Trabalho Prático: Autômatos Finitos

Aluno em Graduação da Universidade Federal de Ouro Preto do curso Ciência da Computação:

Halliday Gauss Costa dos Santos.

Matrícula: 18.1.4093.

Área: Teoria da Computação.

Introdução:

O trabalho prático consistiu na implementação de um programa em Python que simula um autômato finito. Através de um arquivo texto é passado para o programa os estados iniciais e finais do autômato, as transições que o compõe e as palavras que devem ser verificadas se pertencem ou não a Linguagem representada. O programa deve retornar, para cada palavra a ser testada, o processamento de cada símbolo, dever retornar também se a palavra em questão pertence ou não a linguagem representada pelo autômato, caso pertença, deve ser impresso os estados finais ativos após o processamento de todos os símbolos da palavra.

Desenvolvimento:

Existem várias funções que compõe o código e será mostrado o funcionamento de cada uma delas logo abaixo:

- **Função lerArquivo:**

```
def lerArquivo(nomeDoArquivo):
    """ funcao que le o arquivo e retorna uma lista onde cada posicao
        da lista possui uma linha do arquivo"""
    conteudoDoArquivo = ""
    with open(nomeDoArquivo) as arquivo:
        conteudoDoArquivo = arquivo.readlines()

    return conteudoDoArquivo
```

Essa função recebe o nome do arquivo texto a ser aberto para leitura e retorna uma lista, e cada posição dessa lista é uma linha do arquivo texto**.**

- **Função definirEstadosIniciaisEFinais:**

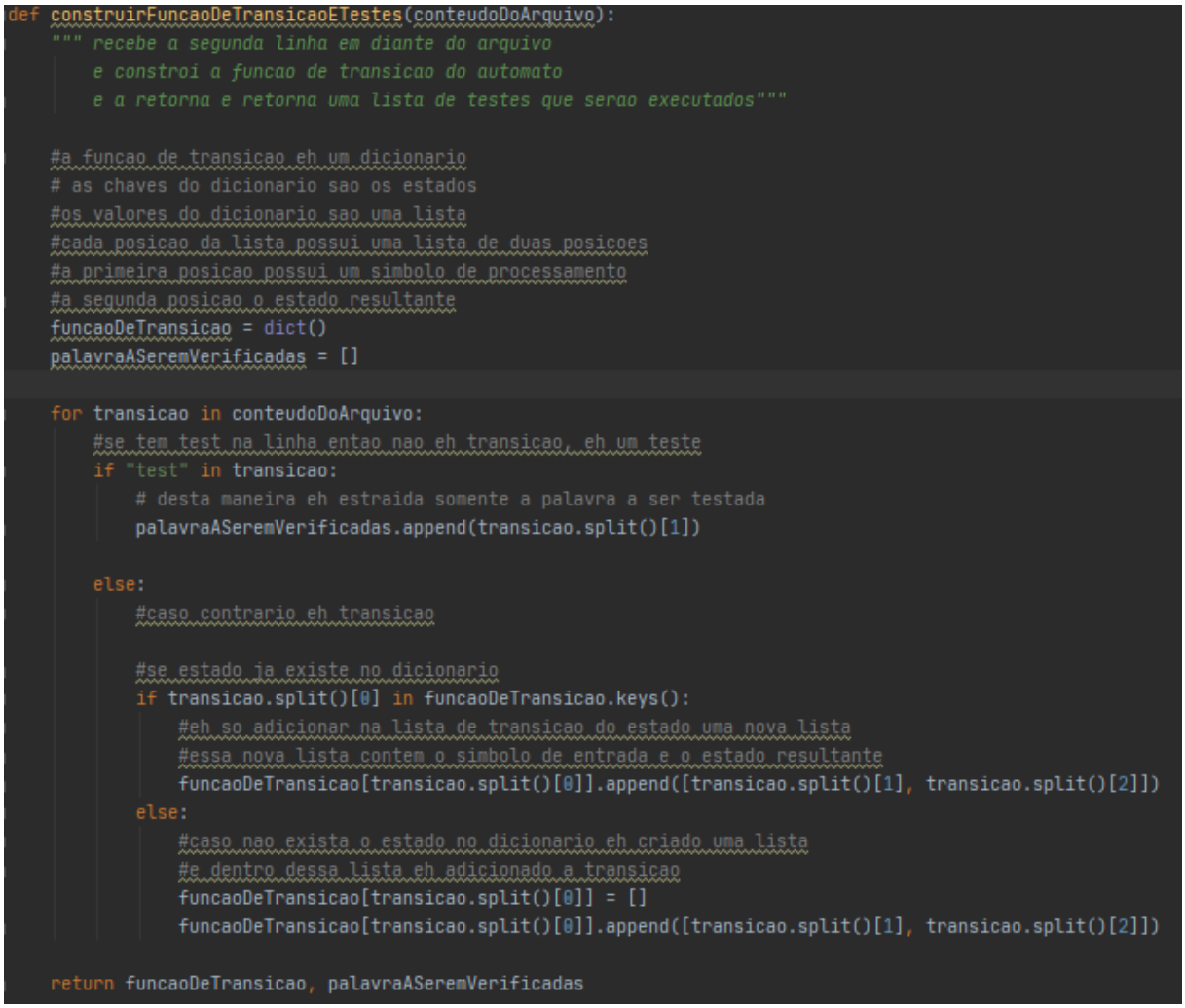
```
def definirEstadosIniciaisEFinais(linha):
    "dada a primeira linha do arquvio separa os estados iniciais e finais"
    estadosIniciais = []
    estadosFinais = []

    #separa os estados por ponto e virgula
    #a posicao 0 terah uma string contendo os estados iniciais separados por espaco
    #a posicao 1 terah uma string contendo os estados finais separados por espaco
    estados = linha.split(';')

    for estadoIncial in estados[0].split():
        estadosIniciais.append(estadoIncial)

    for estadoFinal in estados[1].split():
        estadosFinais.append(estadoFinal)


    return estadosIniciais, estadosFinais
```

