UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

Guilherme Fortunato Da Silva

RELATÓRIO TÉCNICO - LFA

APUCARANA - PR
2024

Guilherme Fortunato Da Silva

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

Linguagens Formais, Autômatos e Computabilidade formam a base teórica da Ciência da Computação. Este campo de estudo se concentra na definição e análise de sistemas de processamento de informação, através do uso de modelos matemáticos abstratos. As Linguagens Formais são conjuntos definidas por regras gramaticais específicas, enquanto os Autômatos são máquinas teóricas que reconhecem essas linguagens. A Computabilidade, por sua vez, investiga quais problemas podem ser resolvidos por algoritmos e quais não podem, além de explorar a eficiência desses algoritmos.

Este curso visa fornecer uma compreensão aprofundada desses conceitos fundamentais, que são essenciais para o desenvolvimento de tecnologias modernas, desde compiladores até inteligência artificial. Os estudantes serão introduzidos a gramáticas formais, autômatos finitos, máquinas de Turing, e teoria da complexidade, adquirindo habilidades críticas para a análise e construção de sistemas computacionais.

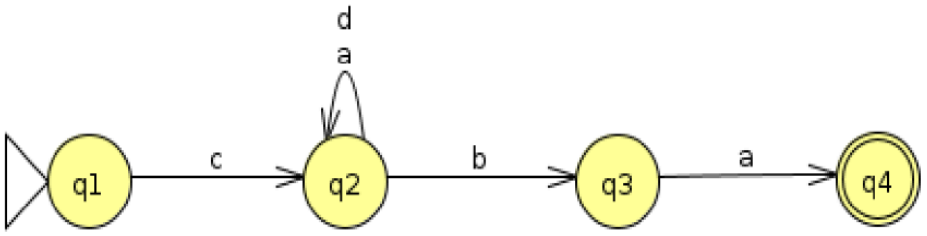
Neste trabalho, irei criar um AFD que em resumo são modelos matemáticos usados para reconhecer linguagens regulares. Eles consistem de um conjunto finito de estados, um alfabeto de símbolos de entrada, uma função de transição que determina o próximo estado para cada combinação de estado atual e símbolo de entrada, um estado inicial e um conjunto de estados de aceitação ou finais.

Um AFD processa uma string de símbolos do alfabeto de entrada, começando no estado inicial e seguindo as transições definidas pela função de transição. Se, após consumir toda a string, o autômato termina em um estado de aceitação, a string é considerada aceita pela linguagem que o AFD reconhece. Caso contrário, a string é rejeitada.

CAPÍTULO 1: OBJETIVOS

O objetivo principal do código criado pode ser interpretado como estabelecer o funcionamento prático de um Autômato Finito Determinístico, mas, em um programa. De modo geral, o código fonte tem como objetivo receber uma descrição formal, ou seja, as características de um conjunto de estados que representam um AFD, com isso, poderá avaliar a aprovação ou não de palavras mediante a linguagem informada por meio das transições, e então teste cadeias de caracteres para verificar se são aceitas ou não pelo AFD.

O programa é uma ferramenta interativa para a construção e teste de AFDs, que pode ser usada para entender melhor o funcionamento dos autômatos e como eles processam linguagens regulares. Na figura 1 está um exemplo de um Autômato finito Determinístico que o programa deve ser capaz de reconhecer.



	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 1

CAPÍTULO 2 MOTIVAÇÃO E RECURSOS UTILIZADOS

Baseando-se no exposto anteriormente, devemos explicitar os motivos para a realização do trabalho, ou seja, o objetivo final e os recursos utilizados para que isso seja cumprido.

2.1 Motivação

Como citado no capítulo que trata acerca dos objetivos do projeto em questão, a motivação também seria a realização de um código fonte funcional em que possamos demonstrar o pleno funcionamento de um Autômato Finito Determinístico (AFD). Este projeto visa proporcionar uma compreensão prática e teórica de como um AFD opera, permitindo que possamos determinar se as palavras requisitadas fazem ou não parte da linguagem reconhecida pelo autômato. Para tal, precisamos receber os dados, tais como a descrição formal e a tabela de transições. Após isso, é necessário criar uma lógica de programação para atingir o objetivo de maneira eficaz. Na figura 2, por exemplo, podemos visualizar como a descrição formal deverá ser recebida, de maneira a imprimir o seu conteúdo formatado no modelo contido nesta.

