UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

Paloma de Castro Leite

RELATÓRIO TÉCNICO - LFA

APUCARANA – PR
2024

Paloma de Castro Leite

RELATÓRIO TÉCNICO – LFA

Trabalho apresentado à disciplina de Linguagens Formais, Autômatos e Computabilidade do curso de Bacharelado em Ciência da Computação.

Professor: Guilherme Henrique de Souza Nakahata;

APUCARANA – PR
2024

SUMÁRIO

INTRODUÇÃO 03

CAPÍTULO 1: OBJETIVOS 04

CAPÍTULO 2: MOTIVAÇÃO E RECURSOS UTILIZADOS 05

2.1 Motivação..... 05

2.2 Estrutura de Dados 06

2.3 Linguagem de programação e demais informações..... 08

CAPÍTULO 3: RESULTADOS 09

CONCLUSÃO 15

REFERÊNCIAS 16

INTRODUÇÃO

O estudo de Linguagens Formais e Autômatos é fundamental para a Ciência da Computação, pois fornece as bases para a compreensão e especificação de linguagens, bem como o reconhecimento de padrões. Esta disciplina investiga classificações, estruturas, propriedades e relações entre diferentes tipos de autómatos e linguagens, além de embasar outros aspectos mais teóricos, como decidibilidade, computabilidade e complexidade computacional.

Neste trabalho, iremos analisar especificamente um Autômato Finito Determinístico (AFD), um modelo matemático que representa uma máquina em que descreve sistemas com um número finito de estados e transições executáveis. Um AFD tem cada estado associado a cada símbolo do alfabeto, resultando em um comportamento determinístico, isso significa que, ao processar uma cadeia de entrada, um AFD sempre exibe um único caminho definido, tornando-o ideal para reconhecimento de padrões e validação de sequências.

CAPÍTULO 1
OBJETIVOS

Neste trabalho em questão, o objetivo principal é simular um Autômato Finito Determinístico (AFD), um modelo matemático que é normalmente usado para simular máquinas que tem um número finito de estados, onde a partir de um estado e o processamento de um símbolo (cadeia ou palavras), o próximo estado é único e sempre conhecido. Para isso, o código fonte do AFD lê a cadeia inserida e exibe o resultado após percorrer a sequência completa de símbolos, indicando se a cadeia é aceita ou rejeitada, possibilitando uma forma interativa para o desenvolvimento e teste de um autômato finito determinístico.

Para garantir a validade do teste, este trabalho também apresentará a descrição formal do autômato e a tabela de transição, para auxiliar na análise do funcionamento do AFD e dos testes realizados durante a simulação até que o usuário saia do programa ou deseje testar um novo AFD.

CAPÍTULO 2
MOTIVAÇÃO E RECURSOS UTILIZADOS

Baseando-se no que foi descrito anteriormente, devemos explicitar os motivos para a realização do trabalho e seu objetivo final, além dos recursos utilizados para que o trabalho seja executado de maneira eficiente e concisa .

2.1 Motivação

Conforme mencionado no capítulo que trata acerca dos objetivos do trabalho, a motivação seria a execução de um código fonte funcional em que possamos demonstrar e exemplificar o funcionamento de um Autômato Finito Determinístico (AFD), em que recebemos as informações necessárias pelas entradas do usuário e assim determinamos se a cadeia faz ou não parte daquela linguagem, demonstrando como um AFD funciona de maneira prática e visual.

Para esse fim, precisamos receber os dados que possibilitem o funcionamento desse autômato, tais como a descrição formal e a tabela de transições. Na figura a seguir, por exemplo, podemos visualizar como a descrição formal deverá ser recebida.

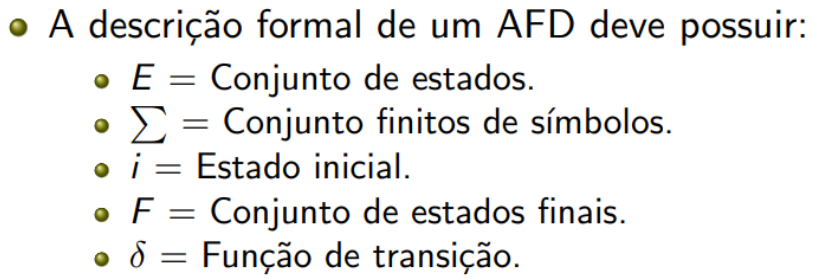


Figura 1 - Descrição formal de um AFD

Portanto, faz-se necessário, com as informações pertinentes acerca dos objetivos e motivações, detalhar os dados mais relevantes à Estrutura de Dados, Linguagem de Programação e demais questões acerca da implementação do código fonte em questão para melhor análise de suas funcionalidades, a fim de obter uma conclusão geral.

