UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO - UFES
CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO - CEUNES
CIÊNCIA DA COMPUTAÇÃO
LINGUAGENS FORMAIS E AUTÔMATOS

JOÃO PAULO SOUZA FERRETE

RELATÓRIO

SIMULADOR UNIVERSAL DE AUTÔMATOS FINITOS DETERMINÍSTICOS

SÃO MATEUS
15 DE AGOSTO DE 2021

INTRODUÇÃO

O objetivo deste relatório é apresentar, de forma mais detalhada, o algoritmo implementado para o simulador de autômatos finitos, que foi desenvolvido a partir dos conceitos discutidos e aprendidos em aula, incluindo suas funções e explicando seu funcionamento.

IMPLEMENTAÇÕES

O programa foi desenvolvido em linguagem C, mas utilizando a definição de novos tipos, de forma a facilitar a implementação e o entendimento do código durante e após o desenvolvimento.

Inclusão de Bibliotecas e Definição de Tipos

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum{
   true = 1,
   false = 0
}boolean;
```

Inicialmente, foram incluídas as bibliotecas padrões da linguagem, e a biblioteca *'string.h'*, já que o programa lidará, principalmente, com cadeias de caracteres. Após as inclusões, foi definido o novo tipo, chamado de boolean, para facilitar o processo de implementação, a partir de um enumerador, onde a palavra-chave *true* passa a ser equivalente a 1 (ou ligado, em C) e a palavra-chave *false* passa a ser equivalente a 0 (ou desligado, em C).

Função *estadoFinal()*

```c
boolean estadoFinal(char estado, int numEstados, char
estadosFinais[]){
   for(int i = 0; i < numEstados; i++){
       if(estado == estadosFinais[i]){
           return true;
       }
   }
   return false;
}
```

A função *estadoFinal* é responsável por verificar se um estado, passado por parâmetro, é um estado final. Ela também recebe por parâmetro o número de estados, para utilização no loop, e o vetor de estados finais. Sua execução acontece a partir de cada iteração do laço que, se encontrar um estado final igual ao estado atual, retorna *true*. Se, ao fim do laço, nenhum elemento corresponder ao estado atual, a função retorna *false*.

Função *taNoDicionario()*

```
boolean taNoDicionario(char elemento, char simbolosFinais[],
int numSimbolosFinais){
   for(int i = 0; i < numSimbolosFinais; i++){
       if(elemento == simbolosFinais[i]){
           return true;
       }
   }
   return false;
}
```

Esta função tem por objetivo verificar se um elemento, passado por parâmetro, pertence ao dicionário (ou alfabeto) aceito pelo autômato. Ela também recebe o vetor que contém o alfabeto e o número de elementos neste vetor. Seu funcionamento é de forma similar à função anterior, consistindo em um laço para verificar se o elemento é igual a um dos elementos do dicionário. Em caso positivo se retorna *true, mas,* caso o laço termine, então não foi encontrado nenhum elemento correspondente, então é retornado *false*.

Função *pertence()*

```
boolean pertence(char estadosFinais[],int numEstados, char transicoes[][3],
int numTransicoes, char simbolosFinais[], int numSimbolos){

   char cadeia[51];
   char estadoAtual;
   estadoAtual = '0';
   scanf("%s", cadeia);
   boolean entrou = false;

   for(int i = 0; i < strlen(cadeia); i++){
       for(int j = 0; j < numTransicoes; j++){
           if (cadeia[i] == '-' && estadoFinal(estadoAtual, numEstados,
estadosFinais)) return true;
           if(!taNoDicionario(cadeia[i], simbolosFinais, numSimbolos))
return false;
           if(cadeia[i] == transicoes[j][1] && estadoAtual ==
transicoes[j][0]){
               estadoAtual = transicoes[j][2];
               entrou = true;
               break;
           }
           else entrou = false;
       }
       if(!entrou) return false;
       if (i == strlen(cadeia)-1 && !estadoFinal(estadoAtual, numEstados,
```