Essa função recebe a primeira linha lida do arquivo e retorna duas listas, uma contendo os estados iniciais do autômato e outra contendo os estados finais.

É utilizada a função split na variável ‘linha’ de maneira que gere uma lista de duas posições que é salva na variável estados, ou seja, separa os estados por ponto e vírgula. Portanto, a posição 0 da lista chamada ‘estados’ contém os estados iniciais separados por espaço e a posição 1 da lista contém os estados finais também separados por espaço.

Em seguida, os estados iniciais são adicionados na lista vazia chamada de ‘estadosIniciais’, e os estados finais são adicionados na lista vazia chamada de ‘estadosFinais’. E por fim, essas listas são retornadas pela função.

- **Função construirFuncaoDeTransicaoETestes:**



```
def construirFuncaoDeTransicaoETestes(conteudoDoArquivo):
    """ recebe a segunda linha em diante do arquivo
        e constroi a funcao de transicao do automato
        e a retorna e retorna uma lista de testes que serao executados"""

    #a funcao de transicao eh um dicionario
    # as chaves do dicionario sao os estados
    #os valores do dicionario sao uma lista
    #cada posicao da lista possui uma lista de duas posicoes
    #a primeira posicao possui um simbolo de processamento
    #a segunda posicao o estado resultante
    funcaoDeTransicao = dict()
    palavraASeremVerificadas = []

    for transicao in conteudoDoArquivo:
        #se tem test na linha entao nao eh transicao, eh um teste
        if "test" in transicao:
            # desta maneira eh estraida somente a palavra a ser testada
            palavraASeremVerificadas.append(transicao.split()[1])

        else:
            #caso contrario eh transicao

            #se estado ja existe no dicionario
            if transicao.split()[0] in funcaoDeTransicao.keys():
                #eh so adicionar na lista de transicao do estado uma nova lista
                #essa nova lista contem o simbolo de entrada e o estado resultante
                funcaoDeTransicao[transicao.split()[0]].append([transicao.split()[1], transicao.split()[2]])
            else:
                #caso nao exista o estado no dicionario eh criado uma lista
                #e dentro dessa lista eh adicionado a transicao
                funcaoDeTransicao[transicao.split()[0]] = []
                funcaoDeTransicao[transicao.split()[0]].append([transicao.split()[1], transicao.split()[2]])

    return funcaoDeTransicao, palavraASeremVerificadas
```

Essa função recebe uma lista como parâmetro a qual contém a segunda linha lida do arquivo até a última.

