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 CONVERSÃO DE AUTÔMATO

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.

Outro modelo seria o autômato finito não-determinístico que (AFN) é uma máquina de estados finita onde para cada par de estado e símbolo de entrada pode haver vários próximos estados possíveis, diferente do AFD. Para esse projeto, foi utilizado como exemplo o autómato logo abaixo em suas formas de AFD e AFN, onde seu objetivo é reconhecer cadeias de caracteres que terminam com “01”. Nesse projeto, foi implementado um código que faz a conversão de um AFN em um AFD, na linguagem de codificação Python.

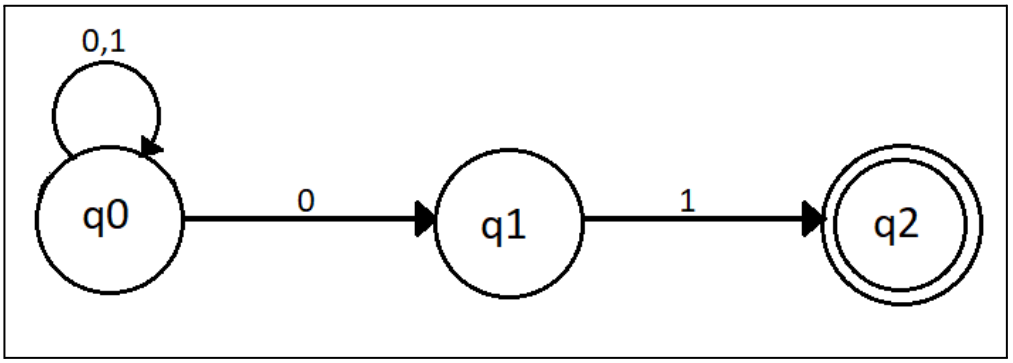


Figura 1: Autômato Finito não-Determinístico (AFN).

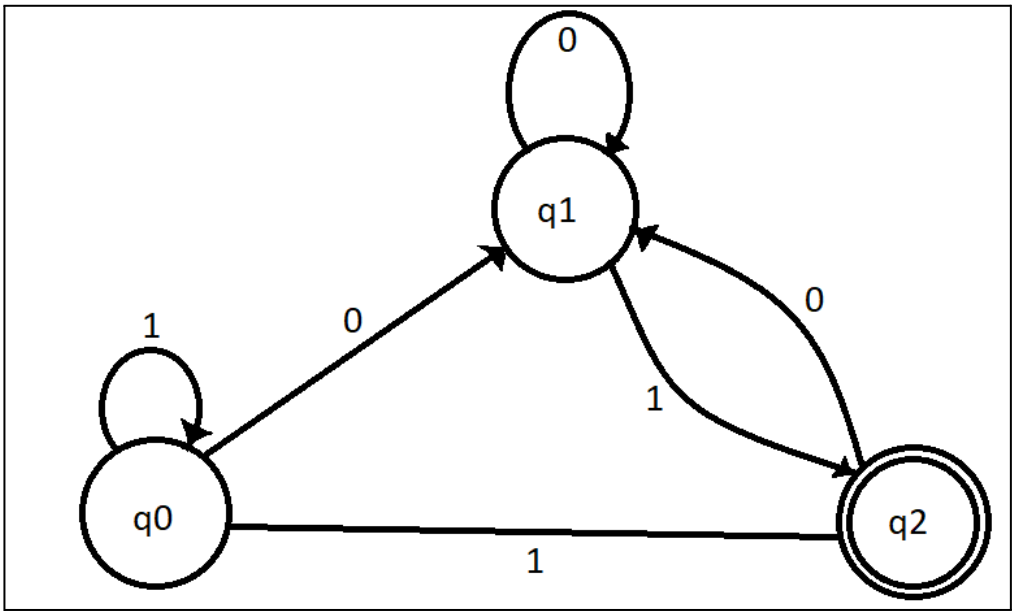


Figura 2: Autômato Finito Determinístico (AFD).

1.1 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”.

