getEstadosFinales().contains(estado)) {
            g2d.drawOval(pos.x - 20, pos.y - 20, 40, 40);
        }

        // Si es estado inicial, dibujar flecha de entrada
        if (estado.equals(automata.getEstadoInicial())) {
            g2d.drawLine(pos.x - 50, pos.y, pos.x - 25, pos.y);
            g2d.fillPolygon(
                new int[]{pos.x - 25, pos.x - 35, pos.x - 35},
                new int[]{pos.y, pos.y - 5, pos.y + 5},
                3
            );
        }

        // Dibujar etiqueta del estado
        g2d.drawString(estado, pos.x - 10, pos.y + 5);
    }
```

```
        // Dibujar transiciones
        for (String estadoOrigen : automata.getTransiciones().keySet()) {
            Map<String, String> transiciones = automata.getTransiciones().g

            for (String simbolo : transiciones.keySet()) {
                String estadoDestino = transiciones.get(simbolo);
                dibujarTransicion(g2d, estadoOrigen, estadoDestino, simbolo
            }
        }
    }

    private void dibujarTransicion(Graphics2D g2d, String origen, String de
        // Implementaci n para dibujar flechas de transici n
    }
}
```

8. Reportes Generados

8.1. Reporte de Tokens

El sistema genera un reporte detallado de todos los tokens encontrados durante el análisis léxico del archivo de entrada.

Listing 4: Generación del reporte de tokens

```
public class ReporteTokens {
    private ArrayList<Token> tokens;

    public ReporteTokens(ArrayList<Token> tokens) {
        this.tokens = tokens;
    }

    public void generarReporteHTML(String rutaArchivo) {
        StringBuilder html = new StringBuilder();
        html.append("<!DOCTYPE html>\n");
        html.append("<html>\n");
        html.append("<head>\n");
        html.append("<title>Reporte de Tokens</title>\n");
        html.append("<style>\n");
        html.append("table { border-collapse: collapse; width: 100%; }\n");
        html.append("th, td { text-align: left; padding: 8px; }\n");
        html.append("tr:nth-child(even) { background-color: #f2f2f2; }\n");
        html.append("th { background-color: #4CAF50; color: white; }\n");
        html.append("</style>\n");
        html.append("</head>\n");
        html.append("<body>\n");
        html.append("<h1>Reporte de Tokens</h1>\n");
        html.append("<table>\n");
```

```
        html.append("<tr><th>No.</th><th>Tipo</th><th>Lexema</th><th>Posic

        int contador = 1;
        for (Token token : tokens) {
            html.append("<tr>");
            html.append("<td>").append(contador++).append("</td>");
            html.append("<td>").append(token.getTipo()).append("</td>");
            html.append("<td>").append(token.getLexema()).append("</td>");
            html.append("<td>").append(token.getPosicion()).append("</td>"
            html.append("</tr>\n");
        }

        html.append("</table>\n");
        html.append("</body>\n");
        html.append("</html>");

        try (FileWriter writer = new FileWriter(rutaArchivo)) {
            writer.write(html.toString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

8.2. Reporte de Errores Léxicos

También se genera un reporte de errores léxicos encontrados durante el análisis.

Listing 5: Generación del reporte de errores

```
public class ReporteErrores {
    private ArrayList<Error> errores;

    public ReporteErrores(ArrayList<Error> errores) {
        this.errores = errores;
    }

    public void generarReporteHTML(String rutaArchivo) {
        // C digo similar al reporte de tokens, pero para errores
    }
}
```

