UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO TÉCNICO - TEORIA DA COMPUTAÇÃO

LEANDRO MARTINS TOSTA

CONVERSÃO DE AUTÔMATO FINITO NÃO-DETERMINÍSTICO EM DETERMINÍSTICO

APUCARANA, 2022

LEANDRO MARTINS TOSTA

CONVERSÃO DE AUTÔMATO FINITO NÃO-DETERMINÍSTICO EM DETERMINÍSTICO

Relatório Técnico do Trabalho Final (TF) apresentado como requisito parcial para obtenção de créditos na disciplina de Teoria da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Lucio Agostinho Rocha

APUCARANA, 2022

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um conversor de autômato não-determinístico (AFN) em um autômato determinístico (AFD), utilizando a linguagem de programação Python, o código foi desenvolvido de maneira simples, implementado na IDE PyCharm. A escolha da linguagem, foi devida a sua fácil manipulação com tabelas e pelo seu alto nível, sendo mais compreensiva aos novos programadores.

Palavras-chave: AFND. AFD. python. Conversor.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 DESCRIÇÃO DOS ARQUIVOS

O projeto do software foi desenvolvido completamente na linguagem de programação python, na versão 3.19. Possui 4 arquivos em python e, além deles, foi necessário outros dois arquivos de entrada, no formato “.txt”.

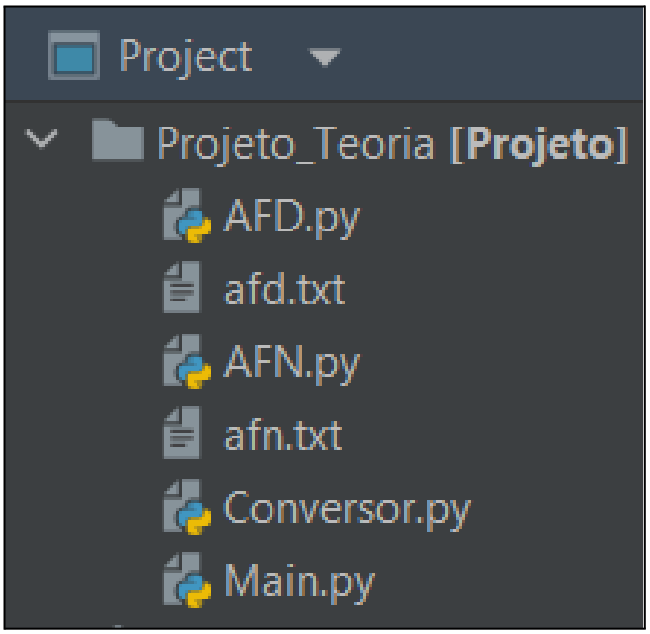

Figura 1: Arquivos do projeto

Primeiramente, uma breve descrição do conteúdo de cada arquivo proposto no projeto:

- ***AFD.py***: implementação do autômato finito deterministico.
- ***AFN.py***: implementação do autômato finito não-determinístico.
- ***Conversor.py***: Implementação da conversão dos autómatos.
- ***Main.py***: código principal, onde podemos fazer os testes e instanciar as funções.
- ***afd.txt****:* arquivo contendo o autômato finito deterministico.
- ***afn.txt****:* arquivo contendo o autômato finito não-determinístico.

Os arquivos de entrada, são padronizados para que o programa consiga efetuar a leitura do conteúdo corretamente, a formatação do *“afn.txt”* e *“afd.txt”*, são praticamente identificados, seguindo o modelo abaixo:

