
**UNIVERSIDADE FEDERAL DO MARANHÃO**
**CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS**
**ENGENHARIA DA COMPUTAÇÃO**

# MINIMIZAÇÃO DE UM AUTÔMATO FINITO DETERMINÍSTICO (AFD): UM ESTUDO E IMPLEMENTAÇÃO

João Lucas Silva Da Silva e Bruno Carvalho Da Silva

São Luís–MA
2025

**SUMÁRIO**

## 1. Introdução

A minimização de um autômato finito determinístico (AFD) é uma etapa fundamental no estudo da teoria da computação e na prática da engenharia de software. Esse processo tem como objetivo simplificar um AFD, reduzindo-o ao menor número possível de estados, sem alterar seu comportamento ou a linguagem que ele aceita. Essa simplificação é essencial para garantir maior eficiência computacional e economia de recursos, aspectos indispensáveis em aplicações que demandam alta performance.

O conceito de minimização de AFDs começou a ser explorado na década de 1950, com o desenvolvimento da teoria dos autômatos e linguagens formais, por meio de contribuições de cientistas como Kleene, Rabin e Scott. Esses estudos estabeleceram as bases para a compreensão do comportamento e da estrutura dos autômatos. Desde então, métodos sistemáticos para a minimização de AFDs foram desenvolvidos, como o uso de tabelas de distinção e o algoritmo de equivalência de estados.

A minimização de AFDs tem como principal finalidade aumentar a eficiência de sistemas que utilizam autômatos, como compiladores, análise léxica, verificação de sistemas e compressão de dados. Além disso, a simplificação facilita a análise e implementação de AFDs, reduzindo a complexidade de projetos que dependem dessas estruturas.

Neste trabalho, será explorado o processo de minimização de AFDs, abordando os fundamentos teóricos e apresentando uma implementação prática que demonstra a aplicação desse conceito em cenários reais.

### 1.1. Objetivos

Os autômatos finitos determinísticos frequentemente apresentam estados redundantes, o que aumenta a complexidade e o custo computacional de sua implementação em sistemas práticos. A eliminação dessa redundância, sem comprometer a linguagem aceita pelo autômato, é um desafio que pode ser superado com o uso de algoritmos de minimização eficientes, como o algoritmo de Hopcroft. Nesse contexto, a questão central deste trabalho é: como implementar e validar um algoritmo de minimização de AFDs, garantindo a preservação de sua funcionalidade original?

O objetivo geral deste estudo é investigar e aplicar o processo de minimização de autômatos finitos determinísticos, utilizando o algoritmo de Hopcroft, com o propósito de otimizar a eficiência e reduzir a complexidade estrutural desses autômatos. Para alcançar esse objetivo, foram definidos os seguintes objetivos específicos: implementar o algoritmo de Hopcroft para a minimização de AFDs; validar a eficiência do algoritmo por meio de casos de teste reais e variados; comparar o desempenho do AFD original e do AFD minimizado, considerando a aceitação de palavras e a redução de estados; e, por fim, visualizar graficamente os AFDs antes e após o processo de minimização.

## 2. Metodologia

A metodologia adotada neste trabalho combina o desenvolvimento teórico com a validação prática. Inicialmente, o AFD original é projetado manualmente, incluindo a definição de estados, transições, estado inicial e estados de aceitação. Em seguida, o algoritmo de

Hopcroft é implementado em Python, utilizando a lógica de divisão de estados em classes de equivalência. Cada iteração do algoritmo verifica a possibilidade de refinar as partições com base nas transições e no alfabeto do AFD.