- $E =$ Conjunto de estados.
- $\sum =$ Conjunto finitos de símbolos.
- $i =$ Estado inicial.
- $F =$ Conjunto de estados finais.
- $\delta =$ Função de transição.

figura 2

2.2 Estrutura de Dados

O código começa criando um objeto `Scanner` para capturar a entrada do usuário. Em seguida, imprime uma mensagem inicial para indicar que é um Autômato Finito Determinístico (AFD) de fim estético. Após o programa solicita ao usuário que insira o número de estados do AFD, que é lido e convertido para um inteiro. Após isso, pede que o usuário forneça o alfabeto, com os símbolos separados por espaços, e armazena esses símbolos em um array de strings chamado `sigma`.

Em seguida, o programa solicita ao usuário o estado inicial do AFD, que é lido e convertido para um inteiro. Também solicita os estados finais (de aceitação), que são inseridos separados por espaços. Esses estados são armazenados em um conjunto (`Set`) chamado `aceitacao` para evitar duplicatas.

Para definir as transições, o programa cria uma matriz `transicoes` de tamanho `numEstados` por `sigma.length` que seria o número de estados pelo número de símbolos, inicializando todas as posições com -1, indicando transições indefinidas. O usuário é solicitado a definir as transições para cada estado e símbolo do alfabeto. Para cada estado, o usuário deve fornecer o estado de transição para cada símbolo do alfabeto. Se uma transição não existir, o usuário deve inserir -1.

Depois que todas as transições são definidas, o programa imprime uma tabela de transições formatada, mostrando como cada estado se comporta para cada símbolo do alfabeto. Na tabela, 'x' indica transições indefinidas.

O uso do `while` é para permitir que o usuário teste diferentes cadeias de caracteres para ver se elas são aceitas. O loop começa solicitando ao usuário que insira uma cadeia. Se o usuário digitar "sair", o programa termina; caso contrário, o loop continua.

Dentro do loop, o estado atual do autômato é definido como o estado inicial. Uma flag chamada `naoAceita` é inicializada como `false`. Esta flag será usada para indicar se a cadeia deve ser rejeitada durante o processamento.

O próximo passo é verificar repetidamente cada símbolo da cadeia. Para cada símbolo, o programa verifica se o símbolo está no alfabeto. Isso é feito convertendo o array `sigma` em uma lista e buscando o índice do símbolo na lista. Se o índice for -1, significa que o símbolo não pertence ao alfabeto, e a cadeia é imediatamente rejeitada. Neste caso, uma mensagem é impressa indicando que a cadeia contém um símbolo inválido e a flag `naoAceita` é definida como `true`. O loop interno é então interrompido.

Se o símbolo estiver no alfabeto, o programa usa o índice do símbolo para buscar o próximo estado na matriz de transições. Se a transição não estiver definida (indicada por -1 na matriz), a cadeia é rejeitada. O programa imprime uma mensagem informando que a transição é indefinida e define a flag `naoAceita` como `true`, interrompendo o loop interno.

Se a transição estiver definida, o estado atual será atualizado para o próximo estado. O programa continua processando os símbolos da cadeia até que todos tenham sido processados ou até que uma transição indefinida ou símbolo inválido seja encontrado.

Após o loop interno, o programa verifica se a cadeia deve ser rejeitada. A cadeia é rejeitada se a flag `naoAceita` for `true` ou se o estado atual não for um estado de aceitação (ou seja, se o estado atual não estiver no conjunto de estados de aceitação). Se qualquer uma dessas condições for verdadeira, o programa imprime "Nao aceita". Caso contrário, imprime "Aceita".

O loop `while` então recomeça, solicitando ao usuário que insira outra cadeia. Esse processo continua até que o usuário digite "sair", momento em que o loop é interrompido e o programa se encerra.

Finalmente, o objeto `Scanner` é fechado para liberar os recursos associados à entrada do usuário. Este ciclo permite que o usuário teste várias cadeias de caracteres de forma interativa, verificando se cada uma é aceita ou rejeitada pelo AFD configurado.

2.3 Linguagem de Programação e demais informações

A linguagem escolhida para a implementação do código foi Java, de maneira que se utiliza-se a linguagem principalmente abordada no segundo ano de formação para implementar o ciclo de instruções. Java é uma linguagem de alto nível desenvolvida pela Sun Microsystems, orientada a objetos e amplamente utilizada em vários ramos da programação. No código usei uma instrução que permite importar todas as classes e interfaces do pacote `java.util`, permitindo que sejam utilizadas no código sem necessidade de especificar o nome completo de cada uma delas.

