UNIVERSIDADE FEDERAL DE OURO PRETO INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS - ICEA

Implementação de AFDs e Operações para Reconhecimento Prático de Linguagens Formais

Gustavo E. Sena[1] Jordan D. Bragon[1] Yuri S.Ribeiro[1]

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto(UFOP)

Caixa Postal 24 – 35.931-008 – João Monlevade – MG – Brazil

{gustavo.estevam, jordan.bragon, yuri.sr}@aluno.ufop.edu.br

Abstract. *The main purpose of this work is to present the practical implementation of a tool coded in C to work with theoretical formalisms of language recognition through algorithms representing the operations of visualization of an AFD, complement of AFD, intersection between AFDs, union between AFDs, minimization and word recognition.*

Resumo. *O propósito principal desse trabalho é apresentar a implementação prática de uma ferramenta codificada em C para trabalhar com formalismos teóricos de reconhecimento de linguagens através de algoritmos representando as operações de visualização de um AFD, complemento de um AFD, interseção entre AFDs, união entre AFDs , minimização e reconhecimento de palavra.*

1. Introdução

De acordo com Michael Sipser [1]:

> Um autômato finito possui um conjunto de estados e regras para ir de um estado para outro, dependendo do símbolo de entrada. Possui um alfabeto de entrada que indica os símbolos de entrada permitidos. Tem um estado inicial e um conjunto de estados de aceitação. A definição formal diz que um autômato finito é uma lista de cinco objetos: conjunto de estados, alfabeto de entrada, regras para movimentação, estado inicial e estados de aceitação.

A definição formal desse tipo de autômato é uma quíntupla, representada por (E, Σ, δ, i, F) , onde:

- E é o conjunto finito de estados.
- Σ é o conjunto finito chamado de Alfabeto.
- δ : E × Σ → E é a função de transição
- i: é um estado inicial de E
- F: é o conjunto de estados finais(de aceitação) contido em E

Um estado de erro em um AFD: e' (estado de erro) ocorre quando não existe transição de e sob a no diagrama de estados, então existe um estado e'. Nesse caso: 1) Existe uma transição de e para e' sob a – 2) e' não é um estado final – 3) Existe um transição de e' para e' sob cada símbolo do alfabeto. Um diagrama sem os estados de erros é chamado de diagrama simplificado [4] .

Um diagrama simplificado de um Autômato Finito para a linguagem L1 = linguagem que reconhece palavras com subcadeias 001, com alfabeto Σ = {0, 1} é como segue [2]:

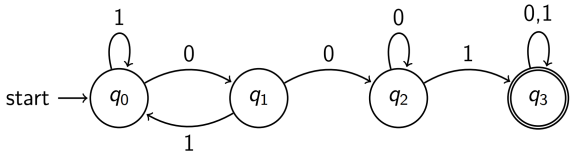


Figura 1- AFD para L1 (Evandro Eduardo Seron Ruiz, USP, 2020) [2]

2. Solução das Funcionalidades Propostas na Implementação

As entradas para as operações seguintes são AFDs representados por txts, contendo sua especificação. Os AFDs abaixo estão em formato .txt e são dois exemplos utilizados.

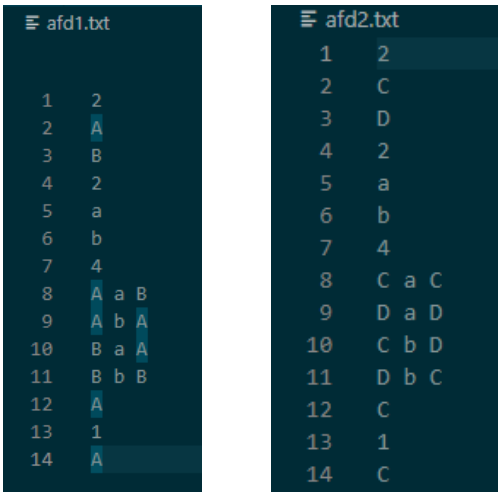
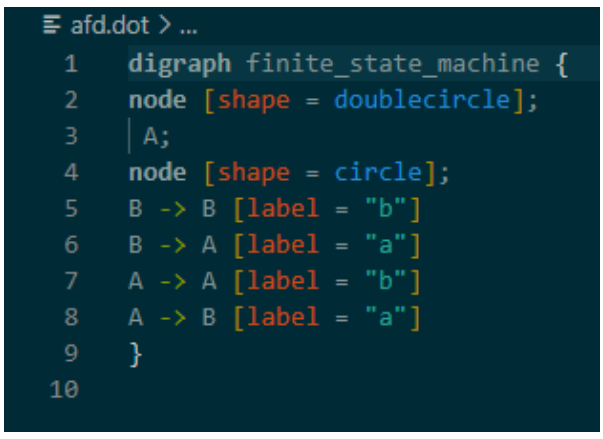


