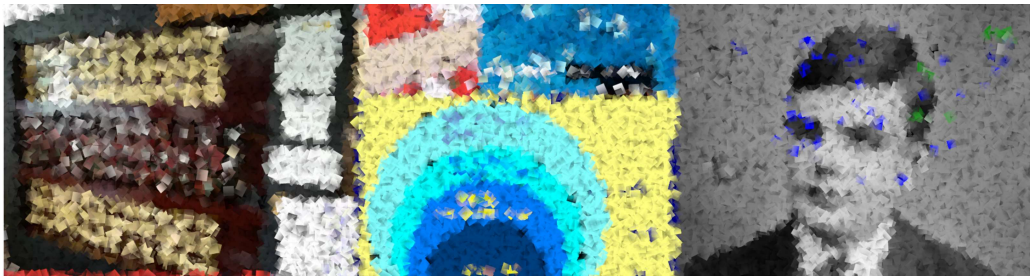
Tutorial Iterativo em Jupyter sobre Máquinas de Turing para Operações Aritméticas



Reginaldo Gregório de Souza Neto

RA: 2252813

e-mails: reginaldoneto@alunos.utfpr.edu.br

Universidade Tecnológica Federal do Paraná (UTFPR)

Departamento Acadêmico de Computação (DACOM-CM)

Curso de Bacharelado em Ciência da Computação.

GitHub do Projeto

Resumo

Este relátorio se trata da implementação de uma máquina de Turing para resolver as 4 operações aritméticas básicas. Para tal, será realizada a prototipação dos modelos das máquinas de Turing no aplicativo JFLAP, assim como a programação da máquina de Turing em Python com o auxílio da biblioteca automota_lib. *Serão realizados testes parar verificar o funcionamento do algoritmos, assim como uma análise de seus resultados.*

Introdução

Autômatos são máquinas de estados que são capazes de reconhecer linguagens e/ou palavras de acordo com seu alfabeto e suas funções de transições. Para que isso ocorra, cada máquina possui um modo de funcionamento específico de acordo as necessidades do ambiente em que foram elaboradas.

Neste trabalho, abordaremos as Máquinas de Turing, que serão melhor explicadas no tópico a seguir. Entretanto, é válido tomar ciência de que o prícipio de elaboração desse projeto condiz com a prototipação e criação de um analisador léxico para a execução das 4 operações básicas aritméticas (soma, subtração, divisão e multiplicação).

Máquinas de Turing

Máquina de Turing (MT) se trata de um autômato proposto por Alan Turing em 1936, semelhante a um autômato finito com estados e funções de transições, entretanto a MT possui memória infinita e irrestrita. Além de atuar sobre uma ou mais fitas, as máquinas de Turing são os autômatos que mais se aproximam com os computadores de hoje em dia, deste modo, pode-se afirmar que uma máquina de Turing pode fazer tudo o que um computador real pode fazer.

O conceito da máquina de Turing se dispõe principalmente sobre ações na fita (memória), sendo possível ler, escrever e determinar qual o movimento do cabeçote de leitura sobre a fita. Por conta disso, é possível realizarmos diversas operações e desempenhar funções complexas apenas com esses princípios básicos de funcionamento da MT. Dentre elas, as 4 operações

Preparação do Ambiente

Para a execução do algoritmo, é necessário que a biblioteca automota-lib esteja instalada no computador, assim como uma versão recente do Python, que deve estar adicionado às variáveis de ambiente.

Em primeiro momento, o relatório foi editado no Google Colab e em seguida foi feita a instalação da extensão do Jupyter no Visual Studio Code para que a formatação das imagens sejam dispostas corretamente no documento.

A biblioteca que auxiliam na implementação de Máquinas de Estados e Autômatos, como automata-lib.

```
!pip3 install automata-lib
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: automata-lib in /home/alunos/a2252813/.local/lib/pytho
Requirement already satisfied: pydot in /home/alunos/a2252813/.local/lib/python3.9/si
Requirement already satisfied: pyparsing>=2.1.4 in /usr/lib/python3/dist-packages (fr
```

Instalação de alguns *plugins* para o Jupyter:

```
!jupyter nbextension install https://rawgit.com/jfbercher/small_nbextensions/master/highli
!jupyter nbextension enable highlighter/highlighter
```

```
/bin/bash: linha 1: jupyter: comando não encontrado
/bin/bash: linha 1: jupyter: comando não encontrado
```

```
%%javascript
require("base/js/utils").load_extensions("highlighter/highlighter")
```

Operações Aritméticas

Para realizar o cálculo das operações aritméticas, as máquinas de Turing foram elaboradas de acordo com o artigo (Construction of a Basic Calculator through the Turing Machine – A Review), onde foram padronizados os caracteres possíveis de entrada, escrita e ações sobre a fita.