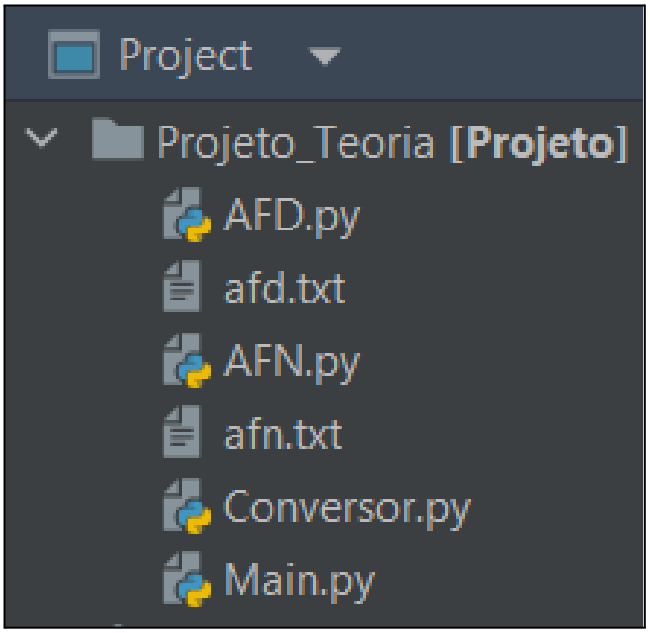


Figura 3: Arquivos do projeto

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:

