**INSTITUTO TECNOLÓGICO DE AERONÁUTICA - ITA**

**3º PROJETO DE CTC - 34**



**Projeto 3: Simulador de Autômato de Pilha**

**Aluno**

Felipe Tuyama - [ftuyama@gmail.com](mailto:ftuyama@gmail.com)

**PROFESSOR**

Carlos Henrique Q. Forster forster@ita.br

São José dos Campos, 09 de Novembro de 2015

**1 INTRODUÇÃO**

O projeto consiste no desenvolvimento de um simulador de Autômato de Pilha, na linguagem de programação Python (versão 2.7). A partir da leitura da especificação do AP (Autômato de Pilha) e de uma dada cadeia a ser simulada, o programa realiza uma busca em largura (BFS) através do Autômato, mostrando a configuração instantânea do AP a cada iteração da busca. Ao final, informa se a dada cadeia é aceita (ou não) pelo Autômato.

**2 METODOLOGIA**

O desenvolvimento do projeto consistiu nas seguintes etapas:

1. Estudo do algoritmo BFS para simular AP.
2. Definição da entrada e saída do programa.
3. Uso da implementação de AFD como base para o AP.
4. Implementação do algoritmo na linguagem Python 2.7

**2.1 Estudo do algoritmo**

O algoritmo, assim como a simulação para Autômatos Finitos Determinísticos (AFD), possui um conjunto de estados , uma cadeia formada por símbolos de um alfabeto , um conjunto de estados de aceitação e um estado inicial .

A principal mudança ocorre na Função de Transição do Autômato de Pilha, que teve de ser estendida para o seguinte formato, com o auxílio de uma Pilha:

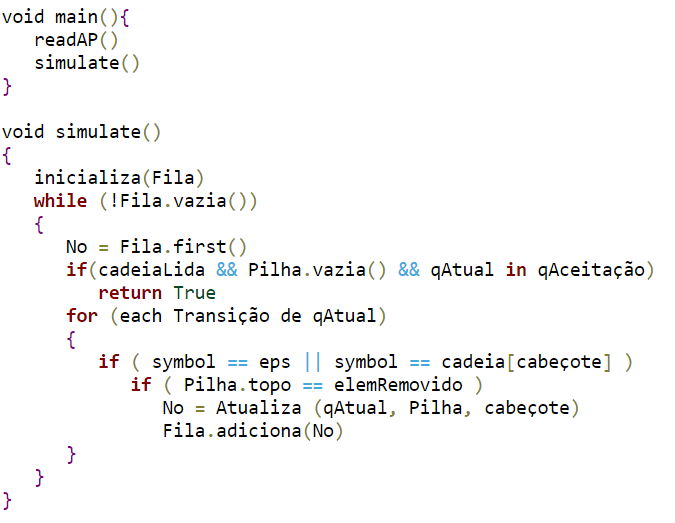
Para implementá-la, foi empregada uma lista de transições em Q. Ou seja, para cada estado Q, temos uma lista de transições possíveis, que irão ocorrer conforme o símbolo lido da cadeia (ou conforme épsilon, se possível) e o conteúdo do topo da pilha, que deve ser removido conforme algumas transições (dois if’s justapostos). Caso ocorra a transição, o estado atual deve mudar, assim como um dado símbolo pode ser adicionado à pilha.

Note que a busca em largura deve ser realizada com auxílio de uma Fila. Cada elemento dessa Fila é uma configuração instantânea do nosso AP, que pode ser determinada pela Pilha do autômato, o estado atual do AP e a posição do cabeçote de leitura sobre a cadeia sendo simulada.

A cada transição possível identificada conforme descrito no penúltimo parágrafo, um novo elemento chamado Nó (configuração instantânea do AP, descrito no último parágrafo) é adicionado à nossa Fila de busca em largura. Esse procedimento continua até que a Fila (correspondente a todas as possibilidades do nosso AP) se esvazie, ou até que ocorra a aceitação da cadeia em questão.

A aceitação da cadeia ocorre se e somente se três eventos acontecem simultaneamente: a cadeia é totalmente percorrida, a pilha se encontra vazia e o estado atual é um estado de aceitação do autômato. Caso esses três critérios sejam obedecidos, a dada cadeia pertence à linguagem reconhecida pelo AP.