Dentro desse escopo, a função de transição é definida como um dicionário, e a ideia é que cada estado que possua transição seja uma chave do dicionário e cada valor correspondente a sua chave é uma lista, e essa lista pode possuir várias outras listas, e essas listas mais internas possuem duas posições, a posição 0 é responsável por armazenar o símbolo que representa a transição, e a posição 1 é responsável por armazenar o estado resultante da transição.

É declarada também uma lista chamada de ‘palavrasASeremVerificadas’, onde cada posição da lista deve armazenar as palavras que serão verificadas se pertencem ou não a linguagem representada pelo autômato.

Na sequência, é percorrido as linhas do arquivo, a partir da segunda linha, e caso uma linha contenha a palavra ‘test’, quer dizer então que não é uma transição, mas sim uma palavra a ser verificada, então essa palavra é adicionada na lista ‘palavrasASeremVerificadas’. Caso contrário, é uma transição, então o programa verifica se o estado já existe no dicionário, se existe, então sua transição é adicionada, se não existe, é criado a nova chave com um valor que é uma lista vazia, e dentro dessa lista é adicionado uma lista de duas posições que contém o símbolo de uma aresta e o estado resultante da transição. Após esse procedimento a função de transição e as palavras a serem verificadas são retornadas.

- **Função transitar:**

```
def transitar(estadosAtivos, estadoATransitar, funcaoDeTransicao, simbolo, estadosResultantes):
    """ essa funcao recebe um estado que deve processar um simbolo 'estadoATransitar'
        recebe tambem os estados ativos, a funcao de transicao, o simbolo a ser processado,
        e os estados que serao resultantes da transicao, que eh parecido com os estados ativos
        , essa funcao realiza a transicao sobre um simbolo a partir de um estado
        e adiciona os estados resultante como proximos estados a serem transitados"""

    #recebe as transicoes do estado a ser transitado, se ele nao tiver transicoes retorna uma lista vazia
    transicoesDoEstado = funcaoDeTransicao.get(estadoATransitar, [])

    for transicao in transicoesDoEstado:
        #se o simbolo a ser processado for igual ao simbolo do estado e o estado resultante ja nao estiver ativo
        #entao eh adicionado como estado ativo na proxima iteracao
        if transicao[0] == simbolo:
            estadosResultantes.append(transicao[1])
            if transicao[1] not in estadosAtivos:
                estadosAtivos.append(transicao[1])
```

A função ‘transitar’ recebe como argumentos os estados ativos, o estado que realizará a transição chamado de ‘estadoATransitar’, a função de transição, o símbolo a ser processado, e os estados resultantes. Essa função é responsável por realizar as transições no autômato.

Inicialmente a variável ‘transicoesDoEstado’, recebe a lista de transições do estado que realizará a transição. Para cada transição dessa lista é conferido se existe o símbolo a ser processado, se não existir não faz nada, mas se existir, o estado resultante dessa transição é adicionado na lista estadosResultantes, o estado resultante será adicionado na lista de estados ativos para a próxima execução de símbolos, somente se ele já não foi adicionado.

- **Função palavrasAlcancaveiComLambda:**

```python
def palavrasAlcancaveiComLambda(estadosAtivos, funcaoDeTransicao):
    """a partir de um conjunto de estados ativos, essa funcao realiza as transicoes lambdas a partir deles"""
    mudou = True
    while mudou:
        mudou = False
        for estadoAtivo in estadosAtivos:
            for transicao in funcaoDeTransicao.get(estadoAtivo, []):
                if (transicao[0] == "lambda") and (transicao[1] not in estadosAtivos):
                        #caso tenha encontrado um transicao lambda
                        # o estado resultante deve ser adicionado a lista de estados ativos
                        #e o loop deve repetir mais uma vez em busca de novas transicoes lambdas
                        estadosAtivos.append(transicao[1])
                        mudou = True
```

Essa função recebe como parâmetro os estados ativos no momento e a função de transição, e a partir do conjunto de estados ativos são adicionados como ativos os estados que são alcançáveis por transições lambdas, ou seja, os estados alcançáveis por transições lambdas a partir dos estados ativos também deverão ser ativos.

