Universidade Federal de São João Del Rei (CTAN)
Curso Ciência da Computação

Trabalho Prático

# Simulador de Autômatos de Pilha

Professor : Vinicius H. S. Durelli
Aluno: Luiz Gustavo Colzani Monti Sousa
Tulio Ribeiro Torres
Yan de Lucena Rayvel Pedroso

**Sumário**

**Introdução**

Para podermos falar do tema do trabalho, precisamos definir o que é um Autômato e o que ele representa.

Autômatos são definidos para estudar máquinas úteis sobre o formalismo matemático. Então, a definição de um autômato é aberta a variações de acordo com a “máquina do mundo real”, que nós queremos modelar usando o autômato, que serão o conjunto de estados e suas transições para a representação da nossa modelagem.

Este trabalho tem como objetivo a implementação de um simulador de Autômatos de Pilha (que é capaz de representar linguagens livre de contexto) a partir de uma tabela de transição. Assim mostrando sua implementação, funcionamento, testes e resultados.

**Implementação**

O simulador se consiste em 4 arquivos, tp.c , tp.h , main.c e Tabela.txt, onde tp.c é constituído pelas funções de todo o sistema, o tp.h será o arquivo de cabeçalho, contendo os nomes de todas as funções juntamente com seus parâmetros e as implementos dos TADs( Tipos Abstratos de Dados), main.c é a função principal que irá interagir com o usuário e fazer a chamada de todas as funções, e a Tabela.txt é um arquivo de texto que irá servir de parâmetro, onde o usuário irá preencher a tabela de transição do autômato e o programa irá montar o mesmo a partir dela.

- **tp.h**

_TAD

```
typedef struct Struct1 Estados;

struct Struct1{
    char transicao;
    int tipo;
    char estadoc;
    int estadoi;
    char empilha;
    char desempillha;

    struct Struct1* ProximoEstado
};
```

```
typedef struct Struct2 Pilha;

struct Struct2{
    char elemento;
    int contador;

    struct Struct2* Prox;
};
```

*struct Struct1 : tipo abstrato de dado para a representação de estados de um autômato.

*char transição : variável que irá guardar o autômato ir para o atual estado.

*int tipo : fazer a verificação se o estado atual é um estado inicial, final ou normal, sendo o estado inicial referenciado como o inteiro ‘1’, final como ‘2’ e normal como ‘0’.

*char estadoc : guarda o símbolo caractere de representação do estado do autômato.

*int estadoi: guarda o número de representação que se vem junto ao símbolo do estado do autômato.

*int empilha:

*int desempilha:

*struct Struct1* ProximoEstados : ponteiro para utilização de listas para guardar todas as transições e seus respectivos estados de chegadas que um estado de saída tem.

*struct Struct2: tipo abstrado de dado para a implementação de uma pilha.

*char elemento: guarda o elemento à ser empilhado.

*int contador: guarda a quantidade e elemento que tem na minha pilha.

*struct Struct2* Prox: ponteiro para o próximo elemento da minha pilha.

_Variáveis Globais

```c
#define MAX 500

char Palavra[MAX];
int TotalEstados;
int NSimbolos;
int PalavraValida;
int AF;
Estados *Automato;
Pilha *pilha;
```

*#define MAX 500 : atribuição do valor 500 para a variável MAX para utilização de limite superior para a quantidade de caracteres nas palavras de teste do usuário.

*char Palavra[MAX] : vetor de caracteres para salvar a palavra de teste do usuário.

*int TotalEstados : variável que guarda a quantidade total de estados do autômato.

*int NSimbolos : variável que guarda a quantidade de símbolos do alfabeto permitido pelo autômato.

*int PalavraValida : variável que valida se a palavra utilizada pelo usuário é reconhecida pelo autômato.

*int AF : variável que verifica se é um AFD ou um AFN.

*Estados *Automato : variável do TAD Estados que representa minha lista de estados do Automato.

*Pilha *pilha : variável do TAD Pilha que representa minha pilha.