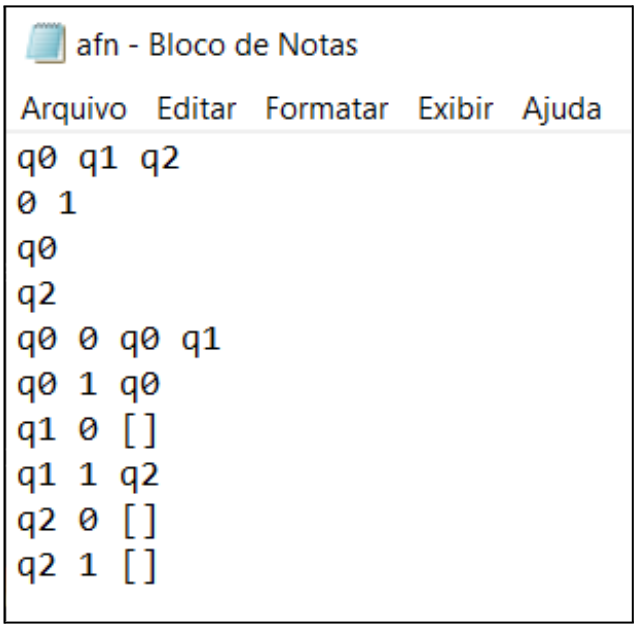


Figura 4: Modelo do *"afn.txt".*

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

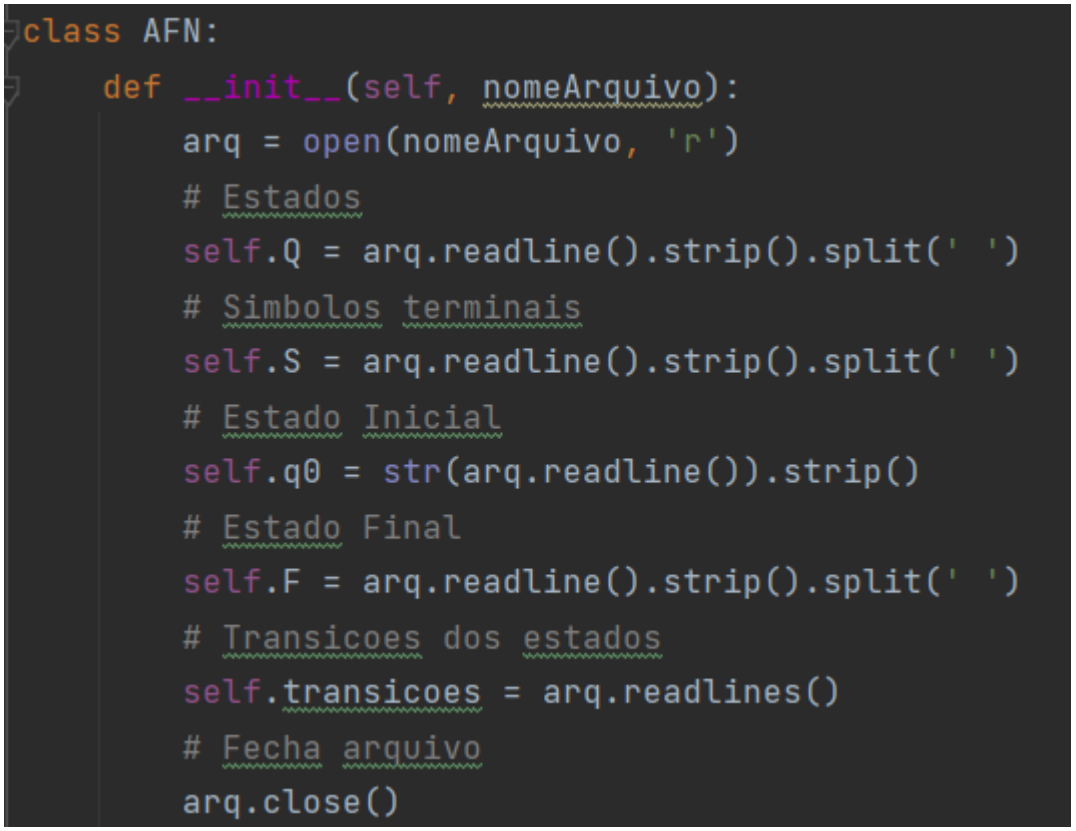
LINHA 3: Indica estados iniciais

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.

1.2 Implementação Autômato Finito não-Determinístico (AFN)

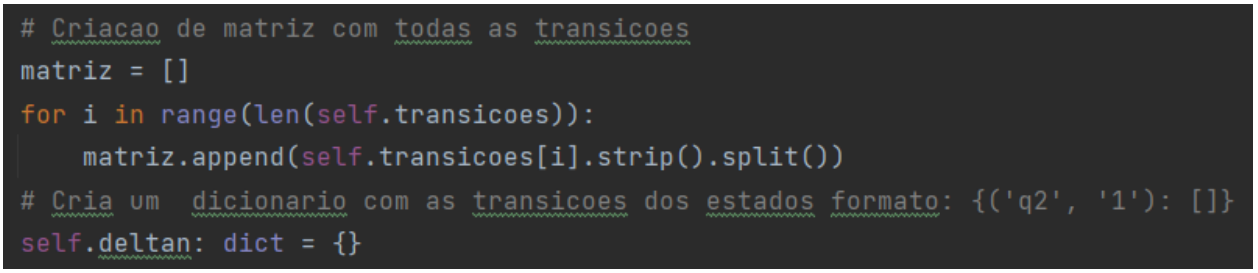
Para a implementação do autômato não determinístico, foi desenvolvida uma classe AFN, que inicialmente, abre o arquivo com nome passado por referência na página principal *main.py*. Após abertura do arquivo, cada linha lida é armazenada em um atributo da classe AFN, explicado anteriormente, e em seguida o arquivo é fechado.



```
class AFN:
    def __init__(self, nomeArquivo):
        arq = open(nomeArquivo, 'r')
        # Estados
        self.Q = arq.readline().strip().split(' ')
        # Simbolos terminais
        self.S = arq.readline().strip().split(' ')
        # Estado Inicial
        self.q0 = str(arq.readline()).strip()
        # Estado Final
        self.F = arq.readline().strip().split(' ')
        # Transicoes dos estados
        self.transicoes = arq.readlines()
        # Fecha arquivo
        arq.close()
```

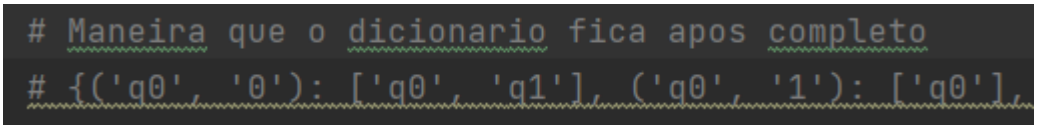
Figura 5: Abertura e leitura de arquivo AFN.

Com os atributos armazenados, é criado uma matriz com todas as transições possíveis do autômato, através de um laço de repetição. Para a melhor organização das informações, a matriz é passada para um dicionário. O formato de como o dicionário fica organizado na estrutura, está sendo exibido na imagem abaixo.



```
# Criacao de matriz com todas as transicoes
matriz = []
for i in range(len(self.transicoes)):
    matriz.append(self.transicoes[i].strip().split())
# Cria um  dicionario com as transicoes dos estados formato: {('q2', '1'): []}
self.deltan: dict = {}
```

Figura 6: Matriz e criação do dicionário.



```
# Maneira que o dicionario fica apos completo
# {('q0', '0'): ['q0', 'q1'], ('q0', '1'): ['q0'],
```

Figura 7: Organização do conteúdo.

Em decorrer da implementação, notamos a necessidade de algumas funcionalidades, como a verificação dos caracteres da sequência informada pelo usuário contém somente caracteres finais definidos no arquivo texto.

```
# Verifica se a cadeia digitada pelo usuario contem somente simbolos terminais contidos na linguagem
def verificaCadeiaAFN(self, sequencia):
    for i in sequencia:
        if(self.pertence(i) == False):
            return False
    return True
```

Figura 8: Verifica caracteres do AFD.

