

CONVERSOR DE AUTÔMATO FINITO COM MOVIMENTOS VAZIOS (AFE) PARA AUTÔMATO FINITO NÃO DETERMINÍSTICO (AFN)

Disciplina: Linguagens Formais
e Autômatos

Orientador: Dr. Thales L. A. Valente



Universidade Federal do Maranhão

São Luís - Maranhão. | 14 e 16 de jan. 2025

INTEGRANTES - GRUPO 3

FERNANDA SOUSA DE ASSUNÇÃO VALE
Matrícula: 2022024316

JHONES DE SOUSA SOARES
Matrícula: 2020002730

■ ÍNDICE

04 Introdução

06 Fundamentação Teórica

08 Desenvolvimento

19 Análise de Resultados

23 Conclusão

24 Programa em Funcionamento

25 Referências

1. INTRODUÇÃO

■ O que é um autômato finito?

Dispositivo de tamanho finito com certas partes especificadas como entradas e saídas, de modo que o que acontece nas saídas é determinado, ou pelo menos sua função de distribuição de probabilidade é determinada, pelo que aconteceu nas entradas.

Fita

Unidade de controle

Programa ou função de transição

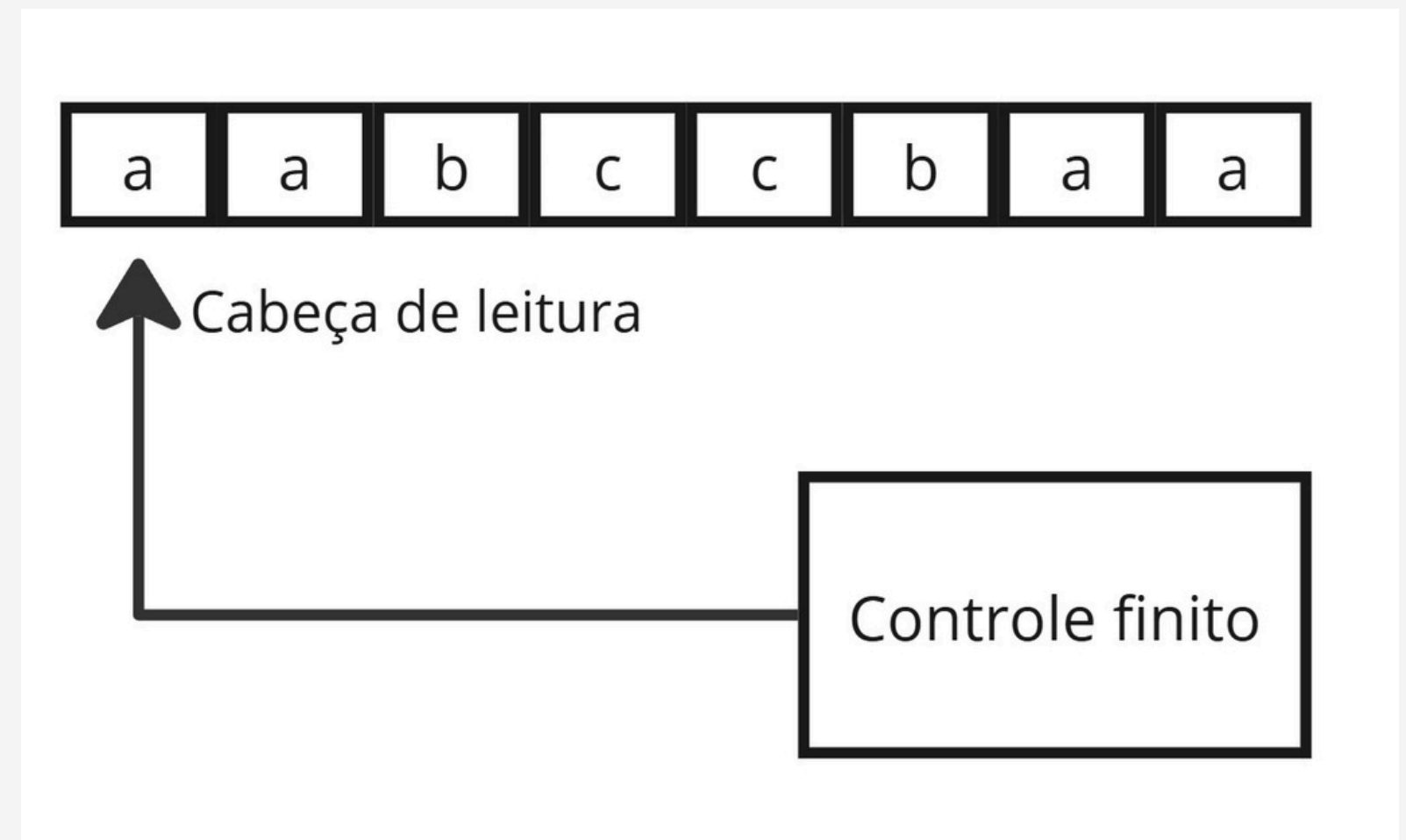


Figura 1.1 - Exemplificação da composição de um automato finito

1. INTRODUÇÃO

■ Objetivos do trabalho

Implementar com sucesso a transformação de um Autômato Finito de movimentos vazios(AF ϵ) para um Autômato Finito não determinístico(AFD).

