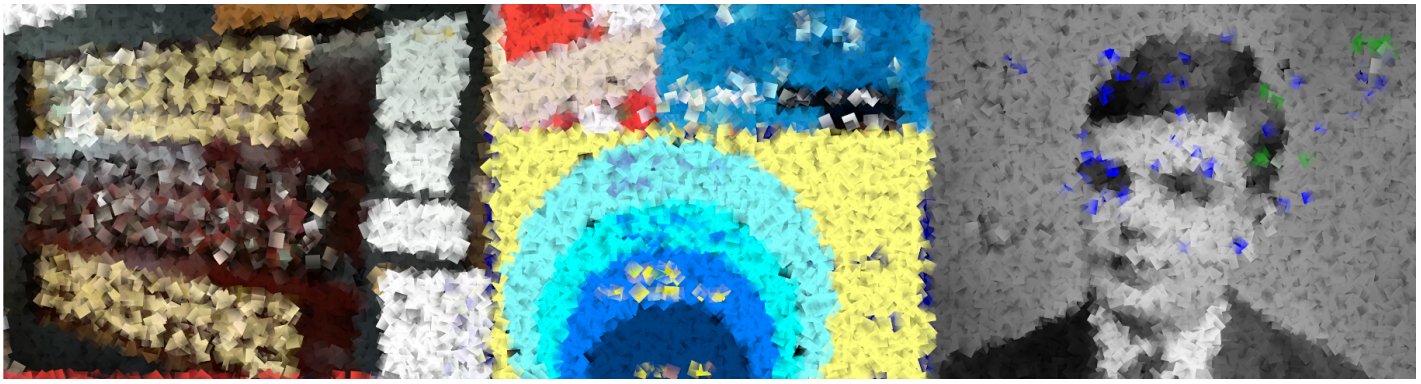
Tutorial Iterativo em Jupyter sobre Máquinas de Turing para Operações Aritméticas



Gregully Willian e William Reticena

e-mails: gregullywillian@alunos.utfpr.edu.br, williamreticena@alunos.utfpr.edu.br

Universidade Tecnológica Federal do Paraná (UTFPR)

Departamento Acadêmico de Computação (DACOM-CM)

Curso de Bacharelado em Ciência da Computação.

Resumo

Este trabalho tem como tema principal, conceitos de uma máquina de Turing. Estes conceitos são abordados com uma explicação do tema em questão e junto a ele, alguns exemplos para o seu melhor entendimento. É detalhado também, as ferramentas e bibliotecas utilizadas para a construção dos exemplos.

Introdução

Este trabalho tem como objetivo implementar quatro Máquinas de Turing, cada uma realizando uma operação aritmética básica, as operações necessárias que este projeto solicitava eram de adição, subtração, divisão e múltiplicação. Para a construção deste autômato, foi utilizada a linguagem de programação Python, e junto dela, uma biblioteca que trabalha com vários tipos de automatos, denominada de automata-lib. Foram usados alguns diagramas de estado, construídos com a ferramenta JFLAP para o auxilio da construção do mesmo.

Máquinas de Turing

O conceito de máquina de Turing é semelhante ao de uma fórmula ou equação. Assim, há uma infinidade de possíveis máquinas de Turing, cada uma correspondendo a um método definido ou algoritmo. Uma máquina de Turing pode fazer tudo que um computador de propósito geral

faz, contudo, ainda vai possuir algumas limitações. Uma máquina de Turing utiliza uma fita como memória, sendo essa fita infinita. A cabeça de fita pode ler e escrever símbolos sobre ela e movimentar-se. A fita encontra-se com uma entrada (programa) e possui todo o restante em branco, se a máquina precisar armazenar informações, ela o pode fazer ao escrever sobre a fita, enquanto para ler uma informação basta mover a cabeça para a posição onde se encontra a informação.

