

Relatório Trabalho Final de LFA

Lucas Gabriel e Santiago Cardoso

Para implementar um autômato com pilha, primeiro foram criadas 3 classes: Transition, State e AutomatonStack.

Cada classe contém os seguintes atributos:

Transition:

- string : symbol_read (símbolo da fita que será lido)
- string : symbol_top_pull (símbolo a ser desempilhado)
- string : symbol_push (símbolo a ser empilhado)

State:

- string: name (nome do estado)
- ArrayList : transitions(Lista de transições do estado)
- ArrayList: stack (pilha)

AutomatonStack:

- State : start_state (estado inicial do autômato)
- ArrayList : states (lista de estados)
- ArrayList: final_state (lista de estados finais)
- string : input (palavra a ser lida)
- ArrayList : stack (pilha)

Obs:

Vale destacar que a pilha está sendo representada por um ArrayList de forma que o topo da pilha é o último elemento do ArrayList (topo = stack[-1]) . Essa representação foi escolhida pois é uma maneira mais assertiva de se representar uma pilha em python, e podemos usar as funções internas do Python, sem necessidade de criar novos algoritmos para manipular a pilha. Por exemplo, para adicionar um elemento no topo podemos utilizar stack.append(), e para eliminar um elemento stack.pop().

Dessa forma, cada estado possui diversas transições, e cada autômato possui diversos estados.

O método responsável por iniciar a execução do autômato foi denominado "read_input()", esse método define um "estado atual" que no início começa sendo o estado **inicial** passado pelo usuário, e após isso imprime para o usuário o estado inicial no qual a pilha se encontra, que por padrão é vazia.

A partir do estado atual, para cada letra presente na fita, o autômato faz a leitura desta letra, verifica se há algum movimento vazio presente na lista de transições do estado atual, executa as operações do autômato e após isso imprime para o usuário o estado atual, a letra lida e a pilha. Se a pilha estiver vazia e todas as letras da fita foram lidas, então a palavra é aceita.

Para situações em que há movimento vazio, utilizamos uma função para verificar a existência de movimentos vazios, e caso essa função retorne True, o algoritmo cria um novo autômato que é uma cópia do autômato atual, e que possui estado inicial sendo equivalente ao estado indicado pela transição. Dessa forma conseguimos simular dois estados ativos ao mesmo tempo sem que a pilha de um interfira na pilha do outro.

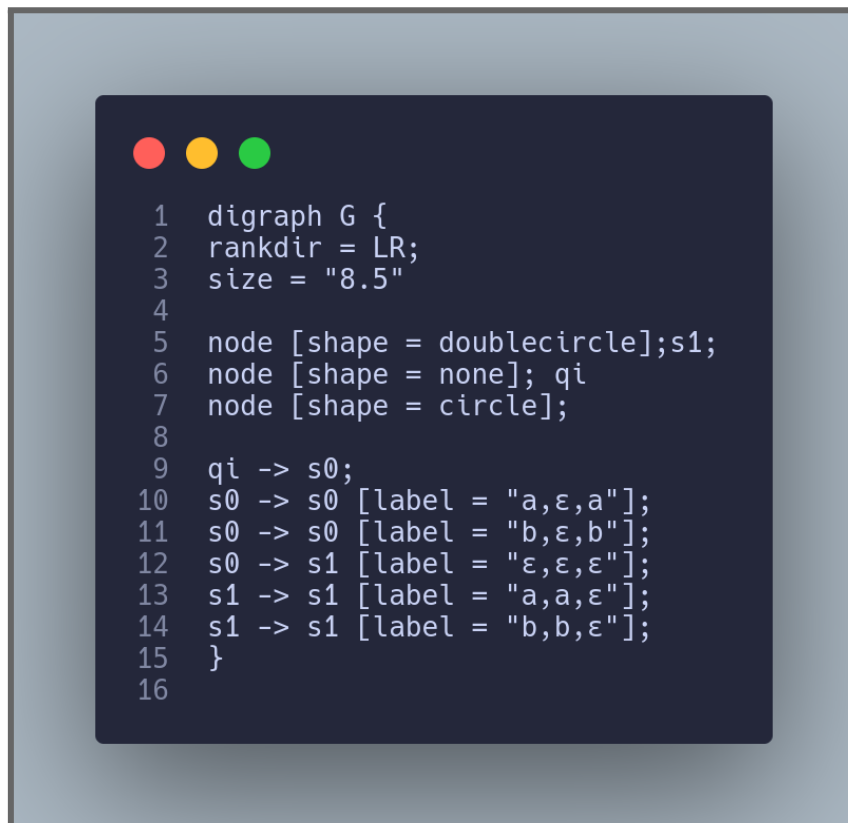
Para fazer a leitura do autômato, passamos um arquivo .json formatado da seguinte forma:



```
1  {
2    "type": "PDA",
3    "pda": {
4      "states": {
5        "s0": [
6          {
7            "label": "a,,a",
8            "next": "s0"
9          },
10         {
11           "label": "b,,b",
12           "next": "s0"
13         },
14         {
15           "label": ",, ",
16           "next": "s1"
17         }
18       ],
19       "s1": [
20         {
21           "label": "a,a,",
22           "next": "s1"
23         },
24         {
25           "label": "b,b,",
26           "next": "s1"
27         }
28       ]
29     }
30   },
31   "start": "s0",
32   "final": "s1",
33   "tests": {
34     "accept": "abba"
35   }
36 }
37
```