- Remoção das operações movimentos vazios;
- Visualização dos automatos formados

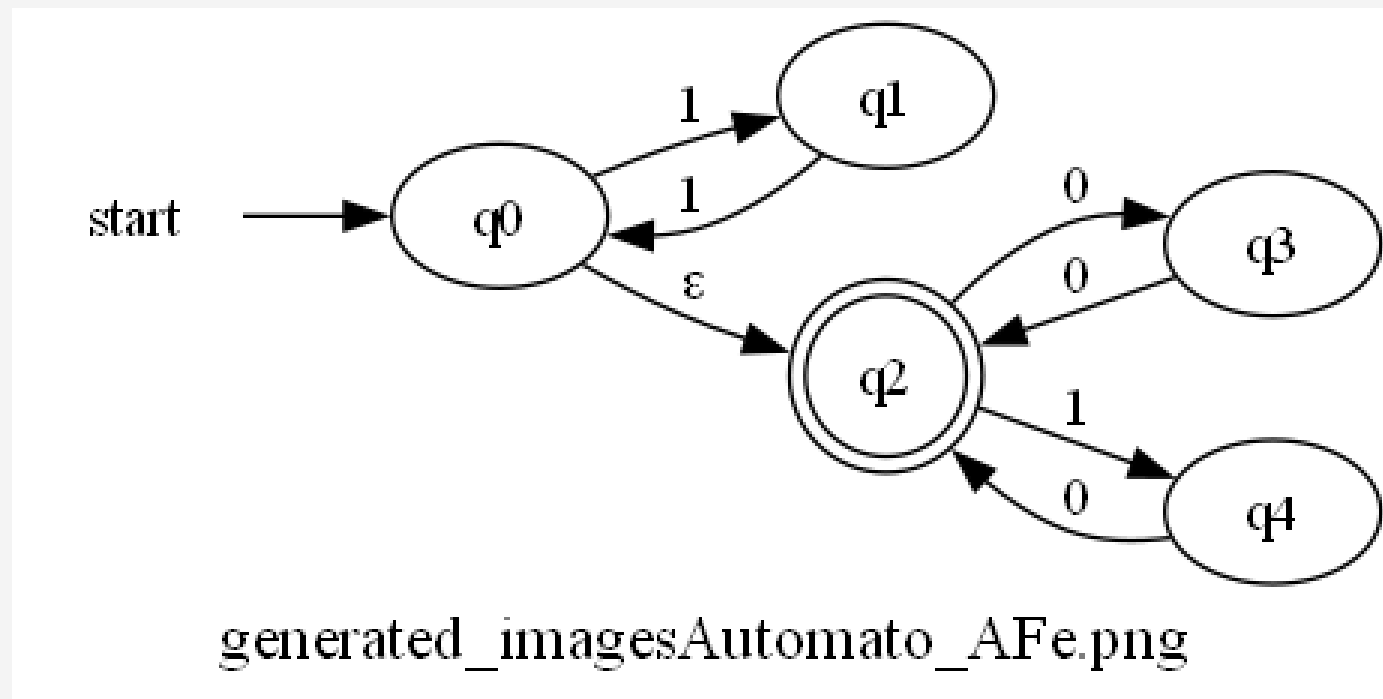


Figura 1.2 - Automato Finito com movimentos vazios

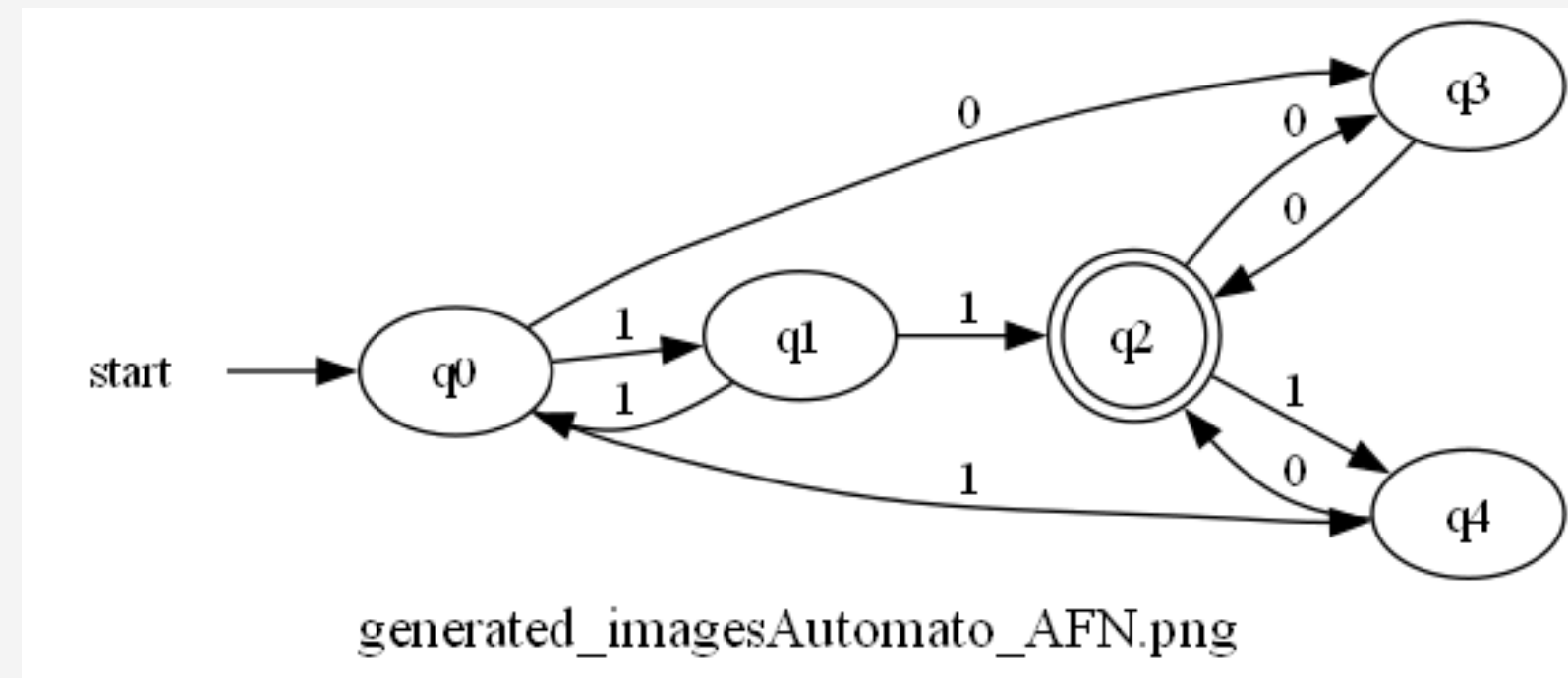


Figura 1.3 - Automato Finito não Determinístico equivalente

2. FUNDAMENTAÇÃO TEÓRICA

■ Autômato Finito não determinístico (AFN)

Um Autômato Finito Não-Determinístico (AFN) M é uma 5-upla:

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

Σ : alfabeto de símbolos de entrada;

Q : conjunto de estados possíveis do autômato, o qual é finito;

δ : função programa ou função de transição:

- $\delta: Q \times \Sigma \rightarrow 2^Q$

q_0 : estado inicial, tal que q_0 é elemento de Q ;

F : conjunto de estados finais, tal que F está contido em Q .