Ficando estipulado que:

(#) = Simboliza espaços em branco;

(E) = Simboliza uma unidade, ou seja, EEE = 3;

(1) = Simboliza uma unidade, ou seja, 111 = 3;

(Z) = Marcador de posição lógica na fita;

(+) = Símbolo de Adição;

(-) = Símbolo de Subtração;

(/) = Símbolo de Divisão;

(*) = Símbolo de Multiplicação;

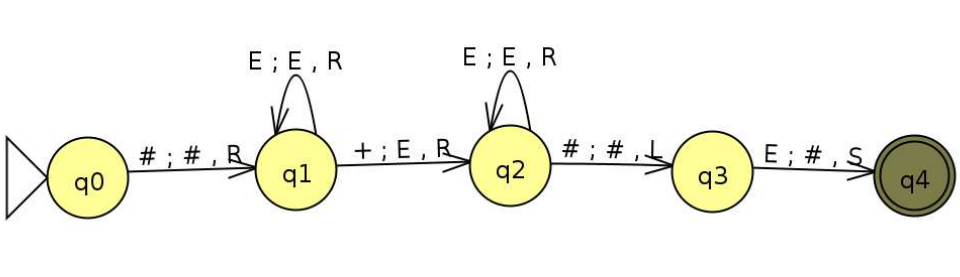
(R) = Simboliza a movimentação do cabeçote para a DIREITA;

(L) = Simboliza a movimentação do cabeçote para a ESQUERDA;

(S) = Simboliza que o cabeçote não deve se mover. (Na biblioteca automota-lib, esse símbolo é substituído pelo caracter 'N', abreviação da palavra NONE, que simboliza que não haverá movimento no cabeçote de leitura).

Adição

Exemplo da máquina de Turing de ADIÇÃO replicada no JFLAP:



Como podemos observar, essa máquina de Turing elaborada para a ADIÇÃO possui 5 estados, sendo 1 (q0) inicial e 1 (q4) final. Seu princípio de funcionamento é bem simples:

A MT identifica o início (e o final) da fita com o caraceter #, em seguida permanece no estado q1 enquanto estiver lendo caracteres 'E' na entrada ela os replica sobre a fita deslocando o cabeçote para a direita.

Ao encontrar o símbolo de adição (+) ela o substitui por um 'E' na fita e continua a replicação de entradas.

No fim da fita, a máquina volta o cabeçote uma casa para a esquerda e substitui o último 'E' por '#' para que a soma seja finalizada de maneira correta.

A implementação da MT pode ser vista no código I.

Código I: Implementação da Máquina de Turing para Adição

```
from automata.tm.ntm import NTM
# NTM que faz a operação de adição.
ntm = NTM(
    states={'s0', 's1', 's2', 's3', 'halt'},
    input_symbols={'0', '1', '+'},
    tape_symbols={'0', '1', '+', '#', 'E'},
    transitions={
        's0': {
            '#': {('s1', '#', 'R')}
        },
        's1': {
            'E': {('s1', 'E', 'R')},
            '+': {('s2', 'E', 'R')}
        },
        's2': {
            'E': {('s2', 'E', 'R')},
            '#': {('s3', '#', 'L')}
        },
        's3': {
            'E': {('halt', '#', 'N')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)
```

Função de Validação para verificar se a especificação está correta.

```
ntm.validate()  # returns True
```

```
True
```

```
ntm.read_input_stepwise('#EEEEE+EE#')
```

```
<generator object NTM.read_input_stepwise at 0x7f7fde7ec7b0>
```

O método/função `read_input(palavra)` verifica se a *palavra* é aceita pela Máquina de Turing e

```
ntm.read_input('#EEEEE+EE#').pop().print()
```

```
     halt: #EEEEEEE##
                   ^
```

```
if ntm.accepts_input('#EEEEE+EE#'):
    print('accepted')
else:
    print('rejected')
```

```
     accepted
```

```
ntm.read_input_stepwise('#EEEEE+EE#')
```

```
     <generator object NTM.read_input_stepwise at 0x7f7fde7ecac0>
```

