UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CHRISTIAN JONAS OLIVEIRA

SIMULADOR UNIVERSAL DE AUTÔMATOS FINITOS DETERMINÍSTICOS

SÃO MATEUS - ES
2021

INTRODUÇÃO

O termo autômato tem como propósito de descrever um mecanismo que modela, opera ou imita movimentos humanos. No passado era usado como ferramenta científica, brinquedo e usado também com princípios religiosos. Os autômatos exploram modelos teóricos de computação, que explicam e que estão diretamente ligados aos fenômenos computacionais. O presente trabalho veio com o intuito de implementar um simulador de autômatos finitos afim de melhor compreender e entender LF's.

IMPLEMENTAÇÃO

→ Estrutura utilizada

Para a implementação deste simulador foi necessário o uso de uma estrutura que armazena as transições entre os estados. Para isso foi utilizado dois inteiros (*int*), que indicam estado atual e o estado ao qual este está ligado, respectivamente, além de um caractere (*char*) para indicar a transição entre um estado e outro.

```
typedef struct Transicao{
  int i, f;
  char elem;
}Transition;
```

→ Funções

A funcao *"verificaPertinencia"* é responsável por varrer todo o alfabeto afim de verificar se um elemento específico pertence a ele, retornando 1 caso pertença ou 0 caso não pertença.

```
int verificaPertinencia(char e, char s[]){
  int j=0;
  for(int i=0; i<strlen(s); i++) if(e==s[i]) j=1;
  return j;
}
```

A função *"verificaFinal"* é responsável por percorrer o vetor Aceitação, com o intuido de verificar se dado estado (ou "posição") é final.

```
int verificaFinal(int pos, int Aceitacao[], int qntFinais){
  int aux = 0;
  for(int i=0; i<qntFinais; i++)
    if(Aceitacao[i]==pos) aux = 1;
  return aux;
}
```

A funcao *"verificaCaminho"* é responsável por percorrer o vetor de transições afim de verificar se determinado elemento pode ser obtido através de alguma transição a partir do estado atual, retornando o estado seguinte, resultado da operação, ou -1 caso não encontre.

```
int verificaCaminho(char e, Transition **t, int tam, int pos){
  int k = -1;
  for(int c = 0; c<tam; c++){
    if(t[c]->i==pos && t[c]->elem==e){
        k = t[c]->f;
        break;
    }
  }
  return k;
}
```

A função *"verificaCadeia"* é responsável por unir todas as funções anteriores, realizando todas as verificações pertinentes.

```
int verificaCadeia(char *cadeia, Transition *t[], int Aceitacao[],
char s[],                    int qntFinais, int qTransicoes){
  int p = 1;
  int aux = 0;

  // verifica se a cadeia passada eh uma cadeia vazia
  if(verificaFinal(0, Aceitacao, qntFinais) && cadeia[0]=='-') return
1;

  // verifica os caracteres da cadeia estao no alfabeto
  for(int i = 0; i<strlen(cadeia); i++){
    if (verificaPertinencia(cadeia[i], s)==0) return 0;
  }

  // verifica se a cadeia eh valida
  for(int i=0; i<strlen(cadeia); i++){
    aux = verificaCaminho(cadeia[i], t, qTransicoes, aux);
    if(aux==-1){
      p = 0;
      break;
    }
  }
```

```
  // verifica se a cadeia passada termina em um estado final
  if(p==1)
    p = verificaFinal(aux, Aceitacao, qntFinais);
  return p;
}
```

DISCUSSÕES

Apesar da solução implementada em Python ser mais simples em termos de ferramentas disponibilizadas pela linguagem e em termos de memória, a solução em C se mostra mais rápida tendo um custo maior de memória, o que levou a adoção dessa linguagem para o desenvolvimento do presente trabalho. Pode-se dizer que os resultados foram bem satisfatórios considerando que o problema se dava em pequena escala (máximo de 50 transições).