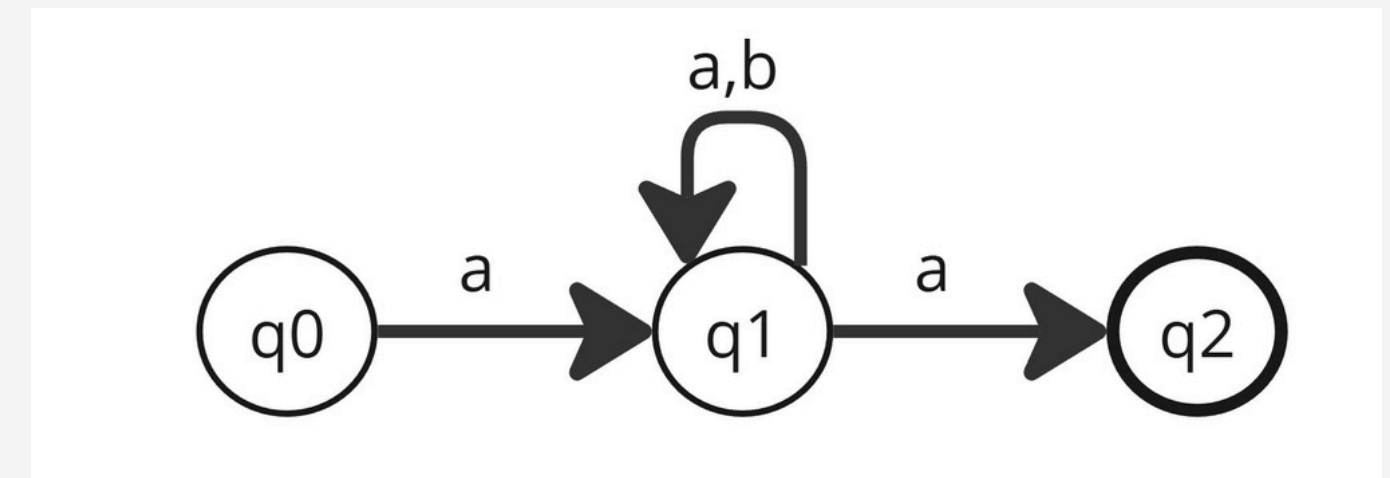


Figura 2.1 - Automato Finito não determinístico

2. FUNDAMENTAÇÃO TEÓRICA

■ Autômato Finito com Movimentos Vazios(AFN_{ϵ})

Um Autômato Finito de movimentos vazios(AFN_{ϵ}) M é uma 5-upla:

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

Σ : alfabeto de símbolos de entrada;

Q : conjunto de estados possíveis do autômato, o qual é finito;

δ : função programa ou função de transição:

- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$

q_0 : estado inicial, tal que q_0 é elemento de Q ;

F : conjunto de estados finais, tal que F está contido em Q .

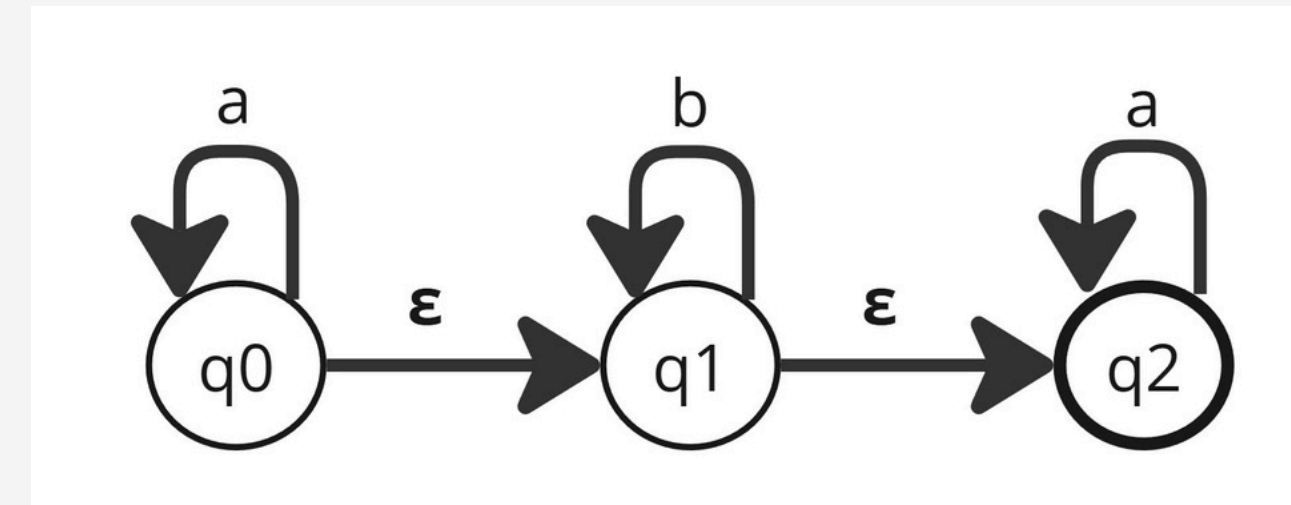


Figura 2.2 - Autômato Finito com movimentos vazios

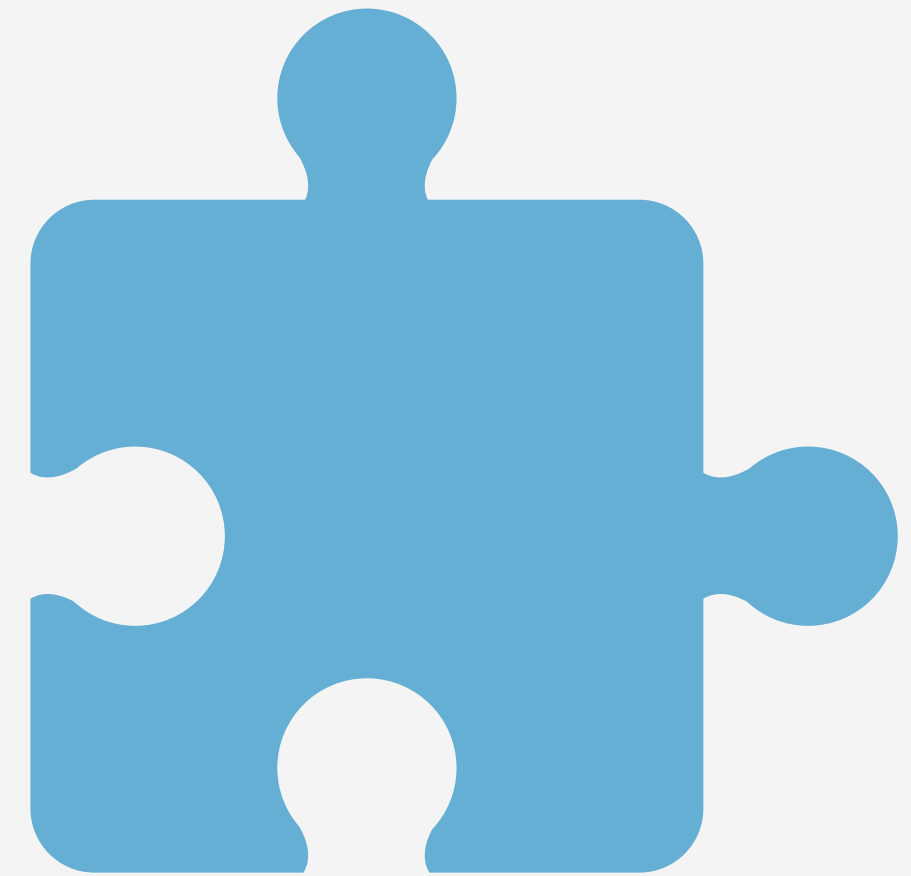
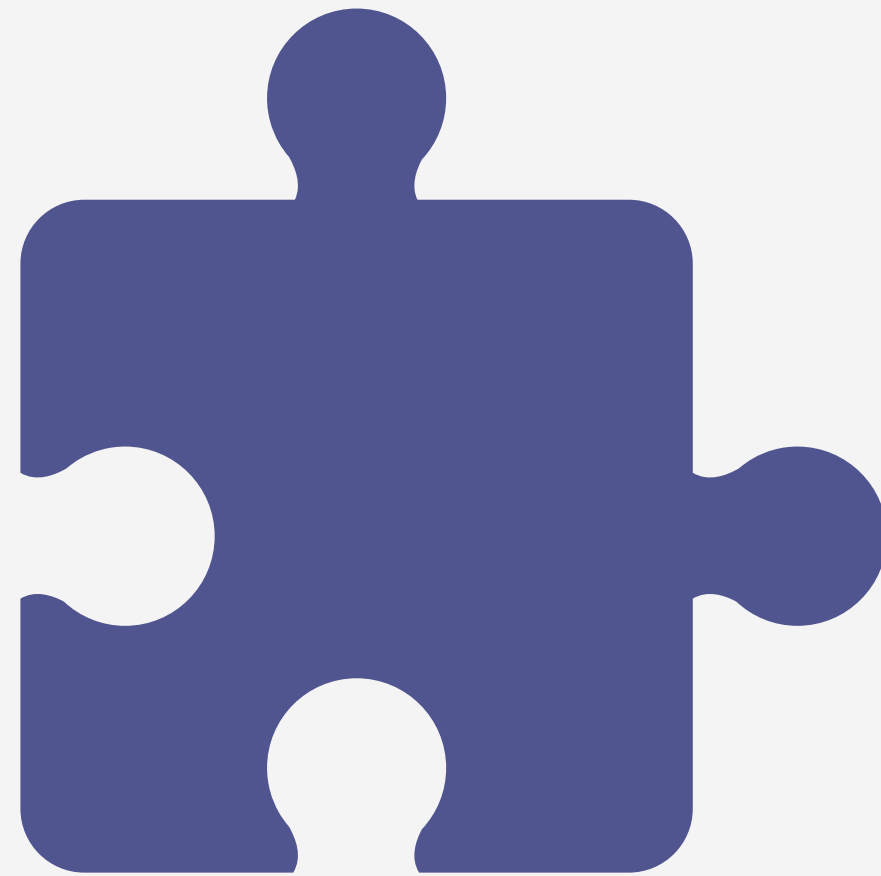
3. DESENVOLVIMENTO