- **tp.c**

_NEstados

```c
void NEstados(){
    FILE *arquivo;
    arquivo = fopen("Tabela.txt","r");

    char texto[MAX];
    int i=-1;
    while(!feof(arquivo)){
        fscanf(arquivo,"%[^\n] %s",texto,texto);
        i++;
    }
    fclose(arquivo);
    TotalEstados = i;
}
```

Função que calcula a quantidade de estado que se tem o autômato, percorrendo linha por linha por arquivo Tabela.txt. Porém o contador começa do -1 ao invés do 0 para não contar a primeira linha de transições.

A função “%[^\n] %s” vai salvar no vetor de caracteres tudo que tem na linha até o ‘\n’, podendo alterar o ‘\n’ para qualquer outro caractere.

_NSimb

```
void NSimb(){
    FILE *arquivo;
    arquivo = fopen("Tabela.txt","r");

    char texto[MAX];
    int i=0, cont=0;
    fscanf(arquivo,"%[^\n]",texto);
    while(1){
        if(texto[i] == '\0') break;
        if(texto[i] != ' ') cont++;
        i++;
    }

    fclose(arquivo);
    NSimbolos = cont;
}
```

Função que calcula a quantidade de símbolos do alfabeto, percorrendo a primeira linha e verificando os caracteres que são diferentes de ‘espaço’ e ‘\0’ que é o elemento que identifica o final da linha no vetor de caracteres.

_MontaSimbolos

```
void MontaSimbolos(char Alfabeto[]){
    FILE *arquivo;
    arquivo = fopen("Tabela.txt","r");

    char texto[MAX];
    int i=0,cont=0;
    fscanf(arquivo,"%[^\n]",texto);
    while(1){
        if(texto[i] == '\0') break;
        if(texto[i] != ' '){
            Alfabeto[cont] = texto[i];
            cont++;
        }
        i++;
    }

    fclose(arquivo);
}
```

Função que lê a primeira da tabela e atribui a um vetor de caracteres cada símbolo do alfabeto aceito pelo autômato, diferenciando os espaços de cada símbolo na linha do arquivo.

_AdicionaTransicao

```
void AdicionaTransicao(Estados *e, char tr, char ec, int ei, char desemp, char empi){

    Estados *Aux;
    Estados *NovoEstado;

    Aux = e;

    while(Aux->ProximoEstado != NULL){
        Aux = Aux->ProximoEstado;
    }

    NovoEstado = (Estados*)malloc(sizeof(Estados));
    NovoEstado->transicao = tr;
    NovoEstado->estadoc = ec;
    NovoEstado->estadoi = ei;
    NovoEstado->desempillha = desemp;
    NovoEstado->empilha = empi;
    NovoEstado->ProximoEstado = NULL;

    Aux->ProximoEstado = NovoEstado;
}
```

Função de alocação e carregamento de transições para as listas encadeadas de cada estado do autômato, percorrendo a lista e adicionando sua transição e o estado em que a mesma o leva. A função recebe a lista, o símbolo da transição, o caractere do estado que irá o autômato após a transição, o valor do estado, o caractere à ser desempilhado e o caractere à ser empilhado, assim respectivamente. Então ela copia a lista para um ponteiro auxiliar para poder percorrer a lista e fazer o carregamento ao final da lista com a alocação da nova transição.

_ConverteNumero

```
int ConverteNumero(char t[]){

    char texto[MAX];
    int i = 2;
    while(t[i] != ','){
        texto[i-2] = t[i];
        i++;
    }
    int valor = atoi(texto);
    return valor;
}
```

Função que converte o número do estado de caractere para inteiro, para melhor manipulação dos estados e leitura da TAD, percorrendo o vetor de caracteres da transição copiando o vetor do estado após o caractere que o representa até achar a virgula, que separa o estado do caractere a ser empilhado que está contido no vetor para um vetor auxiliar, utilizando a função atoi para a conversão e retornando o resultado da mesma.