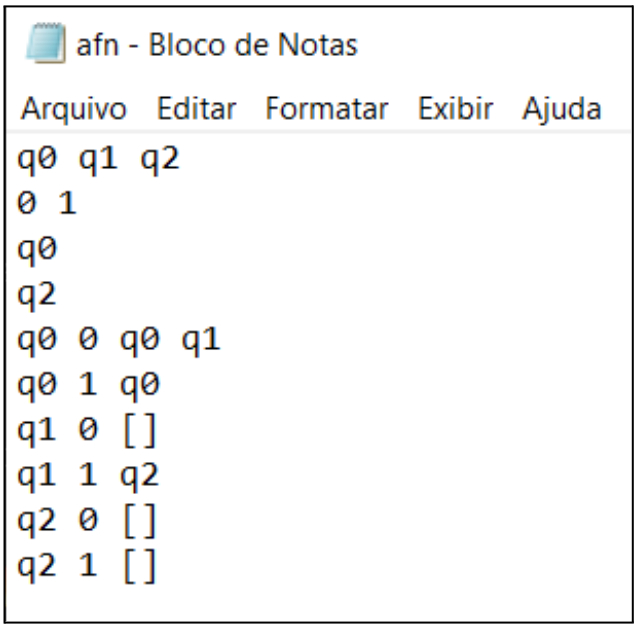

Figura 2: Modelo do *"afn.txt"*

A padronização do *"afn.txt",* é determinada pela linha, onde cada linha corresponde a uma determinada configuração do autômato.

LINHA 1: Estados contidos no autômato

LINHA 2: Símbolos terminais

LINHA 3: Indica estados iniciais

LINHA 4: Indica estado final

LINHA 5-11: Cada linha refere-se às transições contidas no autômato. A primeira coluna, é o estado atual, já a segunda coluna corresponde ao valor lido pelo estado atual, a terceira em diante, são os novos estados após a leitura da sentença.

2 AUTÔMATO DETERMINÍSTICO (AFD)

Na Teoria dos Autômatos, um autômato finito determinístico (AFD), é uma máquina de estados finita que aceita ou rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada. Desse modo, o autômato, ao receber um caractere, tende a ir para somente um novo estado.

Abaixo, temos a representação do autômato utilizado na implementação do projeto, trata-se de um autômato que reconhece palavras que seus últimos caracteres são "01".

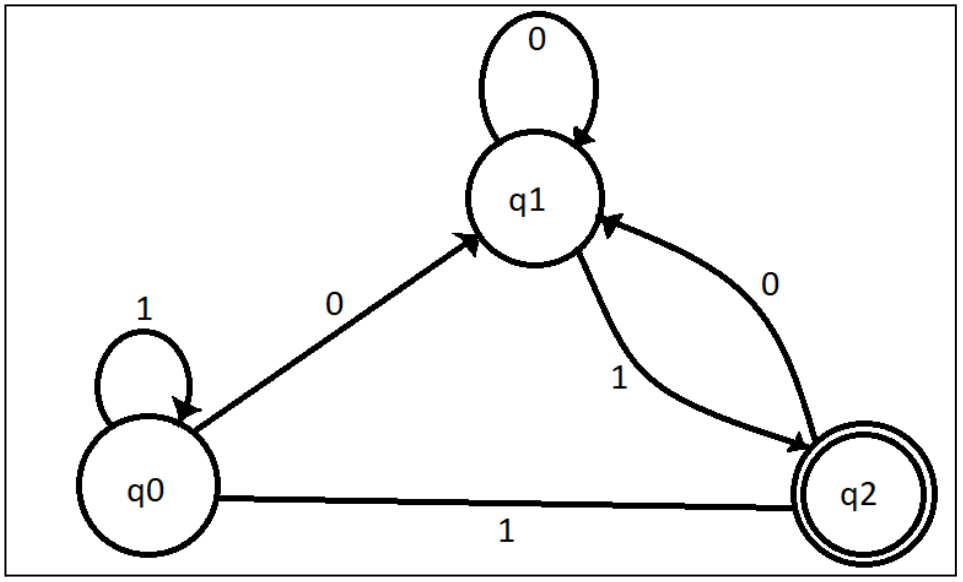

Figura 3: Autômato Determinístico (AFD)