Preparação do Ambiente

Para a construção dos autômatos na linguagem python foi utilizado a biblioteca [automata-lib](), esta que auxilia na implementação de Máquinas de Estados e Autômatos.

```
!pip install ply
```

```
Collecting ply
  Downloading https://files.pythonhosted.org/packages/a3/58/35da89ee790598a0700ea49b
     |[redacted]|
51kB 3.9MB/s Installing collected packages: ply
Successfully installed ply-3.11
```

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

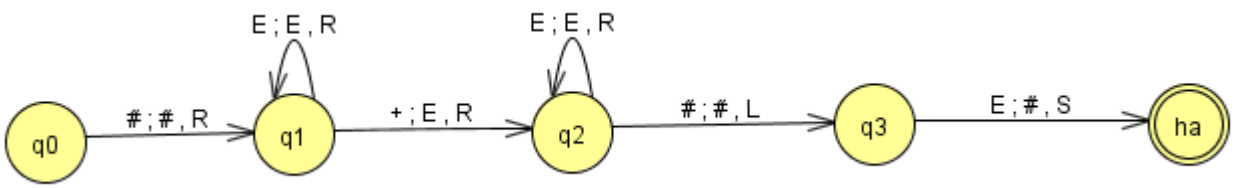
```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: automata-lib in /var/data/python/lib/python3.9/site-p
Requirement already satisfied: pydot in /var/data/python/lib/python3.9/site-packages
Requirement already satisfied: pyparsing>=2.1.4 in /var/data/python/lib/python3.9/si
```

Operações Aritméticas

Com base no artigo [Construction of a Basic Calculator through the Turing Machine – A Review](), no qual é apresentado os autômatos que realizam as quatro operações basicas (adição, subtração, multiplicação e divisão) e implementados tanto em liguagem python quanto no JFLAP como uma forma de validar o seu funcionamento.

Adição

Temos o seguinte autômato implementado no JFLAP para a resolução da operação de adição:



Para que seja possível executar a operação com tal máquina, temos que a fita deve estar no seguinte formato: "#E+E#", a máquina deve receber inicialmente o símbolo #, a partir deste ela irá ler várias sequências de E antes da entrada +, com isso basta ler os caracteres E seguintes e ir fazendo a soma destes, por fim, quando a máquina ler o símmbolo # a operação é encerrada.

Implementando o autômato em python temos o seguinte código:

```
from automata.tm.dtm import DTM

dtm_add = DTM(
  states={"q0", "q1", "q2", "q3", "ha"},
  input_symbols={"E", "+"},
  tape_symbols={"E", "#", "+"},
  transitions={
    "q0": {
      "#": ("q1", "#", "R")
    },
    "q1": {
      "E": ("q1", "E", "R"),
      "+": ("q2", "E", "R")
    },
    "q2": {
      "E": ("q2", "E", "R"),
      "#": ("q3", "#", "L")
    },
    "q3": {
      "E": ("ha", "#", "N")
    }
  },
  initial_state="q0",
  blank_symbol="#",
  final_states={"ha"}
)
```

Função de Validação para verificar se a especificação está correta.

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

```
     True
```

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

```
     <generator object DTM.read_input_stepwise at 0x000001F12C582B90>
```

