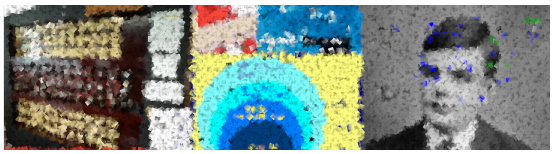
Máquinas de Turing para Operações Aritméticas



Alunos:

- Felipe Archanjo da Cunha Mendes, RA: 2252740
- Pamella Lissa Sato Tamura, RA: 2254107

e-mails:

- felipemendes.1999@alunos.utfpr.edu.br
- pamellalissa@alunos.utfpr.edu.br

Universidade Tecnológica Federal do Paraná (UTFPR)

Departamento Acadêmico de Computação (DACOM-CM)

Curso de Bacharelado em Ciência da Computação.

Resumo

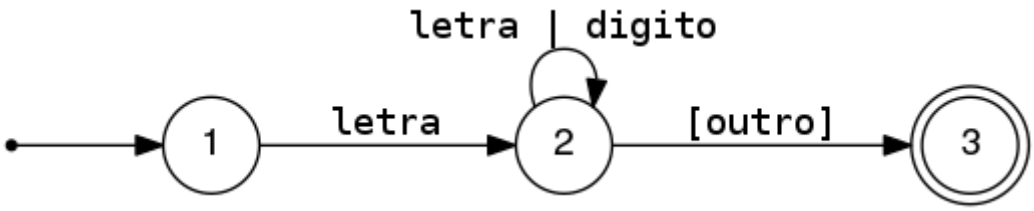
Máquinas de Turing são considerados um modelo computacional amplamente utilizando na Ciência da Computação. O artigo Construction of a Basic Calculator through the Turing Machine – A Review traz modelos de Máquinas de Turing que executam as quatro operações aritméticas (adição, subtração, multiplicação e divisão). A ideia do trabalho é que sejam reproduzidos os modelos de Máquinas de Turing para cada operação aritmética no JFLAP. Com execução de testes para verificarmos o funcionamento. A implementação das Máquinas de Turing também é para ser feita em Python com o uso da biblioteca automata-lib.

Introdução

Uma máquina de Turing é um modelo matemático de computação que descreve uma máquina abstrata que manipula símbolos em uma fita de acordo com uma tabela de regras. Apesar da simplicidade do modelo, ele é capaz de implementar qualquer algoritmo de computador.

A máquina opera em uma fita de memória infinita dividida em células discretas, cada uma das quais pode conter um único símbolo extraído de um conjunto finito de símbolos chamado alfabeto da máquina. Possui uma "cabeça" que, em qualquer ponto da operação da máquina, é posicionada sobre uma dessas células, e um "estado" selecionado a partir de um conjunto finito de estados. A cada passo de sua operação, a cabeça lê o símbolo em sua célula. Então, com base no símbolo e no estado atual da própria máquina, a máquina escreve um símbolo na mesma célula e move a cabeça um passo para a esquerda ou para a direita, ou interrompe a

computação. A escolha de qual símbolo de substituição escrever e qual direção mover é baseada em uma tabela finita que especifica o que fazer para cada combinação do estado atual e o símbolo que é lido.



Máquinas de Turing

Com o auxilios das tabelas de estados do artigo [Construction of a Basic Calculator through the Turing Machine – A Review] e com o auxilio da biblioteca automat-lib é possivel criar as de turing para a realização das principais operações aritmeticas matematicas.

Preparação do Ambiente

Para a realização desse projeto foi necessario a instalação apenas da biblioteca [automata-lib].

```
# Instalando a biblioteca
!pip3 install automata-lib
```

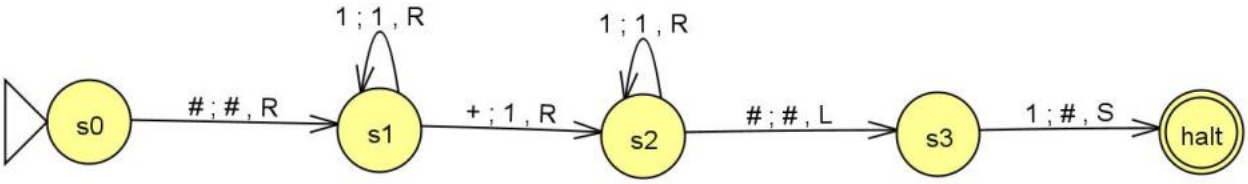
```
Requirement already satisfied: automata-lib in c:\users\usuario\anaconda3\lib\site-pa
Requirement already satisfied: pydot in c:\users\usuario\anaconda3\lib\site-packages
Requirement already satisfied: pyparsing>=2.1.4 in c:\users\usuario\anaconda3\lib\sit
```

```
# Importando a classe NTM da biblioteca automata
from automata.tm.ntm import NTM
```

