

Relatório de Linguagens Formais e Autômatos

Hugo Bourguignon Rangel¹

*Curso de Engenharia de Computação,
Instituto Politécnico do rio de janeiro (IPRJ),
Rio de Janeiro State University,
Rua Bonfim 25, Nova Friburgo, RJ 28625-570, Brazil
27 de maio de 2021*

Resumo

Neste relatório será especificado todos os passos e considerações para a construção de uma Máquina de Estado Finito (MEF) e uma Máquina de Turing (MT) para um dado conjunto de entradas, Além de analisar suas saídas. Trabalho do curso de Linguagens Formais e Autômatos (Graduação em Engenharia de Computação) - Instituto Politécnico, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2021.

O trabalho consiste na criação de uma máquina de estados finitos (MEF) e uma máquina de Turing (MT) visando corresponder ao problema 1(MEF) e problema 2 (MT) apresentados no objetivo do trabalho.

Palavras chave: Maquina de estados finitos, Maquina de Turing, Autômato finito, Linguagem formal.

Abstract

This report will specify all steps and considerations for the construction of a Finite State Machine (MEF) and a Turing Machine (MT) for a given set of inputs, in addition to analyzing its outputs. Formal Languages and Automata course work (Graduation in Computer Engineering) - Polytechnic Institute, State University of Rio de Janeiro, Nova Friburgo, 2021.

The work consists of the creation of a finite state machine (MEF) and a Turing machine (MT) in order to correspond to problem 1 (MEF) and problem 2 (MT) presented in the objective of the work.

Keywords: Finite state machine, Turing machine, Finite automaton, Formal language.

¹ Endereço de e-mail (email address): hugob.rangel@gmail.com

LISTA DE ILUSTRAÇÕES

Figura 1 – Hierarquia de linguagens formais [1].....	2
Figura 2 – Relação entre hierarquia de linguagens formais e hierarquia de autômatos [1]	3
Figura 3 – Relação entre linguagens, autômatos, gramáticas e reconhecimento [2]	4
Figura 4 – Grafo de estados da máquina M construído no JFLAP.....	6
Figura 5 – Representação de uma MT	7
Figura 6 – Esquema resumido do código da MEF no Visual Studio Code.	8
Figura 7 – Esquema resumido do código da MT no Visual Studio Code.	9
Figura 8 – Grafo da máquina M [1].....	10
Figura 9 – Parte do código com os estados definidos.....	10
Figura 10 – Terminal solicitando a entrada (entrada utilizada = “01101”).	11
Figura 11 – Terminal com as respostas do código da MEF.	11
Figura 12 – Trecho do código referente ao estado de aceitação.....	11
Figura 13 – Novo grafo para a máquina M.	12
Figura 14 – Terminal com as respostas (+ Informação sobre o reconhecimento da Entrada).	12
Figura 15 – Parte do código com as quintuplas definidas.	13
Figura 16 - Terminal solicitando a entrada e máximo de passos (entrada utilizada = “0110”, máximo de passos = ‘20’).	13
Figura 17 - Terminal com as respostas do código da MT.	14
 Tabela 1 – Tabela de estados e saídas para a máquina M.	6

SUMÁRIO

1. OBJETIVOS	1
2. INTRODUÇÃO TEÓRICA	2
2.1. Máquina de estados finitos (MEF)	4
2.2. Máquina de Turing (MT).....	7
3. DESENVOLVIMENTO.....	8
3.1. Máquina de estados finitos (MEF)	8
3.1. Máquina de Turing (MT).....	9
4. RESULTADOS	10
4.1. Máquina de estados finitos (MEF)	10
4.2. Máquina de Turing (MT).....	13
5. CONCLUSÃO.....	14
6. BIBLIOGRAFIA e REFERÊNCIAS	14
7. ANEXOS.....	15
7.1. Código Comentado da MEF.....	15
7.2. Código Comentado da MT	17

1. OBJETIVO

Como objetivo do trabalho temos os seguintes problemas computacionais a serem solucionados.

Problema 1:

Escreva um simulador de máquina de estado finito. Isto é, dado:

- um número inteiro positivo n , $n \leq 50$, representando o número de estados de uma máquina de estados finitos com alfabeto de entrada = alfabeto de saída = $\{0, 1\}$.
- uma matriz $n \times 3$ que representa a descrição da tabela de estado de tal máquina.

O programa deve solicitar strings de entrada e escrever as strings de saída correspondentes pelo tempo que o usuário desejar.

Problema 2:

Escreva um simulador de máquina de Turing. Ou seja, dado um conjunto de quintuplas que descreve uma máquina de Turing, seu programa deve solicitar o conteúdo inicial da fita e escrever uma sequência de configurações de fita sucessivas.

Suponha que haja no máximo 100 quintuplas, que o número de células usadas na fita seja no máximo 70, e permita que o usuário defina um número máximo de etapas caso o cálculo não pare antes disso.

2. INTRODUÇÃO TEÓRICA

Temos que a teoria dos autômatos visa o entendimento e estudo de máquinas abstratas que possuem instruções pré-definidas, de forma que trabalhem automaticamente a partir dessas instruções.

Podemos classificar esses autômatos em classes, dependendo de sua Hierarquia na leitura e processamento de uma gramática formal.