PARTE 1

Código
“AFe_to_AFN.py”.

PARTE 2

Código “main.py” +
arquivos JSON



PARTE 1

Código

AFe_to_AFN.py

1. Sobre as Bibliotecas

```
1  #Biblioteca para gerar visualizações gráficas de autômatos.  
2  import graphviz  
3  #Estrutura de dicionário que inicializa automaticamente  
4  #valores padrão, útil para evitar KeyError.  
5  from collections import defaultdict
```

Figura 3.1 - Bibliotecas do arquivo AFe_to_AFN.py

PARTE 1

Código

AFe_to_AFN.py

2. Sobre as Funções

```
10 def visualize_automaton(transitions, start_state, final_states, title):
11     #ver os autômatos:
12     #Cria um objeto Digraph do graphviz para gerar o grafo em formato PNG
13     dot = graphviz.Digraph(format="png", engine="dot")
14     #Define a direção do grafo da esquerda para a direita
15     dot.attr(rankdir="LR")
16
17     #Adiciona os estados ao grafo.
18     for state in transitions:
19         #Se um estado estiver na lista de estados finais,
20         #ele recebe a forma de "doublecircle" (indica estado final).
21         if state in final_states:
22             dot.node(state, shape="doublecircle")
23         else:
24             dot.node(state) #Estados comuns são representados normalmente.
25     #add q0
26     #Cria um nó fictício "start" sem forma (shape="none") para indicar visualmente o estado inicial.
27     dot.node("start", shape="none")
28     dot.edge("start", start_state) #Conecta "start" ao start_state com uma aresta.
```

Figura 3.2 - Função para visualizar autômato do arquivo

PARTE 1

Código

AFe_to_AFN.py

2. Sobre as Funções

```
29     #adiciona transições
30     #Percorre as transições do autômato.
31     for state, moves in transitions.items():
32         #Usa defaultdict(list) para armazenar transições organizadas por estado origem → destino.
33         edges = defaultdict(list)
34         #Agrupar os símbolos que levam de um estado a outro.
35         for symbol, next_states in moves.items():
36             for next_state in next_states:
37                 edges[(state, next_state)].append(symbol)
38         #Para cada transição src -> dest, cria uma aresta com um rótulo contendo os
39         #símbolos separados por vírgula.
40         for (src, dest), symbols in edges.items():
41             dot.edge(src, dest, label=",".join(symbols))
42     #Define o título do grafo.
43     dot.attr(label=title, fontsize="16")
44     #Renderiza e exibe o grafo, nomeando o arquivo com base no título.
45     dot.render(f"{title.replace(' ', '_')}", cleanup=True, view=True)
```

Figura 3.2 - Função para visualizar autômato do arquivo

PARTE 1

Código

AFe_to_AFN.py

2. Sobre as Funções

```
47 #Calcula o fecho-ε (ε-closure) de um estado, ou seja,  
48 # todos os estados acessíveis apenas por transições vazias (ε).  
49 def e_closure(state, transitions):  
50     #Inicializa uma pilha (stack) e um conjunto (closure) com o estado inicial  
51     stack = [state]  
52     closure = set(stack)  
53     #Enquanto houver estados na pilha, remove o topo (pop()).  
54     while stack:  
55         current_state = stack.pop()  
56         #Para cada estado acessível via transição ε, adiciona ao  
57         # conjunto closure se ainda não estiver lá.  
58         #O novo estado é colocado na pilha para continuar a busca recursiva.  
59         for next_state in transitions[current_state].get("ε", []):  
60             if next_state not in closure:  
61                 closure.add(next_state)  
62                 stack.append(next_state)  
63     #Retorna o fecho-ε do estado inicial.  
64     return closure
```

Figura 3.3 - Função para calcular o fecho-ε para um estado específico

2. Sobre as Funções

PARTE 1

Código

AFe_to_AFN.py

```
66 #Converte um Autômato Finito com  $\epsilon$ -transições (AFE) em um
67 # Autômato Finito Não Determinístico (AFN).
68 def convert_afe_to_afn(afe_transitions, start_state, final_states):
69
70     #Inicializa a estrutura de transições do AFN, onde cada
71     # estado tem um dicionário de símbolos mapeando para conjuntos de estados alcançáveis.
72     afn_transitions = defaultdict(lambda: defaultdict(set))
73     #calcula o fecho- $\epsilon$  para cada estado no autômato e armazena no dicionário closures.
74     closures = {state: e_closure(state, afe_transitions) for state in afe_transitions}
75     #Coleta todos os símbolos usados no autômato, excluindo  $\epsilon$ .
76     all_symbols = set(
77         symbol
78         for state_transitions in afe_transitions.values()
79         for symbol in state_transitions.keys()
80         if symbol != " $\epsilon$ "
81     )
```