9. Diagrama de Clases

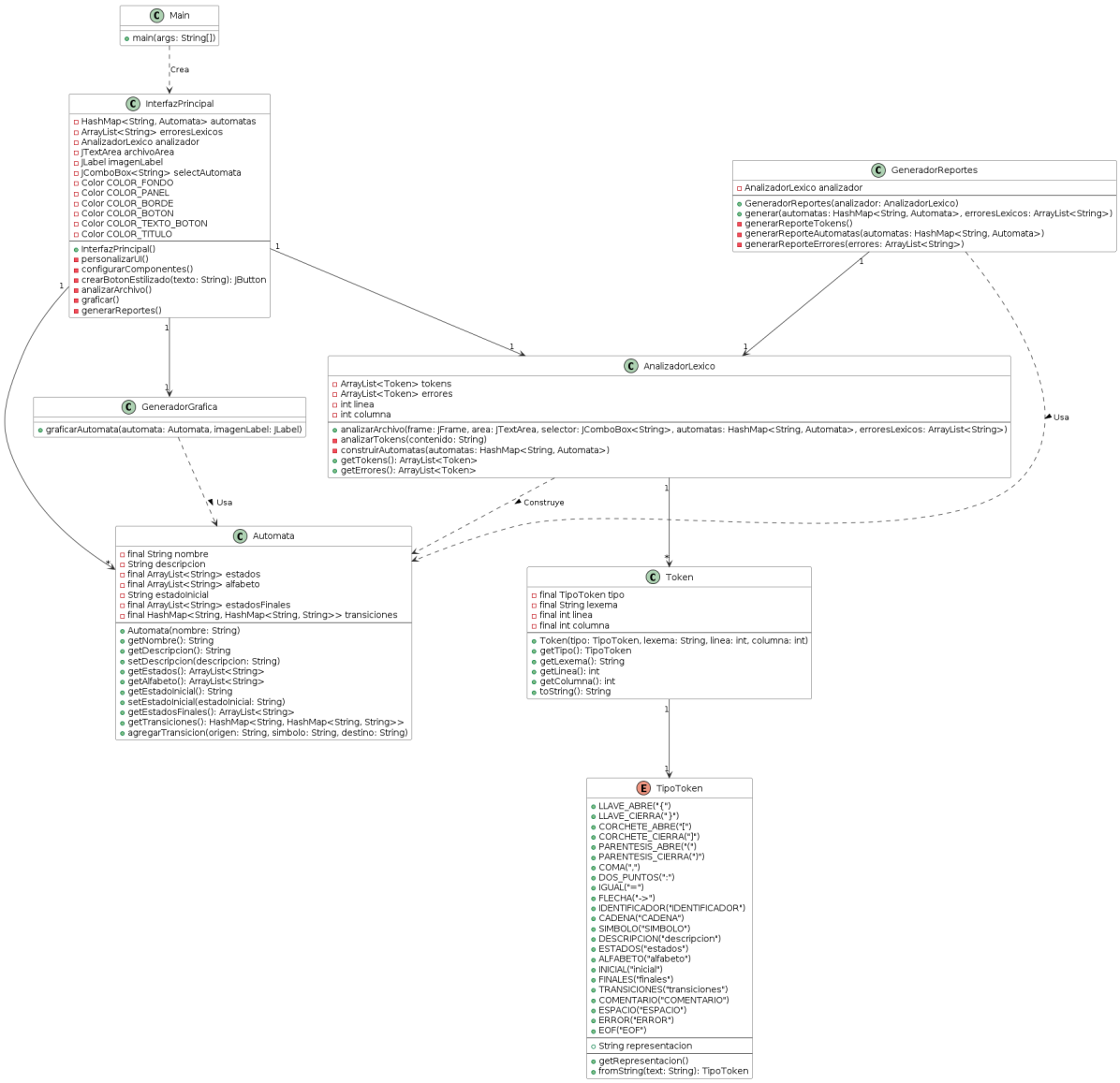


Figura 2: Diagrama de clases del sistema AFDGraph.

