maquina-turing-operacoes-aritmeticas

June 22, 2022

1 Tutorial Iterativo em Jupyter sobre Máquinas de Turing para Operações Aritméticas

Breno Farias da Silva e **Thaynara Ribeiro Falcão dos Santos**

e-mails: BrenoFarias@alunos.utfpr.edu.br, thaynararibeiro@alunos.utfpr.edu.br

Universidade Tecnológica Federal do Paraná (UTFPR)

Departamento Acadêmico de Computação (DACOM-CM)

Curso de Bacharelado em Ciência da Computação.

1.1 Resumo

O relatório está dividido em duas partes, a primeira reservada para dar uma introdução, de forma breve, ao conceito de Máquina de Turing, utilizado para desenvolver este trabalho, além da prepação de ambiente para a execução apropriada do projeto. Logo após, será discutido sobre as funcionalidades do projeto que conciste em como a Máquina de Turing realiza operações aritméticas, sendo elas a soma, subtração, multiplicação e divisão, assim como, a explicação do raciocínio desenvolvido nos trechos principais do algoritmo. Por fim, junto com as implementações feitas, será apresentado alguns testes para verificar a sua funcionalidade.

1.2 Introdução

Este trabalho possui como objetivo reproduzir os modelos de Máquinas de Turing para cada operação aritmética no JFLAP com a execução de testes para a verificação do funcionamento do código. Bem como, a implementação das Máquinas de Turing utilizando a linguagem Python com o uso da biblioteca automata-lib. A máquina de Turing é composta pela 7-upla dada por: M = (Q, E, Γ, Q0, F, QAceita, QRejeita,). Q é um conjunto finito de estados, E é um alfabeto finito de símbolos, Γ é o alfabeto da fita (conjunto finito de símbolos), Q0 é o estado inicial, QAceita é o conjunto dos estados finais, QRejeita é o conjunto dos estados não finais e é a função de transição.

2 Máquinas de Turing

A Máquina de Turing criada em 1936 por Alan Turing, possui o objetivo definir o que é ou não computavél. Ela é formada por três partes: fita, unidade de controle e função de transição. A fita é dividida em células que só armazenam um símbolo por vez, e também, é usada como dispositivo de entrada, saída e memória. Já a unidade de controle é constituída por uma unidade de leitura e gravação, podendo ser deslocada tanto para a direita, quanto para a esquerda, no decorrer da fita. A última parte, função de transição, comanda as leituras e gravações, o sentido e o estado da máquina.

2.1 Preparação do Ambiente

Inicialmente é necessário verificar se em sua máquina está instalado a linguagem de programação [Python] (https://www.tutorialsteacher.com/python/install-python).

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

Como iremos utilizar uma biblioteca externa, é necessário instalá-la usando o seguinte comando:

[ ]:
```
$ pip install automata-lib
```

Instalação do Jupyter no VSCode:

[ ]:
```
jupyter notebook: https://code.visualstudio.com/docs/datascience/
  ↪jupyter-notebooks
```

3 Operações Aritméticas

A seguir, será apresentado a implementação, concatenada a explicação da lógica utilizada em cada função da calculadora desenvolvida na Máquina de Turing.

3.1 Adição

Aa implementação da adição utilizando Máquina de Turing começa recebendo um input de entrada, por exemplo, #EEEEE+EE#. A partir disso, a máquina verifica se a entrada está correta, ou seja, se o primeiro símbolo é o símbolo branco "#". Em seguida, se a verificação for dada como verdadeira, a máquina de Turing começa a computar a string de entrada. O algoritmo segue com a leitura de todos os símbolos "E" até que o primeiro símbolo de adição "+" seja encontrado. Quando isso acontece, é feito a troca do símbolo de adição "+" pelo símbolo anterior "E". Após isso, a máquina de Turing segue lendo a fita para a direita, até encontrar o primeiro símbolo branco "#". Por fim, ao encontrar o símbolo branco, faz a troca reversa, ou seja, troca o último símbolo "E" pelo símbolo branco "#, indicando o novo final da string. Com isso, o resultado da string fica: #EEEEEEE##, estando a máquina no estado de aceitação ("halt).