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

```
dtm_add.read_input('#EEEEE+EE#').print()
```

```
     ha: #EEEEEEE##
```

```
                ^


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


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

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


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

Subtração

Este tópico trata-se da operação de subtração, este autômato aceita uma fita com a seguinte configuração, "#EEEEE-EE#", ele esta programado para receber uma quantidade de letras 'E's, onde esta quantidade representa um número inteiro. Para que uma operação de subtração seja validada com este autômato, a quantidade de argumentos (letras 'E') da fita a ser recebida deve estar entre cerquilhas - estas representam os limites direito e esquerdo da fita, respectivamente - e também para que a operação funcione, os argumentos devem estar separados pelo operador de subtração, por exemplo, "EE-E", onde esta operação numéricamente equivale a "2-1" e resultará em "1", ou como saída da fita depois da computação: "#E#".##E###

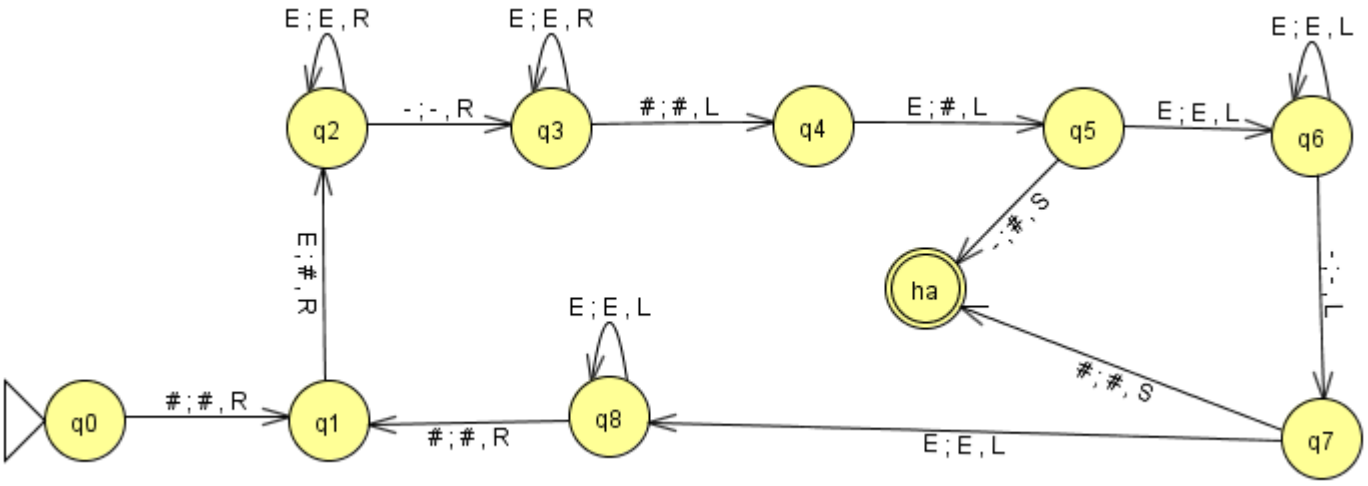


Figura 2: Máquina de Turing que faz Subtração

O seguinte código do autômato, foi implementado em Python:

```
from automata.tm.dtm import DTM
dtm_sub = DTM(
  states={"q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "ha"},
  input_symbols={"E", "-"},
  tape_symbols={"E", "#", "-"},
  transitions={ "q0": {
      "#": ("q1", "#", "R")
    },
    "q1": {
      "E": ("q2", "#", "R")
    },
    "q2": {
      "E": ("q2", "E", "R"),
      "-": ("q3", "-", "R")
    },
    "q3": {
      "E": ("q3", "E", "R"),
      "#": ("q4", "#", "L")
    },
    "q4": {
      "E": ("q5", "#", "L")
    },
    "q5": {
      "E": ("q6", "E", "L"),
      "-": ("ha", "#", "N")
    },
    "q6": {
      "E": ("q6", "E", "L"),
      "-": ("q7", "-", "L")
    },
    "q7": {
      "E": ("q8", "E", "L"),
      "#": ("ha", "#", "N")
    },
    "q8": {
      "E": ("q8", "E", "L"),
      "#": ("q1", "#", "R")
```

```
    }
  },
  initial_state="q0",
  blank_symbol="#",
  final_states={"ha"}
)
```

Função de Validação para verificar se a especificação está correta.

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

```
     True
```

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

```
     <generator object DTM.read_input_stepwise at 0x0000012FB6B66E30>
```

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

```
dtm_sub.read_input('#EEEEE-EE#').print()
```

```
     ha: ###EEE####
                ^
```

```
if dtm_sub.accepts_input('#EEEEE-EE#'):
    print('accepted')
else:
    print('rejected')
```