O algoritmo em pseudocódigo teria então o seguinte formato:



**2.2 Definição da entrada e saída do programa**

**2.2.1 Entrada**

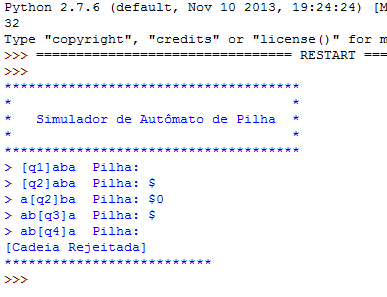
A entrada do programa é um arquivo .txt contendo as especificações do AP e a cadeia a ser simulada. É suposto que os dois arquivos de entrada estejam no mesmo diretório do programa desenvolvido. As informações são divididas por blocos necessariamente entre “#”, um elemento da lista por linha, na seguinte ordem:

Para mais detalhes da especificação, ver a seção de testes.

**2.2.2 Saída**

A saída do arquivo é escrita no próprio terminal de execução, exibindo cada possível configuração instantânea do AP contemplada pelo algoritmo BFS. No final, entre colchetes, é informado se a dada cadeia pertence à linguagem.

Exemplo de saída:



**3 CONCLUSÃO**

A implementação do *Pushdown Automaton* (Autômato de Pilha) compreendem a classe das linguagens livres de contexto, sendo muito úteis para no desenvolvimento de parsers computacionais, com aplicação na escrita de compiladores e interpretadores.

Depreende-se que AP’s não determinísticos podem lidar com situações onde várias escolhas de ação são possíveis em dada situação. No entanto, a maioria de suas escolhas resultam em falha. Portanto, o algoritmo BFS embora certeiro, pode consumir grande quantidade de memória devido à Fila implementada, em que cada elemento deve possuir uma Pilha embutida.

**4 REFERÊNCIAS BIBLIOGRÁFICAS**

[1] RIBEIRO, C. H; FORSTER, C. H. Q. Slide 7: CTC-34 Automata e Linguagens Formais, Outubro de 2015. Notas de Aula.

**5 TESTES**

**5.1) Caso A**

***APa.txt***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Simulador de Autômato de Pilha \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Alfabeto \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

a

b

c

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

q2

q3

q4

q5

q6

q7

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estado Inicial \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados de Aceitação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q4

q7

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Função de Transferência\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1 q2 ,/$

q2 q2 a,/a

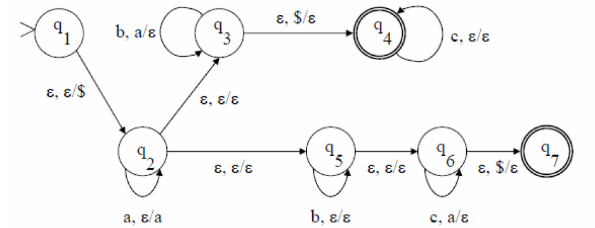
q2 q3 ,/

q2 q5 ,/

q3 q3 b,a/

q3 q4 ,$/

q4 q4 c,/

q5 q5 b,/

q5 q6 ,/

q6 q6 c,a/

q6 q7 ,$/

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Cadeia simulada \*

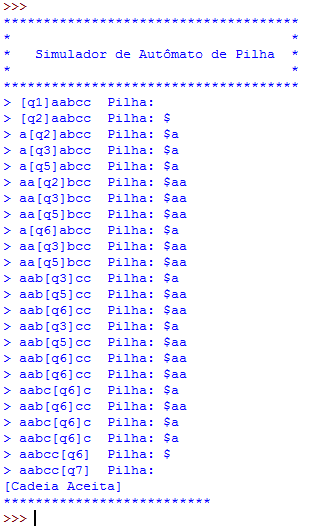
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

aabcc

#

**Resultado (Saída) obtido:**



O resultado está conforme o esperado para o AP ilustrado abaixo. Pode-se identificar, conforme visto em sala de aula, que o dado AP aceita cadeias (dentre elas aabcc) geradas pela seguinte linguagem:

**5.2) Caso B**

***APb.txt***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Simulador de Autômato de Pilha \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Alfabeto \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