```
estadosFinais)) return false;
   }
   return true;
```

A função *pertence* é a principal função do programa, responsável por verificar se uma cadeia de caracteres é, ou não, aceita pelo autômato finito inserido. Ela recebe por parâmetro o vetor de estados finais, a matriz com as transições, o vetor que contém o dicionário, e seus respectivos tamanhos.

Inicialmente, definimos o estado inicial como '0', considerando que é um autômato determinístico, e contém apenas um estado inicial, sendo este q_0. Então, a cadeia é lida, e a variável 'entrou' é definida, inicialmente, como *false*, visto que a condição que ela representa ainda não foi satisfeita.

O trabalho consiste, principalmente, em dois laços, o primeiro para percorrer a cadeia lida, para analisar o seus elementos, e o segundo para percorrer as transições.

A primeira etapa de verificação é de se a cadeia é o elemento lambda, e a posição atual é uma das posições finais. Em caso positivo, se retorna True. Caso a condição não seja satisfeita, é verificado se o símbolo atual está no dicionário e, caso não esteja, é retornado *false*.

Então, após essas verificações, vemos se existe uma transição que, saindo do estado atual, possui o caractere atual da cadeia. Em caso positivo, o estado atual é atualizado para o próximo estado da transição, a variável *entrou* é atualizada, sinalizando que o elemento foi encontrado, e interrompemos o laço de procura de transições, já que ela já foi encontrada. Caso contrário, fazemos 'encontrou' receber falso.

Ao fim da verificação de todas as transições, caso a variável *entrou* seja falsa, então não existe uma transição compatível, retornando *false*. E, caso esteja no último elemento da cadeia, e o estado atual não seja o final, então também é retornado *false*.

Caso, ao fim de todos os laços, a função ainda não tenha retornado *false*, então a cadeia é lida pelo autômato, retornando *true*.

Função *main()*

```
int main(){
   int estados, numSimbolosTerminais, numEstadosFinal,
numeroTransicoes, numCadeias, a;
   char b;
```

```
    scanf("%d", &estados);
    scanf("%d", &numSimbolosTerminais);
    char simbolos[numSimbolosTerminais];

    for (int i = 0; i < numSimbolosTerminais; i++) {
        getchar();
        scanf("%c", &simbolos[i]);
    }

    scanf("%d", &a);
    scanf("%d", &numEstadosFinal);
    char estadosFinais[numEstadosFinal];
    for (int i = 0; i < numEstadosFinal; i++) {
        getchar();
        scanf("%c", &estadosFinais[i]);
    }
    scanf("%d", &numeroTransicoes);

    char transicoes[numeroTransicoes][3];
    for (int i = 0; i < numeroTransicoes; i++) {
        getchar();
        scanf("%c",&transicoes[i][0]);
        getchar();
        scanf("%c",&transicoes[i][1]);
        getchar();
        scanf("%c",&transicoes[i][2]);
    }
    getchar();

    scanf("%d", &numCadeias);

    while(numCadeias--){
        boolean verifica;
        verifica = pertence(estadosFinais, numEstadosFinal, transicoes,
numeroTransicoes, simbolos, numSimbolosTerminais);
        if(verifica)
            printf("aceita\n");
        else
            printf("rejeita\n");
    }

}
```

Na função *main*, os dados de entrada são lidos, e, com um laço para chamar a quantidade de vezes necessário, chamamos a função *pertence*, passando os parâmetros, e, caso a função retorna *true* é impresso “aceita”, caso contrário é impresso “rejeita”.

CONCLUSÃO

Neste relatório foi possível descrever todas as funções implementadas, aplicando os conceitos aprendidos em aula e exercitando os conhecimentos adquiridos anteriormente, o que ocasionou uma melhora nas práticas de programação e de entendimento de códigos.

Em relação a estruturação do código, é possível que implementações mais otimizadas sejam produzidas. Porém, se tratando das restrições dos problemas aceitos pelo programa, a solução implementada é suficiente para suprir as necessidades. Soluções em outras linguagens como Python ocupariam uma quantidade relativamente menor de linhas, facilitando o entendimento do código, mas, por ser uma linguagem interpretada, o consumo de tempo seria similar.

Se tratando de tempo, cada caso de teste foi executado, em média, em 0.0018s, resultando em uma execução final de 0.072s, no teste de submissão no Coffee, o que é um tempo razoável, dado a complexidade dos problemas a serem solucionados.