Operações Aritméticas

Adição

Criação da MT de soma no JFLAP:



Processo de criação do algoritmo da MT de soma:

Utilizando a classe NTM, instanciamos uma Máquina de Turing com os seguintes parametros:

- Estados: s0, s1, s2, s3 e halt
- Simbolos de entrada: 1 e +
- Simbolos da fita: 1, + e #
- Transições: Conjunto das funções de transição
- Estado inicial: s0
- Simbolo em branco na fita: #
- Estado final: halt

A maquina de turing que realiza as operações de soma conseguem ler fitas da seguinte formatação:

- **#11+1#**
- **#1111+11#**
- **#1+1**
- **#111111+1**

Os valores dos lados direito e esquerdo da soma são representados por 1's, sendo que "111" representa o valor 1+1+1 = 3. As #'s representam os espaços vazios, e devem ser utilizados sempre uma no inicio e outra no final da expressão. Com isso, a expressão #11+1#= 2 + 1 = 3 = 111

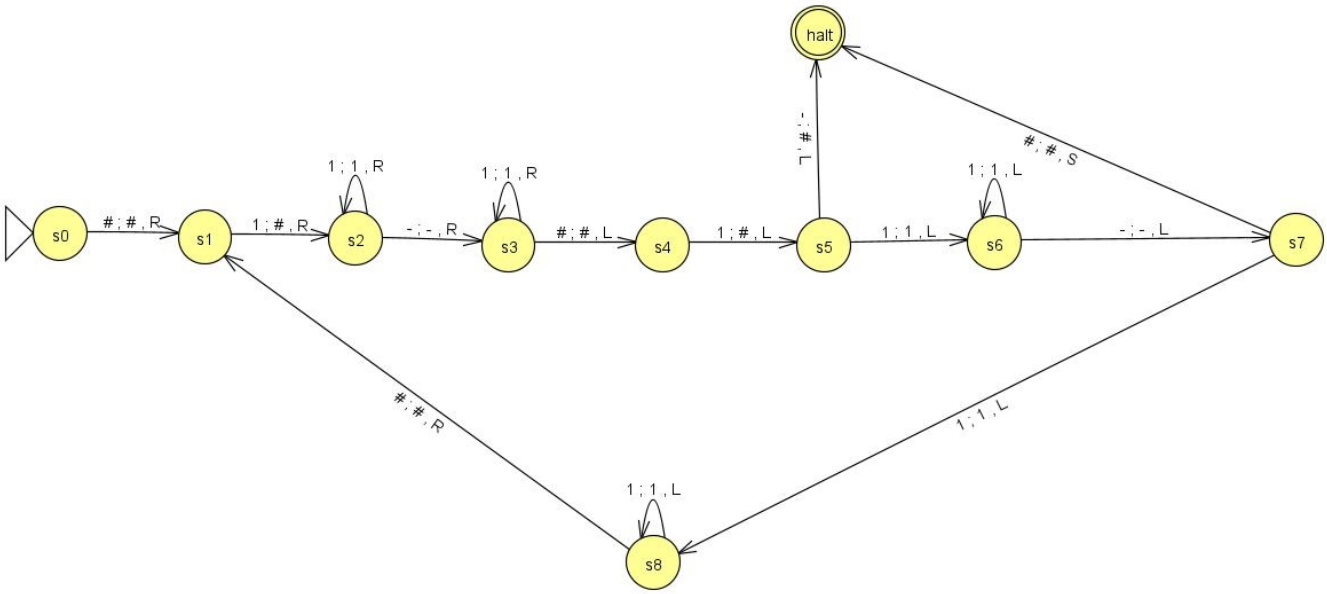
```
# Maquina de turing para soma
soma = NTM(
    states={'s0', 's1', 's2', 's3', 'halt'},
    input_symbols={'1', '+'},
    tape_symbols={'1', '+', '#'},
    transitions={
        's0': {
            '#': {('s1', '#', 'R')}
```

```
        },
        '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'}
)


# Lista com as operações de soma
op_soma = [
    '#11111+11#',
    '#11111+111#',
    '#111+11#',
    '#1+11#',
    '#11111111+1111111#',
    '#11111+1#'
]

# Exibição do resultado das somas
print("------ Resultado das somas ------")
for op in op_soma:
    if soma.accepts_input(op):
        soma.read_input(op).pop().print()
```

```
 ------ Resultado das somas ------
 halt: #1111111##
              ^
 halt: #11111111##
               ^
 halt: #11111##
            ^
 halt: #111##
          ^
 halt: #111111111111111##
                       ^
 halt: #111111##
              ^
```