Foi desenvolvida uma funcionalidade para reconhecer a sequência de caracteres informados pelo usuário, que percorre a sentença transitando pelos estados contidos no dicionário.

```
# Faz a derivacao da palavra para verificar se e reconhecida pelo afn
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: Verifica a sentença do usuário no 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.

```
# Exibe o afn no console
def printAFN(self):
    print('\n\n--------AUTOMATO FINITO NAO DETERMINISTICO--------\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: Função que exibe o autômato.

1.3 Implementação Autômato Finito Determinístico (AFD)

Assim como a implementação do autômato não-determinístico, foi desenvolvida uma classe AFD, 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 designa
            arq = open(nomeArquivo, 'r')
            # Faz a leitura dos estados iniciais e finais
            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()

            matriz = []
```

Figura 11: Abertura e leitura do arquivo.

```
for i in range(len(self.transicoes)):
    matriz.append(self.transicoes[i].strip().split())
self.deltad = {}
for k in range(len(matriz)):
    self.deltad[(matriz[k][0], matriz[k][1])] = matriz[k][2]
```

Figura 12: Alocando as transições no dicionário.

A verificação da cadeia de caracteres, é da mesma forma que no AFN, retornando o verdadeiro caso a cadeia conter somente símbolos informados no arquivo texto.

```
def verificaCadeiaAFD(self, seq):
    for i in seq:
        if(self.pertence(i) == False):
            return False
    return True
```

Figura 13: Verificação de caracteres.

Outra funcionalidade em comum com AFN é a verificação da sentença, analisando as transições pelos estados, retornando a sequência de transições ocorridas pelos diferentes estados. A diferença entre a implementação ocorre devido ao fator do AFD não possuir múltiplos estados para o mesmo valor de leitura.

```
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 14: Verificação de sentença.

O modo de impressão no console é o mesmo para os dois autômatos, exibido abaixo:

```
def printAFD(self):
    print('\n\n----------AUTOMATO FINITO DETERMINISTICO----------\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 15: Exibindo o 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. Porém sua implementação é necessário alguns passos adicionais, pois possui a possibilidade de com a mesmo caractere ir para dois estados diferentes.

2 CONVERSÃO AFN PARA AFD

Inicialmente, a conversão é feita após o AFN ser definido, após o código gera novos estados e transições, que serão passadas para o novo AFD. Para efetuar a conversão foi necessário a implementação de uma funcionalidade, que calcula os novos valores dos atributos.

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

Figura 16: Gera novo estado.

A função geraDelta que recebe como parâmetro os estados e o AFN, faz a criação das novas transições.

```
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 17: Geração de transições.

Após as transições, estados finais e iniciais gerados, foi necessário a implementação de uma função para a formatação melhor dos dados, para serem exibidos de forma mais didática ao usuário.

```
def formataDelta(delta):
    entrada = []
    simbolo = []
    saida = []
    leuDigito = False
    j = 0
    d = []

    for i in range(len(delta)):
        while True:
            if j >= len(delta[i]):
                break
```

Figura 18: Formatação do autômato.

3 UTILIZAÇÃO DO SOFTWARE

Para testar o software é bem simples, 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.txt"*.

```
import AFN
import AFD

from Conversor import *

afn = AFN('afn.txt')
#afd = AFD('afd.txt',True)
afd = converte(afn)
afn.printAFN()
afd.printAFD()
```

Figura 19: 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 e verificaAFN.

```
while(1):
    # inserindo uma cadeia para testar no afn e no afd
    sequencia = input('Insira uma sequencia para o reconhecimento: ')

    # garantir que todos os caracteres pertencem ao alfabeto
    while((afd.verificaCadeiaAFD(sequencia) != True and afn.verificaCadeiaAFN(sequencia) != True)):
        print(' ')
        print('Cadeia inválida!')
        sequencia = input('Insira uma sequencia para o reconhecimento: ')
    print(' ')
```

Figura 20: Reconhecimento de cadeia de caracteres.