2.2 Estrutura de Dados

Este código em Java simula o funcionamento de um autômato finito determinístico. Inicialmente, o código foi separado nas classes *ControlaAFD* e *UsarAFD,* e em uma interface chamada *AFD* para facilitar a compreensão e gerenciamento de cada parte do código, além de possibilitar novas implementações e alterações em suas funcionalidades sem afetar outras partes do sistema. A seguir serão explicadas o funcionamento de cada classe e da interface para melhor entendimento:

a) ControlaAFD: inicialmente são incluídos o pacote padrão *java.util.** que contém uma coleção de classes e interfaces utilitárias e *java.util.regex.Pattern* que é usada para definir padrões em operações de texto , além dos atributos do AFD e um método construtor para ler as entradas do usuário. Essa classe é a implementação do autômato em questão que lê e processa entradas para simular seu comportamento, implementando a interface AFD, que define três métodos principais:

- *leEntradas():* lê as definições usadas no AFD a partir das entradas do usuário, que serão solicitadas e armazenadas em suas respectivas arrays. Esse método também valida algumas entradas do usuário a partir de dois métodos denominados *lerInteiroPositivo* e *validarAlfabeto* que serão explicados futuramente e são usados para garantir o desempenho adequado do simulador.
- *tabelaTransicaoDescricao()*: exibe a tabela de transições do AFD, substituindo ‘-1’ por ‘x’ para indicar as transições não definidas, e sua descrição formal que conta com duas funções auxiliares: *getEstados* - usada para obter o número total de estados do AFD, e *getEstadoFinal* - retorna uma string com

todos os estados finais formatados da maneira que foi especificada no código.

- *testaCadeia()*: permite o teste de diversas cadeias de caracteres por meio de um loop que solicita a entrada desses caracteres, verificando se os mesmos pertencem ao alfabeto inserido anteriormente e realizando as transições de estados, assim verificando se essa cadeia é aceita ou não com base no estado final. Isso se repete até que o usuário digite ‘sair’ para fechar o programa ou digite ‘novo’ para testar um novo AFD.

b) AFD: interface que estabelece os métodos *lerEntradas*, *tabelaTransicaoDescricao* e *testarCadeia*, necessários para configurar, descrever e testar o AFD. Definindo assim o que a classe *ControlaAFD* deve fornecer ao implementar essa interface.

c) UsarAFD: classe usada para instanciar a implementação do AFD, utilizando os métodos inseridos na interface *AFD* e fornecendo uma maneira de executar essas operações. Ela serve como intermediário na interação entre o usuário e a simulação do AFD.

Agora, tendo em foco a classe *ControlaAFD* que controla todo o processo de implementação do autômato finito determinístico, temos o método *lerInteiroPositivo,* que solicita e lê um número inteiro positivo, garantindo que apenas entradas válidas sejam aceitas e em seguida retorna esse valor para ser utilizado pelo restante do programa, o método *validarAlfabeto*, que recebe como parâmetro um array de strings e em seguida inicia um loop que itera sobre cada símbolo, assim verificando através do método *Pattern.matches (*parte do pacote *java.util.regex.Pattern)* se eles correspondem ao padrão especificado (letras minúsculas e números de 0 a 9), e por último o método *lerEstadoInicial* que recebe o número de estados como parâmetro,

inicial um loop com o while para iterar sobre cada estado, assim, se o estado for menor do que 0 ou igual ou maior ao número total de estados do AFD é exibido um aviso para o usuário, caso contrário o estado inicial inserido é validado e o programa segue a execução para as próximas entradas.

A execução do programa finaliza ao usuário digitar “sair”, garantindo que o programa feche apenas após termos obtido e visualizado todos os testes de cadeia desejados pelo usuário. Algumas outras maneiras de organizar e gerir o código fonte foram utilizadas, como o código ANSI para cores que foi empregado apenas para fins estéticos e de melhor visualização, porém apenas com o objetivo de facilitar e possibilitar o desempenho geral do mesmo.