Em relação a implementação do autômato determinístico, foi desenvolvida uma classe, e inicialmente, começamos com a abertura e leitura do arquivo passado por referência na *main.py*.

```
class AFD:
    def __init__(self, nomeArquivo, permisaoLeitura):
        # Se a leitura do arquivo = TRUE
        if(permisaoLeitura == True):
            # Faz a abertura do arquivo, com nome designado na main
            arq = open(nomeArquivo, 'r')
            # Faz a leitura dos estados iniciais e finais, transicoes
            self.Q = arq.readline().strip().split(' ')
            self.S = arq.readline().strip().split(' ')
            self.q0 = str(arq.readline()).strip()
            self.F = arq.readline().strip().split(' ')
            self.transicoes = arq.readlines()
            # Fecha o arquivo
            arq.close()
```

Figura 4: Abertura e leitura de arquivo AFD.

O arquivo é lido e inserido em listas, com suas respectivas funcionalidades. Para manipulação e verificação dos caracteres, utilizamos uma função que faz análise das sequências de símbolos.

```python
def verificaAFD(self, seq, estado):
    if seq == '':
        return estado in self.F

    else:
        proxcaract = seq[0]
        dupla = (estado, proxcaract)
        print(dupla, ' -> ', self.deltad[(dupla)])
        if dupla in self.deltad:
            return self.verificaAFD(seq[1:], self.deltad[dupla])
        else:
            return False
```

Figura 5: Verifica caracteres do AFD.

Para mostrar o autômato no console, foi necessário a implementação de uma função que exibia os estados, estado inicial, estados finais e cada transição.

```python
def printAFD(self):
    print('\n\n----------\033[1;34mAUTOMATO FINITO DETERMINISTICO\033[0;0m----------\n')
    print('Estados: ', self.Q)
    print('Estado inicial: ', self.q0)
    print('Estados finais: ', self.F)
    print('Transições:')
    print('(Estado atual, Simbolo) -> Estado resultante\n')
    for i in self.deltad:
        print(i,' -> ',self.deltad[i])
    print('\n-------------------------------------------------\n')
```

Figura 6: Função que exibe o autômato.

3 AUTÔMATO NÃO-DETERMINÍSTICO (AFN)

Na teoria da computação, uma máquina de estados finita não-determinística ou um autômato finito não-determinístico é uma máquina de estados finita onde para cada par de estado e símbolo de entrada pode haver vários próximos estados possíveis. Para esse projeto, foi utilizado como exemplo o autómato logo abaixo, que faz reconhecimento de sentenças que terminam com “01”.

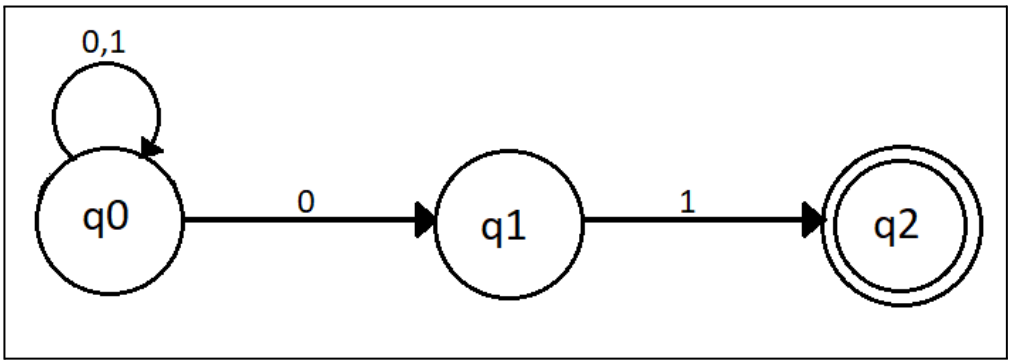

Figura 7: Autômato Finito não-Determinístico (AFN).