Após a implementação, um conjunto de palavras é utilizado para testar a funcionalidade do AFD original e do AFD minimizado, garantindo que ambos aceitem as mesmas linguagens. Para facilitar a análise, ferramentas como Graphviz são empregadas na geração de diagramas que ilustram os AFDs antes e após a minimização. Por fim, os resultados são avaliados com base na redução do número de estados e na manutenção da consistência na aceitação das palavras, assegurando que o processo de minimização

## 3.Fundamentação teórica

A minimização de autômatos finitos determinísticos (AFDs) é um processo essencial que visa reduzir a complexidade estrutural de um AFD, eliminando estados redundantes enquanto preserva a linguagem aceita pelo autômato. Essa técnica é crucial para otimizar o uso de recursos computacionais, especialmente em aplicações que demandam alta eficiência. Um AFD é formalmente definido como uma 5-upla $(Q, \Sigma, \delta, q0, F)$, onde Q é um conjunto finito de estados, $\Sigma$ é o alfabeto finito de entrada, $\delta: Q \times \Sigma \rightarrow Q$ é a função de transição, $q0 \in Q$ é o estado inicial e $F \subseteq Q$ é o conjunto de estados de aceitação.

O objetivo da minimização é identificar estados equivalentes em Q que possam ser agrupados sem alterar o comportamento do autômato. Dois estados são considerados equivalentes se, para qualquer sequência de símbolos do alfabeto, ambos levam ao mesmo resultado de aceitação ou rejeição. Entre os métodos disponíveis para a minimização de AFDs, o algoritmo de Hopcroft se destaca por sua eficiência e simplicidade.

Esse algoritmo utiliza um particionamento iterativo dos estados em classes de equivalência, reduzindo o conjunto de estados de forma sistemática. Sua complexidade é $O(n \log n)$, o que o torna especialmente adequado para aplicações em larga escala. O processo de minimização pode ser resumido em três etapas principais: divisão inicial dos estados em dois grupos, sendo eles os estados de aceitação (F) e os estados de não aceitação $(Q \setminus F)$; refinamento iterativo das partições com base nas transições induzidas pelo alfabeto; e construção de um novo autômato com os estados minimizados e as transições ajustadas.

A aplicação da minimização de AFDs tem um impacto significativo em diversas áreas, como compiladores, onde reduz a complexidade de tabelas de transições em análises léxicas; verificação de modelos, facilitando a validação de propriedades de sistemas; e compressão de dados, permitindo a representação compacta de autômatos em sistemas de armazenamento. Por meio da aplicação de técnicas de minimização, é possível melhorar a eficiência de sistemas computacionais, reduzindo o custo computacional e simplificando a manutenção de sistemas baseados em autômatos. Essa abordagem não apenas otimiza o desempenho, mas também contribui para a criação de soluções mais robustas e escaláveis.

## 4.Processo de Minimização de um AFD

O processo de minimização de um autômato finito determinístico (AFD) utilizando o algoritmo de Hopcroft segue uma sequência de etapas bem definidas, baseadas no particionamento de estados e na análise de transições. Essas etapas garantem a redução do número de estados do autômato sem alterar a linguagem aceita.

Inicialmente, o AFD é definido com seus elementos constitutivos: o conjunto de estados {q0, q1, q2, q3, q4, q5}, o alfabeto {a, b}, a tabela de transições (que especifica para qual estado cada símbolo leva a partir de um estado dado), o estado inicial q0 e os estados de aceitação {q0, q4, q5}. Com base nessas definições, os estados são particionados em dois grupos iniciais: os estados de aceitação {q0, q4, q5} e os estados de não aceitação {q1, q2, q3}. A fila de partições a serem analisadas começa com os estados de aceitação, formando a partição inicial P = [{q0, q4, q5}, {q1, q2, q3}] e a fila de análise Q = [{q0, q4, q5}].

Na primeira iteração de análise, a partição {q0, q4, q5} é examinada com base nas transições para cada símbolo do alfabeto. Para o símbolo 'a', todos os estados dessa partição transitam para o mesmo estado (q2), o que não gera divisões. No entanto, para o símbolo 'b', as transições levam a estados distintos (q1, q2, q3), o que resulta na divisão da partição em dois subconjuntos: {q0}, que transita para q1, e {q4, q5}, que transitam para q2 e q3. Após essa análise, a partição é atualizada para P = [{q0}, {q4, q5}, {q1, q2, q3}] e a fila de análise torna-se Q = [{q4, q5}, {q1, q2, q3}].