As bibliotecas da linguagem utilizadas para o funcionamento do código foram as bibliotecas `java.util..Scanner`, ‘java.util.Set’, ‘java.util.HashSet’, e a ‘java.util.Arrays’. A classe `Scanner` é usada para ler entradas do usuário, como o número de estados, o alfabeto, o estado inicial, os estados finais, as transições e as cadeias a serem testadas pelo autômato. A interface `Set` é usada em conjunto com a classe `HashSet` para criar um conjunto que armazena os estados de aceitação, isso permite evitar estados duplicados. A classe `Arrays` é utilizada para manipular arrays. No código, o método `Arrays.fill` é utilizado para inicializar a matriz de transições com o valor `-1`, indicando transições indefinidas. Essas funcionalidades combinadas permitem , a definição de seus estados, transições e a verificação se uma cadeia de entrada é aceita ou não pelo autômato.

O programa tem a limitação de sempre precisar começar de um estado q0 impossibilitando que o estado inicial comece de um q1 por exemplo.

CAPÍTULO 3: RESULTADOS

Mediante os objetivos apresentados, o resultado esperado seria o pleno funcionamento de um código que exemplifica como um Autômato Finito Determinístico (AFD) reconhece ou não palavras de uma determinada linguagem, dado a sua descrição formal e tabela de transições. Dessa maneira, com a implementação completa e revisada, os resultados foram atingidos, resultando em uma aplicação interativa e funcional, recebendo a quantidade de letras no alfabeto, as letras, a quantidade de estados, a quantidade de estados finais, o estado inicial, os estados finais. Na figura 3 podemos ver como o programa se comporta na fase inicial, que seria pedindo, e registrando a descrição e por fim mostrando a tabela de transições.

```
Digite o numero de estados: 4

Digite o alfabeto, separados por espacos: a b c d

Digite o estado inicial: 0

Digite os estados finais (separe por espacos): 3

TRANSICOES
OBS: Caso nao exista uma transicao, insira -1

Estado q0, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': -1
Para simbolo 'c': 1
Para simbolo 'd': -1

Estado q1, digite a transicao:
Para simbolo 'a': 1
Para simbolo 'b': 2
Para simbolo 'c': -1
Para simbolo 'd': 1

Estado q2, digite a transicao:
Para simbolo 'a': 3
Para simbolo 'b': -1
Para simbolo 'c': -1
Para simbolo 'd': -1

Estado q3, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': -1
Para simbolo 'c': -1
Para simbolo 'd': -1

TABELA DE TRANSICOES:
    a b c d
q0 x x q1 x
q1 q1 q2 x q1
q2 q3 x x x
q3 x x x x
```

figura 3

A seguir o programa executou o reconhecimento de cadeia dos AFDs mostrados nas figuras 4 e 5, seus resultados estão sendo respectivamente mostrados nas figuras 6 e 7.