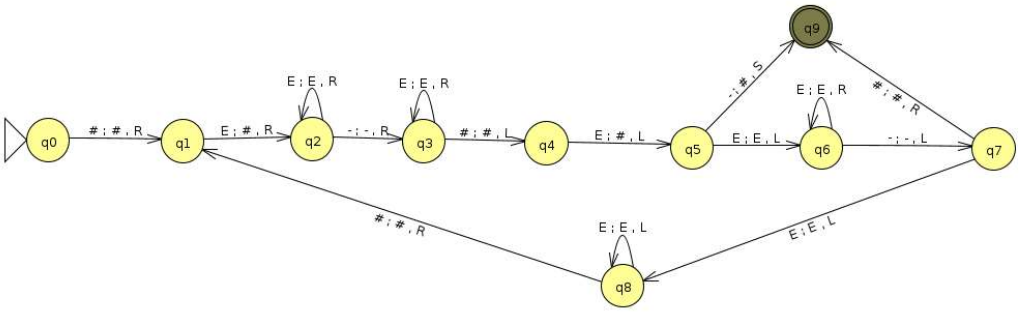
```
palavras = ['#E+E#','#E+EE#','#EEEEE+EE#']

for w in palavras:
  print("Verificando palavra:", w)
  if ntm.accepts_input(w):
    print('aceita')
  else:
    print('rejeitada')
  ntm.read_input(w).pop().print()
```

```
     Verificando palavra: #E+E#
     aceita
     halt: #EE##
              ^
     Verificando palavra: #E+EE#
     aceita
     halt: #EEE##
               ^
     Verificando palavra: #EEEEE+EE#
     aceita
     halt: #EEEEEEE##
                   ^
```

Subtração

Exemplo da máquina de Turing de SUBTRAÇÃO replicada no JFLAP:



Como podemos observar, essa máquina de Turing elaborada para a SUBTRAÇÃO possui 9 estados, sendo 1 (q0) inicial e 1 (q9) final. Seu princípio de funcionamento é um pouco mais complicado:

A MT identifica o início (e o final) da fita com o caraceter #, em seguida substitui o primeiro 'E' lido na entrada por '#' na fita e permanece no estado q2 replicando os caracteres 'E' na fita e descolando o cabeçote para a direita.

Ao encontrar o símbolo de subtração (-) ela o copia na fita e continua a replicação de entradas.

No fim da fita, a máquina volta o cabeçote uma casa para a esquerda e substitui o último 'E' por '#' para que a subtração seja finalizada de maneira correta.

Para a validação do resultado, a MT compara se o caracter de subtração (-) foi encontrado novamente para que possa ser finalizado o programa (caso em que a subtração dá zero). Ou então, verifica se o termo à esquerda do (-) possui mais unidades do que o da direita, caso isso ocorra, a MT replica seu funcionamento a partir do estado q1 para que sempre que haja uma unidade em cada lado do operados, ambas são sobrescritas por '#' eliminando uma a uma da fita para o cálculo correto da subtração.

A implementação da MT pode ser vista no código II.

Código II: Implementação da Máquina de Turing para Subtração

```
from automata.tm.ntm import NTM
# NTM que faz a operação de subtração.
ntm = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 'halt'},
    input_symbols={'0', '1', '-'},
    tape_symbols={'0', '1', '-', '#', 'E'},
    transitions={
        's0': {
            '#': {('s1', '#', 'R')}
        },
        's1': {
            'E': {('s2', '#', 'R')}
        },
        's2': {
            'E': {('s2', 'E', 'R')},
            '-': {('s3', '-', 'R')}
```

```
        },
        's3': {
            'E': {('s3', 'E', 'R')},
            '#': {('s4', '#', 'L')}
        },
        's4': {
            'E': {('s5', '#', 'L')}
        },
        's5': {
            'E': {('s6', 'E', 'L')},
            '-': {('halt', '#', 'N')}
        },
        's6': {
            'E': {('s6', 'E', 'L')},
            '-': {('s7', '-', 'L')}
        },
        's7': {
            'E': {('s8', 'E', 'L')},
            '#': {('halt', '#', 'N')}
        },
        's8': {
            'E': {('s8', 'E', 'L')},
            '#': {('s1', '#', 'R')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)
```