Subtração

Criação da MT de subtração no JFLAP:



Processo de criação do algoritmo da MT de subtração:

Utilizando a classe NTM, instanciamos uma Máquina de Turing com os seguintes parametros:

- Estados: s0, s1, s2, s3, s4, s5, s6, s7, s8 e halt
- Simbolos de entrada: 1 e -
- Simbolos da fita: 1, - e #
- Transições: Conjunto das funções de transição
- Estado inicial: s0
- Simbolo em branco na fita: #
- Estado final: halt

A maquina de turing que realiza as operações de subtração conseguem ler fitas da seguinte formatação:

- **#11-1#**
- **#1111-11#**
- **#1-1**
- **#111111-1**

Os valores dos lados direito e esquerdo da soma são representados por 1's, sendo que "111" representa o valor 1+1+1 = 3. As #'s representam os espaços vazios, e devem ser utilizados sempre uma no inicio e outra no final da expressão. Com isso, a expressão #111-1# = 3 - 1 = 2 = 11

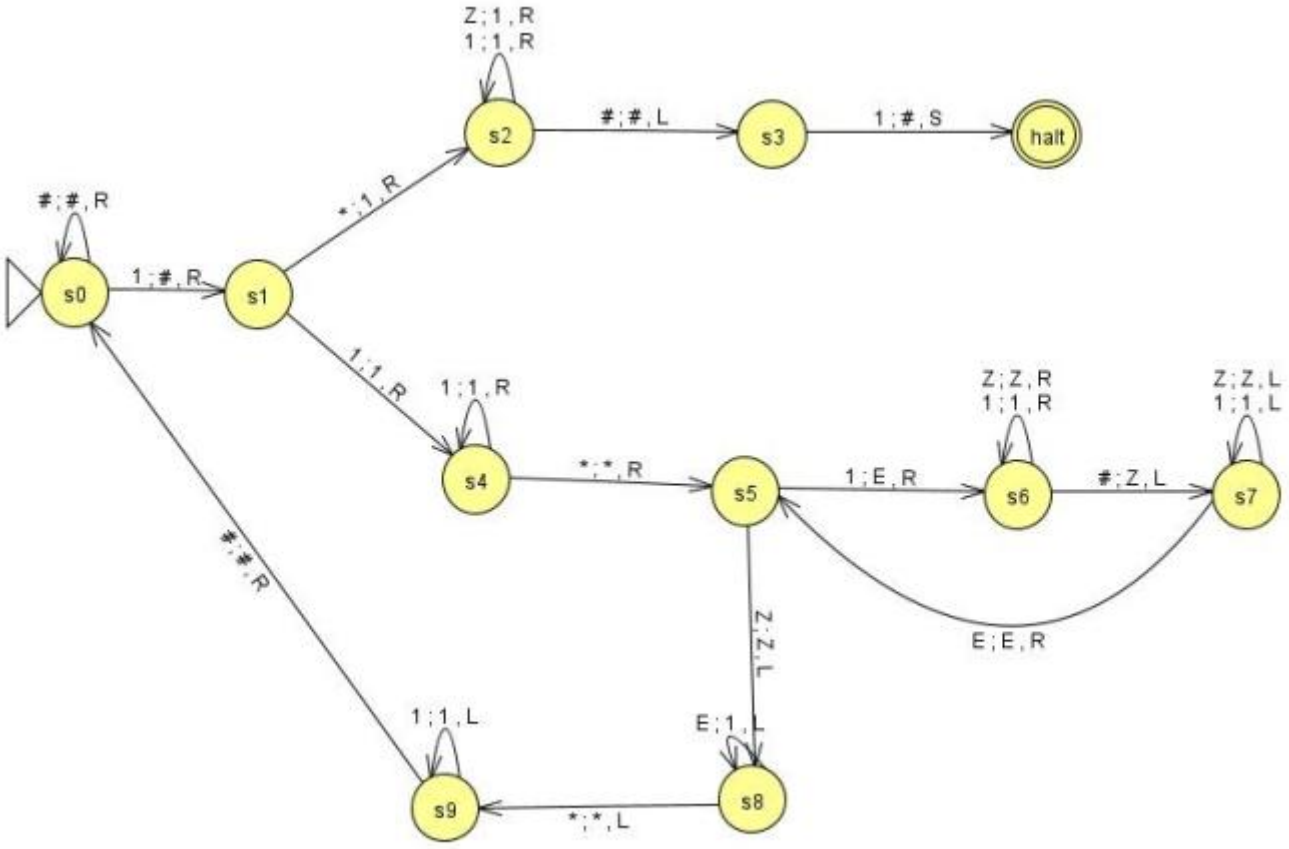
```
# Maquina de turing para subtracao
subtracao = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 'halt'},
    input_symbols={'1', '-'},
    tape_symbols={'1', '-', '#'},
    transitions={
```

```
            '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'}
    )


    # Lista com as operações de soma
    op_subtracao = [
        '#11111-11#',
        '#11111-111#',
        '#111-11#',
        '#1-1#',
        '#11111111-1111111#',
        '#11111-1#'
    ]

    # Exibição do resultado das somas
    print("------ Resultado das subtrações ------")
    for op in op_subtracao:
```

```
if subtracao.accepts_input(op):
    subtracao.read_input(op).pop().print()

------ Resultado das subtrações ------
halt: ###111####
           ^
halt: ####11#####
           ^
halt: ###1####
          ^
halt: #####
        ^
halt: ########1#########
              ^
halt: ##1111###
```

