Trabalho de Linguagens Formais e Autômatos

Ariel Nogueira Kovaljski
<arielnogueirak@gmail.com>

Computer Engineering Course,
Instituto Politécnico (IPRJ) — Rio de Janeiro State University,
Rua Bonfim 25, Nova Friburgo, RJ 28625-570, Brazil

<dia-atual> de maio de 2021

Abstract

In this assignment we build and analyze the behavior and output of a Finite State Machine (FSM) and a Turing Machine (TM) for a given set of inputs.

Keywords: finite state machine, turing machine, finite automata

Resumo

Neste trabalho nós construímos e analisamos o comportamento e a saída de uma Máquina de Estado Finito (MEF) e uma Máquina de Turing (MT) para um dado conjunto de entradas.

Palavras-chave: máquina de estado finito, máquina de turing, autômatos finitos

1. Introdução

A teoria de autômatos trata do estudo de máquinas abstratas que seguem instruções pré-determinadas automaticamente.

A hierarquia de Chomsky define quatro tipos (ou classes) de gramáticas formais, do tipo 3 (mais restrito) ao tipo 0 (mais geral). Cada gramática de classe superior, é um superconjunto das classes anteriores. Desta forma, uma classe mais geral é capaz de gerar uma linguagem de classe mais restrita, mas não o contrário.

Para cada gramática, responsável pela geração de uma linguagem, há o seu respectivo autômato, responsável pelo processamento e aceitação da mesma. Similarmente, os autômatos de classes mais gerais também podem processar e aceitar linguagens de classe mais restrita, mas não o contrário.

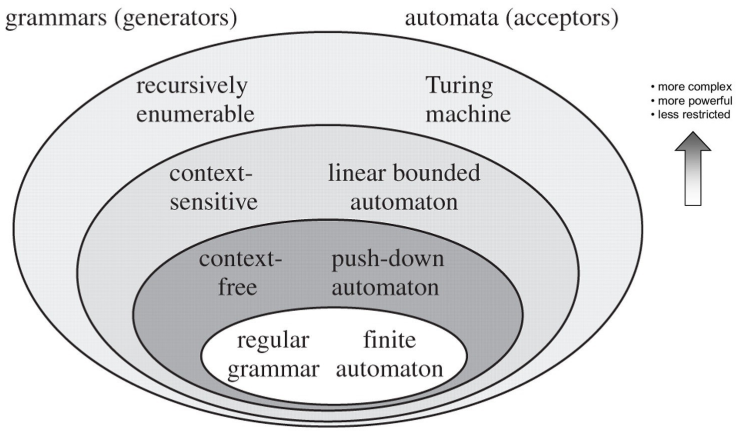

Figura 1: Hierarquia de Chomsky [1]

Uma gramática pode gerar uma linguagem possivelmente infinita, sendo assim, a melhor maneira de representá-la é através da definição de um autômato.

Dentre os tipos de autômatos existentes, neste trabalho iremos abordar a construção e o funcionamento das *máquinas de estado finito* e das *máquinas de Turing*. Estes são responsáveis pelo reconhecimento das *gramáticas regulares* e *recusivamente enumeráveis*, respectivamente.

2. Teoria

2.1. Máquina de Estado Finito

Máquinas de estado finito (MEF) são o tipo mais simples e restrito de autômato, capazes apenas de processar linguagens do tipo 3: "gramática regular", geradas por expressões regulares.

Uma MEF pode ser considerada como um modelo simplificado do funcionamento de um computador. Esta pode ser definida como uma quíntupla 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);
- $f_S : S \times I \to S$ é uma função que retorna o próximo estado $s_{t_{k+1}} \in S$ dado o estado anterior $s_{t_k} \in S$ e um símbolo de entrada $i_{t_k} \in I$;
- $f_O : S \to O$ é a função *output*, que retorna o símbolo de saída $o_{t_k} \in O$ do estado atual $s_{t_k} \in S$.

