Universidad Rafael Landívar

Facultad de Ingeniería

Laboratorio de Arquitectura del Computador

Catedrático: Ing. Juan Carlos, Soto Santiago

DOCUMENTACION

Godinez Gudiel, Javier Estuardo

Carné: 1179222

Cuevas Lau, Ubaldo Sebastian

Carne: 1034222

Ovalle Montenegro, Saul Alejandro

Carne: 1226122

Guatemala de la Asunción, 11 de Mayo del 2024

Contenido

OBJETIVO

El objetivo del programa es proporcionar una herramienta para manejar autómatas finitos no deterministas (AFND) en C#, permitiendo cargar la configuración del autómata desde un archivo, agregar transiciones, verificar cadenas y mostrar resultados de estas verificaciones.

FUNCIONALIDADES

- **Cargar un Autómata**: El programa puede leer desde un archivo los estados, el estado inicial, los estados finales y las transiciones de un autómata finito no determinista (AFND).
- **Agregar Transiciones**: Permite añadir nuevas transiciones al autómata finito no determinista (AFND) en tiempo de ejecución.
- **Verificación de Cadenas**: Verifica si una cadena dada es aceptada por el autómata finito no determinista (AFND), mostrando las transiciones utilizadas durante el proceso.
- **Interfaz de Usuario**: Proporciona una interfaz de consola para interactuar con el usuario, permitiendo realizar las operaciones anteriores y visualizar resultados.

ESPECIFICACIONES

Entradas:

- Archivos de configuración del AFND en formatos TXT, JSON y CSV. Cada formato tiene una estructura específica que debe ser documentada para que los usuarios sepan cómo preparar sus archivos.
- Cadenas ingresadas por el usuario para verificar contra el autómata.

Procesos:

- Lectura y análisis del archivo para construcción del autómata finito no determinista (AFND).
- Adición de transiciones al autómata finito no determinista (AFND).
- Evaluación de cadenas con registro de transiciones utilizadas.

Salidas:

- Mensajes de error o confirmación sobre la carga y análisis del archivo.
- Resultado de la evaluación de cadenas (aceptada o no).
- Listado de transiciones utilizadas durante la evaluación.

