

# Reporte de Ejercicio Simulación de un Autómata Finito Determinista

LENGUAJES FORMALES

ALUMNO: Ulises Herrera Rodríguez



## 1. Introducción

Este reporte describe la implementación de un Autómata Finito Determinista (AFD) en C++. Un AFD es un modelo matemático de computación utilizado para reconocer patrones en cadenas de caracteres. Su importancia radica en su aplicación en el análisis de lenguajes formales y compiladores.

Los autómatas finitos deterministas (AFD) son un modelo matemático ampliamente utilizado en la teoría de la computación para el reconocimiento de lenguajes formales. Estos autómatas son herramientas fundamentales en el diseño de analizadores léxicos, validadores de cadenas y otros sistemas de procesamiento de información. Un AFD se define formalmente como una 5-tupla  $Q, \Sigma, \delta, q_0, F$ , donde  $Q$  es un conjunto infinito de estados,  $\Sigma$  es un alfabeto finito,  $\delta$ , es una función de transición que define el cambio de estados en respuesta a una entrada,  $q_0$  es el estado inicial y  $F$  es el conjunto de aceptación.

## **2. Marco teórico**

### **Autómatas Finitos Deterministas**

Los autómatas finitos deterministas son máquinas abstractas que procesan cadenas de símbolos de acuerdo con un conjunto de reglas de transición.

### **Estados y transiciones**

Un AFD está definido por un conjunto de estados, un alfabeto, una función de transición y estados de aceptación.

### **Importancia de los autómatas**

Los AFD se utilizan en compiladores, validación de cadenas, expresiones regulares y diseño de sistemas lógicos.

### **3. Herramientas empleadas**

1. C++: Lenguaje utilizado para la implementación.
2. Compilador GCC/G++: Para la ejecución del código.
3. Entorno de desarrollo: Visual Studio Code.
4. Suit del Ejercicio: OmegaUP.

## 4. Desarrollo

### Descripción

Un autómata finito determinista (AFD) es una 5-tupla  $(Q, \Sigma, \delta, q_0, F)$ , donde:

1.  $Q$  es un conjunto finito llamado **estados**.
2.  $\Sigma$  es un conjunto finito llamado **alfabeto**.
3.  $\delta : Q \times \Sigma \rightarrow Q$  es la **función de transición**.
4.  $q_0 \in Q$  es el **estado inicial**.
5.  $F \subseteq Q$  es el **conjunto de estados de aceptación**.

Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un AFD y sea  $w = w_1 w_2 \cdots w_n$  una cadena de símbolos donde  $w_i \in \Sigma$ . Entonces,  $M$  **acepta**  $w$  si existe una secuencia de estados  $r_0, r_1, \dots, r_n$  en  $Q$  con tres condiciones:

1.  $r_0 = q_0$ .
2.  $\delta(r_i, w_{i+1}) = r_{i+1}$ , para  $i = 0, \dots, n-1$ .
3.  $r_n \in F$ .

La condición 1 establece que la máquina comienza en el estado inicial. La condición 2 establece que la máquina cambia de estado de acuerdo con la función de transición. La condición 3 establece que la máquina acepta la cadena de entrada si termina en un estado de aceptación. Entonces,  $M$  **reconoce el lenguaje**  $A$  si  $A = \{w | M \text{ acepta } w\}$ .

Escribir un programa para determinar si un conjunto de cadenas  $W$  pertenecen o no al lenguaje  $A$  reconocido por un AFD  $M$ .

### Entrada

La primera línea de entrada contiene seis enteros  $N, S, D, q_0, T$  y  $C$ , donde:

- $1 \leq N, S, C \leq 100$ .
- $1 \leq D \leq 10^4$ .
- $1 \leq q_0 \leq N, 0 \leq T \leq N$ .

Cada estado  $q \in Q$  se representa por un número entero entre 1 y  $N$ . La segunda línea contiene el alfabeto  $\Sigma$ , representado por una secuencia de  $S$  símbolos separados por espacios. La tercera línea contiene el conjunto de estados de aceptación  $F$ , representado por  $T$  enteros separados por espacios.

Las siguientes  $D$  líneas contienen transiciones en la forma  $(I, X, J)$ , indicando que desde el estado  $I$ , al leer el símbolo  $X$ , se transita al estado  $J$ .

Finalmente, las siguientes  $C$  líneas contienen cadenas  $W$ , de longitud entre 0 y 100 caracteres.

### Salida

Para cada cadena  $W$ , imprimir:

- **ACEPTADA** si el autómata acepta  $W$ .
- **RECHAZADA** en caso contrario.