As operações da MEF são sincronizadas por pulsos discretos de um relógio (*clock*) representados por t_k, onde $k \in \mathbb{N}_0$ representa o ciclo de *clock* atual. Partindo de um estado inicial $s_0 = s_{t_0}$, após um pulso de *clock*, a função f_S retornará um novo estado s_{t_1} dado a entrada anterior i_{t_0} e o estado anterior s_0. Generalizando para qualquer pulso de *clock*, é possível afirmar que a MEF possui um comportamento determinístico, ou seja, o novo estado sempre dependará do estado e entrada anteriores. Cada estado funciona como uma memória dos *inputs* anteriores, além de possuir um símbolo de saída, que é retornado pela função *output* f_O.

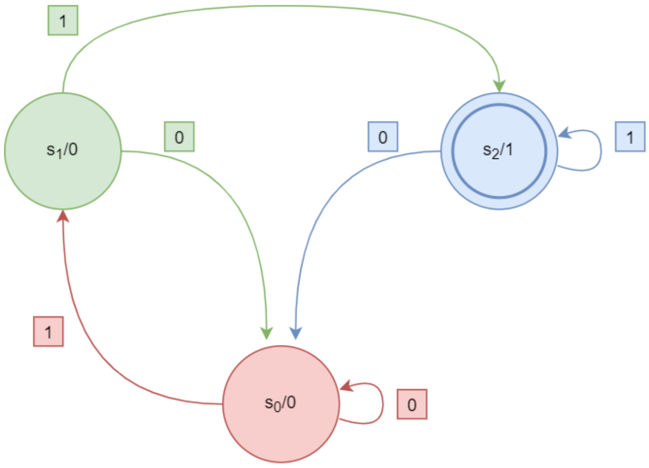

Figura 2: Exemplo de máquina de estado finito

Tomando como exemplo a MEF acima, começamos no estado s_0 e temos como saída o símbolo **0**. Aqui, há duas possibilidades: enquanto a entrada for **0**, permanecemos neste estado; caso a entrada seja **1**, seguimos para o estado s_1. Ao seguirmos para o estado s_1, temos como saída o símbolo **0**. Novamente, há duas possíveis entradas: caso a entrada seja **0**, retornamos ao estado s_0; caso a entrada seja **1**, seguimos para o estado s_2. Por fim, ao seguirmos para o estado s_2, temos como saída o símbolo **1**. As alternativas são: caso a entrada seja **0**, retornarmos ao estado s_0; enquanto a entrada seja **1**, permanecemos neste estado.

Esta forma um tanto verbosa de se descrever uma MEF pode ser colocada em uma tabela, com o estado atual, possíveis entradas e saídas como colunas da mesma.

Estado Atual	Próximo Estado		Saída
	0	**1**	
s_0	s_0	s_1	**0**
s_1	s_0	s_2	**0**
s_2	s_0	s_2	**1**

Tabela 1: Tabela de estados da MEF representada na Fig. 2

Nota-se que dentre todas as possíveis entradas, esta MEF só aceitará, isto é, terminará no estado final s_2, para *strings* de entrada que terminem com dois ou mais **1** (**11**, **111**, **1111**, ...).

2.2. Máquina de Turing

Máquinas de estado finito são capazes de processar apenas gramáticas do tipo 3: "gramática regular". Sendo assim é necessário o uso de outros tipos de autômatos para o processamento de gramáticas do tipo 2, 1 e 0: "livre de contexto", "sensível ao contexto" e "recursivamente enumerável", respectivamente.

Segundo a hierarquia de Chomsky, cada classe de gramática é um superconjunto da gramática anterior. Sendo assim, a gramática mais geral, a "recursivamente enumerável" com o seu respectivo autômato, a máquina de Turing, é capaz de representar e processar linguagens formais de qualquer outra classe.

A máquina de Turing (MT), criada por Alan Turing [3], é o modelo mais geral do funcionamento de um computador. Considera-se uma fita de comprimento infinito que armazena os dados de entrada da MT. Cada dado ocupa uma posição da fita. Anexado a esta fita, há um cabeçote, que pode ler e escrever dados da fita sob a posição em que se encontra. Este pode mover-se para a esquerda e para direita apenas uma posição por vez. O cabeçote serve como dispositivo de entrada/saída da MT.