A Partir da hierarquia de Chomsky podemos definir quatro tipos (ou classes) de gramáticas formais, começando na gramática formal mais restrita (tipo 3) até a gramática formal mais geral (tipo 0). Cada tipo de gramática formal engloba o conjunto dos tipos de gramáticas formais de nível mais restrito que a mesma. Assim, uma classe mais geral de gramática formal é capaz de gerar uma linguagem formal de classe mais restrita, mas não o contrário.

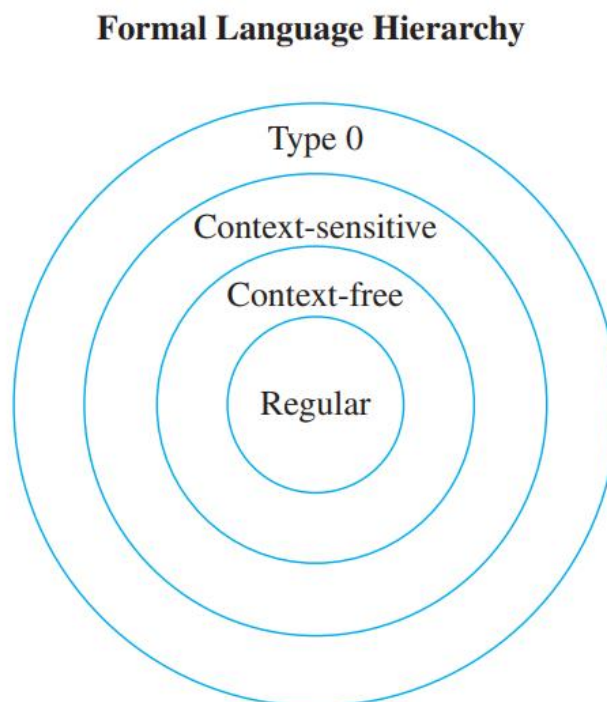


Figura 1 – Hierarquia de linguagens formais [1]

- **Tipo-3: Gramática regular** - mais restritiva do conjunto, geram linguagens regulares.
- **Tipo-2: Gramática Livre de Contexto** - gera linguagens livres de contexto.
- **Tipo-1: Gramática sensível ao contexto** - Tem o nível programável mais alto, eles geram linguagens sensíveis ao contexto.

- **Tipo-0: gramática recursivamente enumerável** – são gramáticas muito genéricas e irrestritas. Pode descrever a sintaxe das linguagens de programação ou linguagens naturais.

Temos que, para cada gramática formal (responsável pela geração de uma linguagem formal), existe um autômato responsável pelo processamento e aceitação da mesma. De forma semelhante as gramáticas formais, os autômatos de classes mais gerais (processam linguagens mais gerais) também podem processar e aceitar linguagens de classe mais restrita.

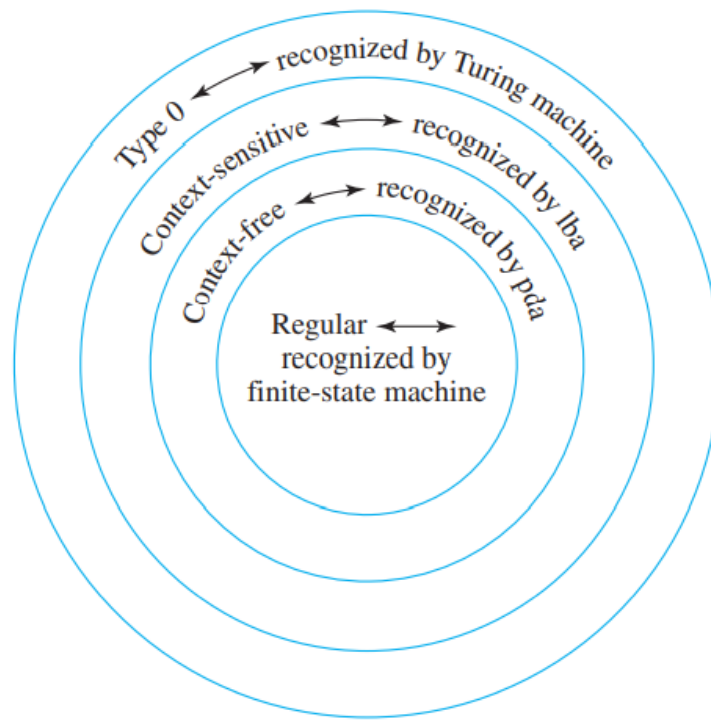


Figura 2 – Relação entre hierarquia de linguagens formais e hierarquia de autômatos [1]

Assim como as linguagens formais, agora podemos definir os autômatos em classes de tipos.

Tipo-3: Autômatos de Estados Finitos (MEF) - Para calcular construções para uma linguagem regular, a consideração mais importante é que não há requisito de memória. Temos uma máquina de propósito único ou um algoritmo sequencial. O autômato conhece o estado atual e os próximos estados permitidos, mas não "lembra" das etapas anteriores.

Tipo 2: Autômato push-down (PDA) - Para corresponder às dependências definidas, este autômato requer uma pilha de memória com uma extremidade de funcionamento. Por exemplo, para combinar o número de frases 'if' e 'else', o autômato precisa 'lembrar' o último 'if' ocorrendo. Só então ele pode encontrar o "outro" correspondente.