_VerificaAFNAFD

```
int VerificaAFNAFD(char texto[]){
    int i=0, verifica=0;
    while(texto[i] != '\0'){
        if(texto[i] == '/'){
            verifica = 1;
            break;
        }
        i++;
    }
    return verifica;
}
```

Função que faz a verificação se o autômato é determinístico ou não determinístico de acordo com a tabela de transição, verificando se há mais um estado para o mesmo símbolo, sendo separados por ‘/’ ao percorrer o vetor de caracteres de transição.

_VetorAFN

```
void VetorAFN(char texto1[], char texto2[]){

    int i=0, aux=0, i2 =0;
    while(texto1[i] != '\0'){
        if(aux > 0){
            texto2[i2] = texto1[i];
            texto1[i] = ' ';
            i2++;
        }
        if(texto1[i] == '/'){
            aux++;
            texto1[i] = ' ';
        }
        i++;
    }
    texto2[i2] = texto1[i];
}
```

Função que separa estado por estado de um AFN de uma mesma transição, para poder fazer o carregamento do mesmo para a lista encadeada do estado de transição. A função irá percorrer o vetor de caractere até encontrar ‘/’ , após o encontro, ela copia o restando do vetor para um segundo vetor auxiliar.

_VerificaPalavra

```
int VerificaPalavra(Estados *Automato,char estadoc, int estado,int estadoanterior, int contador){
    int cont1 = 0;
    Estados *P;
    Estados *verif;

    if(PalavraValida == 0)
    printf("->%c%d\n",estadoc,estado);

    if(contador == 0 && Palavra[contador] == 'E' || contador != 0 && Palavra[contador] == '\0'){
        verif = Automato[estado].ProximoEstado;
        while(verif != NULL){
            if(verif->transicao == '?')
                break;
            verif = verif->ProximoEstado;
        }

        if(verif != NULL){
            if(verif->transicao == '?'){
                if(pilha->contador == 0){
                    if(Palavra[contador] == 'E' || Palavra[contador] == '\0'){
                        printf("->%c%d\n",verif->estadoc,verif->estadoi);
                        PalavraValida = 1;
                    }
                    else
                        PalavraValida = 0;
                    return;
                }
            }
        }
    }
    else{
        P = Automato[estado].ProximoEstado;

        while(P!=NULL && Palavra[contador]!='\0'){
            if(P->transicao == Palavra[contador]){
                if(PalavraValida == 0){
                    if(P->empilha != 'E'){
                        printf("Empilha: %c\n",P->empilha);
                        Empilhar(P->empilha);
                    }
                    if(P->desempillha != 'E'){
                        printf("Desempilha: %c\n",P->desempillha);
                        if(Desempilhar(P->desempillha) == 0)
                        printf("\nErro ao Desempilhar, palavra nao valida\n");
                    }
                }
                VerificaPalavra(Automato,P->estadoc,P->estadoi,estado,contador+1);
                if(P->desempillha != 'E'){
                    printf("Retorna estado\nEmpilha novamente: %c\n",P->desempillha);
                    Empilhar(P->desempillha);
                }
                if(P->empilha != 'E'){
                    printf("Retorna estado\nDesempilha caminho invalido: %c\n",P->empilha);
                    if(Desempilhar(P->empilha) == 0)
                    printf("\nErro ao Desempilhar, palavra nao valida\n");
                }
            }
            P = P->ProximoEstado;
        }
    }

    if(PalavraValida == 1){
        return 1;
    }

    return 0;
}
```

Função recursiva para a validação da palavra testada, onde a mesma irá percorrer a lista encadeada do estado atual até achar a transição referente ao caractere atual da palavra no vetor, após achar ele irá empilhar ou desempilhar um elemento da pilha dependendo da sua composição, se o seu empilhar for diferente de ‘E’ ele empilha seu elemento, caso o seu desempilhar for diferente de ‘E’ ele desempilha seu elemento, e depois continua a chamar a função de novo passando a lista encadeada do estado referente a transição e olhar o próximo caractere do vetor que guarda a palavra testada até chegar o ultimo caractere e verificar se a pilha está vaiza e o estado aonde a última transição se resultou o leva para um estado final ou não para a sua validação. O uso de uma função recursiva serve para o caso do autômato ser não determinístico, assim olhando as ramificações de cada estado de um mesmo símbolo de transição.