DISEÑO DE LA SOLUCIÓN

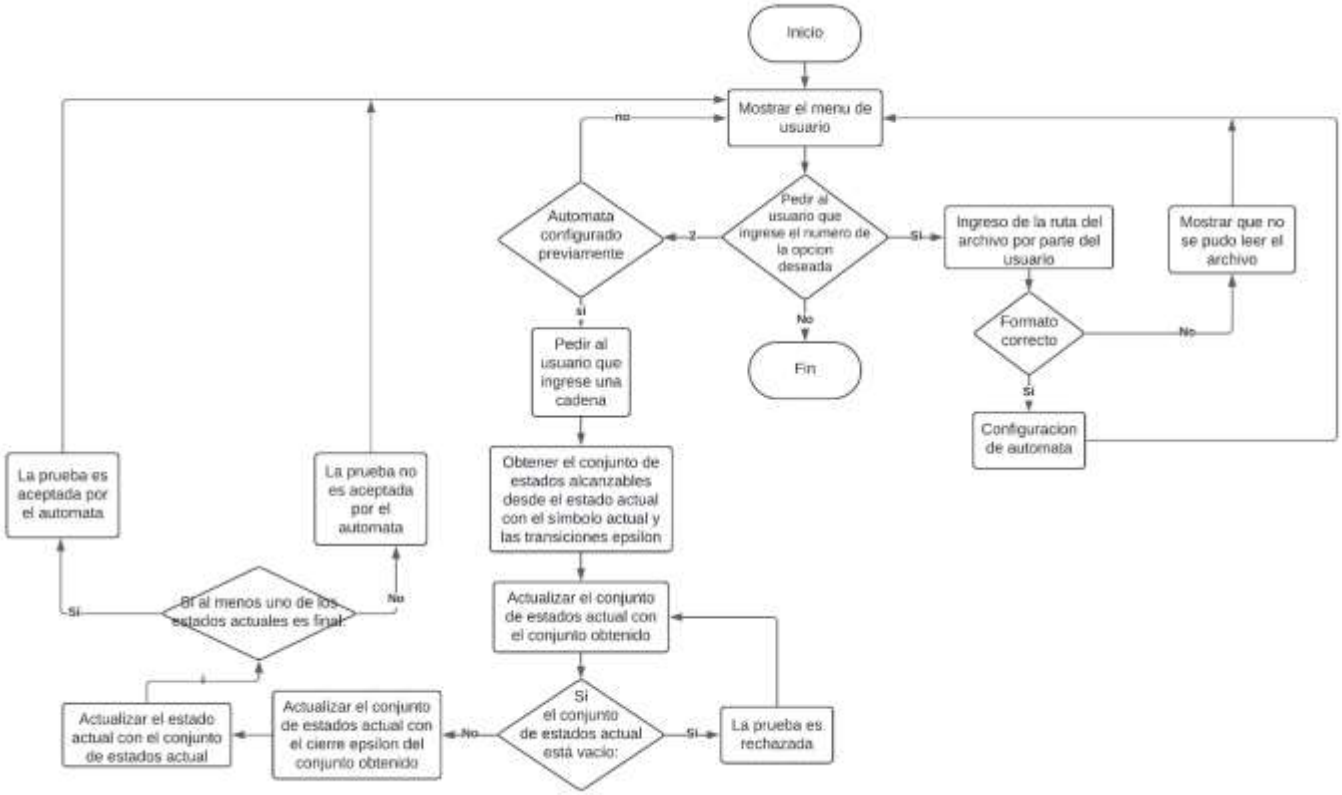
NFA

- **Atributos**:
    - **numberOfStates**: Cantidad de estados del autómata.
    - **initialState**: Estado inicial.
    - **finalStates**: Conjunto de estados finales.
    - **transitions**: Diccionario para almacenar las transiciones.
    - **acceptedPaths**: Lista de caminos aceptados durante la verificación de cadenas.
- **Métodos**:
    - **Constructor**: Inicializa el autómata finito no determinista (AFND).
    - **LoadFromFile**: Lee y carga la configuración del autómata desde un archivo en formatos TXT, JSON o CSV.
    - **ParseFile**: Analiza el contenido del archivo para configurar el autómata.
    - **LoadFromCSV** y **LoadFromJSON**: Métodos auxiliares para cargar la configuración desde archivos CSV y JSON, respectivamente.
    - **Accepts**: Verifica si una cadena es aceptada por el autómata, registrando los caminos utilizados durante la verificación.
    - **CheckAccepts**: Método recursivo para verificar la aceptación de una cadena por el autómata, explorando todos los posibles caminos.
    - **PrintTransitions**: Imprime todas las transiciones del autómata.

Program

- **Métodos**:
    - **Main**: Bucle principal que maneja la interfaz de usuario y las interacciones.
    - **Menu**: Muestra opciones disponibles al usuario.