Tipo 1: Autômato Linear-Bounded (LBA) - É uma forma de máquina de Turing restrita que, em vez de ser ilimitada, é limitada por alguma função linear computável. A vantagem desse autômato é que seu requisito de memória (limitado pela memória RAM) é previsível mesmo que a execução seja recursiva em partes.

Tipo-0: Máquina de Turing (MT) – Sabemos que existem funções não computáveis na Matemática ou na Ciência da Computação. A máquina de Turing, por ser o caso mais geral, permite representar até mesmo essas funções como uma sequência de etapas discretas. O controle da máquina é finito (devido as propriedades físicas da máquina), mesmo que os dados possam ser aparentemente infinitos (teoricamente, memória infinita).

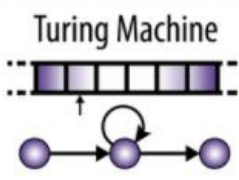

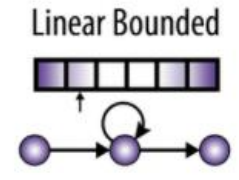

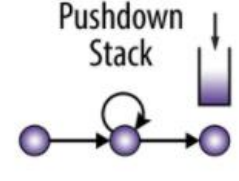

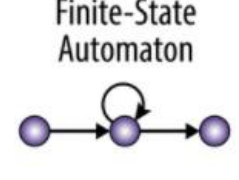

Language	Automaton	Grammar	Recognition
Recursively Enumerable Languages		Unrestricted $Baa \rightarrow A$	Undecidable 
Context-Sensitive Languages		Context Sensitive $A t \rightarrow aA$	Exponential? 
Context-Free Languages		Context Free $S \rightarrow gSc$	Polynomial 
Regular Languages		Regular $A \rightarrow cA$	Linear 

Figura 3 – Relação entre linguagens, autômatos, gramáticas e reconhecimento [2]

Temos que o objetivo do trabalho só requer a construção de uma Máquina de estados finitos e Máquina de Turing.

2.1. Máquina de estados finitos (MEF)

De acordo com a hierarquia de Chomsky, a máquina de estado finito (MEF) é o tipo de autômato mais simples e restrito (tipo 3), sendo capaz de processar somente linguagens do tipo 3 (linguagens regulares).

Uma MEF pode ser definida como uma quintupla ordenada $M = (S, I, O, f_s, f_o)$ onde:

- S é o conjunto finito de estados;
- I é o conjunto finito de símbolos de entrada (alfabeto de entrada);
- O é o conjunto finito de símbolos de saída (alfabeto de saída);
- $f_s : S \times I \rightarrow S$, é uma função que retorna o próximo estado em determinado passo;

$$f_s(\text{estado}(t_i), \text{entrada}(t_i)) = \text{estado}(t_{i+1})$$

$$t_i \rightarrow \text{passo atual} \mid t_{i+1} \rightarrow \text{Próximo passo}$$

- $f_o : S \rightarrow O$, é uma função que retorna o termo de saída em determinado passo;

$$f_o(\text{estado}(t_i)) = \text{saida}(t_i)$$

$$t_i \rightarrow \text{passo atual}$$

Os processos da MEF são sincronizados por pulsos discretos de um relógio (clock). Em um pulso de clock, a entrada pode ser lida, o que pode mudar alguns dos locais de armazenamento e, portanto, alteram o estado da máquina para um novo estado.

O novo estado vai depender da entrada no pulso de clock, assim como todos os outros estados anteriores. Se as entradas e estados forem conhecidos em cada pulso de clock, as mudanças nos parâmetros internos da máquina são previsíveis e não aleatórias.

Temos que o conteúdo de armazenamento está disponível como a saída da máquina. O estado da máquina determina o conteúdo de armazenamento (saída) para determinado momento de pulso de clock. Desta forma, ao longo de uma sucessão de pulsos de clock, a máquina produz uma sequência de saídas em resposta a uma sequência de entradas (as quais definem os estados da máquina).

Para melhor ilustração temos, a seguir, um exemplo de uma máquina de estados finitos.

Temos uma Máquina de estado finito M com as seguintes características:

$$S = \{q_0, q_1, q_2\} \mid I = \{0,1\} \mid O = \{0,1\}$$

Como as duas funções (f_s e f_o) atuam em domínios finitos, elas podem ser definidas através de uma tabela de estados e saídas.

Estado atual	Próximo Estado		Saída
	Termos da entrada		
	1	0	
q_0	q_1	q_0	0
q_1	q_2	q_1	1
q_2	q_2	q_0	1

Tabela 1 – Tabela de estados e saídas para a máquina M.

A partir de uma máquina de estados é possível verificar se uma entrada é reconhecida ou não pela Gramática formal atribuída a máquina.