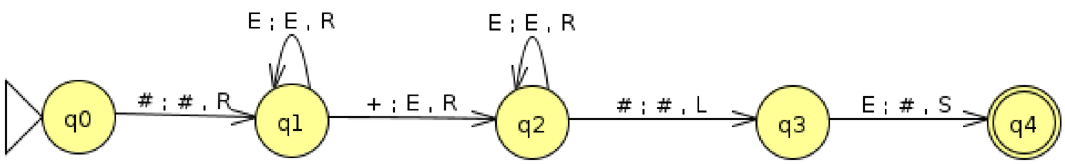


Figura 01: Máquina de Turing que faz Adição

A implementação da MT pode ser vista no código 01.

Código 01: Implementação da Máquina de Turing para Adição

[ ]:
```
from automata.tm.ntm import NTM

# NTM que faz a operação de adição.
ntmSUM = NTM(
    states={'s0', 's1', 's2', 's3', 'halt'},
    input_symbols={'1', '+'},
    tape_symbols={'1', '+', '#'},
    transitions={
        's0': {
            '#': {('s1', '#', 'R')}
        },
        's1': {
            '1': {('s1', '1', 'R')},
            '+': {('s2', '1', 'R')}
        },
        's2': {
```

```
            '1': {('s2', '1', 'R')},
            '#': {('s3', '#', 'L')}
        },
        's3': {
            '1': {('halt', '#', 'N')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)

def sumTest():
    print('TURING MACHINE - SUM:')

    sumInputs = [
        '#11111+11#',
        '#11111+111#',
        '#111+11#',
        '#1+11#',
        '#1111111+1111111#',
        '#11111+1#'
    ]

    for input in sumInputs:
        print('Validate Input:',  ntmSUM.validate()) # Valida o input
        if ntmSUM.accepts_input(input): # Se o input for aceitável, então o␣
 ↪autômato aceita
            ntmSUM.read_input(input).pop().print() # Lê o input e imprime o␣
 ↪resultado

def main():
    sumTest()

# Execução do programa
if __name__ == '__main__':
    main()
```

```
Running cells with 'Python 3.9.7 64-bit' requires ipykernel package.


Run the following command to install 'ipykernel' into the Python environment.

Command: '"c:/Program Files/Python39/python.exe" -m pip install ipykernel -U␣
 ↪--user --force-reinstall'
```

Função de Validação para verificar se a especificação está correta.

[17]:
```python
ntmSum.validate()  # returns True
```

[17]:
```
True
```

[18]:
```python
ntmSum.read_input_stepwise('#EEEEE+EE#')
```

[18]:
```
<generator object NTM.read_input_stepwise at 0x7f7d881ac270>
```

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

[22]:
```python
ntmSum.read_input('#EEEEE+EE#')
```

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

[19]:
```python
if ntmSum.accepts_input('#EEEEE+EE#'):
    print('accepted')
else:
    print('rejected')
```

```
accepted
```

[32]:
```python
ntmSum.read_input_stepwise('#EEEEE+EE#')
```

[32]:
```
<generator object NTM.read_input_stepwise at 0x7f7d8810e120>
```

[30]:
```python
def sumTest(): # Verifica as palavras de entrada
    print('TURING MACHINE - SUM:')

    sumInputs = [
        '#11111+11#',
        '#11111+111#',
        '#111+11#',
        '#1+11#',
        '#1111111+1111111#',
        '#11111+1#'
    ]

    for input in sumInputs:
        print('Validate Input:',  ntmSUM.validate()) # Valida o input
        if ntmSUM.accepts_input(input): # Se o input for aceitável, então o autômato aceita
            ntmSUM.read_input(input).pop().print() # Lê o input e imprime o resultado
```

```
Verificando palavra: #E+E#
aceita
halt: #EE##
         ^
```

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

3.2 Subtração

Para realizar a opreção de subtração a máquina recebe um input, por exemplo, #EE-EE#. A partir disso, a máquina verifica se a entrada está correta, ou seja, se o primeiro símbolo é o símbolo branco "#". Em seguida, se a verificação for dada como verdadeira, a Máquina de Turing começa a computar a string de entrada. O algoritmo seguido é dado pela leitura do primeiro símbolo "E" até que o primeiro símbolo de subtração "-" seja encontrado. Quando isso acontece, a Máquina de Turing vai caminhando para a direita na fita até o último símbolo "E", após o símbolo de subtração seja encontrado. Depois disso, a Máquina de Turing segue movendo a fita para a esquerda até encontrar o primeiro simbolo branco "#" do lado esquerdo do símbolo de subtração. Após isso, o algoritmo segue seguindo a lógica de, para cada símbolo "E" retirado na esquerda, retira-se outro símbolo "E" na direita e substitui pelo símbolo "#". Após todos os símbolos "E" da esquerda terem sido retirados, o resultado será o número de símbolos "E" contidos na string final.