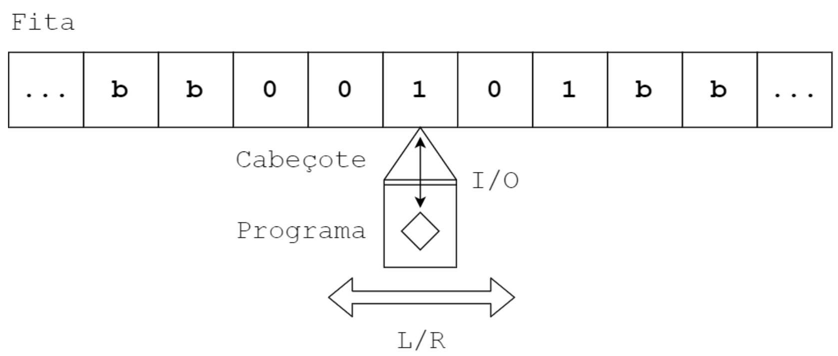

Figura 3: Elementos da máquina de Turing

Para um conjunto finito de estados S e para um conjunto finito de símbolos da fita (alfabeto da fita) I, define-se uma MT como uma quíntupla ordenada $T = (s, i, i', s', d)$ onde:

- $s \in S$ é o estado da MT;
- $i \in I$ é um símbolo de entrada;
- $i' \in I$ é um símbolo de saída;
- $s' \in S$ é o novo estado da MT;
- $d \in \{L, R\}$ é direção de movimento do cabeçote.

Para cada dado i lido pela MT, dado o estado atual s, resultará em uma saída i', um novo estado s' e uma direção de movimento do cabeçote d. Nota-se que exceto pelo componente d, a MT é idêntica a MEF, mas devido a fita com capacidade de memória infinita e a possibilidade de ler e escrever e reler os dados da própria fita, faz com que a MT seja de processar gramáticas que seriam impossíveis em uma MEF.

3. Desenvolvimento

Tendo a teoria como base, foi possível escrever uma implementação computacional destes autômatos. A linguagem escolhida foi *Python*, pois sua orientação a objetos e simplicidade de sintaxe permitiu uma modelagem do problema de forma mais direta, rápida e eficiente.

3.1. Máquina de Estado Finito

A máquina de estado finito foi implementada como uma classe (`FSM`), onde seus estados estão armazenados em uma lista, e cada estado nesta lista é um dicionário com chaves correspondentes ao próprio estado (`state`), próximo estado se o *input* for **0** (`next0`), próximo estado se o *input* for **1** (`next1`) e ao *output* (`output`), nesta ordem. Esta abordagem foi tomada para facilidade de interpretação do código; ao invés de utilizar apenas índices, o uso das chaves nomeadas torna o funcionamento mais explícito.

Esta lista de dicionários é preenchida a partir da leitura de um arquivo `.yaml`, onde o usuário pode configurar os estados da MEF.

Ao iniciar o programa, o mesmo realiza a inicialização de uma instância da MEF, e em seguida pede ao usuário a inserção de um *string* de entrada. Ao pressionar ↲, esta *string* é processada pela MEF, e, para cada símbolo de entrada, o estado interno é atualizado, e o símbolo de saída correspondente é impresso no console. Após todos os símbolos terem sido processados, o programa solicita ao usuário uma nova *string* de entrada. Para encerrar a execução do programa, o usuário pode usar o atalho de teclado Ctrl+C ou Ctrl+D.

```
        *** Finite State Machine ***
[NOTE] Press CTRL+C or CTRL+D anytime to exit

Input:  1011101101
Output: 00111110011

Input:  █
```

Figura 4: Execução da máquina de estado finito

Caso uma *string* contenha símbolos diferentes de **0** ou **1**, uma mensagem de erro é mostrada e o programa encerra sua execução.