Temos para a Máquina M o seguinte grafo de estados construído no programa JFLAP:

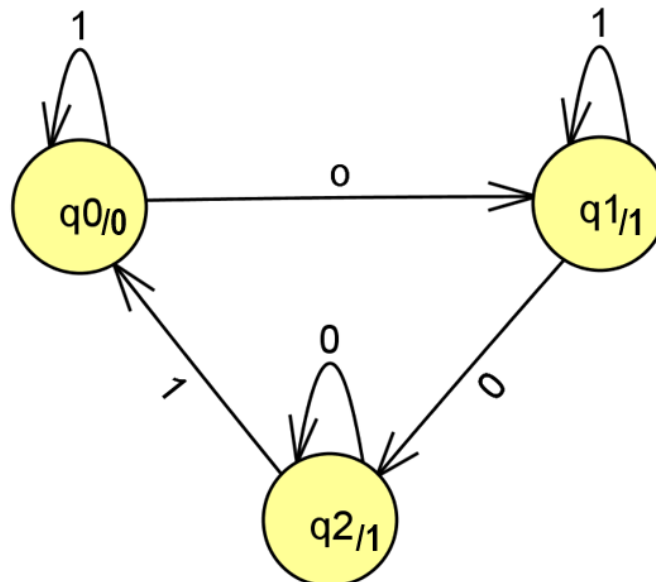


Figura 4 – Grafo de estados da máquina M construído no JFLAP

2.2. Máquina de Turing (MT)

A máquina de Turing é o Autômato mais geral que simula o funcionamento de um computador. Criada por Alan Turing, a MT consiste em uma fita de comprimento infinito que comporta os dados de entrada da Máquina de Turing e um cabeçote capaz de ler e escrever os dados armazenados na fita.

O cabeçote pode percorrer a fita, tanto para a esquerda quanto para direita, apenas uma posição por vez. Abaixo temos uma representação de uma MF.

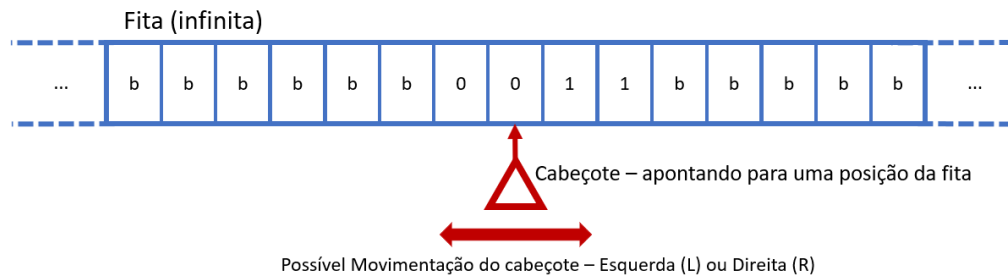


Figura 5 – Representação de uma MT

Assim como uma Máquina de estados finitos, a MT também pode ser representada por uma quintupla ordenada $MT = (s, i, i', s', d)$. Onde, para um conjunto finito de estados S e para um conjunto finito de símbolos da fita I (alfabeto da fita), temos:

- s ($s \in S$), representa um estado da Máquina;
- i ($i \in I$), representa um símbolo de entrada;
- i' ($i' \in I$), representa um símbolo de saída;
- s' ($s' \in S$), representa um novo estado;
- d ($d \in \{L, R\}$), representa a direção de movimento do cabeçote (“L” -Left-esquerda, “R” -Right- direita)

Para cada dado i lido pelo cabeçote da MT, e dado o estado atual s , teremos uma saída i' , um novo estado s' e uma direção de movimento d para o cabeçote.

Percebe-se que, diferente de uma MEF, uma MT pode ler, escrever e guardar dados na Fita (capacidade de memória infinita). Assim, uma MT é capaz de processar gramáticas formais que uma MEF nunca conseguiria processar.

3. DESENVOLVIMENTO

Através da base teórica já apresentada, foi possível definir códigos computacionais para cada autômato (MEF e MT). Foi utilizado a linguagem de programação Python, juntamente com o software Visual Studio Code (editor e compilador), para a construção desses códigos. Através do recurso de orientação a objetos do Python foi possível gerar uma resposta rápida e pratica (via terminal) para ambos os problemas.

3.1. Máquina de estados finitos (MEF)

Utilizando o recurso de orientação a objetos do Python foi definido uma classe Estado com os parâmetros da quintupla ordenada da MEF como objetos.

Cada objeto da classe pode aponta para outros objetos (ponteiros). A partir desse princípio foi definido o código da MEF que, através da entrada de um vetor de dados (string de entrada) e uma lista de estados (matriz de estados), pode simular o comportamento de uma MEF.

Durante a simulação da MEF, o programa irá listar todos os objetos para cada passo interno da computação (mostrado no terminal). Após o termino da simulação da MEF o programa irá retorna um novo vetor de dados (string de saída).