Multiplicação

Criação da MT de multiplicação no JFLAP:



Processo de criação do algoritmo da MT de multiplicação:

Utilizando a classe NTM, instanciamos uma Máquina de Turing com os seguintes parametros:

- Estados: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9 e halt
- Simbolos de entrada: 1 e *
- Simbolos da fita: 1, *, #, Z e E

- Transições: Conjunto das funções de transição
- Estado inicial: s0
- Simbolo em branco na fita: #
- Estado final: halt

A maquina de turing que realiza as operações de multiplicação conseguem ler fitas da seguinte formatação:

- **#11*1#**
- **#1111*11#**
- **#1*1**
- **#111111*1**

Os valores dos lados direito e esquerdo da soma são representados por 1's, sendo que "111" representa o valor 1+1+1 = 3. As #'s representam os espaços vazios, e devem ser utilizados sempre uma no inicio e outras quatro no final da expressão. Com isso, a expressão #111*11# = 3 * 2 = 6 = 111111

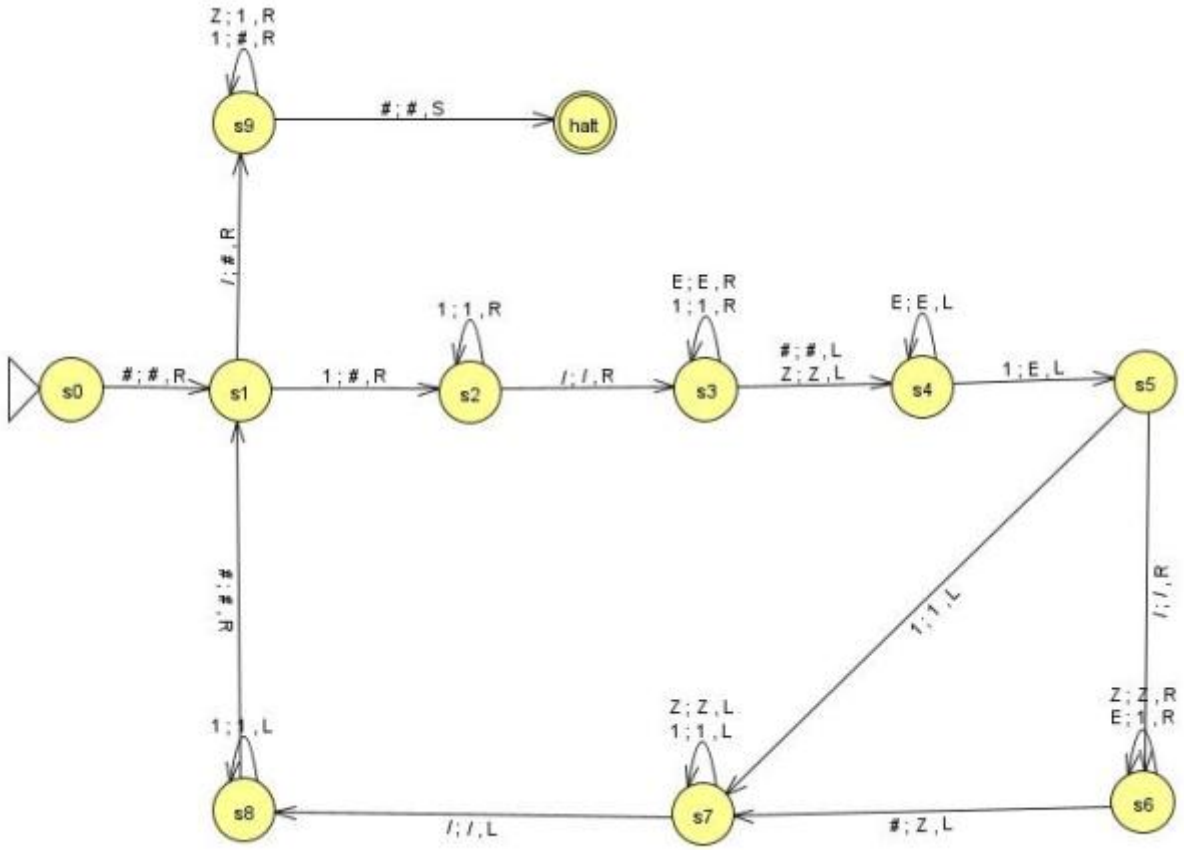
```
# Maquina de turing para multiplicacao
multiplicacao = NTM(
    states={'s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 'halt'},
    input_symbols={'1', '*'},
    tape_symbols={'1', '*', '#', 'Z', 'E'},
    transitions={
        's0': {
            '1': {('s1', '#', 'R')},
            '#': {('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'}
)

# Lista com as operações de soma
op_multiplicacao = [
    '#11111*11####',
    '#11111*111####',
    '#111*11####',
    '#1*1####',
    '#11111111*1111111####',
    '#11111*1####'
]

# Exibição do resultado das somas
print("------ Resultado das multiplicações ------")
for op in op_multiplicacao:
    if multiplicacao.accepts_input(op):
        multiplicacao.read_input(op).pop().print()
```

```
 ------ Resultado das multiplicações ------
 halt: ######111111111##
                       ^
 halt: ######1111111111111##
                            ^
 halt: ####111111##
                ^
 halt: ##1#####
          ^
 halt: #########11111111111111111111111111111111111111111111111111111##