3.2. Máquina de Turing

A máquina de Turing foi implementada como duas classes: uma para a lógica do programa (`TuringMachine`), e a outra para a fita (`Tape`).

Na classe `TuringMachine`, as quíntuplas da MT, são armazenadas em um dicionário, onde cada elemento do dicionário corresponde a um estado, cada estado é um dicionário que contém os símbolos de entrada, e cada símbolo de entrada é um dicionário que contém a própria quíntupla, isto é, estado atual (`present_state`), símbolo de entrada (`present_symbol`), símbolo de saída (`symbol_printed`), direção (`direction`) e o próximo estado (`next_state`).

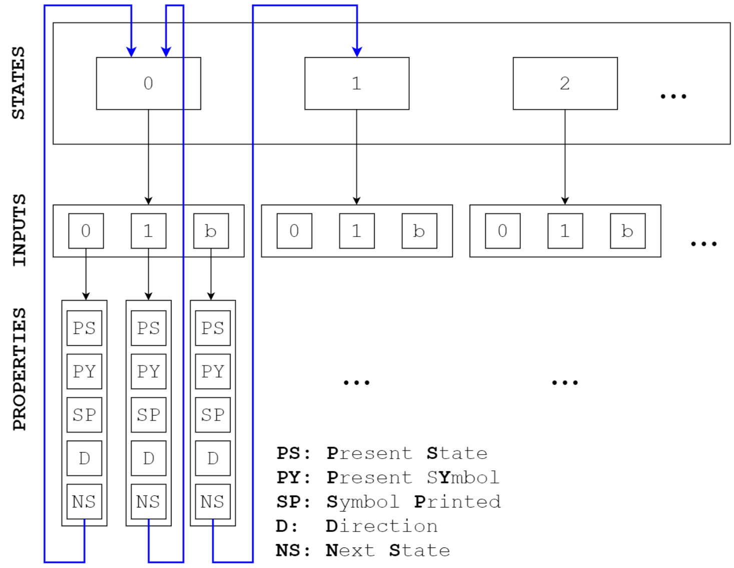

Figura 5: Lógica de funcionamento da máquina de Turing

A essência do funcionamento da MT é apresentada na figura acima, onde, para um dado estado, dependendo do símbolo de *input*, pode-se ter diferentes símbolos de *output*, direções e próximos estados.

Na classe `Tape`, a fita é implementada como uma lista de comprimento 70, que será preenchida com a *string* de entrada.

Tanto o dicionário de quíntuplas quanto os dados de entrada da fita são preenchidos a partir da leitura de um arquivo `.yaml`, onde o usuário pode configurar os parâmetros da MT.

Ao iniciar o programa, o mesmo realiza a inicialização de instâncias da MT e da fita, e exibe no console uma mensagem de confirmação: caso o usuário pressione Enter, a execução da MT é iniciada, caso o usuário pressione Ctrl + C ou Ctrl + D, o programa é encerrado. Partindo do pressuposto que o usuário decidiu iniciar a execução, o programa irá mostrar no console uma representação visual da fita, os dados nela presentes, e do cabeçote de leitura e escrita.

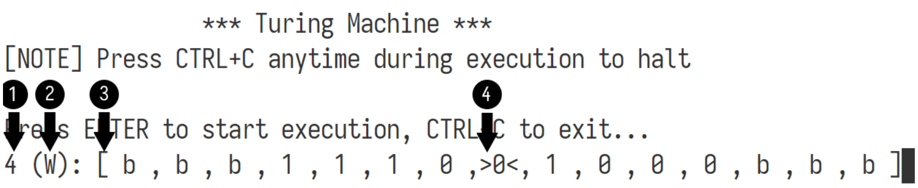

Figura 6: Execução da máquina de Turing

Os seguintes elementos podem ser vistos nesta imagem:

1. Iteração atual;
2. Ação atual do cabeçote (Leitura (`R`) ou Escrita (`W`));
3. Fita e seu conteúdo;
4. Posição atual do cabeçote (representado pelo par `> <`).