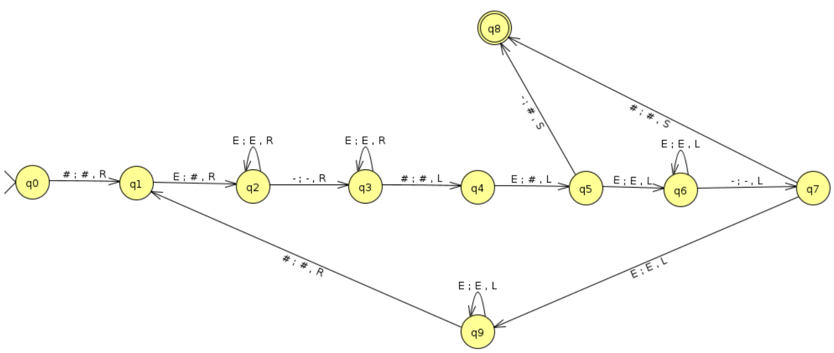


Figura 02: Máquina de Turing que faz Subtração

A implementação da MT pode ser vista no código 02.

[ ]:
```python
from automata.tm.ntm import NTM

# NMT que faz a operação de subtração.
ntmSUBTRACTION = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 'halt'},
    input_symbols={'1', '-'},
    tape_symbols={'1', '-', '#'},
    transitions={
        's0': {
```

```
            '#': {('s1', '#', 'R')}
        },
        's1': {
            '1': {('s2', '#', 'R')}
        },
        's2': {
            '1': {('s2', '1', 'R')},
            '-': {('s3', '-', 'R')}
        },
        's3': {
            '1': {('s3', '1', 'R')},
            '#': {('s4', '#', 'L')}
        },
        's4': {
            '1': {('s5', '#', 'L')}
        },
        's5': {
            '1': {('s6', '1', 'L')},
            '-': {('halt', '#', 'N')}
        },
        's6': {
            '1': {('s6', '1', 'L')},
            '-': {('s7', '-', 'L')},
        },
        's7': {
            '1': {('s8', '1', 'L')},
            '#': {('halt', '#', 'N')}
        },
        's8': {
            '1': {('s8', '1', 'L')},
            '#': {('s1', '#', 'R')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)

def subtractionTest():
    print('TURING MACHINE - SUBTRACTION:')

    subtractionInputs = [
        '#11111-11#',
        '#11111-111#',
        '#111-11#',
        '#1-1#',
        '#11111111-1111111#',
```

```
        '#11111-1#'
    ]

    for input in subtractionInputs:
        print('Validate Input:',  ntmSUBTRACTION.validate()) # Valida o input
        if ntmSUBTRACTION.accepts_input(input): # Se o input for aceitável,␣
 ↪então o autômato aceita
            ntmSUBTRACTION.read_input(input).pop().print() # Lê o input e␣
 ↪imprime o resultado

def main():
    subtractionTest()

# Execução do programa
if __name__ == '__main__':
    main()
```

Função de Validação para verificar se a especificação está correta.

[ ]:
```
ntmSubtraction.validate()  # returns True
```

True

[ ]:
```
ntmSubtraction.read_input_stepwise('#EE-EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d881ac270>

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

[ ]:
```
ntmSubtraction.read_input('#EE-EE#')
```

halt: #######

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

accepted

[ ]:
```
ntmSubtraction.read_input_stepwise('#EEEEE-EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d881ac270>

[ ]:
```
def subtractionTest():
    print('TURING MACHINE - SUBTRACTION:')

    subtractionInputs = [
        '#11111-11#',
        '#11111-111#',
```

```
        '#111-11#',
        '#1-1#',
        '#11111111-1111111#',
        '#11111-1#'
    ]

    for input in subtractionInputs:
        print('Validate Input:',  ntmSUBTRACTION.validate()) # Valida o input
        if ntmSUBTRACTION.accepts_input(input): # Se o input for aceitável,␣
↪então o autômato aceita
            ntmSUBTRACTION.read_input(input).pop().print() # Lê o input e␣
↪imprime o resultado
```

