Conversor de Autômato Finito com Movimentos Vazios (AFe) para Autômato Finito Não Determinístico (AFN)

Fernanda Sousa de Assunção Vale[1]; Jhones de Sousa Soares[2]
[1,2] Universidade Federal do Maranhão – Cidade Universitária Dom Delgado – São Luís – Maranhão
{fernanda.sav, jhones.sousa}@discente.ufma.br

Resumo

Este artigo apresenta o desenvolvimento de um conversor de Autômatos Finitos com movimentos vazios (AFe) para Autômatos Finitos Não Determinísticos (AFN). Utilizando a linguagem de programação Python. Foi implementada uma solução que permite carregar uma definição de AFe por meio de arquivos JSON, processá-la e visualizar graficamente as transições dos autômatos gerados. Este trabalho explora conceitos fundamentais da teoria da computação, incluindo transições ε, fecho-ε e equivalência entre modelos de autômatos. Ademais, propõe uma interface gráfica interativa para facilitar o uso da ferramenta, destacando sua relevância tanto para o ensino quanto para a aplicação prática desses conceitos.

Palavras-chave: AFe, AFN, Python, conversor, JSON.

1. Introdução

Inicialmente, pode-se definir como autômatos finitos, modelos matemáticos amplamente utilizados para descrever e manipular linguagens regulares. Entre suas variações, destacam-se os Autômatos Finitos com movimentos vazios (AFe) e os Autômatos Finitos Não Determinísticos (AFN). O AFe se distingue pela presença de transições ε, que permitem ao autômato mudar de estado sem consumir nenhum símbolo de entrada. Já o AFN, embora não possua tais transições, mantém a característica de não determinismo por meio de transições para múltiplos estados (Marcus, 2009).

A conversão de AFe para AFN é um passo importante na teoria dos autômatos, pois demonstra a equivalência entre esses dois modelos e simplifica análises subsequentes, como a minimização de estados e a construção de expressões regulares equivalentes. Estudos como os de McNaughton (1961) exploram detalhadamente a equivalência estrutural e comportamental, bem como as aplicações práticas dessa transformação em sistemas computacionais. Conforme destacado por McNaughton (1961), a regularidade e a equivalência são elementos centrais para garantir a correção dos processos automáticos. Este trabalho tem como objetivo desenvolver uma ferramenta computacional para realizar essa conversão, explorando conceitos teóricos e técnicas de implementação em Python. Além disso, uma interface gráfica foi criada para permitir uma interação mais intuitiva com os dados de entrada e visualização dos resultados.

2. Fundamentação Teórica

Um sistema de Estados finitos é um modelo matemático de sistema com entradas e saídas discretas. Podendo, assim, assumir um número finito e predefinido de estados. Cada estado assume informações necessárias para determinar as próximas ações (MENEZES,2009).

Um autômato finito é formado, essencialmente, por 3 partes, assim como observado na Figura 1:

a) Fita: dispositivo de entrada que contém as informações a serem processadas. Esta é dividida em células onde cada uma armazena um símbolo. Tais símbolos pertencem a um alfabeto de entrada. Não sendo possível gravar novos dados sobre esta fita, as informações a serem processadas ocupam toda a extensão da mesma;
b) Unidade de controle: Reflete o estado correspondente da máquina. Este também possui uma subunidade, a unidade de leitura(ou cabeça da fita), a qual percorre uma célula por vez. Movimentando-se exclusivamente da direita para a esquerda. Esta possui por sua vez um número finito de estados possíveis;
c) Programa ou Função de transição: Função que comanda as leituras e define o estado da máquina. Esta função é uma função parcial, que, de acordo com o estado atual, determina o novo estado do autômato (MENEZES,2009).

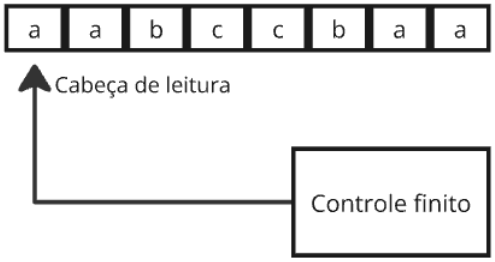

Figura 1: Representação dos componentes de um autômato finito

O Autômato Finito não Determinístico(AFN) representa uma generalização deste modelo de máquina, dando apoio ao estudo da teoria da computação e das linguagens formais.