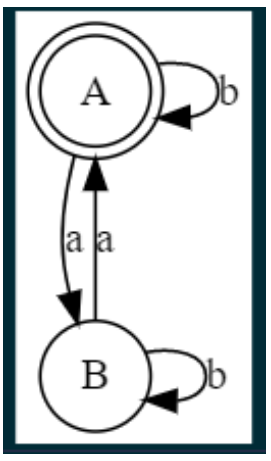
Figura 2- Formato de Entrada de AFDs

2.1 Funcionalidade 1: Visualização

O programa foi concebido para ler um AFD no formato de entrada txt e escrever o mesmo AFD em um formato de saída DOT pronto para visualização com o **GraphViz** [3]. O código do arquivo de saída na linguagem dot para o exemplo 1 , sua visualização e seu pseudocódigo estão contidos nas imagens a seguir:

Figura 3- Visualização de AFD





```
ConverterAFDParaDot(char *nomeArquivoEntrada, char *nomeArquivoSaida)

    Criar estrutura do AFD
    Ler nomeArquivoEntrada e popular o AFD

    int i = 0;
    int j = 0;

    Estados = AFD->Estados
    Transicoes = Estados = AFD->Estados
    Gerar arquivo dot a partir do AFD
        Enquanto i for menor que a quantidades de estados finais faça:
            Escreva o estado[i] final
        fim enquanto

        Enquanto j for menor que a quantidades de transicoes faça:
            Escreva a transicao[j]
        fim enquanto

        Gere o arquivo
        fechar arquivo dot
```

2.2 Funcionalidade 2: Complemento

De maneira direta, o complemento de MA, descrito como MA’ (MA Barra) é o conjunto de todos os elementos sob consideração que não estão em MA. Esse Autômato complementar de MA tem os mesmos estados, o mesmo alfabeto, as mesmas transições e estado inicial de MA. No entanto, a diferença entre eles é que MA’ reconhece as palavras que MA não reconhece, ou seja, coloca como estados finais aqueles que não são finais em MA. Dessa forma, a quíntupla de MA’ é definida por MA′ = (E, Σ, δ, i, E − F) [5]. Para representar essa operação foi construído o pseudocódigo a seguir:

```
ComplementoAFD(char *nomeArquivoEntrada, char *nomeArquivoSaida)
    Criar estrutura do AFD;
    Ler nomeArquivoEntrada e popular o AFD;;

    Estados = AFD->Estados;
    int i = 0;

    Enquanto i for menor que a quantidades de estados faça:
        se o estado for final então:
            atribuir estado[i] como não final;
        se não
            atribuir estado[i] como final;
        fim-se;
    fim enquanto

    Gere o arquivo AFD
    fechar arquivo AFD
```

Figura 4- Complemento de AFD

2.3 Funcionalidade 3: Interseção e União

O método produto de autômatos permite construir um AFD produto P a partir de dois AFDs, MA e MB, de tal forma que P reconhece L(MA) ∪ L(MB), ou então L(MA) ∩ L(MB), dependendo apenas da escolha que se faça do conjunto de estados finais de P [5].

A Interseção de dois AFDs é dada como resultado do produto desses AFDs, enquanto os estados finais dependem dos estados finais dos dois AFDs de origem. A união entre dois AFDs é gerada pelo produto, sendo que os estados finais nesse tipo de operação são os que apresentam pelo menos um dos estados finais nos AFDs de Origem:

- O estado inicial em um produto de AFDs é concebido a partir dos estados iniciais de cada AFD.
- Os símbolos para os AFDs de origem e o AFD produto resultante são os mesmos
- A quantidade de estados é a multiplicação entre a quantidade de estados da AFD MA e a quantidade de estados da AFD MB.
- As transições da AFD Produto são geradas a partir das transições de cada AFD, representadas pela junção da transição do estado da AFD MA com um determinado símbolo X e com a transição do estado da AFD MB em que se utiliza esse mesmo símbolo X.