Nas iterações subsequentes, o mesmo processo de análise e refinamento é aplicado às demais partições. Para cada partição, verifica-se se os estados possuem transições para estados equivalentes ou diferentes, ajustando as divisões conforme necessário. Esse processo é repetido até que nenhuma nova divisão seja possível. Após várias iterações, a partição final é obtida: P = [{q0}, {q1}, {q4}, {q5}, {q2}, {q3}].

Com a partição final definida, o AFD minimizado é reconstruído. Os estados equivalentes são agrupados, e as transições são ajustadas com base nas novas partições. Por exemplo, os estados minimizados podem ser representados como {q0, q1, q4} e {q2, q5}, com transições ajustadas como q0 → q1 (com b) e q2 → q4 (com a).

## 5.Implementação Computacional

O processo de implementação foi realizado em Python, utilizando bibliotecas como NetworkX e Matplotlib para a manipulação e visualização de grafos. Abaixo, apresentamos as principais etapas do código desenvolvido para a minimização de AFDs:

```
1  import networkx as nx
2  import matplotlib.pyplot as plt
3  from collections import defaultdict
4
5  def criar_afd():
6      """Cria um exemplo de AFD para ser minimizado."""
```

```
7     afd = {
8         'estados': {'q0', 'q1', 'q2', 'q3', 'q4', 'q5'},
9         'alfabeto': {'a', 'b'},
10        'transicoes': {
11            ('q0', 'b'): 'q1', ('q0', 'a'): 'q2',
12            ('q1', 'a'): 'q1', ('q1', 'b'): 'q0',
13            ('q2', 'a'): 'q4', ('q2', 'b'): 'q5',
14            ('q3', 'a'): 'q5', ('q3', 'b'): 'q4',
15            ('q4', 'a'): 'q3', ('q4', 'b'): 'q2',
16            ('q5', 'a'): 'q2', ('q5', 'b'): 'q3'
17        },
18        'inicio': 'q0',
19        'aceite': {'q0', 'q4', 'q5'},
20    }
21    return afd
22
23 def minimizacao_hopcroft(afd):
24    """Minimiza o AFD usando o algoritmo de Hopcroft."""
25    estados = afd['estados']
26    alfabeto = afd['alfabeto']
27    transicoes = afd['transicoes']
28    aceite_estados = afd['aceite']
29    nao_aceite_estados = estados - aceite_estados
30
31    # Inicialização
32    P = [aceite_estados, nao_aceite_estados]
33    Q = [aceite_estados] if aceite_estados else []
34
35    while Q:
36        A = Q.pop()
37        for c in alfabeto:
38            X = {s for s in estados if transicoes.get((s, c)) in A}
39            novo_P = []
40
41            for Y in P:
42                intersecao = X & Y
43                diferenca = Y - X
44
45                if intersecao and diferenca:
46                    novo_P.append(intersecao)
47                    novo_P.append(diferenca)
48
49                    if Y in Q:
50                        Q.remove(Y)
```

```
                    Q.append(intersecao)
                    Q.append(diferenca)
                else:
                    Q.append(intersecao if len(intersecao) <= len(diferenca) else diferenca)
            else:
                novo_P.append(Y)

        P = novo_P

    # Construção do AFD minimizado
    minimizado = {}
    for grupo in P:
        rep = next(iter(grupo))
        for estado in grupo:
            minimizado[estado] = rep

    min_estados = set(minimizado.values())
    min_transicoes = {}
    for (estado, simbolo), alvo in transicoes.items():
        if estado in minimizado and alvo in minimizado:
            min_transicoes[(minimizado[estado], simbolo)] = minimizado[alvo]

    min_aceite = {minimizado[s] for s in aceite_estados}
    min_inicio = minimizado[afd['inicio']]

    afd_minimizado = {
        'estados': min_estados,
        'alfabeto': alfabeto,
        'transicoes': min_transicoes,
        'inicio': min_inicio,
        'aceite': min_aceite,
    }

    return afd_minimizado

def teste_palavra(afd, palavra):
    """Testa se uma palavra é aceita pelo AFD."""
    current_state = afd['inicio']
    for symbol in palavra:
        current_state = afd['transicoes'].get((current_state, symbol))
        if current_state is None:
            return False
    return current_state in afd['aceite']