Figura 3.4 - Função para converter o AFe em AFN

PARTE 1

Código

AFe_to_AFN.py

2. Sobre as Funções

```
83     #monta o AFN
84     #Para cada estado e seu fecho-ε, inicia a análise das transições para cada símbolo.
85     for state, closure in closures.items():
86         for symbol in all_symbols:
87             reachable_states = set()
88             #Percorre os estados no fecho-ε e verifica para onde podem ir com determinado símbolo.
89             for intermediate_state in closure:
90                 reachable_states.update(afe_transitions[intermediate_state].get(symbol, []))
91             #Para cada estado acessível diretamente, adiciona ao novo autômato o
92             # conjunto de estados acessíveis via fecho-ε.
93             for reachable_state in reachable_states:
94                 afn_transitions[state][symbol].update(closures[reachable_state])
95
96     #Se qualquer estado no fecho-ε for um estado final, o estado original
97     # também deve ser tratado como final.
98     #atualiza os estados finais
99     if closure.intersection(final_states):
100         final_states.update(closure)
101     #Retorna as transições do AFN e a lista de estados finais atualizada.
102     return {state: dict(moves) for state, moves in afn_transitions.items()}, final_states
103
```

Figura 3.4 - Função para converter o AFe em AFN

PARTE 2

Código “main.py” + arquivos JSON

```
{
  "transitions": {
    "q0": {"ε": ["q1", "q2"]},
    "q1": {"a": ["q1", "q3"]},
    "q2": {"b": ["q2"]},
    "q3": {"ε": ["q4"]},
    "q4": {"c": ["q4"]}
  },
  "start_state": "q0",
  "final_states": ["q4"]
}
```

Figura 3.5 - Estrutura do arquivo JSON

PARTE 2

Código “main.py” + arquivos JSON

```
2  #Utilizado para visualizar o autômato.
3  import graphviz
4  #Para carregar o autômato a partir de um arquivo .json.
5  import json
6  #Um tipo de dicionário que inicializa automaticamente valores padrão.
7  from collections import defaultdict
8  #biblioteca para interface gráfica (tkinter é a
9  # biblioteca principal, messagebox e filedialog são módulos auxiliares).
10 import tkinter as tk
11 from tkinter import messagebox, filedialog
12 #Biblioteca para expressões regulares (não está sendo
13 # usada no código, pode ser um resquício de código antigo).
14 import re
15 #Funções importadas de um módulo (src.AFe_to_AFN),
16 # que convertem autômatos e os visualizam.
17 from src.AFe_to_AFN import convert_afe_to_afn, visualize_automaton
```

Figura 3.6 - Importação das bibliotecas e funções

PARTE 2

Código “main.py” + arquivos JSON

```
18
19 #Carrega os dados do autômato a partir de um arquivo JSON.
20 def load_automaton_from_json(filepath):
21     #Abre o arquivo JSON (filepath) em modo leitura
22     # ('r'), garantindo compatibilidade com caracteres especiais (utf-8).
23     with open(filepath, 'r', encoding='utf-8') as file:
24         #Usa json.load(file) para carregar o conteúdo do arquivo em um dicionário Python.
25         data = json.load(file)
26     #Retorna as transições do autômato, o estado inicial e
27     # os estados finais (convertidos em um set).
28     return data['transitions'], data['start_state'], set(data['final_states'])
29
```

Figura 3.7 - Função para carregar o arquivo JSON

PARTE 2

Código “main.py” + arquivos JSON

```
30 #Abre uma janela para seleção de arquivos JSON e executa a conversão do AFe para AFN.
31 def select_file():
32     #Abre uma caixa de diálogo para o usuário selecionar um arquivo .json
33     filepath = filedialog.askopenfilename(filetypes=[("JSON files", "*.json")])
34     #Verifica se um arquivo foi selecionado.
35     if filepath:
36         #Carrega o autômato do arquivo JSON.
37         afe_transitions, start_state, final_states = load_automaton_from_json(filepath)
38
39         #Exibe no console as transições do AFe, o estado inicial e os estados finais.
40         print("Transições do AFe:")
41         print(afe_transitions)
42         print("\nEstado Inicial:", start_state)
43         print("\nEstados Finais:", final_states)
44
45         #Gera e exibe um gráfico do AFe.
46         visualize_automaton(afe_transitions, start_state, final_states, f"Automato - AFe")
47
48         #Converte o AFe para AFN chamando convert_afe_to_afn().
49         afn_transitions, afn_final_states = convert_afe_to_afn(afe_transitions, start_state, final_states)
50
51         #Exibe no console as transições do AFN e seus estados finais.
52         print("\nTransições do AFN:")
53         print(afn_transitions)
54         print("\nEstados Finais do AFN:", afn_final_states)
55
56         #Gera e exibe um gráfico do AFN.
57         visualize_automaton(afn_transitions, start_state, afn_final_states, f"Automato - AFN")
58
```

Figura 3.8 - Função para geração de display dos resultados

PARTE 2

Código “main.py” + arquivos JSON

```
60 #Cria a interface gráfica para o conversor de AFe para AFN.
61 def main():
62     #Cria a janela principal (Tk()) e define o título.
63     root = tk.Tk()
64     root.title("Conversor de AFe para AFN")
65
66     #Cria um frame dentro da janela principal,
67     # com espaçamento interno e externo de 10 pixels.
68     frame = tk.Frame(root, padx=10, pady=10)
69     frame.pack(padx=10, pady=10)
70
71     #Adiciona um rótulo (Label) com a instrução para selecionar um arquivo JSON.
72     label = tk.Label(frame, text="Selecionar arquivo JSON com o AFe")
73     label.pack(pady=5)
74
75     #Adiciona um botão que, quando clicado, chama select_file().
76     button = tk.Button(frame, text="Selecionar arquivo", command=select_file)
77     button.pack(pady=5)
78
79     #Mantém a janela aberta aguardando interação do usuário.
80     root.mainloop()
```

Figura 3.9 - Função para geração e display da interface

PARTE 2

Código “main.py” + arquivos JSON

```
82  #Verifica se o script está sendo
83  # executado diretamente (__name__ == "__main__").
84  #Se for, chama main() para iniciar a interface gráfica.
85  if __name__ == "__main__":
86      main()
```