Função de Validação para verificar se a especificação está correta.

```
ntm.validate()  # returns True
```

```
True
```

```
ntm.read_input_stepwise('#EEEEE-EE#')
```

```
<generator object NTM.read_input_stepwise at 0x7f7fde7ec6d0>
```

O método/função `read_input(palavra)` verifica se a *palavra* é aceita pela Máquina de Turing e retorna a configuração final para ela.

```
ntm.read_input('#EEEEE-EE#').pop().print()
```

```
halt: ###EEE####
            ^
```

```
if ntm.accepts_input('#EEEEE-EE#'):
    print('accepted')
```

```
else:
    print('rejected')
```

```
 accepted
```

```
ntm.read_input_stepwise('#EEEEE-EE#')
```

```
 <generator object NTM.read_input_stepwise at 0x7f7fdea0e270>
```

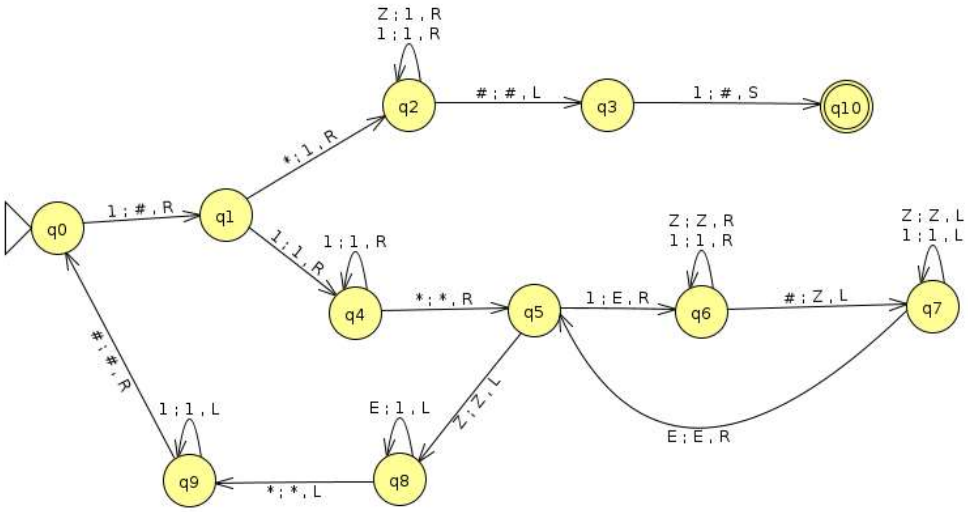
```
palavras = ['#E-E#','#E-EE#','#EEEEE-EE#']

for w in palavras:
  print("Verificando palavra:", w)
  if ntm.accepts_input(w):
    print('aceita')
  else:
    print('rejeitada')
  ntm.read_input(w).pop().print()
```

```
 Verificando palavra: #E-E#
 aceita
 halt: #####
        ^
 Verificando palavra: #E-EE#
 aceita
 halt: ##-E##
       ^
 Verificando palavra: #EEEEE-EE#
 aceita
 halt: ###EEE####
            ^
```

Multiplicação

Exemplo da máquina de Turing de MULTIPLICAÇÃO replicada no JFLAP:



Como podemos observar, essa máquina de Turing elaborada para a MULTIPLICAÇÃO possui 10 estados, sendo 1 (q0) inicial e 1 (q10) final. Seu princípio de funcionamento é descrito a seguir:

Código III: Implementação da Máquina de Turing para Multiplicação

```
from automata.tm.ntm import NTM
# NTM que faz a operação de multiplicação.
ntm = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 'halt'},
    input_symbols={'0', '1', '*'},
    tape_symbols={'0', '1', '*', '#', 'E', 'Z'},
    transitions={
        's0': {
            '#': {('s0', '#', 'R')},
            '1': {('s1', '#', 'R')}
        },
        's1': {
            '*': {('s2', '1', 'R')},
            '1': {('s4', '1', 'R')}
        },
        's2': {
            'Z': {('s2', '1', 'R')},
            '1': {('s2', '1', 'R')},
            '#': {('s3', '#', 'L')}
        },
        's3': {
            '1': {('halt', '#', 'N')}
        },
        's4': {
            '1': {('s4', '1', 'R')},
            '*': {('s5', '*', 'R')}
        },
        's5': {
            '1': {('s6', 'E', 'R')},
            'Z': {('s8', 'Z', 'L')}
        },
```

```
        's6': {
            '1': {('s6', '1', 'R')},
            'Z': {('s6', 'Z', 'R')},
            '#': {('s7', 'Z', 'L')}
        },
        's7': {
            'Z': {('s7', 'Z', 'L')},
            '1': {('s7', '1', 'L')},
            'E': {('s5', 'E', 'R')}
        },
        's8': {
            'E': {('s8', '1', 'L')},
            '*': {('s9', '*', 'L')}
        },
        's9': {
            '1': {('s9', '1', 'L')},
            '#': {('s0', '#', 'R')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)
```

