REPORTE DE PRÁCTICA AFD

UNIVERSIDAD AUTONOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BASICAS E INGENIERIA

LICENCIATURA EN CIENCIAS COMPUTACIONALES

AUTOMATAS Y COMPILADORES

GRAMATICAS Y LENGUAJES FORMALES

ALUMNO: Diego Martinez Ortiz

Dr. Eduardo Cornejo-Velázquez

1. Introducción

Los autómatas finitos deterministas (AFD) son modelos matemáticos utilizados para representar lenguajes formales. Estos autómatas consisten en un conjunto finito de estados, un alfabeto de símbolos de entrada, y transiciones que determinan el siguiente estado en función del estado actual y el símbolo de entrada. En esta práctica, se implementa un programa en C++ que simula el comportamiento de un autómata finito determinista. El objetivo de este programa es leer un conjunto de parámetros de entrada y cadenas, y verificar si dichas cadenas son aceptadas o rechazadas por el autómata, basado en sus transiciones y estados de aceptación.

2. Marco teórico

Automatas Finito Deterministas (AFD)

Un autómata finito determinista (AFD) es una máquina teórica que consiste en los siguientes componentes:

Estados

Un conjunto finito de estados, incluyendo un estado inicial y uno o más estados finales o de aceptación.

Alfabeto

Un conjunto de símbolos que el autómata puede leer. El autómata lee cadenas de este alfabeto y cambia de estado en función de las transiciones.

Transiciones

Funciones que determinan, para cada estado y símbolo de entrada, el siguiente estado al que se mueve el autómata.

Estado inical

El estado en el que comienza el autómata.

Estado de aceptacion

Estados de aceptación: Un conjunto de estados donde el autómata puede aceptar una cadena si termina en cualquiera de estos estados. Un autómata finito determinista procesa una cadena de entrada símbolo por símbolo, siguiendo las transiciones correspondientes, y al final acepta la cadena si termina en un estado de aceptación. Si se llega a un estado que no tiene una transición válida para un símbolo, el autómata rechaza la cadena.

Aceptacion y rechazo

La aceptación de una cadena se determina al procesar cada símbolo de la cadena: - Si el autómata termina en un estado de aceptación, la cadena es aceptada. - Si el autómata llega a un estado sin transición válida o no termina en un estado de aceptación, la cadena es rechazada.

3. Herramientas empleadas

Lenguaje de Programacion C++

Se utilizó el lenguaje C++ para implementar el autómata. C++ es un lenguaje de programación que ofrece control de bajo nivel sobre la memoria y un alto rendimiento, lo que lo hace adecuado para implementaciones eficientes de estructuras de datos y algoritmos complejos.

Estructura de datos

- unorderedmap: Utilizado para almacenar las transiciones del autómata de manera eficiente. Se usa para mapear el estado actual y el símbolo de entrada a un estado destino.
- vector: Usado para almacenar los estados finales y el conjunto de cadenas de entrada.

Algoritmos

- Búsqueda de estado final: Se utiliza find() para verificar si el autómata termina en un estado final después de procesar una cadena.
- Procesamiento de cadenas: Para cada cadena de entrada, se recorre carácter por carácter, y se sigue la transición correspondiente hasta determinar si la cadena es aceptada o rechazada.

4. Desarrollo

Entrada de datos

El programa comienza solicitando la entrada de los parámetros iniciales:
- Número de estados, símbolos, transiciones, estado inicial, número de estados finales y el número de cadenas a procesar.
- Se capturan los símbolos del alfabeto y se inicializan las transiciones entre estados.

Carga de Transiciones y Estados Finales

- Se cargan las transiciones del autómata en un mapa de mapas (unorderedmap¡int, unorderedmap¡char, int), donde las claves son los estados y los valores son los símbolos de entrada que conducen a otros estados.
- Se capturan los estados de aceptación (finales).

Procesamiento de Cadenas

Para cada cadena de entrada:
- Se inicia en el estado inicial.
- Se procesa cada símbolo de la cadena, moviéndose entre estados según las transiciones definidas.
- Si se encuentra una transición inválida (es decir, no existe una transición definida para el estado actual y el símbolo), el autómata rechaza la cadena.
- Al final de la cadena, se verifica si el autómata terminó en un estado de aceptación. Si es así, la cadena es aceptada; de lo contrario, es rechazada.

Salida de datos

El programa imprime "ACEPTADA" si la cadena es aceptada, o "RECHAZADA" si no lo es.

```cpp
#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int main() {
    // Declaración de variables
    int numEstados, numSimbolos, numTransiciones, estadoInicial, numFinales, numCadenas
    int estadoActual;
    char simbolo;
    string cadena;

    // Entrada de parámetros iniciales
    cin >> numEstados >> numSimbolos >> numTransiciones >> estadoInicial >> numFinales
    estadoActual = estadoInicial;

    // Conjunto de estados de aceptación
    vector<int> estadosFinales(numFinales);
    unordered_map<int, unordered_map<char, int>> transiciones;

    // Inicialización del alfabeto con transiciones nulas
    for (int i = 1; i <= numSimbolos; i++) {
        cin >> simbolo;
        for (int j = 1; j <= numEstados; j++) {
            transiciones[j][simbolo] = -1;
        }
    }
```

Figure 1: Caption

```
    // Carga de estados finales
    for (int i = 0; i < numFinales; i++) {
        cin >> estadosFinales[i];
    }

    // Definición de transiciones
    for (int i = 0; i < numTransiciones; i++) {
        int estadoOrigen, estadoDestino;
        char entrada;
        cin >> estadoOrigen >> entrada >> estadoDestino;
        transiciones[estadoOrigen][entrada] = estadoDestino;
    }

    cin.ignore(); // Ignorar salto de línea pendiente

    // Procesamiento de cadenas
    for (int i = 0; i < numCadenas; i++) {
        getline(cin, cadena);
        estadoActual = estadoInicial;

        if (cadena.empty()) {
            cout << (find(estadosFinales.begin(), estadosFinales.end(), estadoActual) !
        } else {
            for (char c : cadena) {
                if (transiciones[estadoActual].count(c) && transiciones[estadoActual][c
                    estadoActual = transiciones[estadoActual][c];
                } else {
                    estadoActual = -1;
                    break;
```

Figure 2: Caption

```
                }
            }
            cout << ((estadoActual != -1 && find(estadosFinales.begin(), estadosFinales
        }
    }

    return 0;
}
```

Figure 3: Caption

5. Conclusiones

El código implementa correctamente un autómata finito determinista en C++, permitiendo procesar múltiples cadenas de entrada y verificar si son aceptadas o rechazadas en función de las transiciones definidas y los estados de aceptación. El uso de estructuras de datos como unorderedmap y vector facilita la implementación eficiente del autómata y la manipulación de las transiciones y los estados. Esta práctica permite entender cómo funcionan los autómatas finitos y su aplicación en la teoría de lenguajes formales y la computación, demostrando cómo se puede modelar y simular el comportamiento de un autómata en un programa de software. El programa es flexible y permite agregar cualquier número de estados, símbolos y transiciones, lo que lo convierte en una herramienta útil para experimentar con autómatas y realizar pruebas sobre cadenas de diferentes longitudes y características.