_CopiaVetor

```
void CopiaVetor(char Texto1[], char Texto2[]){
    int i=0;
    while(Texto2[i] != '\0'){
        Texto1[i] = Texto2[i];
        Texto2[i] = ' ';
        i++;
    }
}
```

Função que copia o vetor auxiliar usado na separação de estados da função AFN percorrendo os vetores e “zerando” o vetor auxiliar, para que os estados restantes permaneçam no vetor principal.

_CaracterEmpilha

```
char CaracterEmpilha(char vetor[]){
    int i=0, cont=0;
    char e;
    while(vetor[i]!='\0'){
        if(cont == 2){
            e = vetor[i];
            break;
        }
        if(vetor[i] == ',')cont++;
        i++;
    }
    return e;
}
```

Função que irá identificar o caractere em que uma transição irá empilhar um elemento, para fazer a atribuição do elemento ao adicionar a transição.

_CaracterDesempilha

```
char CaracterDesempilha(char vetor[]){
    int i=0, cont=0;
    char e;
    while(vetor[i]!='\0'){
        if(cont == 1){
            e = vetor[i];
            break;
        }
        if(vetor[i] == ',')cont++;
        i++;
    }
    return e;
}
```

Função que irá identificar o caractere em que uma transição irá desempilhar um elemento, para fazer a atribuição do elemento ao adicionar a transição.

_Empilhar

```
void Empilhar(char e){
    if(pilha->contador == 0){
        pilha->elemento = e;
        pilha->contador = 1;
    }
    else{
        Pilha *aux;
        aux = pilha;
        while(aux->Prox != NULL){
            aux = aux->Prox;
        }
        Pilha *Novo;
        Novo = (Pilha*)malloc(sizeof(Pilha));
        Novo->contador = aux->contador++;
        Novo->elemento = e;
        Novo->Prox = NULL;
        aux->Prox = Novo;
    }
}
```

Função que irá receber um elemento e empilhar ele na pilha, se minha pilha estiver vazia, com o elemento ‘E’ e o contador igual a 0, eu apenas altero as valores adicionando o elemento e acrescentando ao contador, caso minha pilha não esteja vazia, eu percorro ela até chegar no topo e faço a alocação do novo elemento e adiciono ele a pilha.

_Desempilhar

```
int Desempilhar(char e){
    int valida;
    Pilha *aux;
    Pilha *desemp;

    if(pilha->Prox == NULL){
        if(pilha->elemento == e){
            pilha->contador = 0;
            pilha->elemento = 'E';
            valida = 1;
        }
        else if(pilha->elemento == 'E'){
            pilha->contador = 0;
            pilha->elemento = 'E';
            valida = 1;
        }
    }
    else{
        aux = pilha;
        desemp = aux;
        while(aux->Prox != NULL){
            desemp = aux;
            aux = aux->Prox;
        }
        if(aux->elemento == e){
            desemp->Prox = NULL;
            free(aux);
            valida = 1;
        }
        else
            valida = 0;
    }
    return valida;
}
```

Função que irá receber um elemento e retirar ele do topo da pilha, se o começo a base da minha pilha o próximo dela apontar para nulo, a minha pilha só tem um elemento, caso o elemento seja igual ao que eu quero remover, ai eu apenas altero os valores da pilha para que o elemento seja ‘E’ e o contador igual a 0 para que demonstre que minha pilha está vazia, caso eu não tenha só a base, eu percorro a pilha até o topo, e verifico se o elemento que está no topo é o elemento que eu quero remover, se for, eu faço o elemento anterior a ele apontar para nulo com uma variável auxiliar, e apago o elemento do meu topo. Caso o elemento que eu for remover exista na pilha e é o topo dela, eu retorno verdadeiro (1), caso não seja, retorno falso (0).