- **Função perteceALinguagem:**

```python
def perteceALinguagem(palavra, estadosAtivos, funcaoDeTransicao, estadosFinais):
    """funcao que recebe uma palavra, um conjunto de estados ativos, um funcao de transicao
        e um conjunto de estados finais e imprime na tela se a palavra pertence a Linguagem representada
        pelo automato ou nao, caso pertenca deve imprimir o conjunto de estados finais ativos"""

    print("Iniciando o processamento de", palavra)
    print("Configuracao inicial: [", end="")
    for estadoAtivo in estadosAtivos:
        print(estadoAtivo," ", sep="", end="")

    print(",", palavra, "]", sep="")

    simbolosProcessados = 0

    tamanhoDaPalavra = len(palavra)

    #se a palavra for lambda o tamanho dela eh 0
    if palavra == "lambda":
        tamanhoDaPalavra = 0

    # enquanto os simbolos Processados forem menor que a palavra
    # e ainda tiver estados ativos
    # o loop deve se repetir
    while simbolosProcessados < tamanhoDaPalavra and len(estadosAtivos) > 0:
```

Essa é a primeira parte da função ‘pertenceALinguagem’, ela recebe como parâmetro a palavra a ser processada pelo autômato, os estados ativos, a função de transição e os estados finais.

Inicialmente é impresso a configuração inicial do autômato, a palavra a ser processada e os estados ativos, neste caso, os estados ativos são os estados iniciais sem considerar transições lambdas, pois elas serão realizadas mais à frente. Também é calculado o tamanho da palavra a ser processada, se a palavra for ‘lambda’ é atribuído o valor 0 para o tamanho. Logo em seguida há um loop que realizará todos os processamentos da palavra pelo autômato enquanto a quantidade de símbolos processados não for igual ao tamanho da palavra e enquanto o conjunto de estados ativos não for vazio.

```
while simbolosProcessados < tamanhoDaPalavra and len(estadosAtivos) > 0:
    #ativa os estados que tem transicao lambda
    palavrasAlcancaveisComLambda(estadosAtivos, funcaoDeTransicao)

    #gera uma copia dos estados ativos e o conjunto dos estados ativos eh zerado
    #esse conjunto sera o resultado das transicoes da copia dos estados ativos
    estadosAtivosAux = estadosAtivos.copy()
    estadosAtivos.clear()

    #para cada estado ativo deve ser realizada a transicao
    for estadoAtivo in estadosAtivosAux:
        #inicialmente nao resultou em nenhum estado pois nao transitou
        estadosResultantes = []

        print("Executando a transicao ", estadoAtivo, "-", palavra[simbolosProcessados], "-> ", end="")

        #realiza a transicao
        transitar(estadosAtivos, estadoAtivo, funcaoDeTransicao, palavra[simbolosProcessados], estadosResultantes)

        #imprime o resultado da transicao
        for estadoResultante in estadosResultantes:
            print(estadoResultante, "", end="")

        print("resulta em [ ", end="")
        for estadoResultante in estadosResultantes:
            print(estadoResultante, "", end="")



        #se ja esta no fim da palavra deve imprimir lambda ao inves de vazio
        if simbolosProcessados == tamanhoDaPalavra -1:
            print(",", "lambda]")

        else:
            #caso contrario imprime os simbolos restante a serem processados
            print(",", palavra[simbolosProcessados + 1:], "]")

    #aumenta em um os simbolos processados
    simbolosProcessados +=1
```

A segunda parte da função consiste no loop supracitado. Inicialmente a função palavrasAlcancaveiComLambda é chamada para ativar os estados que podem ser alcançados pelos estados ativos atualmente. Em seguida, é gerado uma cópia desses estados ativos que é armazenada na variável ‘estadosAtivosAux’, e então a variável ‘estadosAtivos’ é zerada, ou seja, fica com uma lista vazia.

Na próxima linha há outro loop, que repete para cada estado em ‘estadosAtivosAux’. Para cada estado ativo em ‘estadosAtivosAux’ é feita a transição processando o símbolo adequado, e é impresso na tela as transições efetuadas por esse estado e os símbolos restantes da palavra que ainda não foram processados, se o símbolo atual for o último símbolo a ser processado, o símbolo restante da palavra é lambda. É importante ressaltar que dentro desse loop ‘for’ a função ‘transitar’ é chamada para realizar a transição e adicionar na lista ‘estadosAtivos’ os resultados da transição, ou seja, a lista ‘estadosAtivos’ estará recebendo os próximos estados ativos.