Figura 3.8 - Função principal



4. ANÁLISE DE RESULTADOS



RESULTADOS ESPERADOS VS OBTIDOS

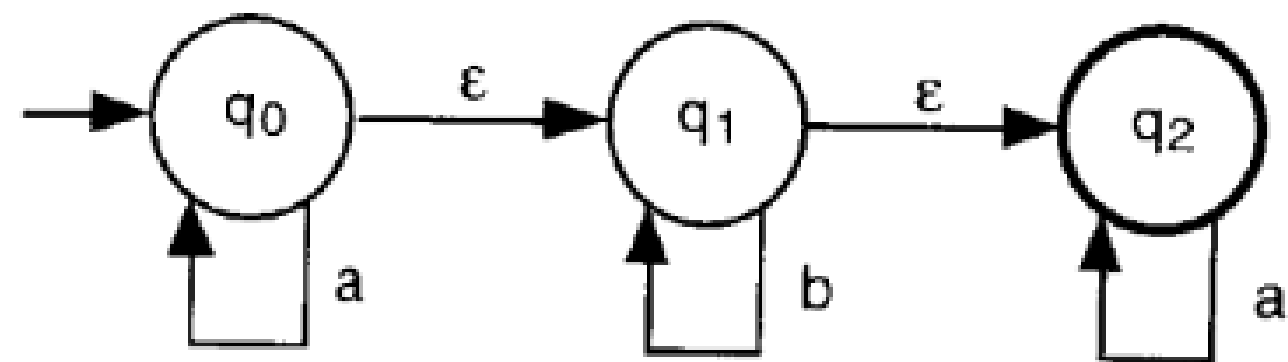


Figura 2.15 Grafo do Autômato Finito com Movimentos Vazios

Figura 4.1 - AFe de exemplo para teste do projeto

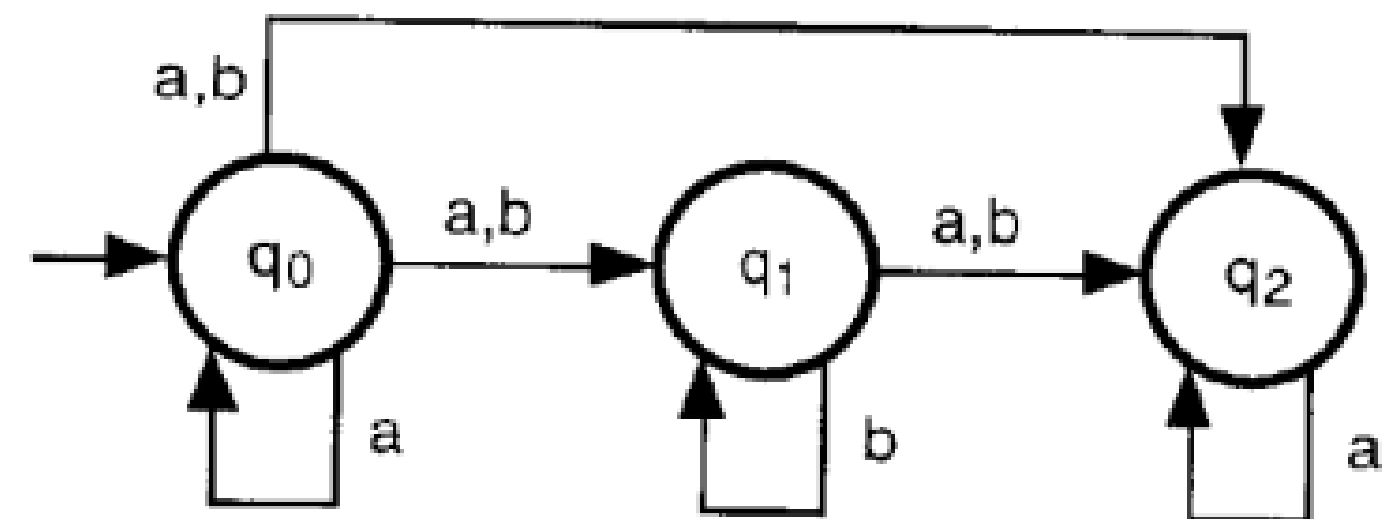


Figura: Grafo do AFN de M_9

Figura 4.2 - AFN equivalente ao AFe anterior

RESULTADOS ESPERADOS VS OBTIDOS

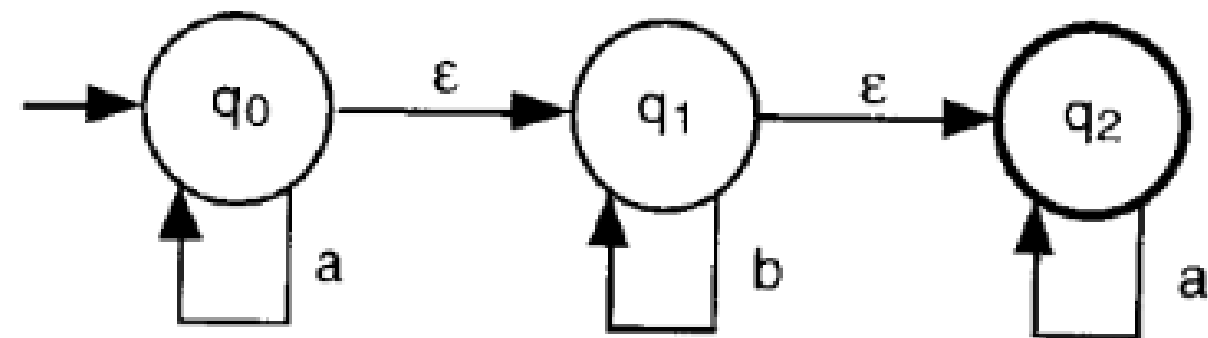
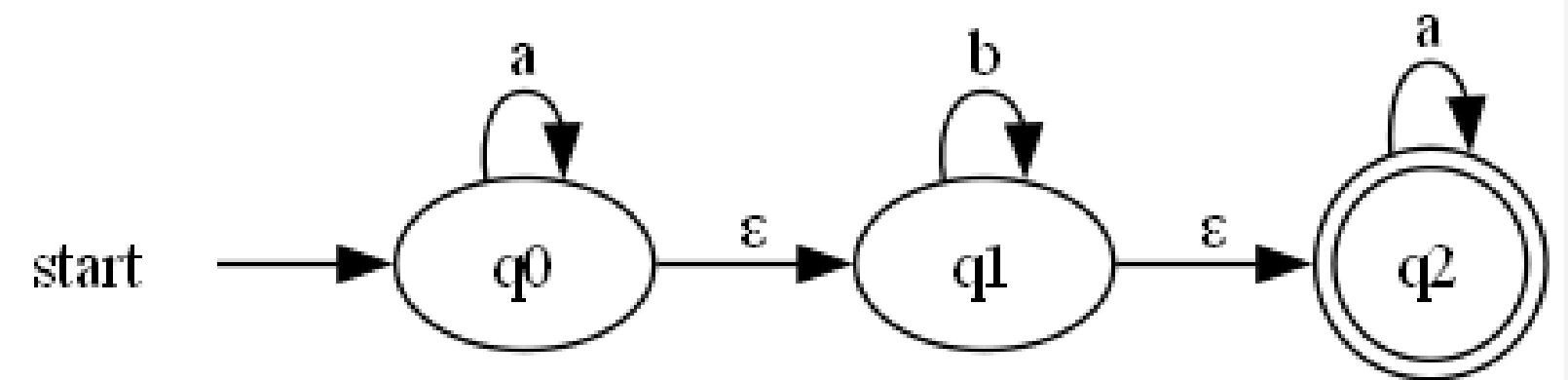