Função de Validação para verificar se a especificação está correta.

```
ntm.validate()  # returns True
```

```
True
```

```
ntm.read_input_stepwise('#11*111########')
```

```
<generator object NTM.read_input_stepwise at 0x7f7fdea0e350>
```

O método/função `read_input(palavra)` verifica se a *palavra* é aceita pela Máquina de Turing e retorna a configuração final para ela.

```
ntm.read_input('#11*111########').pop().print()
```

```
halt: ###111111######
                ^
```

```
if ntm.accepts_input('#111*111#########'):
    print('accepted')
else:
    print('rejected')
```

```
accepted
```

```
ntm.read_input_stepwise('#11111*11#################')
```

```
    <generator object NTM.read_input_stepwise at 0x7f7fdea0e970>
```

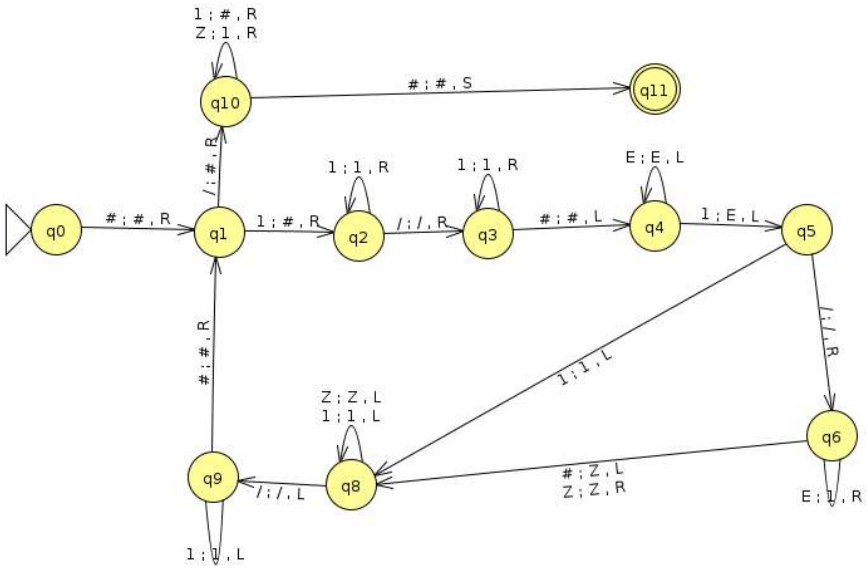
```
palavras = ['#1*1#########','#1*11#########','#1111*11#########']

for w in palavras:
  print("Verificando palavra:", w)
  if ntm.accepts_input(w):
    print('aceita')
  else:
    print('rejeitada')
  ntm.read_input(w).pop().print()
```

```
    Verificando palavra: #1*1#########
    aceita
    halt: ##1##########
              ^
    Verificando palavra: #1*11#########
    aceita
    halt: ##11##########
               ^
    Verificando palavra: #1111*11#########
    aceita
    halt: #####11111111####
                      ^
```

Divisão

Exemplo da máquina de Turing de DIVISÃO replicada no JFLAP:



Como podemos observar, essa máquina de Turing elaborada para a DIVISÃO possui 10 estados (q7 não existe nesse exemplo), sendo 1 (q0) inicial e 1 (q11) final. Seu princípio de funcionamento é descrito a seguir:

A MT identifica o início (e o final) da fita com o caraceter #, em seguida busca por termos '1' no dividendo, caso encontre vai até o final do divisor substituindo seu último carater '1' por 'E'.

Repetindo esse procedimento de "1 pra 1" até que todos os caracteres do divisor sejam 'E'.

Após isso, ela acrescenta um caracter 'Z' ao final, que se trata de um marcador para saber "quantas vezes o dividendo já foi dividido pelo divisor", ou seja, quantas vezes o divisor foi multiplicado até chegar no valor que está sendo dividido.