Considerando as especificações do problema, a string de entrada será solicitada para o usuário no terminal. Também é possível sair da simulação (programa) a qualquer momento utilizando o comando de teclado *Ctrl + C* no terminal. Abaixo segue o esquema do código

```
C:\> Users > Hugo > Desktop > Automatos > Trabalho > MEF.py > ...
1  # ----- Define-se a Classe Estado -----
2
3  > class Estado: ...
9
10 # ----- Define-se a Máquina de Estados Finitos (MEF) -----
11
12 > def MEF(Entrada, Estados): ...
83
84
85 # ----- Main: definir parametros e decalrar a MEF -----
86
87 if __name__ == '__main__':
88
89     # --- vetor de entrada --- #
90     entrada = input('\n Digite uma entrada (Apenas 0s e 1s): ')
91     Entrada = [int(e) for e in entrada]
92
93     # --- Estados da Máquina *(alterar estados aqui)* --- #
94
95     Estados = [Estado(0,1,0,0),
96                Estado(1,2,0,0),
97                Estado(2,2,0,1)]
98
99     # --- Chamar a MEF para os parametros de Entrada e Estados --- #
100
101     MEF(Entrada,Estados)
```

Figura 6 – Esquema resumido do código da MEF no Visual Studio Code.

No anexo do relatório segue código integral (não resumido) e comentado para uma melhor análise.

3.1. Máquina de Turing (MT)

Assim como na MEF, Através da utilização do recurso de orientação a objetos do Python foi definido uma classe *Quíntupla* com os parâmetros da quántupla ordenada da MT como objetos.

Assim, cada objeto da classe pode aponta para outros objetos (ponteiros). A partir desse princípio foi definido o código da MT que, através da entrada de um vetor de dados (Lista com os dados relevantes) e uma lista de quántuplas (matriz de quántuplas), foi possível simular o comportamento de uma MT.

Durante a simulação da MT, o programa irá listar todos os objetos para cada passo interno da computação (mostrado no terminal). Após o termino da simulação da MT o programa irá retorna um novo vetor de dados (Lista depois da computação da MT).

Considerando as especificações do problema, a lista de dados relevantes será solicitada para o usuário no terminal. Também é possível sair da simulação (programa) a qualquer momento utilizando o comando de teclado *Ctrl + C* no terminal. Também é possível definir um número máximo de passos para a MF (dependendo do número de passos do terminal a lista de entrada não será computada totalmente pela MT).

```
C: > Users > Hugo > Desktop > Automatos > Trabalho > MT.py > ...
1  # ----- Define-se a Classe Quíntupla -----
2
3  > class Quíntupla: ...
10
11 # ----- Define-se a Máquina de Turing (MT) -----
12
13 > def MT(Fita, Quíntuplas,max_de_passos):...
69
70 # ----- Main: definir parametros e decalrar a MT -----
71
72 if __name__ == '__main__':
73
74     # ----- construindo a fita ----- #
75     # [0,1,1,0]
76     entrada = input('\n Digite a entrada (Parte não vazia da fita): ')
77     Parte_com_dados= [int(e) for e in entrada]
78     Comprimento_Fita = len(Parte_com_dados)*3
79
80     Fita = ['b']*Comprimento_Fita # (fita preenchida com espaços vazios'b')
81
82 > for i,parte in enumerate(Parte_com_dados):...
84     # Sabemos que uma fita é infinita, mas para sua representação
85     # temos somente um pedaço que comporta os dados (parte relevante)
86
87     # ----- maximo de passos ----- #
88
89     max_de_passos = input('\n Digite o maximo de passos (Parte não vazia da fita): ')
90     max_de_passos = int(max_de_passos)
91
92     # ---- Quíntuplas da Máquina de Turing *(Alterar Quíntuplas aqui)* ---- #
93
94 > Quíntuplas = [Quíntupla( 0 , 0 , 1 , 0 , 'R'),...
99
100 # --- Chamar a MT para os parametros da Fita e Quíntuplas --- #
101
102 MT(Fita,Quíntuplas,max_de_passos)
```

Figura 7 – Esquema resumido do código da MT no Visual Studio Code.

No anexo do relatório segue código integral (não resumido) e comentado para uma melhor análise.

4. RESULTADOS

Para uma melhor ilustração do funcionamento do código foi utilizado um exemplo para cada máquina.

4.1. Máquina de estados finitos (MEF)

Temos uma Máquina de estados finitos M que possui o seguinte grafo:

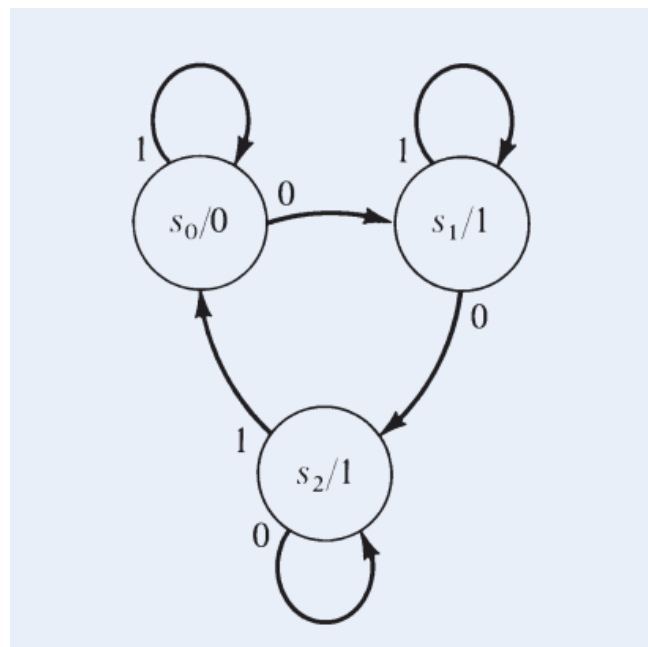


Figura 8 – Grafo da máquina M [1].