Um Autômato Finito não Determinístico(AFN) é definido como uma 5-tupla, conforme demonstra a equação 1 , onde, a estrutura formal permite modelar sistemas computacionais de maneira precisa e lógica:

$$M = (\Sigma, Q, \delta, q_0, F) \quad (1)$$

- **Q** é um conjunto finito de estados;
- **Σ** é um alfabeto finito de símbolos de entrada;
- **δ**: $Q \times \Sigma \rightarrow 2^Q$ é a função de transição;
- **q0** $\in Q$ é o estado inicial;
- **F** $\subseteq Q$ é o conjunto de estados finais.

Sua função programa, ao processar uma entrada composta pelo estado corrente e símbolo lido, tem como saída um conjunto de novos estados. Logo, o autômato assume um conjunto de estados alternativos, processando de forma independente, sem compartilhar recursos com os demais estados, conforme representado na Figura 2:

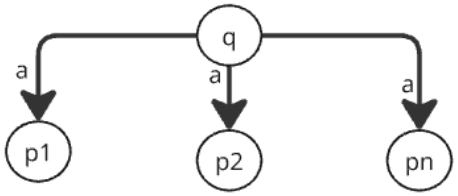

Figura 2: Representação de um Autômato Finito não Determinístico(AFN).

Para o autômato finito de movimentos vazios(AFe), podemos entendê-lo como uma generalização dos modelos não-determinísticos. Um movimento vazio representa a realização de uma transição sem a leitura de um símbolo presente na fita, facilitando assim algumas construções e demonstrações relacionadas ao estudo dos autómatos, assim representado na Figura 3 (MENEZES,2009).

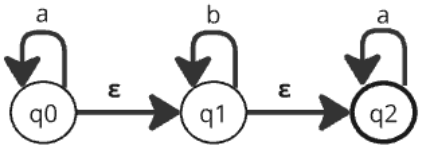

Figura 3: Representação de um Autômato Finito de Movimentos Vazios(AFe).

Podemos observar esta generalização em sua função programa:

$$M = (\Sigma, Q, \delta, q_0, F) \quad (2)$$

- **Q** é um conjunto finito de estados;
- **Σ** é um alfabeto finito de símbolos de entrada;
- **δ**: $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ é a função de transição;
- **q0** $\in Q$ é o estado inicial;
- **F** $\subseteq Q$ é o conjunto de estados finais.

A conversão de um AFe em um AFN requer a eliminação de todas as transições ε. Isso é feito por meio das seguintes etapas:

1. **Cálculo do fecho-ε**: Determina todos os estados que podem ser alcançados a partir de um estado inicial utilizando apenas transições ε;
2. **Redistribuição das transições**: Para cada estado e símbolo de entrada, as transições são reestruturadas para considerar todos os estados do fecho-ε;
3. **Atualização dos estados finais**: Um estado do AFN é final se ele ou qualquer um de seus estados no fecho-ε pertence ao conjunto de estados finais do AFe.

Conforme descrito por McNaughton (1961), a equivalência estrutural entre autômatos garante que modelos mais complexos possam ser simplificados sem perda de generalidade. A implementação computacional desse processo demanda organização eficiente dos dados,

geralmente por meio de estruturas como dicionários ou listas, e algoritmos que garantam a consistência das transições e estados finais (MENEZES,2009).

3. Desenvolvimento

O primeiro passo no desenvolvimento do conversor de AFe para AFN foi entender as definições e diferenças entre os Autômatos Finitos com Movimentos Vazios (AFe) e Autômatos Finitos Não Determinísticos (AFN). Para que a conversão funcionasse corretamente, identificou-se as regras que ditam as transformações, especialmente aquelas voltadas para a remoção de transições vazias, que podem complicar o comportamento do autômato.

Para poder modelar os autômatos, utilizou-se a estrutura dos grafos, onde cada estado do autômato é representado por um nó, e as transições, incluindo as vazias, como arestas. Essa representação permite a visualização do autômato. No Código 1 a seguir, é demonstrado a construção da transformação de um AFe para AFN.