Figura 2.15 Grafo do Autômato Finito com Movimentos Vazios

Figura 4.3 - AFe esperado



Automato - AFe

Figura 4.4 - AFe obtido

RESULTADOS ESPERADOS VS OBTIDOS

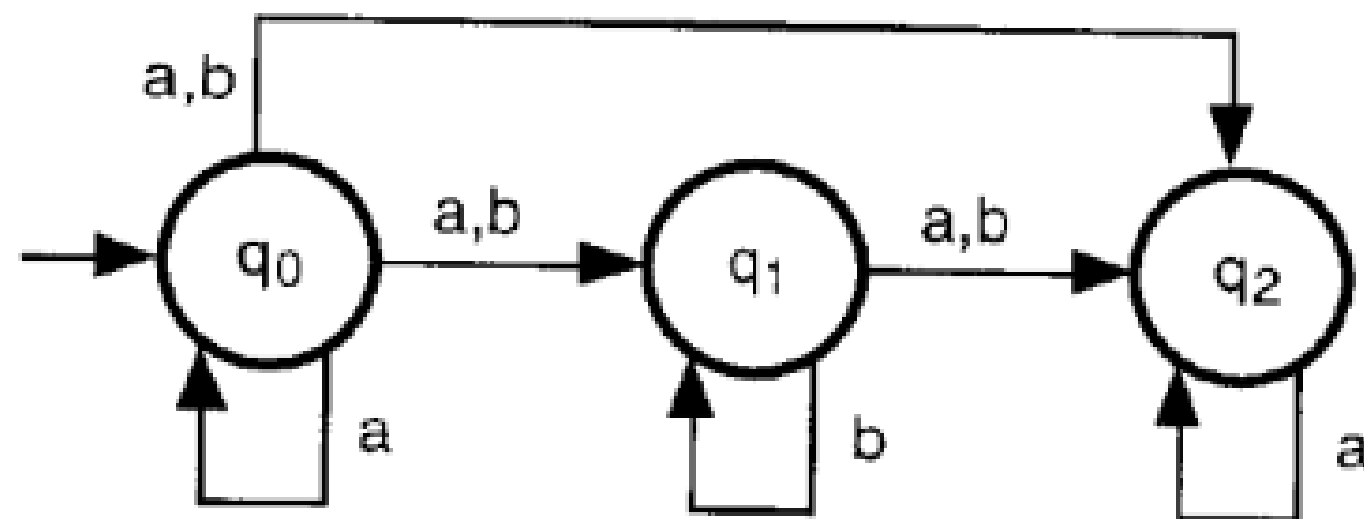
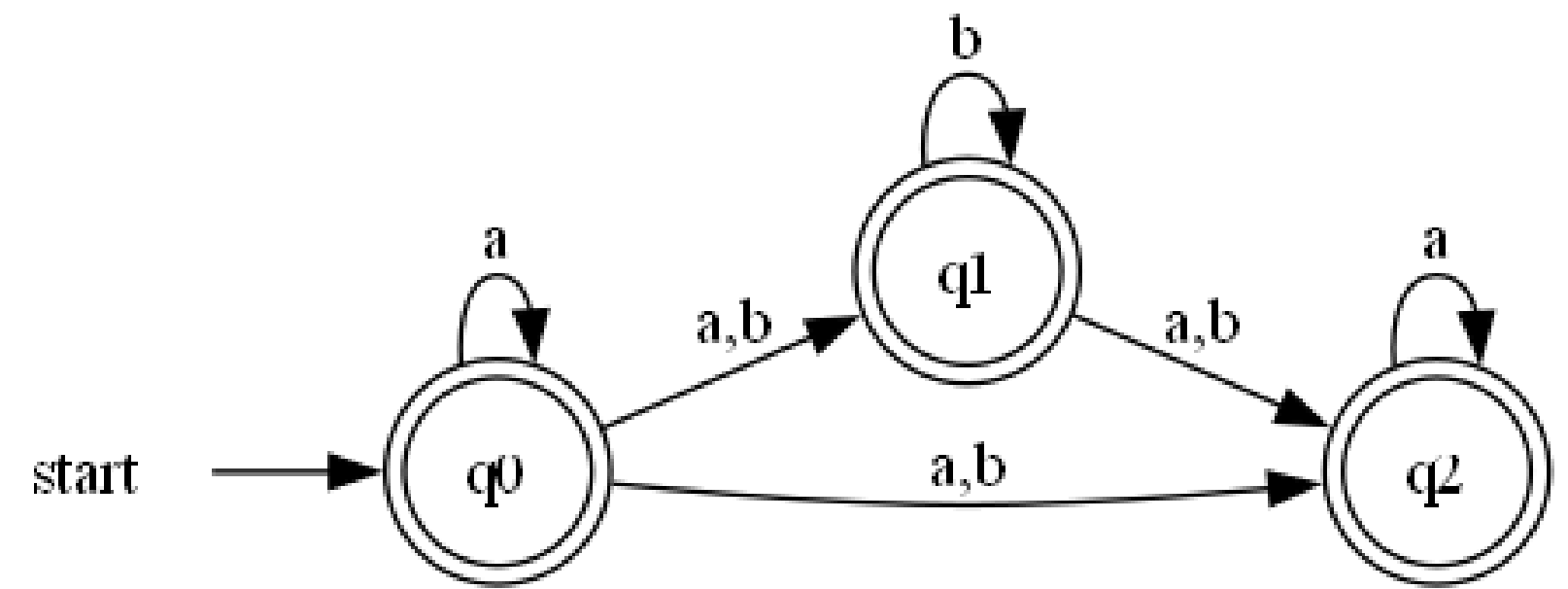