2.3 Linguagem de Programação e demais informações

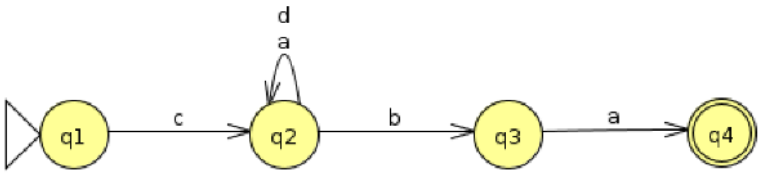
A linguagem de programação Java foi escolhida para o desenvolvimento do código, ela é abordada principalmente durante o segundo ano do curso, e utilizá-la para o desenvolvimento do programa permite conhecer mais sobre a linguagem e desenvolver um aprendizado ao longo do trabalho. Esta linguagem fornece a possibilidade de utilizar orientação a objetos, uma prática de programação que torna possível elaborar um software a partir da geração de objetos que se comunicam entre si, e o uso de classes para representar objetos do mundo real, onde declararmos atributos e métodos, que representam, respectivamente, as características e comportamentos desse objeto.

Dessa maneira, a linguagem de programação Java se tornou ideal para este caso específico, em que o código fonte foi feito para simular um autômato finito determinístico, afinal o uso de classes permite mudanças, compartilhamento de dados, melhor organização e estruturação do código, etc, assim possibilitando um código mais propício a futuras melhorias e implementações sem tê-lo que reescrever por inteiro novamente.

CAPÍTULO 3
RESULTADOS

Diante dos expostos apresentados ao longo do trabalho, a finalidade esperada seria o pleno funcionamento de um código que exemplifica como um simulador de AFD funciona de maneira prática, demonstrando se uma determinada linguagem é aceita ou não, levando em consideração sua tabela de transições e a descrição formal. Dessa maneira, com a implementação total e sua revisão, o objetivo principal do código fonte foi atingido, resultando em uma aplicação funcional no qual recebemos o número total de estados, o alfabeto, o estado inicial e a quantidade de estados finais, e em seguida realizamos testes para ver se uma cadeia de caracteres faz parte da linguagem.

Abaixo, nas figuras 3, 4, 5, 7, 8 e 9 podemos ver o funcionamento do simulador de AFD a partir de dois exemplos dados pelo professor nas especificações do trabalho:



	a	b	c	d
q1	X	X	q2	X
q2	q2	q3	X	q2
q3	q4	X	X	X
q4	X	X	X	X

Figura 2 - Exemplo 1

Resultados:

```
 --- Linguagens Formais, Autômatos e Computabilidade ---

Trabalho de LFA - 1º Bimestre - C.C UNESPAR

 --- SIMULADOR DE AUTÔMATO FINITO DETERMINÍSTICO (AFD) ---

Digite o número de estados:
4

Digite o estado inicial (entre 0 e 3):
0

Digite os estados finais (Se houver mais de um, separe por espaços):
3

Digite o alfabeto (separados por espaços):

Obs: são permitidos apenas letras minúsculas e números.
a b c d

---TRANSIÇÕES --

Preencha as transições a seguir (Apenas números inteiros):

q0 para 'a': -1

q0 para 'b': -1

q0 para 'c': 1

q0 para 'd': -1

q1 para 'a': 1

q1 para 'b': 2
```

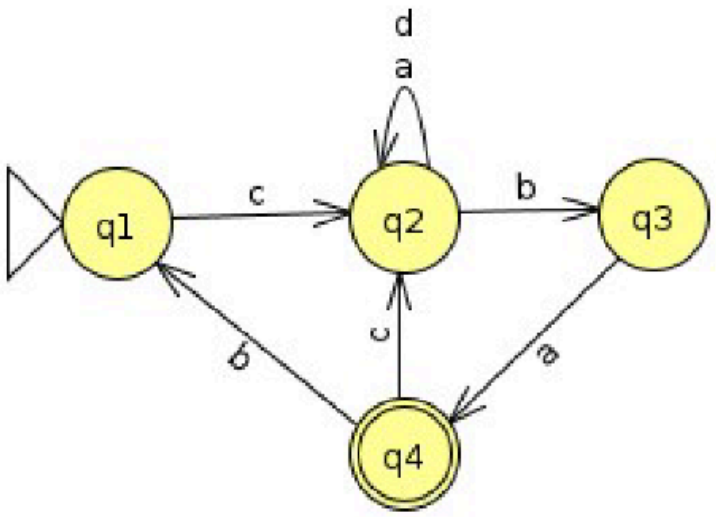
Figura 3

```
q1 para 'c': -1

q1 para 'd': 1

q2 para 'a': 3

q2 para 'b': -1

q2 para 'c': -1

q2 para 'd': -1

q3 para 'a': -1

q3 para 'b': -1

q3 para 'c': -1

q3 para 'd': -1

 --- TABELA DE TRANSIÇÕES ---

    a b c d
q0  x x q1 x
q1  q1 q2 x q1
q2  q3 x x x
q3  x x x x

 --- DESCRIÇÃO FORMAL ---

E = {q0, q1, q2, q3}
Sigma = {a, b, c, d}
i = q0
F = {q3}
```