Assim como a implementação do autômato determinístico, foi desenvolvida uma classe, e inicialmente, começamos com a abertura e leitura do arquivo passado por referência na *main.py*.

```
class AFN:
    def __init__(self, nomeArquivo):
        arq = open(nomeArquivo, 'r')
        self.Q = arq.readline().strip().split(' ')
        self.S = arq.readline().strip().split(' ')
        self.q0 = str(arq.readline()).strip()
        self.F = arq.readline().strip().split(' ')
        self.transicoes = arq.readlines()
        arq.close()
```

Figura 8: Abertura e leitura de arquivo AFN.

O arquivo é lido e inserido em listas, com suas respectivas funcionalidades. Para manipulação e verificação dos caracteres, utilizamos uma função que faz análise das sequências de símbolos. Porém sua implementação é necessário alguns passos adicionais, pois possui a possibilidade de com a mesmo caractere ir para dois estados diferentes.

```python
def verificaAFN(self, sequencia, estadoatual):
    if sequencia == '':
        return estadoatual in self.F
    proxseq = sequencia[0]
    checaestado = (estadoatual, proxseq)
    print(checaestado, ' -> ', end='')
    if checaestado in self.deltan:
        proxest = self.deltan[checaestado]
        for est in proxest:
            if (est not in proxest[0]):
                print('Backtracking!')
                print(checaestado, ' -> ', end='')
            print(est)
            if self.verificaAFN(sequencia[1:], est):
                return True
    return False
```

Figura 9: Função para verificação de sentença.

Para mostrar o autômato no console, utilizamos a mesma função desenvolvida para o AFD, que exibe os estados, estado inicial, estados finais e cada transição.

```python
def printAFN(self):
    print('\n\n--------\033[1;34mAUTOMATO FINITO NAO DETERMINISTICO\033[0;0m--------\n')
    print('Estados: ', self.Q)
    print('Estado inicial: ', self.q0)
    print('Estados finais: ', self.F)
    print('Transições:')
    print('(Estado atual, Simbolo) -> Estados resultantes\n')
    for i in self.deltan:
        print(i, ' -> ', self.deltan[i])
```

Figura 10: Exibir AFN no console.

4 CONVERSÃO AFN PARA AFD

Inicialmente, a conversão é feita após o AFN ser definido, após a definição o código gera novos estados e transições, que serão passadas para o novo AFD. As palavras reconhecidas pelo AFD tem que coincidir com as do autômato AFN.

```
def converte(afn):
    novoQ = geraQ(afn.Q, afn)
    novoS = afn.S
    novoq0 = [afn.q0]
    novoF = geraF(novoQ, afn)
    novoDelta = geraDelta(novoQ, afn)
```

Figura 11: Gera novo estado.

A função geraDelta, fazendo a criação das novas transições, colocando em uma variável auxiliar durante o processo.

```
def geraDelta(Q, afn):
    d = set()
    aux = []
    aux2 = []
    for i in range(len(Q)):
        for j in range(len(afn.S)):
            for k in range(len(Q[i])):
                aux2 = afn.deltan[(Q[i][k], afn.S[j][0])]
                aux.append(aux2)
```

Figura 12: Geração do novo autômato.

Após as transições, estados finais e iniciais gerados, foi necessário a implementação de uma função para a formatação melhor dos dados, a qual ainda possui um pequeno erro de impressão.