_LimpaPilha

```
void LimpaPilha(){
    Pilha *aux;
    Pilha *aux1;
    aux = pilha;
    aux1=aux;
    while(aux->Prox!=NULL){
        aux1 = aux;
        aux = aux->Prox;
        free(aux1);
    }
    aux->contador = 0;
    aux->elemento = 'E';
    aux->Prox = NULL;
}
```

Função que percorre a pilha e remove todos os elementos da mesma.

- **main.c**

  _Alocação do Autômato e da Pilha

```
Automato = (Estados*)malloc(TotalEstados*sizeof(Estados));
pilha = (Pilha*)malloc(sizeof(Pilha));
pilha->elemento = 'E';
pilha->contador = 0;
pilha->Prox = NULL;

int i=0,inicio;

while(i<TotalEstados){
    Automato[i].estadoc = 'q';
    Automato[i].estadoi = i;
    Automato[i].tipo = 0;
    Automato[i].ProximoEstado = NULL;
    Automato[i].empilha = 'E';
    Automato[i].desempillha = 'E';
    i++;
}
```

Vetor do tipo Estados (implementado por uma TAD), do tamanho do número total de estados, onde se faz um vetor de ponteiros para ser a cabeça de cada lista referente a cada estado para representar suas transições, alocando cada posição da lista.

Lista encadeada que irá representar uma pilha.

_Leitura da Tabela

```
fscanf(arquivo,"%[^\n]",texto1);
int contatrans=-1 ,contestado=0, numeroestado=0;

while(!feof(arquivo)){
    fscanf(arquivo,"%s",texto1);
    if(contatrans < 0){
        if(texto1[0] == '>'){
            Automato[contestado].tipo = 1;
            inicio = contestado;
        }
        if(texto1[0] == '*')
            Automato[contestado].tipo = 2;
        contatrans++;
    }
    else if(texto1[0] == '-'){
        contatrans++;
    }
    else{
        if(VerificaAFNAFD(texto1) == 1){
            AF = 1;
            while(VerificaAFNAFD(texto1) == 1){
                VetorAFN(texto1,texto2);
                numeroestado = ConverteNumero(texto1);
                aemp = CaracterEmpilha(texto1);
                adesemp = CaracterDesempilha(texto1);
                if(Alfabeto[contatrans] == '*')
                    AdicionaTransicao(&Automato[contestado],'?',texto1[1],numeroestado,adesemp,aemp);
                else
                    AdicionaTransicao(&Automato[contestado],Alfabeto[contatrans],texto1[1],numeroestado,adesemp,aemp);
                if(VerificaAFNAFD(texto2) == 1)
                    CopiaVetor(texto1,texto2);
            }
            numeroestado = ConverteNumero(texto2);
            aemp = CaracterEmpilha(texto2);
            adesemp = CaracterDesempilha(texto2);
            if(Alfabeto[contatrans] == '*')
                AdicionaTransicao(&Automato[contestado],'?',texto2[1],numeroestado,adesemp,aemp);
            else
                AdicionaTransicao(&Automato[contestado],Alfabeto[contatrans],texto2[1],numeroestado,adesemp,aemp);
        }
        else{
            numeroestado = ConverteNumero(texto1);
            aemp = CaracterEmpilha(texto1);
            adesemp = CaracterDesempilha(texto1);
            if(Alfabeto[contatrans] == '*')
                AdicionaTransicao(&Automato[contestado],'?',texto1[1],numeroestado,adesemp,aemp);
            else
                AdicionaTransicao(&Automato[contestado],Alfabeto[contatrans],texto1[1],numeroestado,adesemp,aemp);
        }
        contatrans++;
    }
    if(contatrans == NSimbolos-1){
        contatrans = -1;
        contestado++;
    }
}

fclose(arquivo);
```