Os estados foram analisados e representados na matriz de estados (no código).

```

Estados = [Estado(0,1,0,0),
            Estado(1,2,1,1),
            Estado(2,2,0,1)]
  
```

Figura 9 – Parte do código com os estados definidos.

Agora foi definido a entrada “01101” (string de entrada). Após Compilar o código, será solicitado a string de entrada no terminal.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\Hugo> conda activate base
PS C:\Users\Hugo> & C:/Users/Hugo/anaconda3/python.exe c:/Users/Hugo/Desktop/Automatos/Trabalho/MEF.py

Digite uma entrada (Apenas 0s e 1s): 01101
```

Figura 10 – Terminal solicitando a entrada (entrada utilizada = “01101”).

Depois de definir a entrada e apertar a tecla “Enter” (dar continuidade ao programa), A Máquina de estados finitos começara a ser computada. Por final, o código irá retornar todos os passos internos da máquina, além de retornar a saída (output).

```
Digite uma entrada (Apenas 0s e 1s): 01101

Entrada → [0, 1, 1, 0, 1]

* Estado_inicial → 0

Estados e saidas de cada etapa de computação da MEF:

* Estado_atual → 1 | Saida → 1
* Estado_atual → 1 | Saida → 1
* Estado_atual → 1 | Saida → 1
* Estado_atual → 2 | Saida → 1
* Estado_atual → 0 | Saida → 0

Saida → [1, 1, 1, 1, 0]
```

Figura 11 – Terminal com as respostas do código da MEF.

Caso a MEF seja utilizada para reconhecer uma entrada, é possível definir o critério de aceitação no código, segue exemplo abaixo.

```
# --- MEF reconhece a Entrada -> verdadeiro --- #

elif (Estado_atual.termo == 0): #(Critério de aceitação, Estado_atual == *Estado de aceitação* )
    print('\n *** A MEF reconhece a Entrada ***\n')
```

Figura 12 – Trecho do código referente ao estado de aceitação.

Assim temos o novo grafo para a MEF, com estado de aceitação em S_0 :

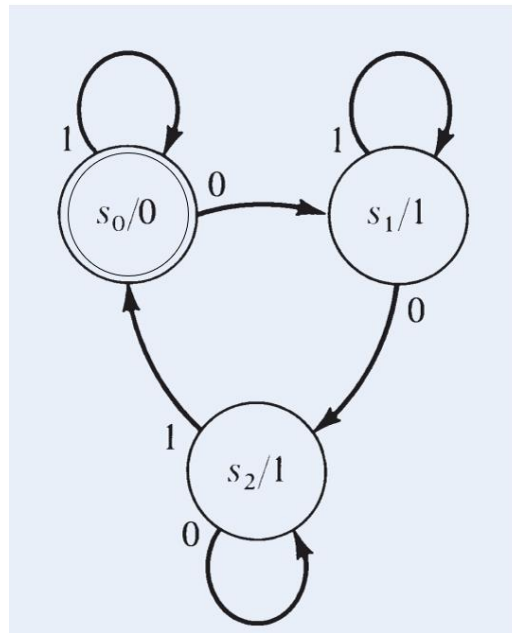


Figura 13 – Novo grafo para a máquina M.

Agora nos resultados do código teremos a informação se a entrada foi ou não reconhecida pela MEF.

```

Digite uma entrada (Apenas 0s e 1s): 01101
Entrada → [0, 1, 1, 0, 1]

* Estado_inicial → 0

Estados e saídas de cada etapa de computação da MEF:

* Estado_atual → 1 | Saída → 1
* Estado_atual → 1 | Saída → 1
* Estado_atual → 1 | Saída → 1
* Estado_atual → 2 | Saída → 1
* Estado_atual → 0 | Saída → 0

Saída → [1, 1, 1, 1, 0]

*** A MEF reconhece a Entrada ***
  
```

Figura 14 – Terminal com as respostas (+ Informação sobre o reconhecimento da Entrada).

4.2. Máquina de Turing (MT)

Para a máquina de Turing temos a definição das quintuplas.

```
# ---- Quintuplas da Máquina de Turing *(Alterar Quintuplas aqui)* ---- #

Quintuplas = [Quintupla( 0 , 0 , 1 , 0 , 'R'),
               Quintupla( 0 , 1 , 0 , 0 , 'R'),
               Quintupla( 0 , 'b', 1 , 1 , 'L'),
               Quintupla( 1 , 0 , 0 , 1 , 'R'),
               Quintupla( 1 , 1 , 0 , 1 , 'R')]
```

Figura 15 – Parte do código com as quintuplas definidas.

Agora foi definido a entrada “0110” (string de entrada) e o máximo de passos “20” (somente para a MT computar toda a parte relevante da fita). Após Compilar o código, será solicitado a string de entrada e o número máximo de passos no terminal.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\Hugo> conda activate base
PS C:\Users\Hugo> & C:/Users/Hugo/anaconda3/python.exe c:/Users/Hugo/Desktop/Automatos/Trabalho/MT.py

Digite a entrada (Parte não vazia da fita): 0110

Digite o maximo de passos: 20
```

Figura 16 - Terminal solicitando a entrada e máximo de passos (entrada utilizada = “0110”, máximo de passos = ‘20’).