Verificando palavra: #E-E#

aceita

halt: ##### ^

Verificando palavra: #EE-E#

aceita

halt: #E#### ^

Verificando palavra: #EEEEE-EE#

aceita

halt: #EEE#####

3.3 Multiplicação

AO algoritmos para realizar a operação de multiplicação na Máquina de Turing começa com a máquina recebendo um input, por exemplo, #11*111####. A partir disso, a máquina verifica se a entrada está correta, ou seja, se o primeiro símbolo é o símbolo branco "#". Em seguida, se a verificação for dada como verdadeira, a Máquina de Turing começa a computar a string de entrada. O algoritmo seguido é dado pela leitura do primeiro símbolo "1". Quando esse símbolo é detectado, a Máquina percorre a fita toda para a direita até encontrar o símbolo "1" após o símbolo da operação de multiplicação "x". Dado isso, a máquina de Turing escreve "E" em todos os símbolos "1" após o símbolo de multiplicação. Após isso, volta novamente para a esquerda até encontrar o primeiro símbolo branco "#" que encontra-se a esquerda do símbolo de multiplicação. Sendo assim, novamente repete-se os processos anteriores de, para cada símbolo "1" do lado esquerdo, ele escrever novos símbolos do lado direito. Por fim, após a string não conter símbolos "1", ela troca todos os símbolos na string que não são o símbolo branco "#" e trocam por "1". Dessa forma, o resultado é dado pela quantidade de símbolos "1" contidos na string.