10. Diagrama de Flujo del Sistema

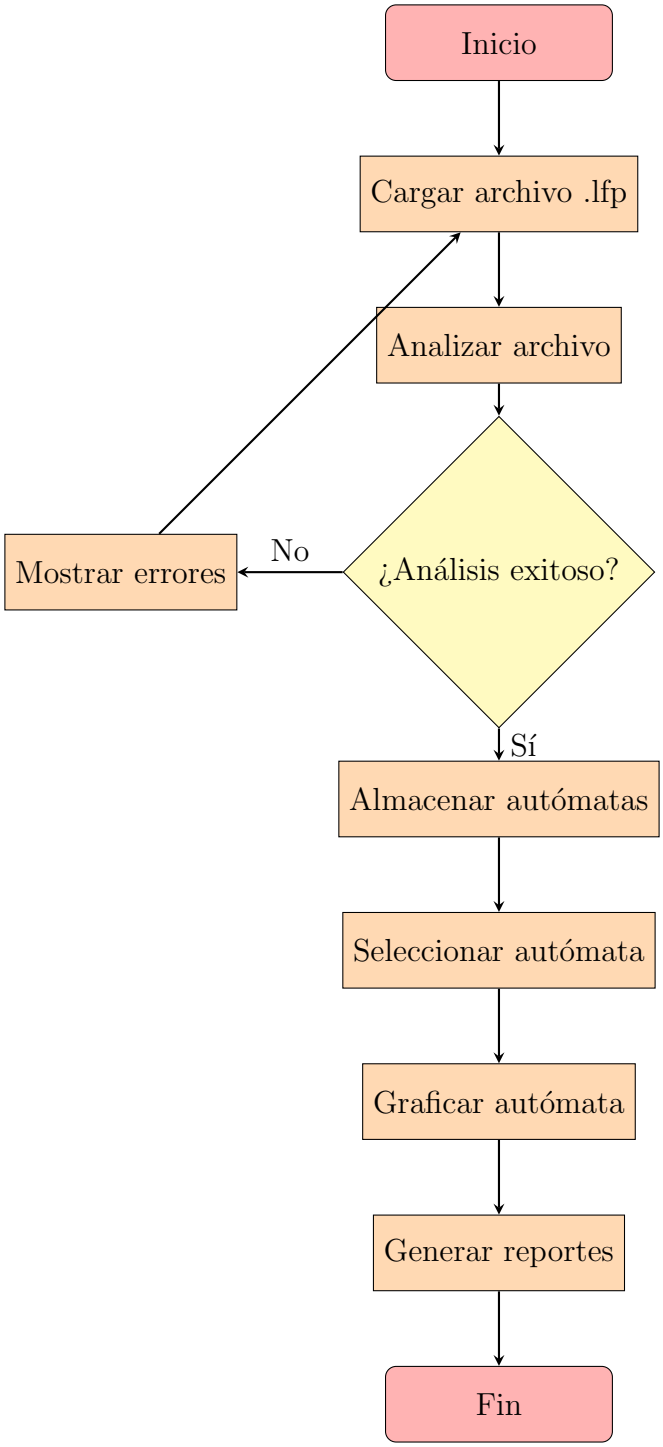
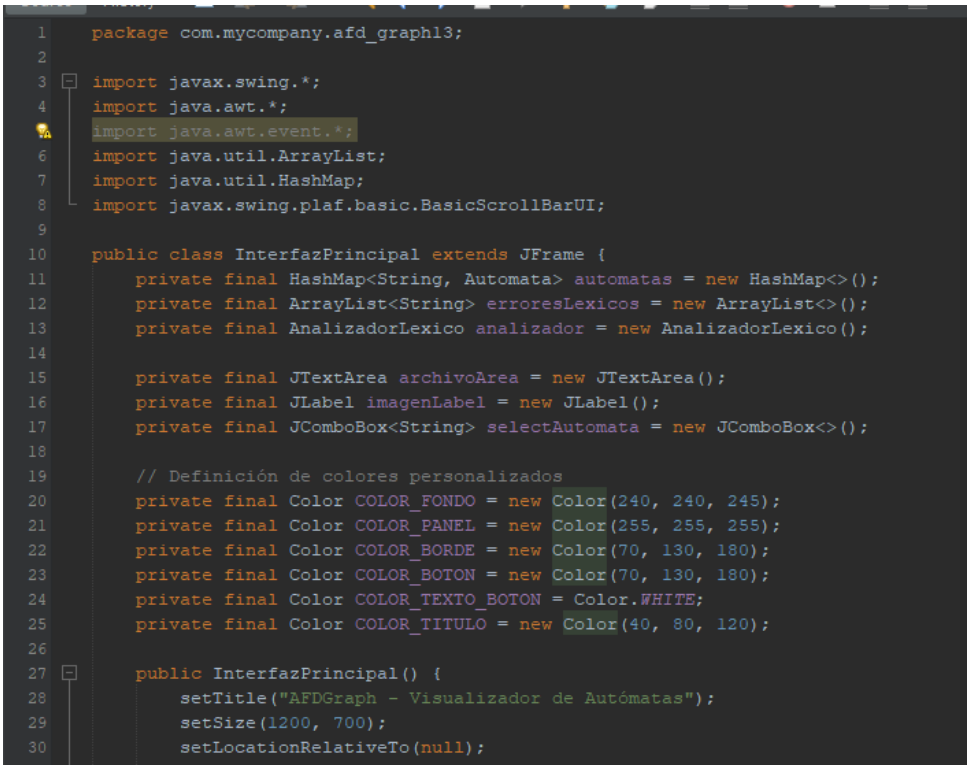


Figura 3: Diagrama de flujo del sistema AFDGraph.