                                                                       ^
 halt: ######11111##
                  ^
```

Divisão

Criação da MT de divisão no JFLAP:



Processo de criação do algoritmo da MT de divisão:

Utilizando a classe NTM, instanciamos uma Máquina de Turing com os seguintes parametros:

- Estados: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9 e halt
- Simbolos de entrada: 1 e /
- Simbolos da fita: 1, /, #, Z e E
- Transições: Conjunto das funções de transição
- Estado inicial: s0
- Simbolo em branco na fita: #
- Estado final: halt

A maquina de turing que realiza as operações de divisão conseguem ler fitas da seguinte formatação:

- **#11/1#**
- **#1111/11#**
- **#1/1**
- **#111111/111**

Os valores dos lados direito e esquerdo da soma são representados por 1's, sendo que "111" representa o valor 1+1+1 = 3. As #'s representam os espaços vazios, e devem ser utilizados

sempre uma no inicio e outras quatro no final da expressão. Com isso, a expressão #1111/11#

```
# Maquina de turing para divisao
divisao = 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')}
        },
        's4': {
            'E': {('s4', 'E', 'L')},
            '1': {('s5', 'E', 'L')}
        },
        's5': {
            '/': {('s6', '/', 'R')},
            '1': {('s7', '1', 'L')}
        },
        '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'}
)


# Lista com as operações de soma
op_divisao = [
    '#1111/11####',
    '#111111/11####',
    '#11/11####',
    '#11111111/1111####',
    '#111111/11####',
]

# Exibição do resultado das somas
print("------ Resultado das divisões ------")
for op in op_divisao:
    if divisao.accepts_input(op):
        divisao.read_input(op).pop().print()
```

```
 ------ Resultado das divisões ------
 halt: ########11##
                ^
 halt: ##########111#
                   ^
 halt: ######1###
             ^
 halt: ###############11##
                       ^
 halt: ##########111#
                   ^
```

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.