Após verificar que todos os caracteres contidos na palavra possuem somente símbolos terminais, é feita a verificação das transições da palavra, checando se a formação da cadeia de caracteres é possível a partir do autômato criado, armazenando na variável "a" e "b", o valor de verdadeiro ou falso.

```
# mostrar o percorrimento da cadeia no afn
print("Reconhecimento da cadeia no AFN\n")
a = afn.verificaAFN(sequencia, afn.q0)
# mostrar o percorrimento da cadeia no afd
print("Reconhecimento da cadeia no AFD\n")
b = afd.verificaAFD(sequencia, afd.q0)
```

Figura 21: Verificação das transições.

Após verificação da cadeia de caracteres, se as variáveis possuírem ambas o valor de verdadeiro, é mostrado no console para o usuário que a palavra é reconhecida pelos autómatos. Vale ressaltar que, a sentença será válida somente se ambos os autômatos reconhecerem a cadeia de caracteres.

```
# veredito do reconhecimento da cadeia
if a and b:
    print('\nA cadeia é reconhecida pelos automatos!\n')
else:
    print('\nA cadeia nao é reconhecida pelos automatos!')
print('-------------------------------------------------\n')
```

Figura 22: Mensagem de reconhecimento.

4 RESULTADOS OBTIDOS

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

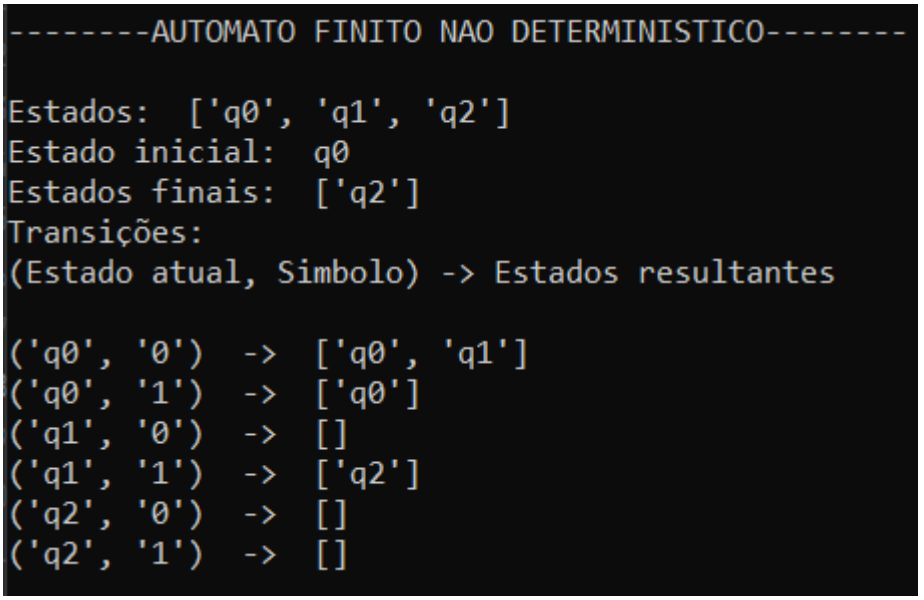


Figura 23: Console com AFN.

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

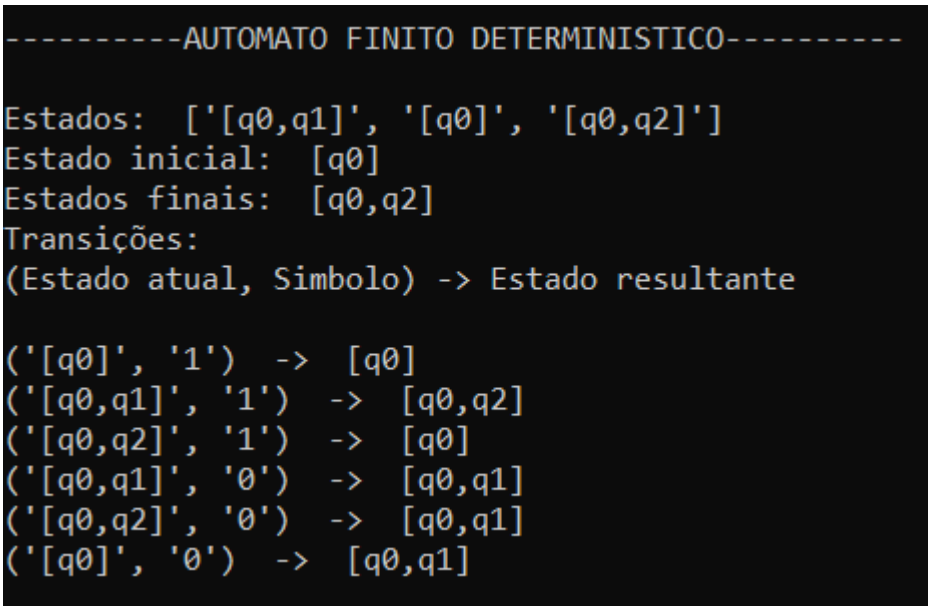


Figura 24: Console com AFD.