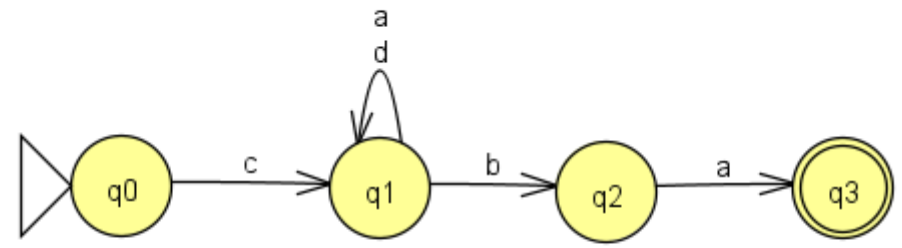


figura 4

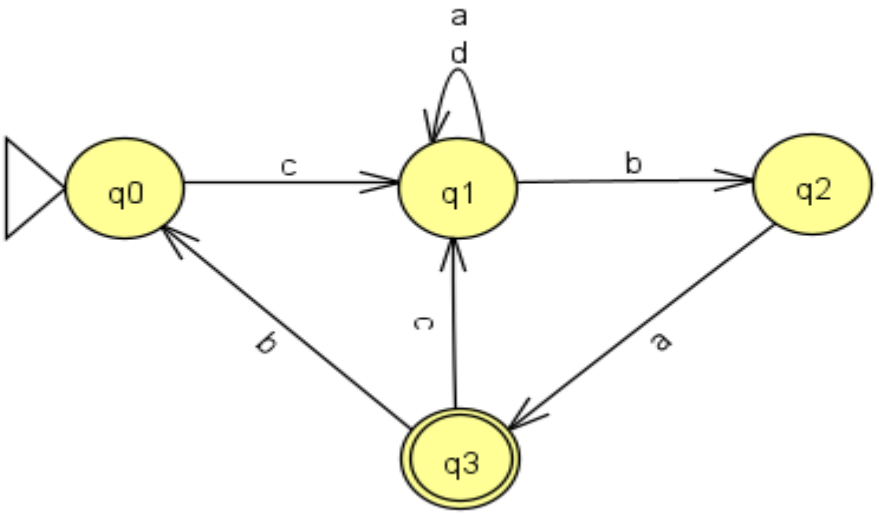


figura 5

Resultado AFD figura 4:

```
Digite o numero de estados: 4

Digite o alfabeto, separados por espacos: a b c d

Digite o estado inicial: 0

Digite os estados finais (separe por espacos): 3

TRANSICOES
OBS: Caso nao exista uma transicao, insira -1

Estado q0, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': -1
Para simbolo 'c': 1
Para simbolo 'd': -1

Estado q1, digite a transicao:
Para simbolo 'a': 1
Para simbolo 'b': 2
Para simbolo 'c': -1
Para simbolo 'd': 1

Estado q2, digite a transicao:
Para simbolo 'a': 3
Para simbolo 'b': -1
Para simbolo 'c': -1
Para simbolo 'd': -1

Estado q3, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': -1
Para simbolo 'c': -1
Para simbolo 'd': -1

TABELA DE TRANSICOES:
    a b c d
q0 x x q1 x
q1 q1 q2 x q1
q2 q3 x x x
q3 x x x x

Informe uma cadeia (Digite 'sair' para fechar o programa): cdddddaaaaba
Aceita.

Informe uma cadeia (Digite 'sair' para fechar o programa): bcdddddba
Transicao indefinida para simbolo 'b' a partir do estado q0
Nao aceita.

Informe uma cadeia (Digite 'sair' para fechar o programa): cdaba
Aceita.
```

figura 6

Resultado AFD figura 5:

```
AUTOMATO FINITO DETERMINISTICO

Digite o numero de estados: 4

Digite o alfabeto, separados por espacos: a b c d

Digite o estado inicial: 0

Digite os estados finais (separe por espacos): 3

TRANSICOES
OBS: Caso nao exista uma transicao, insira -1

Estado q0, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': -1
Para simbolo 'c': 1
Para simbolo 'd': -1

Estado q1, digite a transicao:
Para simbolo 'a': 1
Para simbolo 'b': 2
Para simbolo 'c': -1
Para simbolo 'd': 1

Estado q2, digite a transicao:
Para simbolo 'a': 3
Para simbolo 'b': -1
Para simbolo 'c': -1
Para simbolo 'd': -1

Estado q3, digite a transicao:
Para simbolo 'a': -1
Para simbolo 'b': 0
Para simbolo 'c': 1
Para simbolo 'd': -1

TABELA DE TRANSICOES:
    a b c d
q0 x x q1 x
q1 q1 q2 x q1
q2 q3 x x x
q3 x q0 q1 x

Informe uma cadeia (Digite 'sair' para fechar o programa): cadba
Aceita.

Informe uma cadeia (Digite 'sair' para fechar o programa): cadadadadaba
Aceita.

Informe uma cadeia (Digite 'sair' para fechar o programa): cadadabac
Nao aceita.

Informe uma cadeia (Digite 'sair' para fechar o programa): bca
Transicao indefinida para simbolo 'b' a partir do estado q0
Nao aceita.
```

figura 7

CONCLUSÃO

Em suma, o código apresentado desempenhou o papel esperado, tanto em funcionamento, quanto como um objeto de aprendizado ativamente diferenciado, exemplificando como os AFDs podem ser implementados e utilizados para reconhecer linguagens específicas.

A implementação de um Autômato Finito Determinístico em Java, conforme discutido, oferece uma experiência prática que reforça os conceitos teóricos aprendidos em Linguagens Formais Autômatos e Computabilidade, mostrando como os estados, transições e alfabetos são representados e manipulados programaticamente. Essa abordagem não só facilita a compreensão dos autômatos finitos, mas também demonstra a importância da interdisciplinaridade no ensino de conceitos complexos, promovendo uma aprendizagem mais profunda e integrada.

REFERÊNCIAS

[1] MAZUR, Eric. **Peer Instruction -A Revolução da Aprendizagem Ativa**.Editora Penso. Ano 2015.

[2] "Introduction to Algorithms" de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest e Clifford Stein