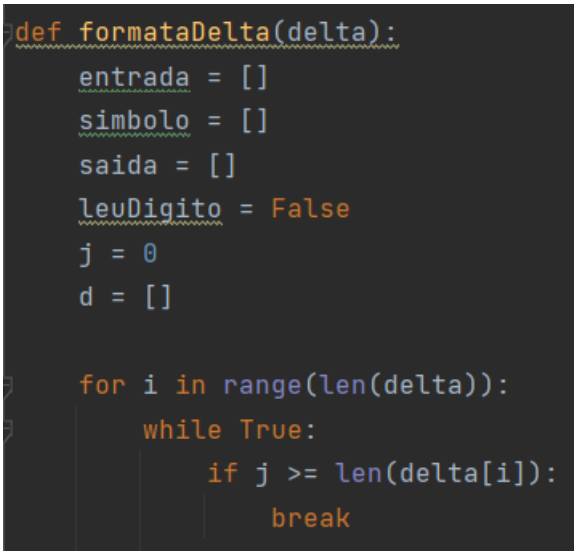

Figura 13: Formatação do autômato.

5 UTILIZAÇÃO DO SOFTWARE

Para testar o software é bem simples, onde basta abrir o arquivo *main.py,* e chamar as funções que foram importadas do conversor. Faça a verificação do nome do arquivo *.txt,* e utilize-o para a criação do AFN. Podemos criar da mesma forma o AFD, porém o mais interessante nesse projeto é fazer a conversão do AFN em AFD chamando a função *converter(afn),* passando como parâmetro o afn.

Figura 14: Inicialização do software.

Além disso, neste software foi implementado um laço de repetição que traz a possibilidade de reconhecer e verificar as sentenças, ao testar palavras, o código entra nos arquivos AFD.py e AFN.py, e a função responsável para fazer tal análise é a

verificaAFD ou verificaAFN.

```
while(1):
    # inserindo uma cadeia para testar no afn e no afd
    sequencia = input('\033[1;34mInsira uma sequencia para o reconhecimento:\033[0;0m')

    # garantir que todos os caracteres pertencem ao alfabeto
    while((afd.verificaCadeiaAFD(sequencia) != True and afn.verificaCadeiaAFN(sequencia) != True)):
        print(' ')
        print('\033[1;34mCadeia inválida!\033[0;0m')
        sequencia = input('\033[1;34mInsira uma sequencia para o reconhecimento:\033[0;0m')
    print(' ')
```

Figura 15: Reconhecimento de cadeia de caracteres.

Caso o reconhecimento da cadeia de caracteres for reconhecido pelos dois autómatos, será exibida uma mensagem para o usuário, informando o reconhecimento da sentença testada.

```
# veredito do reconhecimento da cadeia
if a and b:
    print('\n\033[1;92mA cadeia é reconhecida pelos automatos!\033[0;0m\n')
else:
    print('\n\033[1;91mA cadeia nao é reconhecida pelos automatos!\033[0;0m\n')
print('-------------------------------------------------\n')
```

Figura 16: Mensagem de reconhecimento.

6 RESULTADOS OBTIDOS

Abaixo está sendo exibido o autômato não-determinístico (AFN), e suas respectivas transições.