Código 1: Afe_to_AFN.py

```
import graphviz
from collections import defaultdict
def visualize_automaton(transitions, start_state, final_states, title):
    dot = graphviz.Digraph(format="png", engine="dot")
    dot.attr(rankdir="LR")
    for state in transitions:
        if state in final_states:
            dot.node(state, shape="doublecircle")
        else:
            dot.node(state)
    dot.node("start", shape="none")
    dot.edge("start", start_state)
    for state, moves in transitions.items():
        edges = defaultdict(list)
        for symbol, next_states in moves.items():
            for next_state in next_states:
                edges[(state, next_state)].append(symbol)
        for (src, dest), symbols in edges.items():
             dot.edge(src, dest, label=",".join(symbols))
    dot.attr(label=title, fontsize="16")
    dot.render(f"{title.replace(' ', '_')}", cleanup=True, view=True)
def e_closure(state, transitions):
    stack = [state]
    closure = set(stack)
    while stack:
        current_state = stack.pop()
        for next_state in transitions[current_state].get("ε", []):
            if next_state not in closure:
                closure.add(next_state)
                stack.append(next_state)
    return closure
def convert_afe_to_afn(afe_transitions, start_state, final_states):
    afn_transitions = defaultdict(lambda: defaultdict(set))
    closures = {state: e_closure(state, afe_transitions) for state in
afe_transitions}
    all_symbols = set(
         symbol
         for state_transitions in afe_transitions.values()
         for symbol in state_transitions.keys()
         if symbol != "ε")
     for state, closure in closures.items():
         for symbol in all_symbols:
             reachable_states = set()
```

```
43.                 for intermediate_state in closure:
44.                     reachable_states.update(afe_transitions
                        [intermediate_state].get(symbol, []))
45.                 for reachable_state in reachable_states:
46.                     afn_transitions[state][symbol].
                        update(closures[reachable_state])
47.             if closure.intersection(final_states):
48.                 final_states.update(closure)
49.     return {state: dict(moves) for state, moves in
    afn_transitions.items()}, final_states
```

O código acima mostra o processo de transformação dos autômatos, onde foram criada classes que representassem estados e transições, conforme linha 3 do Código 1. Foi também realizado a manipulação das transições vazias, ocasionando eventual remoção, conforme exigido na conversão para AFN. É possível verificar essa manipulação na linha 22 e 32 do Código 1, sendo possível alcançar todos os estados possíveis, gerando o AFN equivalente ao AFe original. Ao usar bibliotecas como *grapgviz*, gerou-se as representações visuais dos autômatos, visando mostrar os resultados da conversão.

No Código 2 a seguir, é possível notar a chamada do Código 1 e o uso de elementos gráficos e de arquivos JSON para a leitura das informações do autômato AFe usado na conversão.

Código 2: main.py

```
1.  import graphviz
2.  import json
3.  from collections import defaultdict
4.  import tkinter as tk
5.  from tkinter import messagebox, filedialog
6.  import re
7.  from src.AFe_to_AFN import convert_afe_to_afn, visualize_automaton
8.  def load_automaton_from_json(filepath):
9.      with open(filepath, 'r', encoding='utf-8') as file:
10.         data = json.load(file)
11.     return data['transitions'], data['start_state'],
    set(data['final_states'])
12. def select_file():
13.     filepath = filedialog.askopenfilename(filetypes=[("JSON files",
    "*.json")])
14.     if filepath:
15.        afe_transitions, start_state, final_states =
    load_automaton_from_json(filepath)
16.        print("Transições do AFe:")
17.        print(afe_transitions)
18.        print("\nEstado Inicial:", start_state)
19.        print("\nEstados Finais:", final_states)
20.        visualize_automaton(afe_transitions, start_state, final_states,
    f"Automato - AFe")
21.        afn_transitions, afn_final_states =
    convert_afe_to_afn(afe_transitions, start_state, final_states)
22.         print("\nTransições do AFN:")
23.         print(afn_transitions)
24.         print("\nEstados Finais do AFN:", afn_final_states)
25.         visualize_automaton(afn_transitions, start_state,
    afn_final_states, f"Automato - AFN")
26. def main():
27.     root = tk.Tk()
28.     root.title("Conversor de AFe para AFN")
29.     frame = tk.Frame(root, padx=10, pady=10)
30.     frame.pack(padx=10, pady=10)
31.     label = tk.Label(frame, text="Selecionar arquivo JSON com o AFe")
```

```
    label.pack(pady=5)
    button = tk.Button(frame, text="Selecionar arquivo",
command=select_file)
    button.pack(pady=5)
    root.mainloop()
if __name__ == "__main__":
    main()
```

No Código 2 há a parte da possibilidade dos testes para garantir a conversão, pois é possível chamar diferentes arquivos JSON que contém informações de AFe para testar o código. Nele foi também criado uma interface gráfica com tkinter, gerando uma janela para acionar a seleção desses arquivos que contém a descrição dos autômatos, bem como a leitura e carregamento dos dados.

Após esses passos foi possível visualizar os autômatos com o uso das bibliotecas que permitiram gerar as representações visuais que auxiliam tanto na validação do comportamento quanto na apresentação dos resultados.