```

```
95 def compare_afds(afd1, afd2, palavras):
96     """Compara se as palavras aceitas pelo AFD1 são aceitas pelo AFD2."""
97     resultados = {}
98     for palavra in palavras:
99         resultado1 = teste_palavra(afd1, palavra)
100         resultado2 = teste_palavra(afd2, palavra)
101         resultados[palavra] = (resultado1, resultado2)
102     return resultados
103 
104 # Criação do AFD inicial
105 afd = criar_afd()
106 
107 # Minimização do AFD
108 afd_minimizado = minimizacao_hopcroft(afd)
109 
110 def criar_grafo_afd(afd, nome):
111     dot = Digraph(nome, format='png')
112     dot.attr(rankdir='LR')  # Direção da leitura do AFD
113     for estado in afd['estados']:
114         if estado in afd['aceite']:
115             dot.node(estado, shape='doublecircle')  # Estados de aceitação
116         else:
117             dot.node(estado, shape='circle')  # Estados normais
118     dot.node('', shape='none')  # Estado vazio para a seta inicial
119     dot.edge('', afd['inicio'])
120     for (estado_inicial, simbolo), estado_final in afd['transicoes'].items():
121         dot.edge(estado_inicial, estado_final, label=simbolo)
122     return dot
123 
124 # Criar os gráficos para os dois AFDs
125 grafo_afd1 = criar_grafo_afd(afd, 'AFD1')
126 grafo_afd2 = criar_grafo_afd(afd_minimizado, 'AFD2')
127 
128 # Exibir os dois gráficos
129 grafo_afd1.view(cleanup=True)
130 grafo_afd2.view(cleanup=True)
131 
132 # Teste de palavras
133 palavras_para_teste = ["b", "a", "aa", "ab", "ba", "bbb"]
134 comparison_results = compare_afds(afd, afd_minimizado, palavras_para_teste)
135 
136 print("Comparação do aceite de diferentes palavras entre os AFDs:")
137 for palavra, (resultado1, resultado2) in comparison_results.items():
```

```
138     print(f"Palavra '{palavra}': Original AFD = {resultado1}, AFD minimizado =
{resultado2}")
```

## 6. Resultados e Discussão

Os resultados obtidos com a aplicação do algoritmo de Hopcroft foram analisados e validados utilizando um conjunto de palavras de teste, assegurando que o AFD minimizado preservasse a mesma linguagem aceita pelo autômato original. Essa validação foi essencial para confirmar que o processo de minimização não alterou o comportamento funcional do autômato.

Além disso, a redução no número de estados foi destacada visualmente por meio de grafos gerados com a biblioteca Graphviz, permitindo uma análise clara e intuitiva das mudanças estruturais ocorridas no AFD. O AFD original, que possuía 6 estados, foi reduzido para apenas 3 estados após a minimização, evidenciando a eficácia do algoritmo em eliminar redundâncias sem comprometer a funcionalidade.

Os resultados demonstram que o algoritmo de Hopcroft é uma ferramenta eficiente para a simplificação de autômatos finitos determinísticos, contribuindo para a redução da complexidade estrutural e para a otimização do uso de recursos computacionais. A análise visual e quantitativa reforça a importância de técnicas de minimização em aplicações práticas, onde a eficiência e a simplicidade são fatores cruciais.

## 7. Conclusão

O resultado final é um AFD com menos estados, que mantém a mesma linguagem aceita pelo autômato original. Esse processo de minimização não apenas reduz a complexidade estrutural do autômato, mas também otimiza o uso de recursos computacionais, assegurando a consistência e a eficiência do sistema.