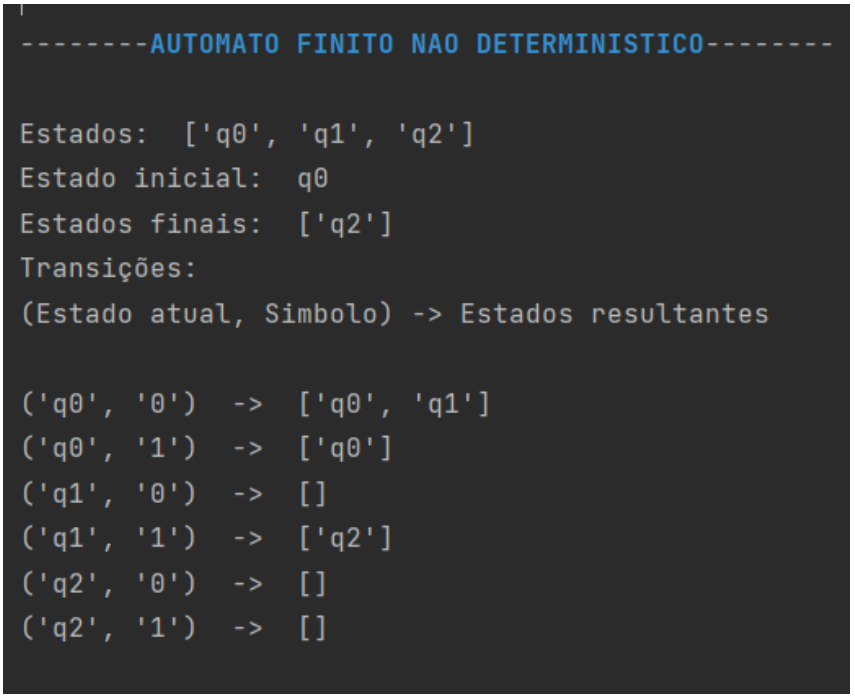

Figura 17: Console com AFN.

Da mesma forma que é exibido o AFN, é mostrado o autômato determinístico, com seus estados e transições possíveis.

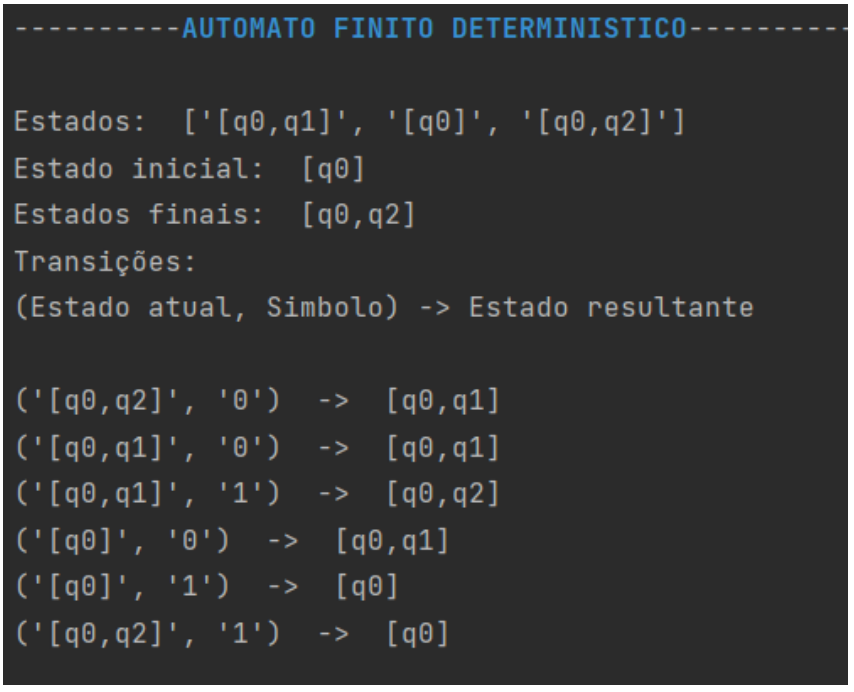

Figura 18: Console com AFD.

Quando a cadeia de caracteres não é reconhecida por nenhum dos autômatos, é enviada a mensagem de negação, e as transições que foram feitas para tentar reconhecer a palavra, que será possível caso a palavra lida completamente chegue ao estado final.