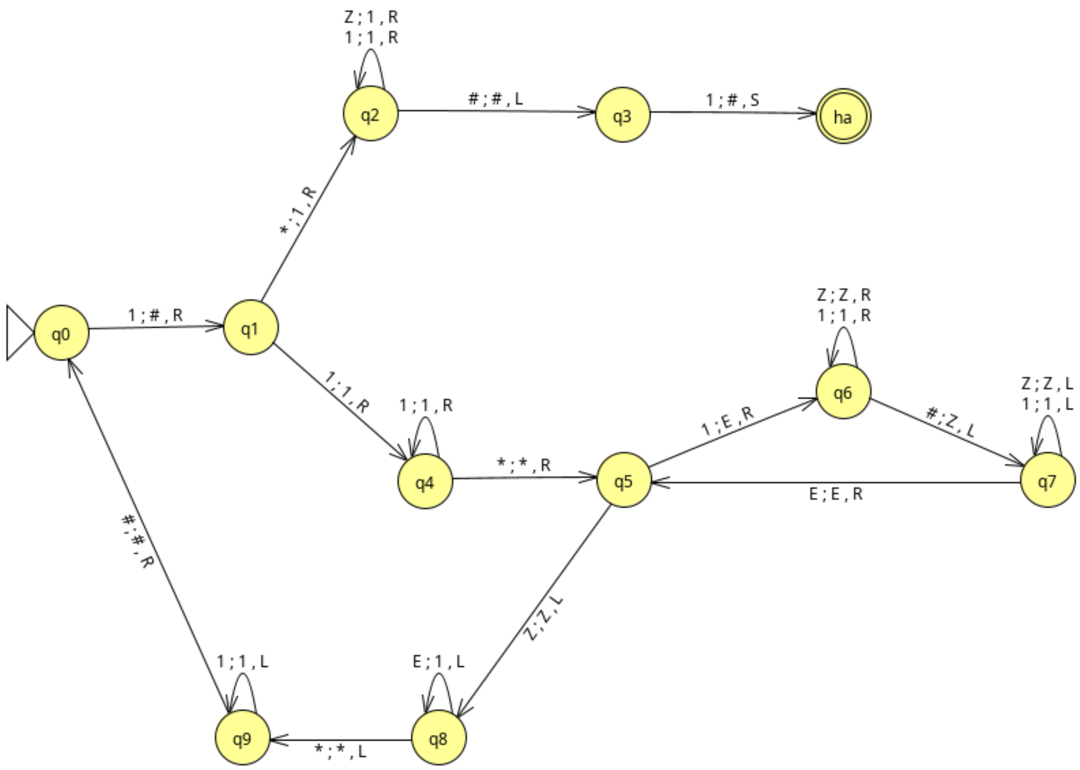


Figura 03: Máquina de Turing que faz Multiplicação

A implementação da MT pode ser vista no código 03.

```
[ ]: from automata.tm.ntm import NTM

# NTM que faz a operação de multiplicação.
ntmMULTIPLICATION = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9' ,'halt'},
    input_symbols={'1', '*'},
    tape_symbols={'1', '*', '#', 'Z', 'E'},
    transitions={
        's0': {
            '1': {('s1', '#', 'R')},
            '#': {('s0', '#', 'R')}
        },
        's1': {
            '1': {('s4', '1', 'R')},
            '*': {('s2', '1', 'R')}
        },
        's2': {
            '1': {('s2', '1', 'R')},
            'Z': {('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': {
            '1': {('s7', '1', 'L')},
            'E': {('s5', 'E', 'R')},
            'Z': {('s7', 'Z', 'L')}
        },
        's8': {
            'E': {('s8', '1', 'L')},
            '*': {('s9', '*', 'L')}
        },
        's9': {
            '1': {('s9', '1', 'L')},
            '#': {('s0', '#', 'R')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)

def multiplicationTest():
    print('TURING MACHINE - MULTIPLICATION:')

    multiplicationInputs = [
            '#11111*11####',
            '#11111*111####',
            '#11*11####',
            '#1*1####',
            '#11111*111111####',
            '#1111*1####'
    ]

    for input in multiplicationInputs:
```

```
        print('Validate Input:',  ntmMULTIPLICATION.validate()) # Valida o input
        if ntmMULTIPLICATION.accepts_input(input): # Se o input for aceitável,␣
 ↪então o autômato aceita
            ntmMULTIPLICATION.read_input(input).pop().print() # Lê o input e␣
 ↪imprime o resultado

def main():
    multiplicationTest()

# Execução do programa
if __name__ == '__main__':
    main()
```

Função de Validação para verificar se a especificação está correta.

[ ]:
```
ntmMULTIPLICATION.validate()  # returns True
```

True

[ ]:
```
ntmMULTIPLICATION.read_input_stepwise('#EEEEE*EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d881ac270>

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

[ ]:
```
ntmMULTIPLICATION.read_input('#EEE*EE#')
```

halt: #EEEEEEE## ^

[ ]:
```
if ntmMULTIPLICATION.accepts_input('#EEE*EE#'):
    print('accepted')
else:
    print('rejected')
```

accepted

[ ]:
```
ntmMULTIPLICATION.read_input_stepwise('#EEE*EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d881ac270>

[ ]:
```
def multiplicationTest():
    print('TURING MACHINE - MULTIPLICATION:')

    multiplicationInputs = [
            '#11111*11####',
            '#11111*111####',
            '#11*11####',
            '#1*1####',
            '#11111*111111####',
            '#1111*1####'
```

```
]

for input in multiplicationInputs:
    print('Validate Input:',  ntmMULTIPLICATION.validate()) # Valida o input
    if ntmMULTIPLICATION.accepts_input(input): # Se o input for aceitável,␣
↪então o autômato aceita
        ntmMULTIPLICATION.read_input(input).pop().print() # Lê o input e␣
↪imprime o resultado
```

3.4 Divisão

Por fim, a operação de divisão é feita pela máquina recebendo um input, por exemplo, # 1111÷11####. A partir disso, a máquina verifica se a entrada está correta, ou seja, se o primeiro símbolo é o símbolo branco "#". Em seguida, se a verificação for dada como verdadeira, a Máquina de Turing começa a computar a string de entrada. O algoritmo seguido é dado pela leitura do primeiro símbolo "1". Quando esse símbolo é detectado, a Máquina percorre a fita toda para a direita até encontrar o último símbolo "E" após o símbolo da operação de divisão "/". Esse processo se repete até que todos os símbolos "1" do lado direito do símbolo de divisão "/" tenham sido trocados. Após isso, todos os símbolos "E" que estão do lado direito do símbolo de divisão "/" serão trocados de volta para "1". Agora, temos que colocar o símbolo "Z" após o último símbolo "1" do lado direito do símbolo de divisão "/". Logo após, vamos percorrer a fita toda para a esquerda até encontrar o primeiro símbolo "1". Agora vamos novamente trocar o símbolo "1" pelo símbolo branco "#' e percorrer a fita toda para a direita até encontrar o último símbolo"1". Quando isso acontecer, trocamos esse símbolo "1" pelo símbolo "E" e voltamos novamente a fita toda para a esquerda e substituímos o "1" restante pelo símbolo branco "#". Do lado direito também trocamos o "1" restante pelo símbolo "E", como feito anteriormente. Por fim, trocamos o símbolo de divisão "/" pelo símbolo branco "#" e adicionamos, do lado direito, após o último símbolo "Z", o símbolo "Z" novamente. A string resultante é dada por "######11ZZ##" Dessa forma, o resultado é dado pela quantidade de símbolos "1" contidos na string. Os símbolos "1" serão trocados pelo símbolo branco "#" e os símbolos "Z" serão trocados pelo símbolo "1", resultando em "########11##" com a Máquina de Turing em um estado de aceitação("halt").