Figura: Grafo do AFN de M_9

Figura 4.5 - AFN esperado



Automato - AFN

Figura 4.6 - AFN obtido



RESULTADOS ESPERADOS VS OBTIDOS

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

Tabela 1 - Tabela obtida pelo AFe da Figura 4.7

RESULTADOS ESPERADOS VS OBTIDOS

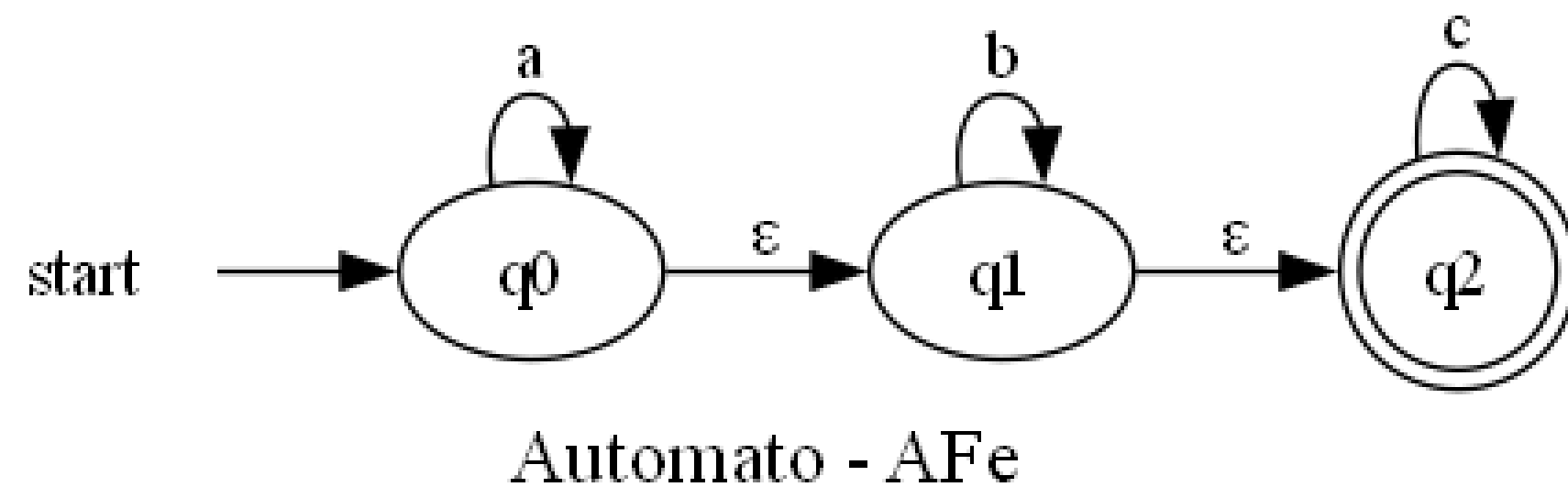


Figura 4.7 - AFe correspondente à tabela 1

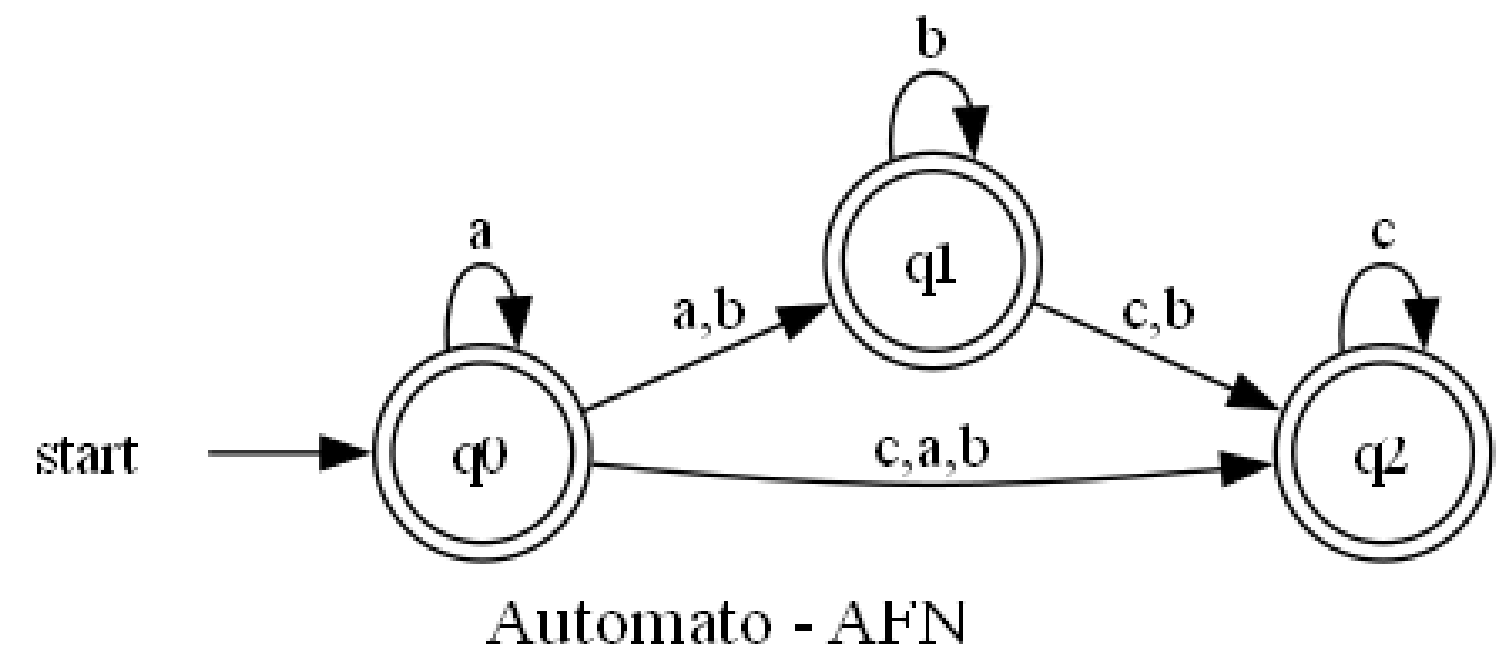


Figura 4.8 - AFN obtido pela conversão do AFe da Figura 4.7

CONCLUSÃO

- **Desenvolvimento de um Conversor Funcional**
- **Contribuição significativa**
- **Potencial para expansão**



PROGRAMA EM FUNCIONAMENTO



REFERÊNCIAS

RAMOS, MARCUS V. M. LINGUAGENS FORMAIS: TEORIA, MODELAGEM E IMPLEMENTAÇÃO. 1ª ED. PORTO ALEGRE: BOOKMAN, 2009.

PAULO BLAUTH MENEZES. LINGUAGENS FORMAIS E AUTÔMATOS: VOLUME 3 DA SÉRIE LIVROS DIDÁTICOS INFORMÁTICA UFRGS. [S.L.] BOOKMAN EDITORA, 2009.

VALENTE, THALES. FORMAL LANGUAGES AND AUTOMATA. 2023. DISPONÍVEL EM: <[HTTPS://GITHUB.COM/THALESVALENTE/TEACHING/TREE/MAIN/FORMAL-LANGUAGES-AND-AUTOMATA/1-LESSONS](https://github.com/thalesvalente/teaching/tree/main/formal-languages-and-automata/1-lessons)>. ACESSO EM: 10 JAN. 2025.

RUI, JOSÉ. LFA10 - AFNE TO AFN. [VÍDEO]. PUBLICADO EM: 8 FEV. 2021. DISPONÍVEL EM: <[HTTPS://WWW.YOUTUBE.COM/WATCH?V=RUTXRCZI9CG](https://www.youtube.com/watch?v=RUTXRCZI9CG)>. ACESSO EM: 10 JAN. 2025.

MCNAUGHTON, R. (1961). THE THEORY OF AUTOMATA, A SURVEY. ADVANCES IN COMPUTERS VOLUME 2, 379-421.



OBRIGADO

CONVERSOR DE AUTÔMATO FINITO COM
MOVIMENTOS VAZIOS (AFE) PARA
AUTÔMATO FINITO NÃO DETERMINÍSTICO
(AFN)

Orientador: Dr. Thales L. A. Valente

■ FERNANDA SOUSA DE
ASSUNÇÃO VALE

■ JHONES DE SOUSA SOARES