### Ejemplo

Ejemplo de entrada y salida:

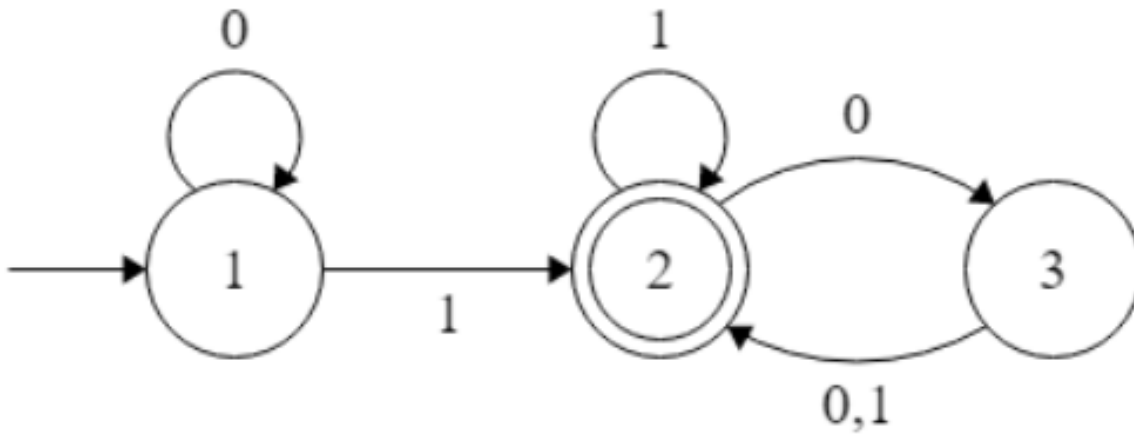
Entrada	Salida
3 2 6 1 1 3	ACEPTADA
0 1	RECHAZADA
2	ACEPTADA

Table 1: Ejemplo de entrada y salida

## Descripción del Ejemplo

En el ejemplo el autómata tiene  $N = 3$  estados, un alfabeto con  $S = 2$  símbolos (segunda línea  $\Sigma = 0, 1$ ), se define  $D = 6$  transiciones (líneas 4 - 9), el estado inicial es  $q_0 = 1$ , solamente hay  $T = 1$ , estados de aceptación (tercer línea  $F = 2$ ) y se verifican  $C = 3$  palabras  $W$  (líneas 10 - 12). La palabra 100 es aceptada, 0 es rechazada y 11 es aceptada. El autómata reconoce palabras que tienen una cantidad par de 0's después del último 1.

La siguiente figura es una representación gráfica del autómata ejemplo.



## Lógica de Implementación

El programa recibe como entrada:

- Conjunto de estados.
- Alfabeto del autómata.
- Estado inicial.
- Estados de aceptación.
- Transiciones del autómata.
- Cadenas de prueba.

## Código fuente

El siguiente fragmento de código representa la implementación del AFD:

Listing 1: Implementación del AFD

```

#include <iostream>
#include <unordered_map>




```

```

#include <unordered_set>
#include <vector>
using namespace std;
int main() {
    int N, S, D, q0, T, C;
    cin >> N >> S >> D >> q0 >> T >> C;
    unordered_set<char> alphabet;
    for (int i = 0; i < S; i++) {
        char symbol;
        cin >> symbol;
        alphabet.insert(symbol);
    }
    unordered_set<int> acceptStates;
    for (int i = 0; i < T; i++) {
        int state;
        cin >> state;
        acceptStates.insert(state);
    }
    unordered_map<int, unordered_map<char, int>> transitions;
    for (int i = 0; i < D; i++) {
        int from, to;
        char symbol;
        cin >> from >> symbol >> to;
        transitions[from][symbol] = to;
    }
    vector<string> words(C);
    cin.ignore();
    for (int i = 0; i < C; i++) {
        getline(cin, words[i]);
    }
    for (const string& word : words) {
        int currentState = q0;
        bool valid = true;
        for (char c : word) {
            if (transitions[currentState].find(c) == transitions[currentState].end()) {
                valid = false;
                break;
            }
            currentState = transitions[currentState][c];
        }
        if (valid && acceptStates.find(currentState) != acceptStates.end()) {
            cout << "ACEPTADA" << endl;
        } else {
            cout << "RECHAZADA" << endl;
        }
    }
    return 0;
}

```

## Ejecución del programa

 Fecha y hora	Lenguaje	Porcentaje	Ejecución	Salida	 Memoria	 Tiempo	Acciones
2025-02-26 10:26	cpp20-gcc	100.00%	Terminada	Correcta 	3.38 MB	0.01 s	



## Conclusiones

El estudio de los autómatas finitos deterministas permite comprender los fundamentos de la computación teórica y su aplicación en diversas áreas, como el análisis léxico en compiladores, la validación de patrones en expresiones regulares y el diseño de sistemas de control de flujo en software.

En este documento, se ha presentado una forma estructurada para definir y evaluar un AFD, estableciendo sus parámetros de entrada, sus reglas de transición y su mecanismo de aceptación o rechazo de cadenas. La representación tabular y la definición de los estados permiten una implementación eficiente del autómata, facilitando su análisis y su uso en sistemas automatizados.

El reconocimiento de cadenas mediante un AFD es un proceso determinista y eficiente, que garantiza resultados consistentes en la evaluación de entradas. Esto demuestra la importancia de los autómatas finitos en la construcción de herramientas computacionales robustas y precisas para la manipulación de lenguajes formales.

## Referencias Bibliográficas

## References

- [1] Hopcroft, J. E., Motwani, R., Ullman, J. D. (**2006**). *Introduction to Automata Theory, Languages, and Computation*. Pearson.