Figura 4

```
 --- TESTAR CADEIA ---

-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
abc

Transição indefinida para 'a' a partir do estado q0.

Cadeia rejeitada.

-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
cba

-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
cadadaba

Resultado: Cadeia aceita pelo AFD.
```

Figura 5



	a	b	c	d
q1	X	X	q2	X
q2	q2	q3	X	q2
q3	q4	X	X	X
q4	X	q1	q2	X

Figura 6 - Exemplo 2

Resultados:

```
 --- Linguagens Formais, Autômatos e Computabilidade ---

Trabalho de LFA - 1º Bimestre - C.C UNESPAR

 --- SIMULADOR DE AUTÔMATO FINITO DETERMINÍSTICO (AFD) ---

Digite o número de estados:
4

Digite o estado inicial (entre 0 e 3):
0

Digite os estados finais (Se houver mais de um, separe por espaços):
3

Digite o alfabeto (separados por espaços):

Obs: são permitidos apenas letras minúsculas e números.
a b c d
```

Figura 7

```
---TRANSIÇÕES --
Preencha as transições a seguir (Apenas números inteiros):
q0 para 'a': -1
q0 para 'b': -1

q0 para 'd': -1
q1 para 'a': 1
q1 para 'b': 2
q1 para 'c': -1
q1 para 'd': 1
q2 para 'a': 3
q2 para 'b': -1
q2 para 'c': -1
q2 para 'd': -1
q3 para 'a': -1
q3 para 'b': 0
q3 para 'c': 1
q3 para 'd': -1
```

Figura 8


--- TABELA DE TRANSIÇÕES ---
a b c d
q0 x x q1 x
q1 q1 q2 x q1
q2 q3 x x x
q3 x q0 q1 x
--- DESCRIÇÃO FORMAL ---
E = {q0, q1, q2, q3}
Sigma = {a, b, c, d}
i = q0
F = {q3}
--- TESTAR CADEIA ---
-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
cadba
Resultado: Cadeia aceita pelo AFD.
-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
cadadadaba
Resultado: Cadeia aceita pelo AFD.
-Informe uma cadeia a ser testada (Digite 'sair' para fechar o programa ou 'novo' para testar um novo AFD):
cadadabac
Resultado: Cadeia não aceita pelo AFD.

Figura 9

CONCLUSÃO

Com base no que foi descrito ao longo do trabalho, podemos ver que essa aplicação prática se torna essencial para demonstrar como os autômatos finitos determinísticos funcionam de maneira mais didática e interativa, facilitando assim o aprendizado da matéria por ser possível realizar testes com diferentes tipos de AFD e visualizar os resultados de maneira imediata, além de obter sua descrição formal e tabela de transições para melhor entendimento do autômato como um todo e algumas de suas limitações.

Portanto, os resultados obtidos durante a execução desse programa podem ser usados durante os estudos, auxiliando na compreensão do conteúdo passado durantes as aulas de LFA, além de tornar evidente a maneira como um AFD permite projetar computadores, algoritmos e programas por serem capazes de reconhecer qualquer gramática regular e resolver diversos problemas computacionais presentes no dia-a-dia.

.

REFERÊNCIAS

Como tratar exceções na linguagem Java. Disponível em:

<https://www.devmedia.com.br/como-tratar-excecoes-na-linguagem-java/39163>.

FURGERI, SÉRGIO. Java 8-Ensino Didático: Desenvolvimento e Implementações de Aplicações. Saraiva Educação SA, 2018.