```
     accepted
```

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

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

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

^

Multiplicação

Este tópico trata-se da operação de multiplicação, este autômato aceita uma fita com a seguinte configuração, "#11111*11#", ele esta programado para receber uma quantidade de números '1's, onde esta quantidade representa um número inteiro. Para que uma operação de multiplicação seja validada com este autômato, a quantidade de argumentos (números '1') da fita a ser recebida deve estar entre cerquilhas - estas representam os limites direito e esquerdo da fita, respectivamente - e também para que a operação funcione, os argumentos devem estar separados pelo operador de multiplicação, por exemplo, "11*1", onde esta operação numéricamente equivale a "2*1" e resultará em "2", ou como saída da fita depois da computação: "##11###".

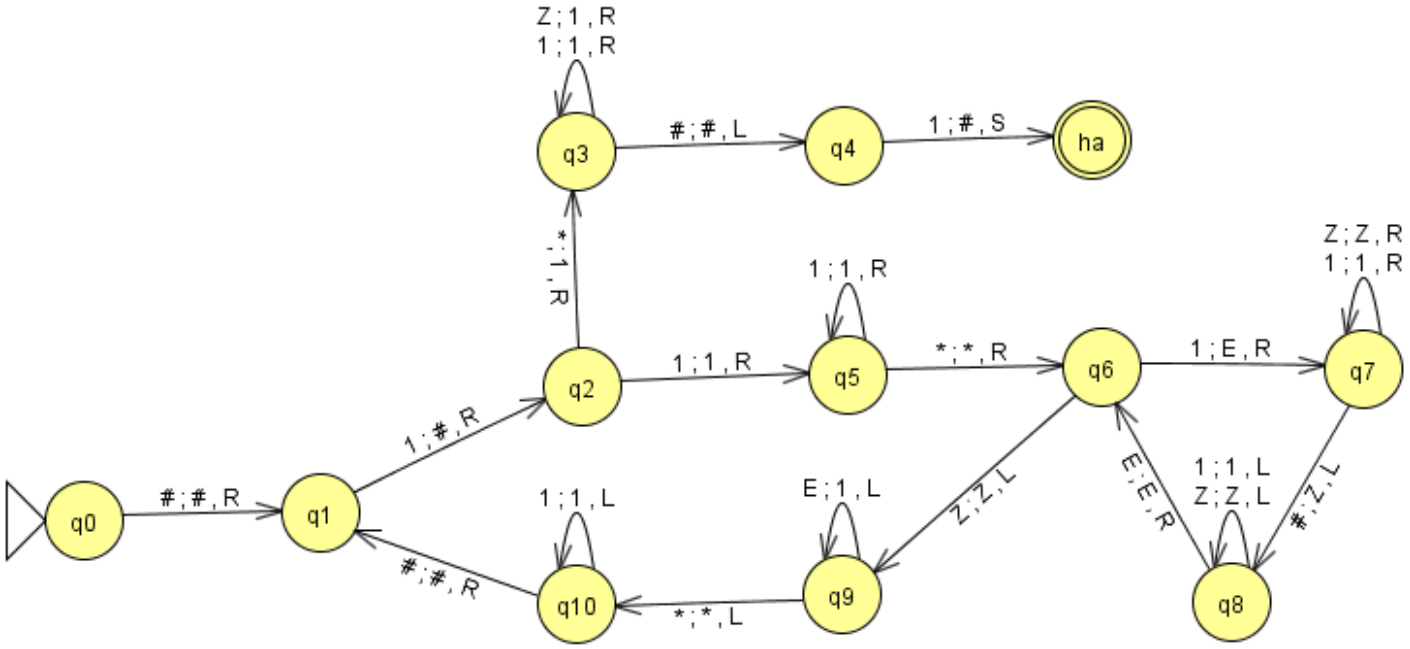


Figura 3:: Máquina de Turing que faz Multiplicação

O seguinte código do autômato, foi implementado em Python:

```
from automata.tm.dtm import DTM
dtm_mult = DTM(
  states={"q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "q10", "ha"},
  input_symbols={"1", "*"},
  tape_symbols={"E", "Z", "#", "*", "1"},
  transitions={
    "q0": {
      "#": ("q1", "#", "R")
    },
    "q1": {
      "1": ("q2", "#", "R")
    },
```

```
    "q2": {
      "*": ("q3", "1", "R"),
      "1": ("q5", "1", "R")
    },
    "q3": {
      "Z": ("q3", "1", "R"),
      "1": ("q3", "1", "R"),
      "#": ("q4", "#", "L")
    },
    "q4": {
      "1": ("ha", "#", "N")
    },
    "q5": {
      "1": ("q5", "1", "R"),
      "*": ("q6", "*", "R")
    },
    "q6": {
      "1": ("q7", "E", "R"),
      "Z": ("q9", "Z", "L")
    },
    "q7": {
      "Z": ("q7", "Z", "R"),
      "1": ("q7", "1", "R"),
      "#": ("q8", "Z", "L")
    },
    "q8": {
      "1": ("q8", "1", "L"),
      "Z": ("q8", "Z", "L"),
      "E": ("q6", "E", "R"),
    },
    "q9": {
      "E": ("q9", "1", "L"),
      "*": ("q10", "*", "L")
    },
    "q10": {
      "1": ("q10", "1", "L"),
      "#": ("q1", "#", "R")
    }
  },
  initial_state="q0",
  blank_symbol="#",
  final_states={"ha"}
)



dtm_mult.validate()  # returns True
```

```
True
```

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

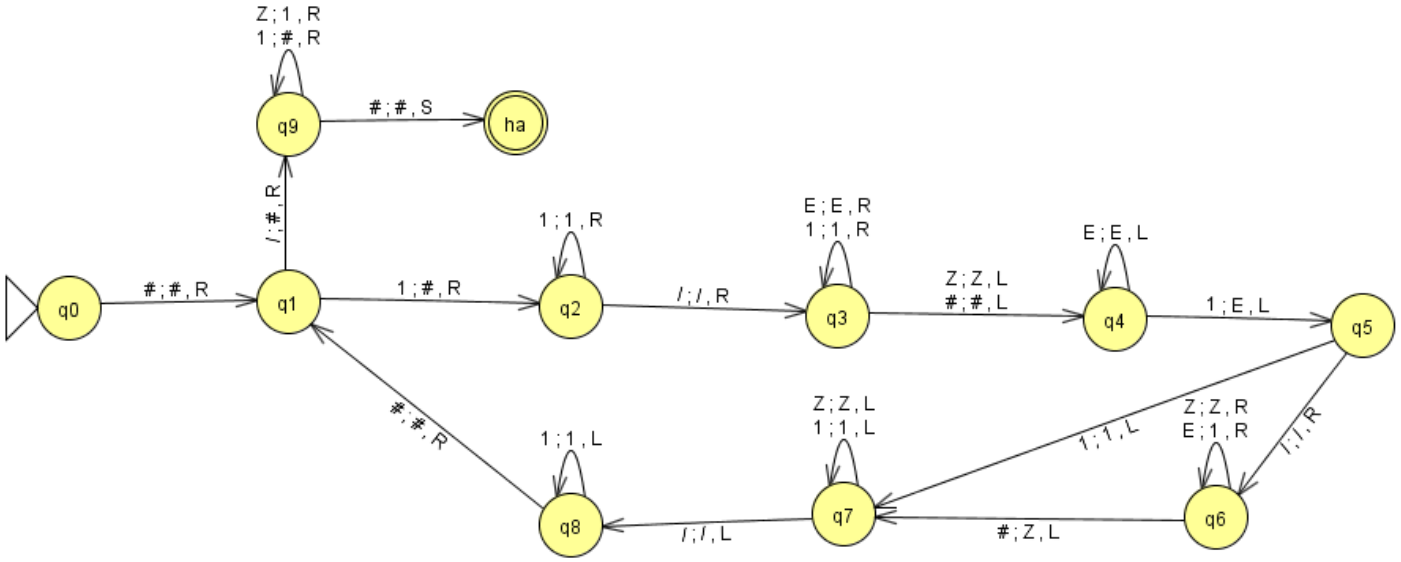
```
<generator object DTM.read_input_stepwise at 0x000002416157AB20>
```

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