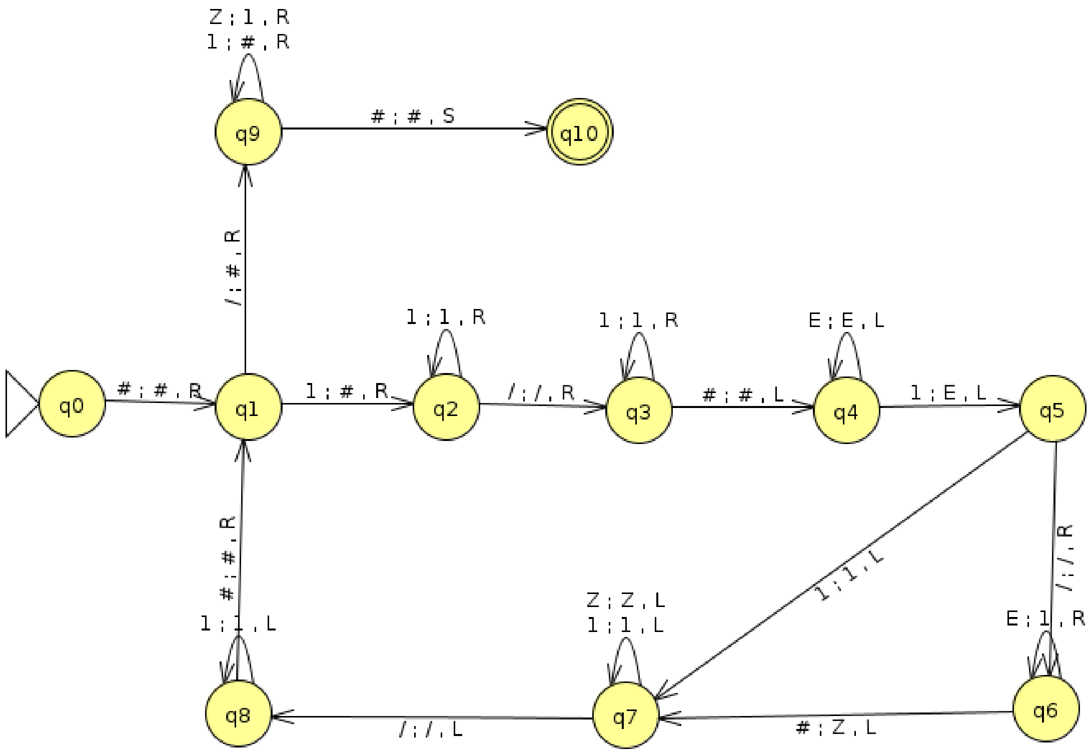
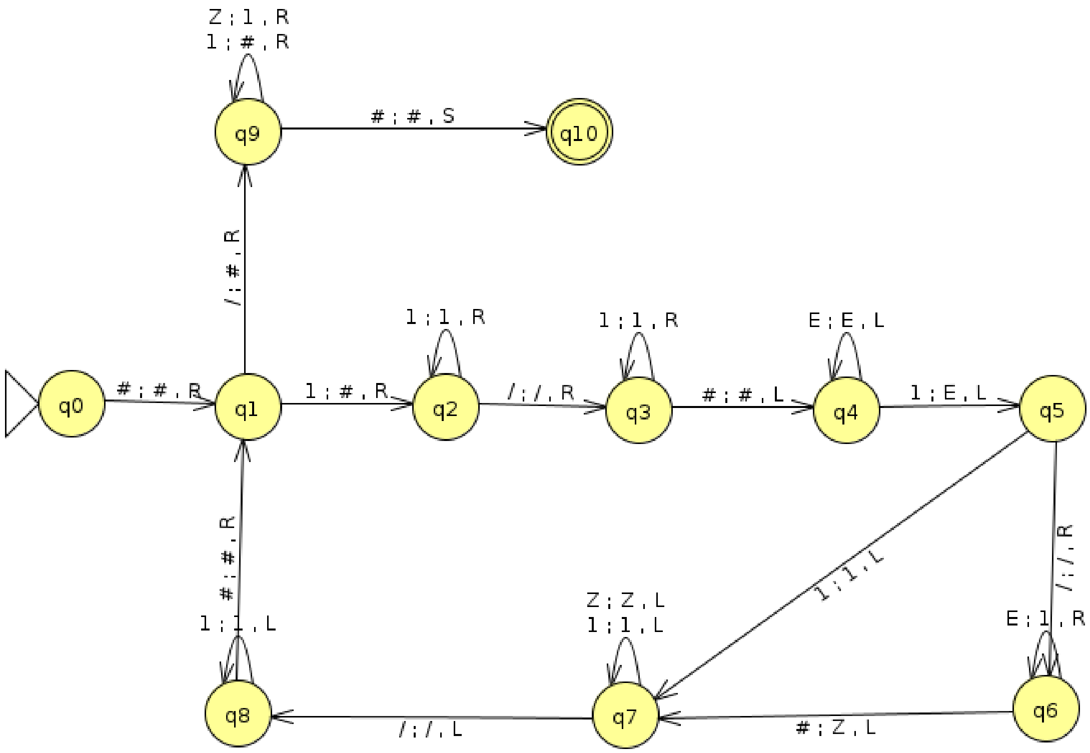