Depois de definir a entrada e apertar a tecla “Enter” (dar continuidade ao programa), A Máquina de Turing começara a ser computada. Por final, o código irá retornar todos os passos internos da máquina, além de retornar a saída (output).


```

Digite a entrada (Parte não vazia da fita): 0110
Digite o maximo de passos: 20
Fita → ['b', 'b', 'b', 'b', 0, 1, 1, 0, 'b', 'b', 'b', 'b']

-----*Computação da MF*-----

Estado, posição na Fita, valor de leitura e valor de escrita em cada passo de computação da MF:

Estado → 0 | Posição na fita → 4 | Valor de leitura → 0 | Valor de Escrita → 1
Estado → 0 | Posição na fita → 5 | Valor de leitura → 1 | Valor de Escrita → 0
Estado → 0 | Posição na fita → 6 | Valor de leitura → 1 | Valor de Escrita → 0
Estado → 0 | Posição na fita → 7 | Valor de leitura → 0 | Valor de Escrita → 1
Estado → 1 | Posição na fita → 8 | Valor de leitura → b | Valor de Escrita → 1
Estado → 1 | Posição na fita → 7 | Valor de leitura → 1 | Valor de Escrita → 0
Estado → 1 | Posição na fita → 8 | Valor de leitura → 1 | Valor de Escrita → 0

-----

Fita (depois da computação da MF) → ['b', 'b', 'b', 'b', 1, 0, 0, 0, 0, 'b', 'b', 'b']

```

Figura 17 - Terminal com as respostas do código da MT.

5. CONCLUSÃO

A partir da construção e execução da máquina de estado finito e máquina de Turing, foi possível ter uma melhor compreensão da teoria das linguagens formais e autômatos.

A teoria das linguagens formais e autômatos tem bastante importância histórica no ramo da computação. A partir da construção dos autômatos ilustrados no trabalho (MEF e MT) foi possível, através de muito desenvolvimento e pesquisa, chegar nos computadores e máquinas que temos atualmente.

Ainda assim, a teoria se faz necessária no dia-a-dia. Temos o seu uso em aplicações (ou máquinas) que envolvam análise sintática, compiladores, inteligência artificial, verificação formal e muitas outras áreas na computação.

6. BIBLIOGRAFIA E REFERÊNCIAS

[1] J.L. Gersting. Mathematical Structures for Computer Science. W. H. Freeman, 2014.

[2] Journal of Neurolinguistics Volume 43, Part B, August 2017, Pages 78-94 : “The Universal Generative Faculty: The source of our expressive power in language, mathematics, morality, and music”, disponível em:

<<https://www.sciencedirect.com/science/article/abs/pii/S0911604416300811?via%3Dihub>>

(Consultada em 25 de maio 2021)

[3] Material disponibilizado pelo Professor Francisco De Moura na plataforma de estudo online Moodle para a matéria Linguagens Formais e Autômatos -2020-2. Disponível em:

<<https://ead.iprj.uerj.br/moodle/course/view.php?id=207>> Necessário Cadastro na matéria.

(Consultada em 25 de maio 2021)

7. ANEXOS

7.1. Código Comentado da MEF

```
# ----- Define-se a Classe Estado ----- #

class Estado:
    def __init__(self, termo, caminho_0, caminho_1, saida):
        self.termo = termo
        self.caminho_0 = caminho_0
        self.caminho_1 = caminho_1
        self.saida = saida

# ----- Define-se a Máquina de Estados Finitos (MEF) ----- #

def MEF(Entrada, Estados):

    print('\n Entrada \u2192', Entrada)
    Comprimento_Entrada = len(Entrada)
    saida = ['b']*Comprimento_Entrada
    m = 0

    # --- Primeiro estado --- #

    n = 0
    while n < len(Estados):
        if Estados[n].termo == 0:
            Estado_atual = Estados[n]
            break
        n+=1

    print('\n* Estado_inicial \u2192', Estado_atual.termo, '\n')

    # --- quebra da MEF -> falso --- #

    Estado_de_quebra = 0

    # --- Computação da Entrada --- #
```

```

print('Estados e saidas de cada etapa de computação da MEF:\n')
while (len(Entrada) != 0):
    aux = Entrada[0]
    Entrada.pop(0)

    if (aux == 0):

        if (Estado_atual.caminho_0 != 'Estado_de_quebra'):
            Prox_estado = Estado_atual.caminho_0
        else:
            Estado_de_quebra = 1

    elif (aux == 1):

        if (Estado_atual.caminho_1 != 'Estado_de_quebra'):
            Prox_estado = Estado_atual.caminho_1
        else:
            Estado_de_quebra = 1

    if (Estado_de_quebra != 1):
        n = 0
        while (n < len(Estados)):
            if (Estados[n].termo == Prox_estado):
                Estado_atual = Estados[n]
                break
            n += 1
        print('* Estado_atual \u2192', Estado_atual.termo, '| Saida \u2192', Estado_atual.saida)

        saida[m] = Estado_atual.saida
        m += 1

    print('\n Saida \u2192', saida)

    # --- quebra da MEF -> verdadeiro --- #

    if (Estado_de_quebra == 1):
        print('\n *** A MEF quebrou, entrou em um estado onde não consegue com-
putar. ***\n')

    # --- MEF reconhece a Entrada -> verdadeiro --- #

    elif (Estado_atual.termo == 0): #(Critério de aceitação, Estado_atual == *
Estado de aceitação*)
        print('\n *** A MEF reconhece a Entrada ***\n')

    # --- MEF reconhece a Entrada -> falso --- #