Dessa forma, para representar as operações foi construído os pseudocódigos a seguir:

```
IntersecaoAFD(char *nomeArquivoAFD1, char *nomeArquivoAFD2, char *nomeArquivoSaida)

    Criar estrutura do AFD1
    Criar estrutura do AFD2
    Criar a estrura do AFDIntersecao

    Ler nomeArquivoAFD1 e popular o AFD1
    Ler nomeArquivoAFD2 e popular o AFD1

    AFDIntersecao = Recebe o produto entre os AFDs 1 e 2

    EstadosFinaisAFD1 = Buscar estados finais do AFD1
    EstadosFinaisAFD2 = Buscar estados finais do AFD2

    int i = 0;
    int j = 0;

    Enquanto i for menor que a quantidades de EstadosFinaisAFD1 faça:
        Enquanto j for menor que a quantidades de EstadosFinaisAFD2 faça:
            Concatenar EstadosFinaisAFD1[i] e EstadosFinaisAFD2[j]
            Buscar o estado concatenado no AFDIntersecao
            Atribuir estado como final no AFDIntersecao
        fim enquanto
    fim enquanto

    Gere o arquivo AFDIntersecao
    fechar arquivo AFDIntersecao
```

Figura 5- Interseção

```
UniaoAFD(char *nomeArquivoAFD1, char *nomeArquivoAFD2, char *nomeArquivoSaida)

    Criar estrutura do AFD1
    Criar estrutura do AFD2
    Criar a estrura do AFDUniao

    Ler arquivo e popular o AFD1
    Ler arquivo e popular o AFD1

    AFDUniao = Recebe o produto entre os AFDs 1 e 2

    EstadosFinaisAFD1 = Buscar estados finais do AFD1
    EstadosFinaisAFD2 = Buscar estados finais do AFD2

    int i = 0;
    int j = 0;

    Enquanto i for menor que a quantidades de AFDUniao->Estados faça:

        Enquanto j for menor que a quantidades de EstadosFinaisAFD1 faça:
            Se o estado final do AFD1 estiver contido no Estado do AFDUniao entao:
                Buscar o estado no AFDUniao;
                Atribuir estado como final no AFDUniao;
            fim se
        fim enquanto

        Enquanto j for menor que a quantidades de EstadosFinaisAFD2 faça:
            Se o estado final do AFD12 estiver contido no Estado do AFDUniao entao:
                Buscar o estado no AFDUniao;
                Atribuir estado como final no AFDUniao;
            fim se
        fim enquanto

    fim enquanto

    Gere o arquivo AFDUniao
    fechar arquivo AFDUniao
```

Figura 6- União

2.4 Funcionalidade 4: Reconhecimento de Palavras

O reconhecimento bem sucedido de uma determinada palavra, resulta no consumo de todos os símbolos que a compõe e após o consumo do último símbolo o AFD encontra-se em um estado final.

Dessa forma, para representar a operação foi construído o pseudocódigo a seguir:

```
ReconhecerPalavra(char *nomeArquivoAFD, char *nomeArquivoPalavras, char *nomeArquivoSaida)
    Criar estrutura do AFD
    Ler nomeArquivoAFD e popular o AFD

    arquivoPalavras = ler arquivo arquivo de Palavras

    Gerar arquivo

    Enquanto a linha do arquivo não for vazia faça:

        Estado estado = buscar estado inicial AFD

        Enquanto simbolo da linha não for vazio
            estado = busca estado destino de acordo com o simbolo
        fim enquanto

        Se o estado é final
            escreva 1 no arquivo de saída
        Se não
            escreva 0 no arquivo de saída
        fim se

    fim enquanto

    Fechar arquivo de saída
```

Figura 7- Reconhecimento de Palavras

3. Guia de Uso do Software

1) O arquivo makefile, deve conter o seguinte comando do gcc para realizar a compilação de todos os arquivos utilizados no projeto:

$ main:
gcc -g -Wall -o afdtool main.c src/header.h src/util.c src/AFD/AFDFuncoes.c src/operacoes/visualizar.c src/operacoes/complemento.c src/operacoes/intersecao.c src/operacoes/uniao.c src/operacoes/reconhecer.c src/operacoes/reconhecer.c

2) Em seguida execute o makefile para compilar:

$ make main

3) Para realizar as operações disponíveis na ferramenta execute:

● Visualização:
$ ./afdtool --dot afd1.txt --output afd1.dot

● Complemento:
$ ./afdtool --complemento afd1.txt --output afdcom.txt

● União:
$ ./afdtool --uniao afd1.txt afd2.txt --output afdu.txt

● Interseção:
$ ./afdtool --intersecao afd1.txt afd2.txt --output afdi.txt

● Reconhecimento de Palavras:
$ ./afdtool --reconhecimento afd1.txt afd2.txt --output afdi.txt