Figura 04: Máquina de Turing que faz Divisão

A implementação da MT pode ser vista no código 04.



[ ]:
```
from automata.tm.ntm import NTM

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

```python
        },
        's4': {
            '1': {('s5', 'E', 'L')},
            'E': {('s4', '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': {
            '1': {('s9', '#', 'R')},
            'Z': {('s9', '1', 'R')},
            '#': {('halt', '#', 'N')}
        }
    },
    initial_state='s0',
    blank_symbol='#',
    final_states={'halt'}
)

def divisionTest():
    print('TURING MACHINE - DIVISION:')

    divisionInputs = [
        '#1111/11####',
        '#111111/11####',
        '#11/11####',
        '#11111111/1111####',
        '#111111/11####',
    ]

    for input in divisionInputs:
        print('Validate Input:',  ntmDIVISION.validate()) # Valida o input
```

```
        if ntmDIVISION.accepts_input(input): # Se o input for aceitável, então
 ↪o autômato aceita
            ntmDIVISION.read_input(input).pop().print() # Lê o input e imprime
 ↪o resultado

def main():
    divisionTest()

# Execução do programa
if __name__ == '__main__':
    main()
```

Função de Validação para verificar se a especificação está correta.

[ ]:
```
ntmDIVISION.validate()  # returns True
```

True

[ ]:
```
ntmDIVISION.read_input_stepwise('#EEEEE+EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d881ac270>

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

[ ]:
```
ntmDIVISION.read_input('#EEEEE+EE#')
```

halt: #EEEEEEE## ^

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

accepted

[ ]:
```
ntmDIVISION.read_input_stepwise('#EEEEE+EE#')
```

<generator object NTM.read_input_stepwise at 0x7f7d8810e120>

[ ]:
```
def divisionTest():
    print('TURING MACHINE - DIVISION:')

    divisionInputs = [
        '#1111/11####',
        '#111111/11####',
        '#11/11####',
        '#11111111/1111####',
        '#111111/11####',
    ]
```

```
    for input in divisionInputs:
        print('Validate Input:',  ntmDIVISION.validate()) # Valida o input
        if ntmDIVISION.accepts_input(input): # Se o input for aceitável, então␣
↪o autômato aceita
            ntmDIVISION.read_input(input).pop().print() # Lê o input e imprime␣
↪o resultado
```

3.5 Referências

EZHILARASU P. **Construction of a Basic Calculator through the Turing Machine – A Review** India: IJETA, 2015. Disponpivel em: http://www.ijetajournal.org/volume-2/issue-6/IJETA-V2I6P1.pdf. Acesso em: 21 jun. 2022

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.