A cada iteração o cabeçote realiza uma etapa de leitura e uma etapa de escrita. Durante a leitura, o símbolo lido é comparado com os símbolos de *input* para aquele estado. Em seguida, o símbolo de *output* é escrito, a MT assume o novo estado, e o cabeçote se move em uma nova direção. Caso o próximo estado não exista, ou o símbolo de entrada não consta dentre os *inputs* do estado atual, o programa encerrará a sua execução.

Devido a impossibilidade de se determinar para quais conjuntos de entradas e quíntuplas a MT irá concluir sua execução ou ficará em um *loop* infinito (Halting Problem [3]), é possível estabelecer um número máximo de iterações para a MT através de um parâmetro no arquivo `.yaml`. Caso o usuário deseje, este também pode parar a execução do programa a qualquer momento, pressionando Ctrl+C.

Devido aos limites físicos de um computador real em relação à MT teórica — no caso, memória finita — caso o cabeçote passe dos limites da fita (estabelecido como 70), a execução do programa será encerrada.

4. Resultados

Com o desenvolvimento do código concluído, testes foram realizados para verificar a acurácia das soluções obtidas.

4.1. Máquina de Estado Finito

Os testes foram realizados a partir de exemplos extraídos do livro [2, cap. 9.3].

p. 730, Example 29:

Estado Atual	Próximo Estado		Saída
	0	**1**	
s_0	s_1	s_0	**0**
s_1	s_2	s_1	**1**
s_2	s_2	s_0	**1**

String de entrada: **01101**

Tabela 2: [2, p. 730, Example 29]

Para a MEF acima e este *input*, temos a seguinte saída:

```
        *** Finite State Machine ***
[NOTE] Press CTRL+C or CTRL+D anytime to exit

Insert input string: 01101

Input string:       0    1    1    0    1
States sequence:  ['s0', 's1', 's1', 's1', 's2', 's0']
Output string:      0    1    1    1    1    0
--------------------------------
Final state:       s0
Final output:       0 (rejected)
```

Figura 7: *Output* da MEF definida na Tabela 2 para o *input* proposto pelo "Example 29"

p. 731, Practice 43:

"For the machine M of Example 29, what output sequence is produced by the input sequence 11001?"

Dado esta entrada, temos a seguinte saída:

```
        *** Finite State Machine ***
[NOTE] Press CTRL+C or CTRL+D anytime to exit

Insert input string: 11001

Input string:       1    1    0    0    1
States sequence:  ['s0', 's0', 's0', 's1', 's2', 's0']
Output string:      0    0    0    1    1    0
--------------------------------
Final state:       s0
Final output:       0 (rejected)
```

Figura 8: *Output* da MEF definida na Tabela 2 para o *input* proposto pela questão "Practice 43"

p. 735, Practice 49

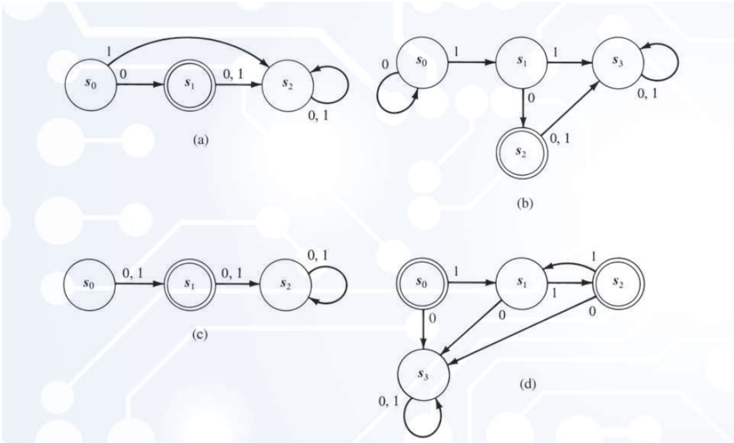

Figura 9: [2, p. 735, Practice 49]

Dado este conjunto de MEFs, cada uma é responsável pelo reconhecimento de uma expressão regular:

(a) 0 (b) 0*10 (c) $0 \vee 1$ (d) (11)*

Verifica-se, então estes conjuntos são de fato reconhecidos por estas MEFs.

```
Input string:        0
States sequence:  ['s0', 's1']
Output string:       0    1
----------------------------------
Final state:        s1
Final output:        1 (accepted)
```

Figura 10: (a) aceito

```
Input string:        0    0
States sequence:  ['s0', 's1', 's2']
Output string:       0    1    0
---------------------------------
Final state:        s2
Final output:        0 (rejected)
```

Figura 11: (a) rejeitado

```
Input string:        0     0     0     1     0
States sequence:  ['s0', 's0', 's0', 's0', 's1', 's2']
Output string:       0     0     0     0     0     1
---------------------------------
Final state:        s2
Final output:        1 (accepted)
```

Figura 12: (b) aceito

```
Input string:        1     0     0     0     1     0
States sequence:  ['s0', 's1', 's2', 's3', 's3', 's3', 's3']
Output string:       0     0     1     0     0     0     0
--------------------------------
Final state:        s3
Final output:        0 (rejected)
```

Figura 13: (b) rejeitado

```
Input string:        1
States sequence:  ['s0', 's1']
Output string:       0     1
----------------------------------
Final state:        s1
Final output:        1 (accepted)
```

Figura 14: (c) aceito

```
Input string:        1     0
States sequence:  ['s0', 's1', 's2']
Output string:       0     1     0
----------------------------------
Final state:        s2
Final output:        0 (rejected)
```

Figura 15: (c) rejeitado

```
Input string:        1     1     1     1     1     1
States sequence:  ['s0', 's1', 's2', 's1', 's2', 's1', 's2']
Output string:       1     0     1     0     1     0     1
---------------------------------
Final state:        s2
Final output:        1 (accepted)
```

Figura 16: (d) aceito

```
Input string:        1     1     1     1     1
States sequence:  ['s0', 's1', 's2', 's1', 's2', 's1']
Output string:       1     0     1     0     1     0
---------------------------------
Final state:        s1
Final output:        0 (rejected)
```

Figura 17: (d) rejeitado

4.2. Máquina de Turing

Os testes foram realizados a partir de exemplos extraídos do livro [2, cap. 9.4], exceto quando indicado.

p. 762, Example 40:

Quíntuplas da máquina:	`(0, 0, 1, 0, R)`
	`(0, 1, 0, 0, R)`
	`(0, b, 1, 1, L)`
	`(1, 0, 0, 1, R)`
	`(1, 1, 0, 1, R)`
		String de entrada: **0110**

Tabela 3: [2, p. 762, Example 40]

Dado os parâmetros acima, a MT executa corretamente, concluindo sua execução ao alcançar um estado não existente ou *input* não existente para aquele estado.

```
                *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
7 (R): [ b , b , b , 1 , 0 , 0 , 1 , 1 ,>b<, b ]
State doesn't exist! Halt!
```

Figura 18: Estado final da fita após a finalização da execução da MT definida na Tabela 3

p. 764, Practice 57:

Quíntuplas da máquina:

```
(0, 0, 0, 1, R)
(0, 1, 0, 0, R)
(0, b, b, 0, R)
(1, 0, 1, 0, R)
(1, 1, 1, 0, L)
```

Tabela 4: [2, p. 764, Practice 57]

a. *String* de entrada: **10**

Para esta *string* de entrada, a MT executa corretamente, concluindo sua execução ao alcançar um estado não existente/*input* não existente para aquele estado.

```
                  *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
2 (R): [ b , b , b , 0 , 0 ,>b<, b , b ]
State doesn't exist! Halt!
```

Figura 19: Estado final da fita após a finalização da execução da MT definida na Tabela 4 para o *input* **10**

b. *String* de entrada: **01**

Para esta *string* de entrada, a máquina fica presa em um *loop*, sendo necessário a parada manual de sua execução (Ctrl+C). Mesmo assim, a fita alcança um estado final, após o qual os seus dados não são mais alterados.

```
                  *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
21 (R): [ b , b , b , 0 ,>1<, b , b , b ]^C
Halting...
```