```
dtm_mult.read_input('#11111*11#####').print()
     ha: ######1111111111##
                         ^


if dtm_mult.accepts_input('#11111*11#'):
    print('accepted')
else:
    print('rejected')

     accepted


palavras = ['#1*1#','#1*11#','#11111*11#']

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


     Verificando palavra: #1*1#
     aceita
     ha: ##1##
            ^
     Verificando palavra: #1*11#
     aceita
     ha: ##11##
             ^
     Verificando palavra: #11111*11#
     aceita
     ha: ######1111111111##
                         ^
```

Divisão

Temos o seguinte autômato implementado no JFLAP para a resolução da operação de divisão:



Para que seja possível executar a operação com tal máquina, temos que a fita deve estar no seguinte formato: "#1/1#####", onde a operação deve iniciar com uma cerquilha e termiar com quatro, além disso divisões que resultam em ponto flutuante não são reconhecidas por esta máquina.

Implementando o autômato em python temos o seguinte código:

```
from automata.tm.dtm import DTM
dtm_div = DTM(
  states={"q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "ha"},
  input_symbols={"1", "/"},
  tape_symbols={"E", "Z", "#", "/", "1"},
  transitions={
    "q0": {
      "#": ("q1", "#", "R")
    },
    "q1": {
      "1": ("q2", "#", "R"),
      "/": ("q9", "#", "R")
    },
    "q2": {
      "1": ("q2", "1", "R"),
      "/": ("q3", "/", "R")
    },
    "q3": {
      "1": ("q3", "1", "R"),
      "E": ("q3", "E", "R"),
      "#": ("q4", "#", "L"),
      "Z": ("q4", "Z", "L")
    },
    "q4": {
      "E": ("q4", "E", "L"),
      "1": ("q5", "E", "L")
      },
    "q5": {
      "1": ("q7", "1", "L"),
      "/": ("q6", "/", "R")
    },
```

```
    "q6": {
      "E": ("q6", "1", "R"),
      "Z": ("q6", "Z", "R"),
      "#": ("q7", "Z", "L")
    },
    "q7": {
      "Z": ("q7", "Z", "L"),
      "1": ("q7", "1", "L"),
      "/": ("q8", "/", "L")
    },
    "q8": {
      "1": ("q8", "1", "L"),
      "#": ("q1", "#", "R")
    },
    "q9": {
      "Z": ("q9", "1", "R"),
      "1": ("q9", "#", "R"),
      "#": ("ha", "#", "N")
    }
  },
  initial_state="q0",
  blank_symbol="#",
  final_states={"ha"}
)
```

Função de Validação para verificar se a especificação está correta.

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

```
     True
```

```
dtm_div.read_input_stepwise('#"#111111/11####')
```

```
     <generator object DTM.read_input_stepwise at 0x0000012FB87A0F90>
```

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

```
dtm_div.read_input("#111111/11####").print() #Este exemplo faz a divisão de 6 por 2, resul
```

```
if dtm_div.accepts_input('#111111/11####'):
    print('accepted')
else:
    print('rejected')
    accepted
```

```
palavras = ['#111111/11####','#1111111111/11####','#1111/1111####']

for w in palavras:
  print("Verificando palavra:", w)
```

```
if dtm_div.accepts_input(w):
  print('aceita')
else:
  print('rejeitada')
dtm_div.read_input(w).print()

   Verificando palavra: #111111/11####
   aceita
   ha: ##########111#
                  ^
   Verificando palavra: #1111111111/11####
   aceita
   ha: ##############11111#
                        ^
   Verificando palavra: #1111/1111####
   aceita
   ha: ##########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