0

1

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

q2

q3

q4

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estado Inicial \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados de Aceitação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

q4

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Função de Transferência\*

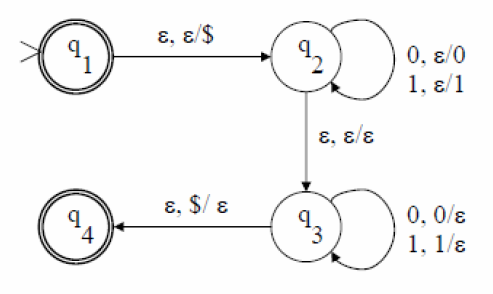
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1 q2 ,/$

q2 q2 0,/0

q2 q2 1,/1

****q2 q3 ,/

q3 q3 0,0/

q3 q3 1,1/

q3 q4 ,$/

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Cadeia simulada \*

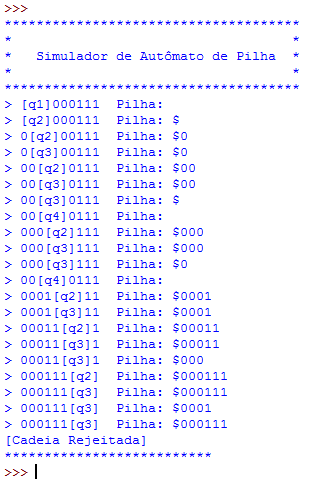
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

000111

#

**Resultado (Saída) obtido:**



O resultado está conforme o esperado para o AP ilustrado abaixo. Pode-se identificar, conforme visto em sala de aula, que o dado AP aceita cadeias geradas pela seguinte linguagem (dos palíndromos pares de 0’s e 1’s):

**5.3) Caso C**

***APc.txt***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Simulador de Autômato de Pilha \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Alfabeto \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

a

b

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

q2

q3

q4

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estado Inicial \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

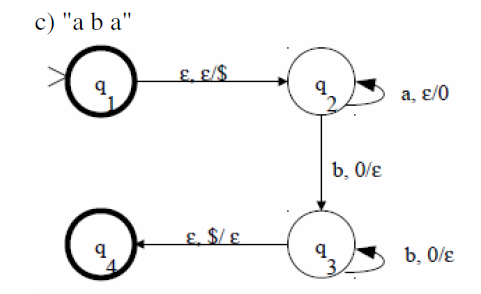
#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados de Aceitação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1

q4

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Função de Transferência\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q1 q2 ,/$

q2 q2 a,/0

q2 q3 b,0/

q3 q3 b,0/

q3 q4 ,$/

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Cadeia simulada \*

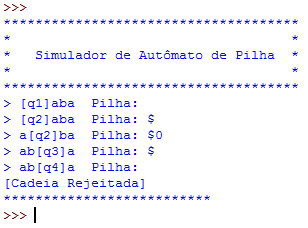
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

aba

#

**Resultado (Saída) obtido:**



O resultado está conforme o esperado para o AP ilustrado abaixo. Pode-se identificar, conforme visto em sala de aula, que o dado AP aceita cadeias geradas pela seguinte linguagem (que trata-se de um Autômato Finito Determinístico, quando fixamos o valor de n para um certo natural, como visto na primeira avaliação. Para um valor de n genérico, temos um Autômato de Pilha):

]

**6 APÊNDICE**

O código a seguir foi desenvolvido em Python 2.7.

# -\*- coding: cp1252 -\*-

#

# Simulador de Autômato de Pilha

#

# Autores: Felipe Tuyama

**import** copy

# Implementação da Fila

**class** Fila(object):

**def** \_\_init\_\_(self):

self.dados = []

**def** insere(self, elemento):

self.dados.append(elemento)

**def** remove(self):

**return** self.dados.pop(0)

**def** vazia(self):

**return** len(self.dados) == 0

**def** length(self):

**return** len(self.dados)

# Implementação da Pilha

**class** Pilha(object):

**def** \_\_init\_\_(self):

self.dados = []

**def** insere(self, elemento):

**if** elemento != "":

self.dados.append(elemento)