Figura 20: Estado da final fita após parada manual da execução da MT definida na Tabela 4 para o *input* **01**

c. *String* de entrada: **00**

Para esta *string* de entrada, o cabeçote começa a se mover para a direita sem fim. Devido aos limites físicos da fita previamente estabelecidos, ao alcançar a extremidade direita, a execução é abortada. Apesar disso, após certo ponto, os dados da fita não são alterados.

```
                  *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
5 (R): [ b , b , b , 0 , 1 , b , b ,>b<]
Right bound of tape reached! Halt!
```

Figura 21: Estado da final fita após parada da execução da MT definida na Tabela 4 ao ultrapassar a extremidade direita para o *input* **00**

Exemplo Próprio

Para demonstrar mais uma peculiaridade no funcionamento de uma MT, abaixo segue um esquema próprio.

Quíntuplas da máquina:

```
(0, 0, 1, 0, R)
(0, 1, 0, 0, R)
(0, b, b, 1, L)
(1, 1, 0, 1, L)
(1, 0, 1, 1, L)
(1, b, b, 0, R)
```

String de entrada: **000111000**

Tabela 5: Exemplo Próprio de MT

Para a MT acima, o seu comportamento independe da string de entrada. Dado que não seja vazia (**b**), a mesma ficará presa em um *loop* infinito, reescrevendo **0** em **1** e **1** em **0**, até alcançar um **b**; após esse ponto, a mesma reverte a direção e desfaz as alterações. Ao alcançar outro **b**, ela novamente reverte de direção e o processo se repete.

```
                  *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
4 (W): [ b , b , b , 1 , 1 , 1 , 0 ,>0<, 1 , 0 , 0 , 0 , b , b , b ]
```

Figura 22: MT definida na Tabela 5 após 4 iterações. O cabeçote está se movendo para a direita.

```
                  *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
17 (W): [ b , b , b , 1 ,>0<, 0 , 1 , 1 , 1 , 0 , 0 , 0 , b , b , b ]
```

Figura 23: MT definida na Tabela 5 após 17 iterações. O cabeçote está se movendo para a esquerda.

```
               *** Turing Machine ***
[NOTE] Press CTRL+C anytime during execution to halt

Press ENTER to start execution, CTRL+C to exit...
50 (R): [ b , b , b , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 ,>1<, b , b , b ]
Max number of iterations reached! Halt!
```

Figura 24: Estado da fita após parada da execução da MT definida na Tabela 5 ao alcançar um número máximo de iterações (50) para o *input* **000111000**

5. Conclusão

A teoria das linguagens formais e autômatos é fundamental no ramo da computação. Dentre as suas aplicações, encontram-se análise sintática, compiladores, inteligência artificial, verificação formal e muitas outras.

Através da construção e execução de máquinas de estado finito e máquinas de Turing, foi possível melhor entender a teoria por trás de seu funcionamento, suas aplicações, e em quais situações se deve preferir o uso de uma ou da outra.

Referências

[1] W. Tecumseh Fitch. "Toward a computational framework for cognitive biology: Unifying approaches from cognitive neuroscience and comparative cognition". Em: *Physics of Life Reviews* 11.3 (2014), pp. 329–364. ISSN: 1571-0645. DOI: https://doi.org/10.1016/j.plrev.2014.04.005. URL: https://www.sciencedirect.com/science/article/pii/S157106451400058X.

[2] J.L. Gersting. *Mathematical Structures for Computer Science*. W. H. Freeman, 2014. ISBN: 9781464193828. URL: https://books.google.com.br/books?id=BLCNBwAAQBAJ.

[3] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". Em: *Proceedings of the London Mathematical Society* s2-42.1 (jan. de 1937), pp. 230–265. ISSN: 0024-6115. DOI: 10.1112/plms/s2-42.1.230. eprint: https://academic.oup.com/plms/article-pdf/s2-42/1/230/4317544/s2-42-1-230.pdf. URL: https://doi.org/10.1112/plms/s2-42.1.230.