AFD

LENGUAJE DE PROGRAMACIÓN Y TRANSDUCCIÓN

Universidad Sergio Arboleda

Ciencias de la computación e Inteligencia Artificial

Docente:

Joaquin Fernando Sanchez Cifuentes

Integrantes:

Juan Camilo Lozano Cortes

Bogota

11 de Febrero del 2026

Documentación Técnica: Simulador de Autómata Finito Determinista (AFD)

1. Introducción:

Este proyecto implica el diseño y la simulación de un Autómata Finito Determinista (AFD) orientado a la validación de lenguajes regulares que emula el funcionamiento de un procesador de estados lógico que evalúa cadenas de texto línea a línea desde un archivo externo. Tiene por objetivo demostrar la capacidad discriminatoria de un algoritmo basado en transiciones de estado que acepta o cancela secuencias bajo reglas preestablecidas.

2. Especificaciones de la Lógica Base:

El software se basa en la definición formal de un autómata, utilizando una estructura de control que garantiza la integridad del alfabeto:

- **Nivel de Alfabeto (Σ):** Se define un conjunto estricto de símbolos válidos: {0, 1}.
- **Nivel de Estados (Q):**
    - **q_0 (Estado Inicial):** Punto de entrada. Evalúa el bit más significativo (MSB) o primer carácter.
    - **q_1 (Estado de Aceptación):** Estado de éxito. Se alcanza únicamente si se cumple la condición de inicio.
    - **q_{error} (Estado de Muerte):** Estado de rechazo absoluto. Cualquier símbolo inválido (como el punto .) o un inicio incorrecto (1) redirige el flujo aquí.

3. Algoritmo de Operación:

El sistema procesa la información siguiendo estos pasos lógicos:

1. **Carga de Argumentos:** El programa utiliza la librería sys para recibir el nombre del archivo (entrada.txt) directamente desde la línea de comandos de Linux.
2. **Filtro de Inicio:**
    - Si el primer símbolo es 0, el autómata transita a **q_1**.
    - Si el primer símbolo es 1 o ., el autómata cae en **q_{error}** y detiene el análisis de esa cadena (break).
3. **Propagación de Estados:** Por cada símbolo adicional, el sistema verifica si pertenece al alfabeto. La presencia de cualquier carácter no binario dispara el estado de error.
4. **Veredicto:** Al finalizar la lectura, solo las cadenas que logran permanecer en el estado **q_1** se marcan como ACEPTA.

4. Tablas de Verdad y Transiciones:

Tabla de Transición de Estados (δ):

Esta tabla define el comportamiento del sistema ante cada entrada posible:

Estado Actual	Entrada: 0	Entrada: 1	Entrada: .
q_0	q_1	q_{error}	q_{error}
q_1	q_1	q_1	q_{error}
q_{error}	q_{error}	q_{error}	q_{error}

Lógica de Selección de Veredicto:

Python
Si Estado Final == q1 -> ACEPTA
Si Estado Final == q_error -> NO ACEPTA

5. Instrucciones de Instalación y Uso en Linux:

1. **Navegación:** Es obligatorio situarse en la carpeta donde reside el código mediante la terminal:
2. Bash

   cd /ruta/de/tu/carpeta/
3.
4. **Archivo de Entrada:** Asegurarse de que el archivo entrada.txt contenga las cadenas deseadas.
5. **Ejecución:** Invoque el script utilizando Python 3:
6. Bash

   python3 AFD.py entrada.txt

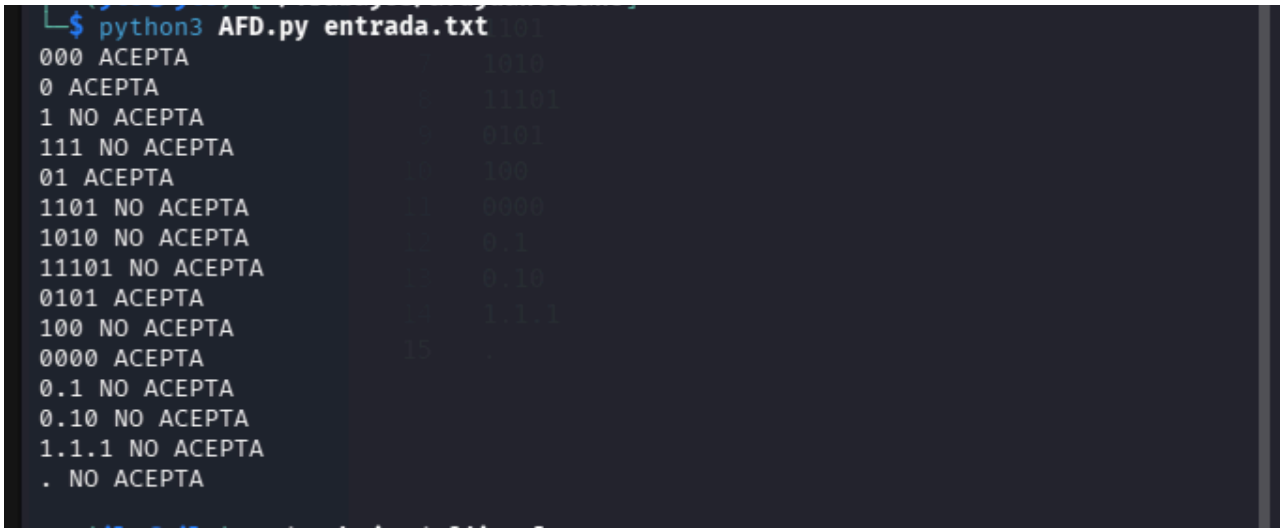
6. Pruebas de Escritorio (Casos de Éxito y Validación)

A continuación, se documentan los resultados obtenidos tras la validación del sistema con los 15 casos de prueba oficiales:

Entrada	Salida del Sistema	Análisis Técnico

000	000 ACEPTA	Inicio en 0, alfabeto válido.
0	0 ACEPTA	Cumple longitud mínima y prefijo.
1	1 NO ACEPTA	Violación de prefijo en q_0.
111	111 NO ACEPTA	Inicio inválido.
01	01 ACEPTA	Transición exitosa $q_0 \to q_1$.
1101	1101 NO ACEPTA	Inicio inválido.
1010	1010 NO ACEPTA	Inicio inválido.
11101	11101 NO ACEPTA	Inicio inválido.
0101	0101 ACEPTA	Cadena binaria válida.
100	100 NO ACEPTA	Inicio inválido.
0000	0000 ACEPTA	Cadena binaria válida.
0.1	0.1 NO ACEPTA	Carácter '.' fuera del alfabeto.

0.10	0.10 NO ACEPTA	Carácter '.' detectado en proceso.
1.1.1	1.1.1 NO ACEPTA	Error múltiple de inicio y alfabeto.
.	. NO ACEPTA	Carácter inválido único.



7. Conclusión:

El simulador desarrollado también obedece a los principios teóricos de la computación en torno a un AFD; es decir, se consiguió una arquitectura que rechazase cadenas por dos vías: por violación de la gramática de inicio y por violación de caracteres ajenos a los del alfabeto definido. Además, el sistema es apto en cuanto a eficiencia, ya que demarcará el procesamiento (mediante break) en el preciso instante que advierte una inconsistencia, tal y como acontece en un circuito lógico real, donde la caída de corriente se produce ante una entrada no válida.