**def** remove(self):

**if** **not** self.vazia():

**return** self.dados.pop(-1)

**def** vazia(self):

**return** len(self.dados) == 0

**def** length(self):

**return** len(self.dados)

**def** inTopo(self, elemento):

**if** elemento == "":

**return** True

**return** elemento == self.topo()

**def** topo(self):

**if** **not** self.vazia():

**return** self.dados[len(self.dados)-1]

**def** log(self):

info = "Pilha: "

**for** i **in** range(0, len(self.dados)):

info += self.dados[i]

**return** info

# Nó de Transição

**class** Transition(object):

**def** \_\_init\_\_(self, st2, sym, rem, add):

self.st = st2

self.sym = sym

self.add = add

self.rem = rem

# Status da Busca BFS (elementos da Fila):

**class** CheckPoint(object):

**def** \_\_init\_\_(self, q, stt, P):

self.q = q

self.stt = stt

self.P = P

# Remove Tabs e Espaços da linha lida

**def** wash(saida):

saida = saida.strip('\t')

saida = saida.strip('\b')

**return** saida

# Leitura da Transição do AP

**def** nextData(char):

**global** reader

begin = 0

**while** begin < len(reader) **and** reader[begin] != char:

begin += 1

**if** begin == 0: info = ""

**else**: info = reader[0:begin]

reader = reader[begin+1:len(reader)]

**return** info

# Leitura de uma linha não vazia sem \n

**def** read():

lido = arquivo.readline().rstrip('\n')

**while** lido == "":

lido = arquivo.readline().rstrip('\n')

**return** lido

# Leitura do Arquivo de Entrada

**def** readAP():

**global** reader

**for** i **in** range(0, 6):

reader = " "

**while** reader[0] != '#':

reader = read()

reader = read()

**while** reader[0] != '#':

**if** i == 0: alphabet.append(wash(reader))

**elif** i == 1: states.append(wash(reader))

**elif** i == 2:

initial.append(wash(reader))

**for** j **in** range(0, len(states)):

TFE.append([])

**elif** i == 3: accept.append(wash(reader))

**elif** i == 4:

st1 = nextData(' ')

st2 = nextData(' ')

sym = nextData(',')

rem = nextData('/')

add = nextData('')

Trans = Transition(st2, sym, rem, add)

TFE[states.index(st1)].append(Trans)

**elif** i == 5: alphabet.append(wash(reader))

reader = read()

# Simulação do Autômato de Pilha

**def** simulate():

**global** P, F

stt = q = 0

No = CheckPoint(states.index(initial[0]),0,P)

F.insere(No)

**while** **not** F.vazia():

No = F.remove()

P = No.P

q = No.q

stt = No.stt

**print** "> "+string[:stt]+"["+states[q]+"]"+string[stt:]+" "+P.log()

**if** No.stt >= len(string):

**if** states[No.q] **in** accept:

**if** P.vazia():

**return** True

**for** j **in** range(0, len(TFE[No.q])):

Trans = TFE[No.q][j]

**if** (No.stt<len(string) **and** Trans.sym == string[No.stt]) **or** Trans.sym == "":

**if** No.P.inTopo(Trans.rem):

q = states.index(Trans.st)

P.insere(Trans.add)

**if** Trans.rem != "":

P.remove()

**if** Trans.sym != "":

stt += 1

Noq = CheckPoint(q, stt, copy.deepcopy(P))

F.insere(Noq)

**return** False

# Dados do Autômato de Pilha

alphabet = [] # Alfabeto de entradas.

states = [] # Estados do AP.

initial = [] # Estado inicial

accept = [] # Estados de aceitação.

TFE = [] # Tabela de Fluxo de Estados.

string = "" # Cadeia a ser simulada.

P = Pilha() # Pilha do Autômato de Pilha

F = Fila() # Fila para a simulação BFS

# Rotina main()

**print** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

**print** "\* \*"

**print** "\* Simulador de Autômato de Pilha \*"

**print** "\* \*"

**print** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

arquivo = open('APc.txt', 'r')

readAP()

string = alphabet.pop()

result = simulate()

**if** result: **print** "[Cadeia Aceita]"

**else**: **print** "[Cadeia Rejeitada]"

**print** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"