4. Testes e Resultado

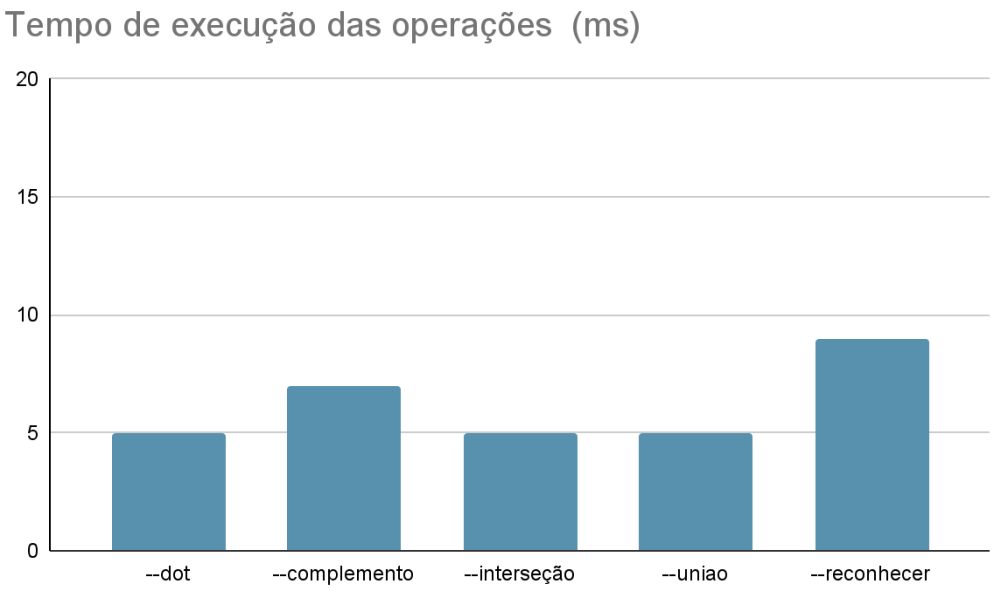


Figura 8 - Tempo de execução das funções do Algoritmo

Pelos dados apresentados, a maioria das funções apresentaram tempo médio de execução em 3 testes de 5 milissegundos. Interseção e união utilizaram os mesmos AFDs. As funções dot, complemento e reconhecimento utilizaram o mesmo AFD, com a função de reconhecimento testando 15 palavras. Baseado nisso, a função de reconhecimento de palavra apresentou o maior tempo médio de execução com 9 milissegundos.

5. Conclusão

De maneira geral esse Trabalho Prático foi importante para aplicar os conceitos de Reconhecimento de Linguagem formal e operações, explorando de forma mais abrangente os Fundamentos Teóricos ministrados em sala durante as primeiras aulas da disciplina. Foi interessante e desafiador por solicitar um conhecimento técnico sobre a disciplina e da linguagem de programação C, assim conseguimos praticar e reforçar o aprendizado. Também foi possível conhecer e usar uma nova ferramenta, Graphviz, o que é um bônus no escopo de abordagem da Disciplina.

Referências

[1]SIPSER, Michael. **Introdução à Teoria da Computação: Trad. 2ª ed. norte-americana**. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9788522108862/pageid/51. Acesso em: 10 out. 2022.

[2]RUIZ, Evandro Eduardo Seron. **Formalização de Autômatos Finitos Determinísticos (AFD)**. 2022. Disponível em: https://edisciplinas.usp.br/pluginfile.php/1817119/mod_resource/content/1/formal_AFD.pdf. Acesso em: 10 out. 2022.

[3]GRAPHVIZ. **What is Graphviz?** 2022. Disponível em: https://graphviz.org/. Acesso em: 10 out. 2022.

[4]ÁLVARES, Andrei Rimsa. **Autômatos Finitos Determinísticos**. 2020. CEFET-MG - Com base no Livro do professor Newton Vieira. Disponível em: http://rimsa.com.br/documents/lectures/decom035/lessons/Aula02.pdf. Acesso em: 10 out. 2022.

[5]VIEIRA, Newton José. **Introdução aos fundamentos da computação: Linguagens e máquinas.** 2018. Disponível em: https://homepages.dcc.ufmg.br/~nvieira/cursos/tl/a18s2/material.html. Acesso em: 10 out. 2022.

[6]MAKE, Gnu. **GNU Make**. 2022. Disponível em: https://www.gnu.org/software/make/. Acesso em: 10 out. 2022.