Feito isso, ela substitui todos os 'E' por '1' novamente, e retorna ao lado esquerdo do operador em busca de mais dígitos '1' para repetir o mesmo processo, até que os dígitos '1' à esquerda do operando se encerrem.

Para a validação do resultado, a MT volta substituindo todos os '1' e '/' por '#' e quando restam apenas caracteres 'Z', ela os substitui por 1, chegando deste modo ao resultado final truncado da divisão.

A implementação da MT pode ser vista no código IV.

Código IV: Implementação da Máquina de Turing para Divisão

```
from automata.tm.ntm import NTM
# NTM que faz a operação de divisão.
ntm = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 'halt'},
    input_symbols={'0', '1', '/'},
    tape_symbols={'0', '1', '/', '#', 'E', 'Z'},
    transitions={
        's0': {
            '#': {('s1', '#', 'R')}
        },
        's1': {
            '1': {('s2', '#', 'R')},
            '/': {('s9', '#', 'R')}
        },
        's2': {
            '1': {('s2', '1', 'R')},
            '/': {('s3', '/', 'R')}
        },
        's3': {
            '1': {('s3', '1', 'R')},
            'E': {('s3', 'E', 'R')},
            '#': {('s4', '#', 'L')},
            'Z': {('s4', 'Z', 'L')}
        },
        's4': {
            'E': {('s4', 'E', 'L')},
            '1': {('s5', 'E', 'L')}
        },
        's5': {
            '1': {('s7', '1', 'L')},
            '/': {('s6', '/', 'R')}
        },
        's6': {
            'E': {('s6', '1', 'R')},
```

```
            'Z': {('s6', 'Z', 'R')},
            '#': {('s7', 'Z', 'L')}
        },
        's7': {
            '1': {('s7', '1', 'L')},
            'Z': {('s7', 'Z', 'L')},
            '/': {('s8', '/', 'L')}
        },
        's8': {
            '1': {('s8', '1', 'L')},
            '#': {('s1', '#', 'R')}
        },
        's9': {
            'Z': {('s9', '1', 'R')},
            '1': {('s9', '#', 'R')},
            '#': {('halt', '#', 'N')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)
```

Função de Validação para verificar se a especificação está correta.

```
ntm.validate()  # returns True
```

```
True
```

```
ntm.read_input_stepwise('#1111/11###########')
```

```
<generator object NTM.read_input_stepwise at 0x7f7fdea0e120>
```

O método/função `read_input(palavra)` verifica se a *palavra* é aceita pela Máquina de Turing e retorna a configuração final para ela.

```
ntm.read_input('#1111/11###########').pop().print()
```

```
halt: ########11#########
                ^
```

```
if ntm.accepts_input('#111111-EE###############'):
    print('accepted')
else:
    print('rejected')
```

```
rejected
```

```
ntm.read_input_stepwise('#1111/11###########')
```

```
     <generator object NTM.read_input_stepwise at 0x7f7fdea0ec10>
```

```
palavras = ['#1111/11###########','#111111/111###########','#11/11###########']

for w in palavras:
  print("Verificando palavra:", w)
  if ntm.accepts_input(w):
    print('aceita')
  else:
    print('rejeitada')
  ntm.read_input(w).pop().print()
```

```
    Verificando palavra: #1111/11###########
    aceita
    halt: ########11#########
                  ^
    Verificando palavra: #111111/111###########
    aceita
    halt: ###########11#########
                     ^
    Verificando palavra: #11/11###########
    aceita
    halt: ######1##########
                ^
```

Referências

SIPSER, M. **Introdução à teoria da computação.** São Paulo: Cengage Learning, 2007. ISBN 9788522104994. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000008725&lang=pt-br&site=eds-live&scope=site. Acesso em: 2 jun. 2022.

MENEZES, P. B. **Linguagens formais e autômatos.** Porto Alegre: Bookman, 2011. ISBN 9788577807994. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000000444&lang=pt-br&site=eds-live&scope=site. Acesso em: 2 jun. 2022.

EZHILARASU, P. **Construction of a Basic Calculator through the Turing Machine – A Review.** Tamil Nadu, índia: International Journal of Engineering Trends and Applications (IJETA) – Volume 2 Issue 6, Nov-Dec 2015. Disponível em: http://www.ijetajournal.org/volume-2/issue-6/IJETA-V2I6P1.pdf. Acesso em: 16 jun. 2022.