DIAGRAMA DE FLUJO



PSEUDOCODIGO

```
// Definición de la clase NFA (Autómata Finito No Determinista)
class NFA {
    // Atributos privados de la clase
    private int numberOfStates;              // Número de estados del autómata
    private int initialState;                // Estado inicial del autómata
    private HashSet<int> finalStates;        // Conjunto de estados finales del autómata
    private Dictionary<(int, string), List<int>> transitions;  // Transiciones del autómata
    private List<string> acceptedPaths;      // Caminos aceptados durante la verificación de cadenas

    // Constructor de la clase
    public NFA() {
        transitions = new Dictionary<(int, string), List<int>>(); // Inicializa el diccionario de transiciones
        finalStates = new HashSet<int>(); // Inicializa el conjunto de estados finales
    }

    // Método para cargar la configuración del autómata desde un archivo
    public void LoadFromFile(string filePath) {
        Console.Clear(); // Limpiar la consola antes de cargar el archivo

        // Verifica la extensión del archivo y carga la configuración correspondiente
        if (Path.GetExtension(filePath).Equals(".csv", StringComparison.OrdinalIgnoreCase)) {
            LoadFromCSV(filePath); // Carga la configuración desde un archivo CSV
        } else if (Path.GetExtension(filePath).Equals(".json", StringComparison.OrdinalIgnoreCase)) {
            LoadFromJSON(filePath); // Carga la configuración desde un archivo JSON
        } else {
            ParseFile(filePath); // Analiza el archivo y carga la configuración
        }
```

```
}

// Método privado para analizar el archivo y cargar la configuración del autómata
private void ParseFile(string filePath) {
    // Lee todas las líneas del archivo
    string[] lines = File.ReadAllLines(filePath);

    // Extrae y asigna el número de estados y el estado inicial
    numberOfStates = int.Parse(lines[0].Trim());
    initialState = int.Parse(lines[1].Trim());

    // Agrega los estados finales al conjunto de estados finales
    foreach (var state in lines[2].Split(',')) {
        finalStates.Add(int.Parse(state.Trim()));
    }

    // Itera sobre las líneas restantes para agregar las transiciones al diccionario de transiciones
    for (int i = 3; i < lines.Length; i++) {
        string[] parts = lines[i].Split(',');
        int fromState = int.Parse(parts[0].Trim());
        string symbol = parts[1].Trim() == "" ? "epsilon" : parts[1].Trim();
        int toState = int.Parse(parts[2].Trim());

        // Crea la clave de la transición y la agrega al diccionario de transiciones
        var key = (fromState, symbol);
        if (!transitions.ContainsKey(key)) {
            transitions[key] = new List<int>();
        }
        transitions[key].Add(toState);
```

```
    }
}

// Método privado para cargar la configuración desde un archivo CSV
private void LoadFromCSV(string filePath) {
    // Utiliza TextFieldParser para parsear el archivo CSV
    using (TextFieldParser parser = new TextFieldParser(filePath)) {
        parser.TextFieldType = FieldType.Delimited;
        parser.SetDelimiters(",");

        // Lee cada línea del archivo y agrega las transiciones al diccionario de transiciones
        while (!parser.EndOfData) {
            string[] parts = parser.ReadFields();
            int fromState = int.Parse(parts[0].Trim());
            string symbol = parts[1].Trim() == "" ? "epsilon" : parts[1].Trim();
            int toState = int.Parse(parts[2].Trim());

            var key = (fromState, symbol);
            if (!transitions.ContainsKey(key)) {
                transitions[key] = new List<int>();
            }
            transitions[key].Add(toState);
        }
    }
}

// Método privado para cargar la configuración desde un archivo JSON
private void LoadFromJSON(string filePath) {
    // Lee el contenido del archivo JSON
```

```
        string jsonContent = File.ReadAllText(filePath);
        // Deserializa el contenido JSON en un diccionario de transiciones
        var data = JsonConvert.DeserializeObject<Dictionary<string, List<(int, string, int)>>>(jsonContent);

        // Itera sobre el diccionario y agrega las transiciones al diccionario de transiciones
        foreach (var item in data) {
            var fromState = int.Parse(item.Key);
            foreach (var transition in item.Value) {
                var symbol = transition.Item2 == "" ? "epsilon" : transition.Item2;
                var toState = transition.Item3;

                var key = (fromState, symbol);
                if (!transitions.ContainsKey(key)) {
                    transitions[key] = new List<int>();
                }
                transitions[key].Add(toState);
            }
        }
    }

    // Método público para verificar si una cadena es aceptada por el autómata
    public List<string> Accepts(string input) {
        acceptedPaths = new List<string>(); // Inicializa la lista de caminos aceptados
        CheckAccepts(initialState, input, ""); // Verifica la aceptación de la cadena
        return acceptedPaths; // Devuelve los caminos aceptados
    }
```

```
    // Método privado recursivo para verificar la aceptación de la cadena por el autómata
    private void CheckAccepts(int currentState, string input, string currentPath) {
        // Construye un nuevo camino basado en el estado actual y la entrada actual
        string newPath = currentPath + (currentPath == "" ? "" : " -> ") + currentState;

        // Si no hay más entrada, verifica si el estado actual es final y agrega el camino a los caminos aceptados
        if (input == "") {
            if (finalStates.Contains(currentState)) {
                acceptedPaths.Add(newPath + " -> Accept");
            }
        } else {
            // Si hay entrada, obtiene el símbolo de entrada y las transiciones desde el estado actual con ese símbolo
            string remainingInput = input.Substring(1);
            string symbol = input.Substring(0, 1);

            // Si hay transiciones desde el estado actual con el símbolo de entrada, continúa verificando la cadena para cada estado de destino
            if (transitions.TryGetValue((currentState, symbol), out var states)) {
                foreach (var state in states) {
                    CheckAccepts(state, remainingInput, newPath + " ->(" + symbol + ")");
                }
            }
        }

        // Verifica las transiciones epsilon desde el estado actual en cualquier punto del proceso
        if (transitions.TryGetValue((currentState, "epsilon"), out var epsilonStates)) {
            foreach (var nextState in epsilonStates) {
                CheckAccepts(nextState, input, newPath + " ->(epsilon)");
```

```
        }
    }
}

// Método público para imprimir todas las transiciones del autómata
public void PrintTransitions() {
    Console.WriteLine("Possible transitions within the NFA:"); // Imprime un encabezado
    // Itera sobre las transiciones y las imprime en el formato especificado
    foreach (var transition in transitions) {
        var (fromState, symbol) = transition.Key;
        var toStates = transition.Value;
        foreach (var toState in toStates) {
            Console.WriteLine($"{fromState} ->({symbol})-> {toState}");
        }
    }
}
}
```