11. Capturas de Pantalla

11.1. Interfaz Principal



```java
package com.mycompany.afd_graph13;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.HashMap;
import javax.swing.plaf.basic.BasicScrollBarUI;

public class InterfazPrincipal extends JFrame {
    private final HashMap<String, Automata> automatas = new HashMap<>();
    private final ArrayList<String> erroresLexicos = new ArrayList<>();
    private final AnalizadorLexico analizador = new AnalizadorLexico();

    private final JTextArea archivoArea = new JTextArea();
    private final JLabel imagenLabel = new JLabel();
    private final JComboBox<String> selectAutomata = new JComboBox<>();

    // Definición de colores personalizados
    private final Color COLOR_FONDO = new Color(240, 240, 245);
    private final Color COLOR_PANEL = new Color(255, 255, 255);
    private final Color COLOR_BORDE = new Color(70, 130, 180);
    private final Color COLOR_BOTON = new Color(70, 130, 180);
    private final Color COLOR_TEXTO_BOTON = Color.WHITE;
    private final Color COLOR_TITULO = new Color(40, 80, 120);

    public InterfazPrincipal() {
        setTitle("AFDGraph - Visualizador de Autómatas");
        setSize(1200, 700);
        setLocationRelativeTo(null);
```

Figura 4: Interfaz principal del programa AFDGraph.

11.2. Visualización de un Autómata

```java
public class GeneradorGrafica {
    public void graficarAutomata(Automata automata, JLabel imagenLabel) {
        try {
            PrintWriter pw = new PrintWriter("automata.dot");
            pw.println("digraph Automata {");
            pw.println("rankdir=LR;");
            pw.println("node [shape = doublecircle]; " + String.join(" ", automata.getEstadosFinales()) + ";");
            pw.println("node [shape = circle];");

            automata.getTransiciones().forEach((origen, mapa) ->
                mapa.forEach((simbolo, destino) ->
                    pw.println(origen + " -> " + destino + " [label=\"" + simbolo + "\"]"))
            );

            pw.println("}");
            pw.close();

            Runtime.getRuntime().exec("dot -Tpng automata.dot -o automata.png").waitFor();
            BufferedImage img = ImageIO.read(new File("automata.png"));
            imagenLabel.setIcon(new ImageIcon(img));
            imagenLabel.revalidate();

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error al graficar: " + e.getMessage());
        }
    }
}
```

Figura 5: Visualización gráfica de un autómata.

11.3. Reporte de Tokens

```
            writer.println("                </thead>");
            writer.println("                <tbody>");

            ArrayList<Token> tokens = analizador.getTokens();
            for (int i = 0; i < tokens.size(); i++) {
                Token token = tokens.get(i);
                writer.println("                    <tr>");
                writer.println("                        <td>" + (i+1) + "</td>");
                writer.println("                        <td><span class='badge badge-primary'>" + token.getTipo() + "</span></td>");
                writer.println("                        <td>" + token.getLexema() + "</td>");
                writer.println("                        <td>" + token.getLinea() + "</td>");
                writer.println("                        <td>" + token.getColumna() + "</td>");
                writer.println("                    </tr>");
            }

            writer.println("                </tbody>");
            writer.println("            </table>");
            writer.println("        </div>");
            writer.println("    </div>");
            writer.println("    <div class='footer'>");
            writer.println("        <p>AFDGraph | Generado: " + java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPatter
            writer.println("    </div>");
            writer.println("</div>");
            writer.println("</body>");
            writer.println("</html>");
        } catch (Exception e) {
            System.err.println("Error al generar reporte de tokens: " + e.getMessage());
        }
```

Figura 6: Reporte de tokens generado por el sistema.

12. Anexos

12.1. Ejemplo de Archivo de Entrada

```
{
AFD1: {
descripcion: "Este autómata reconoce cadenas numéricas.",
estados: [S0, S1, S2, S3, S4, S5, S6, S7, S8],
alfabeto: ["1", "2", "3"],
inicial: S0,
finales: [S0, S1, S2, S3, S5, S6, S7, S8],
transiciones: {
S0 = ("1" -> S1, "2" -> S2, "3" -> S3),
S1 = ("2" -> S1),
S2 = ("2" -> S1, "3" -> S4),
S3 = ("1" -> S5, "2" -> S6, "3" -> S7),
S4 = ("1" -> S8, "3" -> S4),
S5 = ("1" -> S5),
S6 = ("2" -> S6),
S7 = ("1" -> S8, "2" -> S6, "3" -> S7)
}
}
}
```