Após esse loop interno, um símbolo da palavra foi processado, portanto, o símbolo da palavra é incrementado em 1, ou seja, agora o loop irá executar novamente no intuito de processar o próximo símbolo da palavra.

```
#apos terminar a execucao das transicoes eh necessario realizar a transicao lambda dos estados resultantes
palavrasAlcancaveiComLambda(estadosAtivos, funcaoDeTransicao)

#verifica se pelo menos um dos estados ativos eh final
if len(set(estadosFinais).intersection(set(estadosAtivos))) > 0:
    #se for imprime o conjunto de estados finais ativos
    print("Conjunto de Estados finais ativos: ", end="")
    for estado in set(estadosFinais).intersection(set(estadosAtivos)):
        print(estado, "", end="")
    print("\nA palavra", palavra, "eh aceita pelo AF")

else:
    #caso nao tenha pelo menos um estado final ativo, a palavra nao eh aceita pela linguagem
    print("A palavra", palavra, "nao eh aceita pelo AF")

print()
```

A terceira parte da função vem logo após o término do loop. Agora a função ‘palavrasAlcancaveiComLambda’ é novamente executada no intuito de encontrar as transições lambdas a partir dos estados ativos após o processamento completo da

palavra, portanto, essa função irá ativar esses estados alcançáveis por transições lambdas.

Feito isso, é conferido se a interseção entre o conjunto dos estados ativos e conjunto dos estados finais resulta em pelo menos um estado. Se a interseção não é vazia então a palavra é aceita e é impresso na tela o conjunto interseção, caso contrário a palavra não foi aceita.

- **Main:**

```
if __name__ =="__main__":

    #leio o arquivo e armazeno na variavel conteudo
    # o nome do arquivo eh passado como parametro na execucao
    conteudoDoArquivo = lerArquivo(sys.argv[1])

    #passar a primeira linha do arquivo que irá definir os estados iniciais e finais
    estadosIniciais, estadosFinais = definirEstadosIniciaisEFinais(conteudoDoArquivo[0])

    #inicialmente os estados ativos sao os estados iniciais
    estadosAtivos = estadosIniciais.copy()

    #definir transicoes e valores a testar
    #passar como parametro so da segunda linha a diante do arquivo
    funcaoDeTransicao, palavraASeremVerificadas = construirFuncaoDeTransicaoETestes(conteudoDoArquivo[1:])

    #para cada palavra a ser verifica eh verificada se ela pertence a linguagem
    for palavra in palavraASeremVerificadas:
        perteceALinguagem(palavra, estadosAtivos.copy(), funcaoDeTransicao, estadosFinais)
```

O main do programa inicialmente chama a função ‘lerArquivo’, e armazena todo o conteúdo do arquivo na variável ‘conteudoDoArquivo’. Em seguida, a função ‘definirEstadosIniciaisEFinais’ é chamada e é passado como parâmetro somente a primeira linha lida do arquivo, o retorno dessa função permite obter os estados iniciais e finais. Na sequência, a variável estadosAtivos recebe uma cópia dos estados iniciais, pois no início da execução do autômato os estados ativos são os estados iniciais. Seguidamente, a função ‘contruirFuncaoDeTransicaoETestes’ é chamada e retorna à função de transição do autômato e os testes a serem

executados. Finalmente, a função ‘pertenceALinguagem’ é chamada para cada palavra a ser testada.

Vários testes foram executados encima de 7 autômatos, os arquivos de teste e o código do programa estão juntos em um mesmo arquivo .zip, e junto de cada arquivo de teste está a foto do autômato relacionado.

O programa se chama af.py e para o executar basta que o arquivo de testes esteja no mesmo diretório que o programa. Supondo que o arquivo de testes tenha o nome teste.txt, então para executar o programa encima desses testes deveria ser da seguinte forma:

python af.py teste.txt

Cabe ressaltar que todos os testes foram conferidos e geraram os resultados esperados.

É importante ressaltar também que o algoritmo implementado simula não apenas AFD’s’ mas também AFN’s’ e AFN’lambdas’

Conclusão:

Conclui-se que a implementação desse trabalho possibilitou o melhor entendimento da disciplina e resultou numa melhora na lógica de programação na linguagem escolhida, e partir da realização do trabalho haverá uma facilidade ao implementar futuramente, algoritmos que envolve autômatos finitos.