E assim fazemos a leitura, extração e formatação dessas informações passadas a partir do “#Read JSON file”. Abrimos o arquivo .json, fazemos o split da categoria de estados e coletamos as informações salvando-as em objetos “State” em uma lista chamada “states”, após isso, coletamos também o input de palavra, e os estados finais e o inicial, definindo o estado inicial em uma variável e o estado final em uma lista.

Após essa etapa de extração dos dados, criamos um arquivo .dot formatado com base na ferramenta “GraphvizOnline”, as transições epsilon, que estavam representadas como “”, foram transcritas como símbolos ϵ , e formatamos a saída de dados da seguinte maneira:



```
1 digraph G {
2   rankdir = LR;
3   size = "8.5"
4
5   node [shape = doublecircle];s1;
6   node [shape = none]; qi
7   node [shape = circle];
8
9   qi -> s0;
10  s0 -> s0 [label = "a,ε,a"];
11  s0 -> s0 [label = "b,ε,b"];
12  s0 -> s1 [label = "ε,ε,ε"];
13  s1 -> s1 [label = "a,a,ε"];
14  s1 -> s1 [label = "b,b,ε"];
15 }
16
```

Após essas etapas de extração e criação de dados, realizamos a automação executando todos os passos descritos anteriormente neste relatório.

E, por fim, através da biblioteca “keyboard”, fazemos a passagem desse arquivo .dot para o “GraphvizOnline” apresentar a representação gráfica do autômato. Primeiro verificamos o sistema operacional do usuário, no momento, aceitamos somente o Windows para a representação gráfica, caso seja outro sistema, simplesmente executamos o código com as informações que imprimimos no terminal, apresentam-se todos os passo a passos do autômato, porém não temos uma representação gráfica como quando utilizado no Windows com o “GraphvizOnline”.

Selecionamos alguns autômatos para executar o simulador, entre eles, temos os seguintes autômatos:



```
1  {
2    "type": "PDA",
3    "pda": {
4      "states": {
5        "s0": [
6          {
7            "label": "a,,a",
8            "next": "s0"
9          },
10         {
11           "label": "b,,b",
12           "next": "s0"
13         },
14         {
15           "label": ",, ",
16           "next": "s1"
17         }
18       ],
19       "s1": [
20         {
21           "label": "a,a,",
22           "next": "s1"
23         },
24         {
25           "label": "b,b,",
26           "next": "s1"
27         }
28       ]
29     }
30   },
31   "start": "s0",
32   "final": "s1",
33   "tests": {
34     "accept": "abba"
35   }
36 }
37
```



```
1  {
2    "type": "PDA",
3    "pda": {
4      "states": {
5        "s1": [
6          {
7            "label": ",,$",
8            "next": "s2"
9          }
10       ],
11       "s2": [
12         {
13           "label": "0,,0",
14           "next": "s2"
15         },
16         {
17           "label": "1,0,",
18           "next": "s3"
19         }
20       ],
21       "s3" : [
22         {
23           "label": "1,0,",
24           "next": "s3"
25         },
26         {
27           "label": ",$,",
28           "next": "s4"
29         }
30       ],
31       "s4" : []
32     }
33   },
34   "start": "s1",
35   "final": ["s1", "s4"],
36   "tests": {
37     "accept": "0011"
38   }
39 }
40
```

Também estudamos os trabalhos passados como referência, entre eles o “GraphvizOnline” e o simulador de autômatos do Kyle Dickerson, que serviram de base para ideias da criação do nosso simulador.

Para executar o simulador precisa-se ter as seguintes bibliotecas do python instaladas devidamente:

```
import platform
import sys
import json
import time
import keyboard
```

E também, definir-se adequadamente os arquivos de inicialização do simulador, como descrito anteriormente neste relatório. Por fim, digita-se “\$ python3 pda.py” e o código é executado.