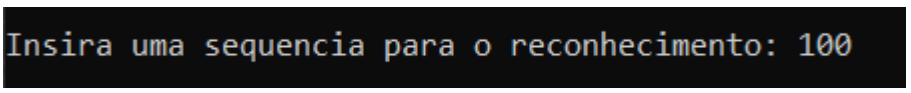


Figura 25: Captura da cadeia de caracteres do usuário.

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.

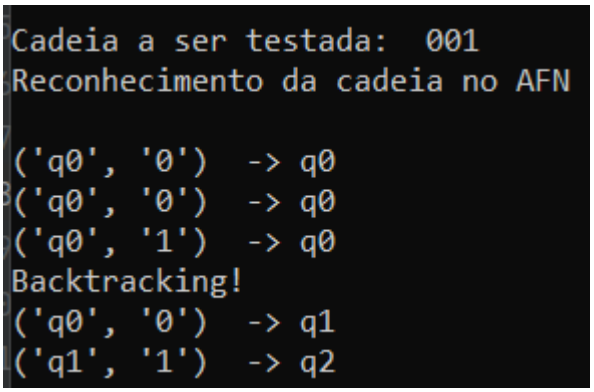


Figura 26: Negação no reconhecimento.

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.

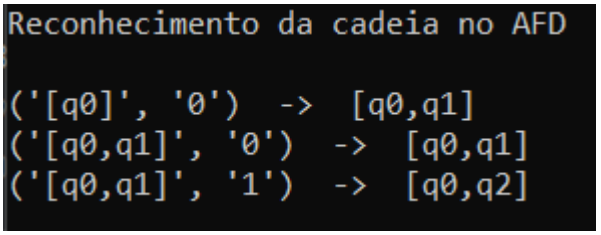


Figura 27: Aprovação no reconhecimento.

De maneira geral o processo de reconhecimento é exibido na tela para o usuário, abaixo temos como seria o reconhecimento da palavra “1001”, que é reconhecida por ambos autômatos, pois a característica dessa linguagem é reconhecer cadeias de caracteres terminados por “01”.

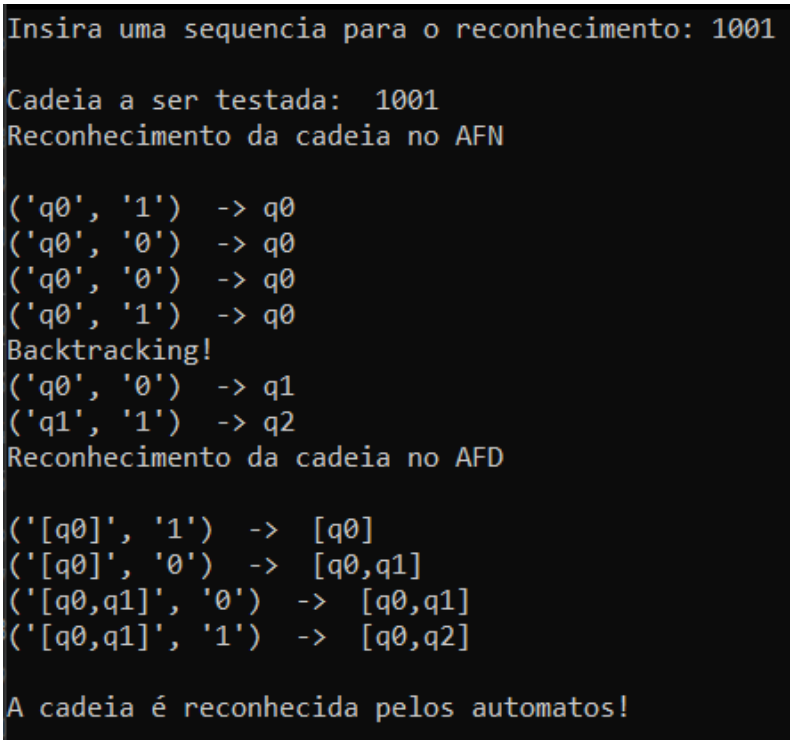


Figura 28: Reconhecimento da palavra “1001”.

5 LINK PARA O CÓDIGO

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

6 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.

7 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.