```

```

else:
    print('\n *** A Entrada não é reconhecida pela MEF ***\n')

# ----- Main: definir parametros e decalrar a MEF ----- #

if __name__ == '__main__':

    # --- vetor de entrada --- #
    entrada = input('\n Digite uma entrada (Apenas 0s e 1s): ')
    Entrada = [int(e) for e in entrada]

    # --- Estados da Máquina *(alterar estados aqui)* --- #

    Estados = [Estado(0,1,0,0),
                Estado(1,2,1,1),
                Estado(2,2,0,1)]

    # --- Chamar a MEF para os parametros de Entrada e Estados --- #

    MEF(Entrada,Estados)

```

7.2. Código Comentado da MT

```

# ----- Define-se a Classe Quintupla ----- #

class Quintupla:
    def __init__(self, Estado_atual, leitura, Escrita, Proximo_Estado, Direcao
    ):
        self.Estado_atual = Estado_atual
        self.leitura = leitura
        self.Escrita = Escrita
        self.Proximo_Estado = Proximo_Estado
        self.Direcao = Direcao

#----- Define-se a Máquina de Turing (MT) ----- #

def MT(Fita, Quintuplas,max_de_passos):
    print('\nFita \u2192', Fita, '\n')
    Estado_atual = 0
    saida = 0
    aux = 0
    Estados = []

```

```

Leituras = []
posicao_Fita = 0

# --- Tratamento dos Estados --- #

for quint in Quintuplas:
    Estados.append(Quintuplas[aux].Estado_atual)
    aux += 1
aux = 0

# --- Tratamento das Leituras --- #

for quint in Quintuplas:
    Leituras.append(Quintuplas[aux].leitura)
    aux += 1
aux = 0

# --- Posição inicial (cabecote inicial da Fita) --- #

while (Fita[posicao_Fita] == 'b'):
    posicao_Fita += 1

# --- Computação da MF --- #

print('-----*Computação da MF*-----\n')

print('Estado, posição na Fita, valor de leitura e valor de escrita em cada passo de computação da MF:\n')

m = 0
while (Estado_atual in Estados and posicao_Fita >= 0 and posicao_Fita < len(Fita)):
    if aux >= len(Quintuplas):
        Estado_atual = 'Saída'
        break

    elif (Quintuplas[aux].leitura not in Leituras):
        Estado_atual = 'Saída'
        break

    elif (Estado_atual == Quintuplas[aux].Estado_atual and Fita[posicao_Fita] == Quintuplas[aux].leitura and m < max_de_passos):
        Fita[posicao_Fita] = Quintuplas[aux].Escrita
        Estado_atual = Quintuplas[aux].Proximo_Estado
        print('Estado \u2192', Estado_atual, ' | Posição na fita \u2192', posicao_Fita, ' | Valor de leitura \u2192', Quintuplas[aux].leitura, ' | Valor de Escrita \u2192', Quintuplas[aux].Escrita)

```

```

        posicao_Fita += 1 if (Quintuplas[aux].Direcao == 'R') else -1
        aux = -1
        m += 1
        aux += 1

    print('\n-----\n')

    print('Fita (depois da computação da MF) \u2192', Fita,'\n')

# ----- Main: definir parametros e decalrar a MT ----- #

if __name__ == '__main__':

    # ----- construindo a fita ----- #
    entrada = input('\n Digite a entrada (Parte não vazia da fita): ')
    Parte_com_dados= [int(e) for e in entrada]
    Comprimento_Fita = len(Parte_com_dados)*3

    Fita = ['b']*Comprimento_Fita # (fita preenchida com espaços vazios'b')

    for i,parte in enumerate(Parte_com_dados):
        Fita[Comprimento_Fita//2-len(Parte_com_dados)//2 + i] = parte
    # Sabemos que uma fita é infinita, mas para sua representação
    # temos somente um pedaço que comporta os dados (parte relevante)

    # ----- maximo de passos ----- #

    max_de_passos = input('\n Digite o maximo de passos: ')
    max_de_passos = int(max_de_passos)

    # ---- Quintuplas da Máquina de Turing *(Alterar Quintuplas aqui)* ---- #

    Quintuplas = [Quintupla( 0 , 0 , 1 , 0 , 'R'),
                   Quintupla( 0 , 1 , 0 , 0 , 'R'),
                   Quintupla( 0 , 'b', 1 , 1 , 'L'),
                   Quintupla( 1 , 0 , 0 , 1 , 'R'),
                   Quintupla( 1 , 1 , 0 , 1 , 'R')]

    # --- Chamar a MT para os parametros da Fita e Quintuplas --- #

    MT(Fita,Quintuplas,max_de_passos)

```