Primeiro se lê a primeira linha do arquivo txt da Tabela, para poder retirar a linha de símbolos do arquivo, após se faz um laço de repetição até que o arquivo acabe. A leitura irá pegar cada conjunto de caracteres separados por ‘espaço’ e por quebra de linha, verificando se a coluna é referente às transições ou aos estados. Caso a coluna seja dos estados, eu verifico se é um estado normal, inicial ou final, diferenciado na tabela com os símbolos ‘>’, ‘ * ‘ , onde ‘>’ representa o estado inicial, ‘ * ‘ representa estado(s) final(ais), e os estados comuns não são acompanhados de símbolos. Caso seja colunas de transição, o programa verifica se contém mais de um estado a transição ou não, caso tenha, chamamos a funções de divisão dos estados e adicionamos os estados até que o último estado seja adicionado, caso não tenha, apenas adicionamos normalmente o estado, onde a adição dos estados, separados o símbolo do estado do seu numero com a função de ‘ConverteNumero’ para depois podermos adicionar o mesmo, depois chamamos as funções ‘CaracterEmpilha’ e ‘CaracterDesempilha’, para ver os elementos à serem empilhados e desempilhados.

_Menu

```
int menu = 0;
int verifica=0;
while(1){
    if(AF == 1) printf("AFN\n");
    else printf("AFD\n")
    printf("1.Testar Palavra\n2.Sair\n");
    scanf("%d",&menu);
    if(menu == 2) break;
    printf("Digite uma palavra: ");
    scanf("%s",&Palavra);
    PalavraValida = 0;
    VerificaPalavra(Automato,Automato[0]->estadoc,Automato[0]->estadoi,-1,0);
    if(PalavraValida == 0)
    printf("\nA palavra %s nao eh valida.\n",Palavra);
    if(PalavraValida == 1)
    printf("\nA palavra %s eh valida.\n",Palavra);
}
```

Menu de interação com o usuário onde ele irá mostrar se o autômato é AFD ou AFN dependendo do autômato carregado pelo arquivo Tabela.txt construído pelo usuário, e as opções de testar palavras ou sair do programa.

- **Tabela.txt**

Tabela de texto construída pelo usuário onde a primeira contém os símbolos do alfabeto reconhecido pelo autômato para a representação dos símbolos de cada transição numa matriz, sendo cada símbolo separado por ‘espaço’, podendo dar mais de uma vez ‘espaço’ para que se possa ter uma melhor visualização, sendo a última coluna a transição para o estado final, sedo representada por um *. Após a primeira linha, na primeira coluna da tabela estará referindo o estado atual do autômato e as seguintes colunas os estados em que cada transição se resulta de acordo com cada transição da primeira linha,sendo subdivido pelos paraenses do seguinte modo (estado,desempilha,empilha), e a última coluna (estado finla,?,E), sendo sempre separado por vírgula. Caso um símbolo leva para mais de um estado, denotar-se o conjunto de estados com ‘/’ separando eles. Na coluna dos estados, para se diferenciar os estados, utilizamos dois símbolos, para o estado inicial, colocamos o símbolo ‘>’ antes do estado, e para o(s) estado(s) final(ais), colocamos o símbolo ‘ * ‘, caso seja um estado comum, não atribuímos nenhum símbolo,onde se deve entrar com os estados em ordem. Para a utilização da palavra vazia, atribuímos o símbolo ‘E’.

Caso o estado não tenha uma transição com algum símbolo, colocar ‘ – ‘ na coluna do símbolo para representar que não existe a transação.

- **Makefile**

Arquivo de compilação para o usuário, atribuindo todos os arquivos para facilitar a utilização do programa para o usuário com o comando make pelo terminal. Ao dar o comando make no terminal, o programa será compilado e irá ser criado um executável com o nome main.exe, para a execução do programa apenas digitar ./main que o programa começará a rodar. Após o termino do programa, digitar o comando make mrproper para a exclusão do arquivo executável.