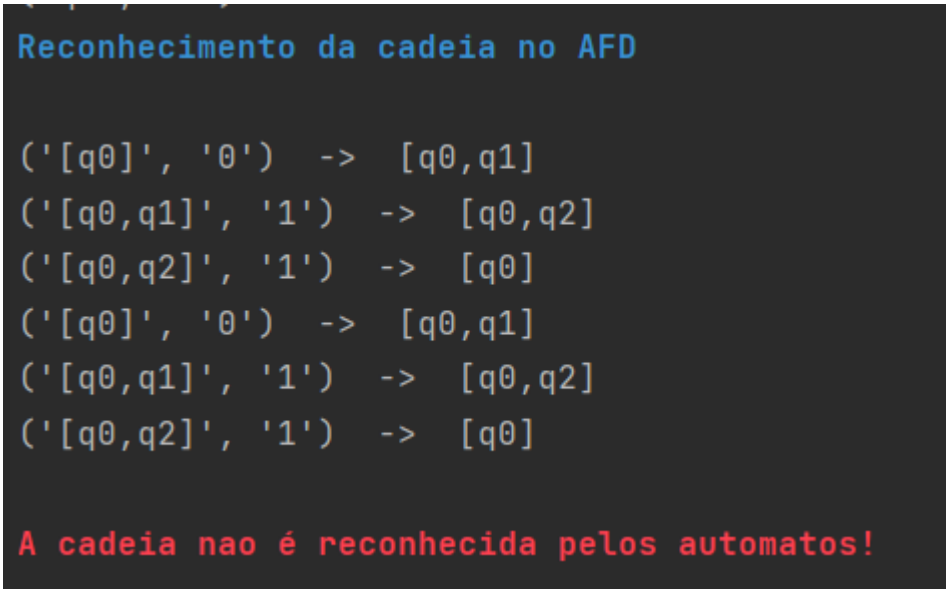

Figura 19: Negação no reconhecimento.

Quando o reconhecimento acontece, a mensagem de aprovação é enviada ao console, e exibido as transições feitas, note que ao final da leitura, o autômato conseguiu chegar ao estado final.

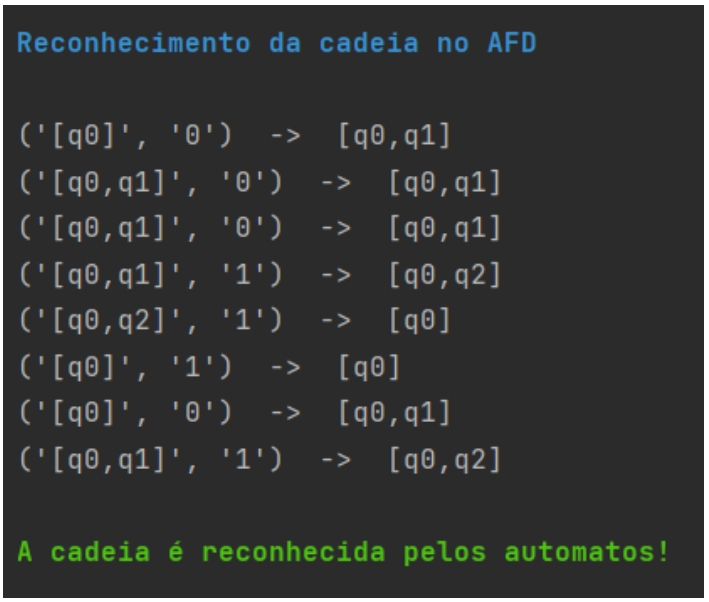

Figura 20: Aprovação no reconhecimento.

7 LINK PARA O CÓDIGO

https://github.com/iRocktys/Teoria-computacao

8 CONSIDERAÇÕES FINAIS

Após o estudo dos autómatos, e a implementação do código desenvolvido em python, nota-se que o conversor funciona de maneira eficiente, ´porém ainda faltando alguns detalhes na manipulação das listas, ao se tratar da impressão dos estados gerados pelo conversor. O algoritmo foi desenvolvido no PyCharm, e funcional no jupyter notebook, sendo necessário algumas adaptações ao importar arquivos.

9 REFERÊNCIAS

Martin, John C. - Introduction to languages and the theory of computation (2ª Edição)

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J D., 2003, "Introdução à Teoria de Autômatos, Linguagens e Computação", 2ª. Edição, Editora Campus, Brasil.

LEWIS, H. R., PAPADIMITRIOU, C. H., 1998, "Elementos de Teoria da Computação", 2ª. Edição, Ed. Bookman, Brasil.