4. Resultados

O processo de conversão de Autômato Finito com Movimentos Vazios (AFe) para Autômato Finito Não Determinístico (AFN) demonstrou ser eficiente e preciso. O algoritmo calculou corretamente o fecho-ε, uma etapa decisiva na eliminação das transições vazias que distinguem o AFe. Nos testes realizados, o conversor lidou efetivamente com uma variedade de casos, desde exemplos simples até aqueles mais complexos, com múltiplas transições e estado

Uma demonstração de seus resultados consta na Figura 4 a seguir, representando um AFe.

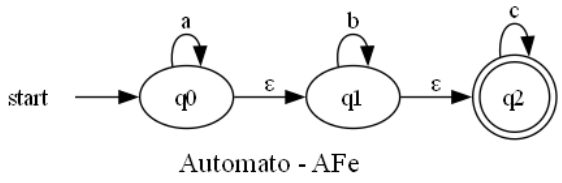

Figura 4: Autômato Finito de Movimentos Vazios (AFe) utilizado para comprovar os resultados

Cuja manipulação resultou na Tabela 1 a seguir:

	a	b	c
q0	{q0, q1, q2}	{q1, q2}	{q2}
q1	{}	{q1, q2}	{q2}
q2	{}	{}	{q2}

Tabela 1: Tabela resultante da manipulação do AFe

A saída do código do projeto gerou o autômato AFN conforme Figura 5 demonstrada logo em seguida. Se comparar com a Tabela 1, é possível notar que o autômato equivale a ela,

comprovando o sucesso na conversão.

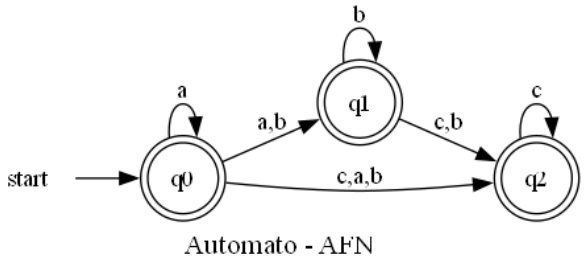

Figura 5: Autômato Finito Não Determinístico (AFN) resultante da conversão.

O AFe e o AFN são funcionalmente equivalentes. Ambos reconhecem a mesma linguagem e permitem identificar cadeias válidas contidas em sua estrutura.

5. Conclusão

O projeto de conversão de Autômato Finito com Movimentos Vazios (AFe) para Autômato Finito Não Determinístico (AFN) foi implementado com sucesso, cumprindo todos os requisitos estabelecidos. Ele permite a leitura e processamento de descrições de autômatos em formato JSON, transformando as informações sobre os mesmos em representações visuais. A inclusão da representação gráfica utilizando a biblioteca *graphviz*, juntamente com uma interface gráfica de usuário construída com *tkinter*, torna o processo tanto visualmente intuitivo quanto fácil de navegar, facilitando a compreensão do fluxo e transformação dos autômatos.

Além disso, se destaca como uma ferramenta prática que contribui para o campo educacional, especialmente para estudantes de Linguagens Formais e Autômatos. A visualização e transformação dos autômatos é realmente benéfica para entender conceitos abstratos de uma maneira visual e intuitiva. Ao tornar este processo interativo, a implementação promovida neste projeto ajuda a aprofundar o entendimento de como os autômatos funcionam e como técnicas de conversão podem ser aplicadas em cenários mais amplos.

Sendo assim, com as implementações, o projeto estabelece uma base para futuras expansões. Há potencial para melhorar e aumentar suas funcionalidades, como incorporar suporte para uma variedade de outros tipos de autômatos, tais como autômatos determinísticos. Portanto, o conversor de AFe para AFN não só oferece uma solução eficaz, mas também um ponto de partida promissor para melhorias no estudo de autômatos e seus usos. Ao facilitar o entendimento prático dos conceitos teóricos através da visualização, ele apoia o desenvolvimento educacional e técnico.

Referências

RAMOS, MARCUS V. M. **Linguagens formais: teoria, modelagem e implementação.** 1ª ed. Porto Alegre: Bookman, 2009.

MENEZES, Paulo Blauth. Linguagens Formais e Autômatos: Volume 3 da Série Livros Didáticos Informática UFRGS. [s.l.] Bookman Editora, 2009.

McNaughton, R. (1961). The Theory of Automata, a Survey. Advances in Computers Volume 2, 379–42