Relatório Trabalho Prático 1

Lucas Rômulo

Março 2021 (UFSJ/ERE-02)

1 Definição

- Autômato: máquina que percorre uma entrada e a qualifica com aceita ou não, de acordo com suas regras de formação.
- AFD (Autômato Finito Determinístico): a cada estado e entrada, vai para um único estado destino.
- AFN (Autômato Finito Não-determinístico): tendo um estado e uma entrada, pode ir para um ou mais estado(s).
- AFNϵ (Autômato Finito Não-determinístico com movimentos vazios): um AFN com movimentos vazios entre os estados.
- ER (Expressão Regular): uma expressão na forma "humana" que é equivalente a um autômato.

2 Introdução

A proposta desse trabalho é tratar linguagens e autômatos. Será visto a transformação de uma ER em um AFD, sendo que essa transformação segue os seguintes passos: ER $->$ AFNϵ $->$ AFN $->$ AFD.

Após a transformação da ER em AFD, o AFD é implementado utilizando a linguagem C. Vale ressaltar que, em algumas entradas, o AFD se comporta como um AFN, então o código implementado é o de um AFN.

O autômato tem como entrada o primeiro nome, segundo nome e o número de matrícula do usuário, sendo que o usuário deve possuir uma matrícula compatível com o formato da matrícula da UFSJ; e as palavras a serem analisadas. Após a análise de cada palavra, uma saída é atribuída a ela, de acordo com a aceitação/rejeição pela máquina.

3 Linguagem

A linguagem utilizada no TP depende das entradas (nomes e matrícula), e é constituída do seguinte alfabeto(Σ) :

- x1: número de letras do primeiro nome
- x2: número de letras do segundo nome

- d2: segundo número da matrícula
- d9: nono número da matrícula
- l1: primeira letra do primeiro nome
- l2: segunda letra do primeiro nome

A ER é da forma x1(d2l1 + d9l2)$^+$x2.

4 Transformação da ER em AFD

4.1 ER $->$ AFNϵ

Para transformar a ER x1(d2l1 + d9l2)$^+$x2, primeiro é necessário dividi-la em ER menores, e então são criados autômatos para essas expressões. A divisão da ER fica como: ER(1) = x1; ER(2) = x2; ER(3) = (d2l1 + d9l2)$^+$.

A partir das ERs 1, 2 e 3, são criados 3 AFNϵ correspondentes.

4.1.1 ER(1) $->$ AFNϵ(1)

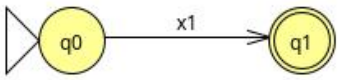

Figure 1: Transformação do ER(1) em AFNϵ(1)

4.1.2 ER(2) $->$ AFNϵ(2)

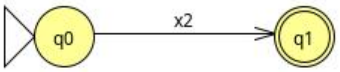

Figure 2: Transformação do ER(2) em AFNϵ(2)

4.1.3 ER(3) $->$ AFNϵ(3)

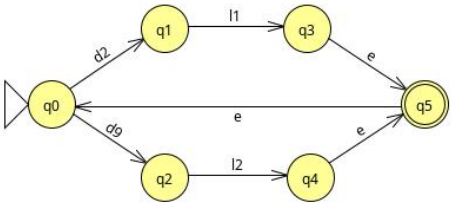

Figure 3: Transformação do ER(3) em AFNϵ(3)

4.2 AFNϵ $->$ AFN

Nesse passo, os AFNϵs obtidos no passo anterior são transformados em AFN.

4.2.1 AFNϵ(1) $->$ AFN(1)

AFN(1) = AFNϵ(1).

4.2.2 AFNϵ(2) $->$ AFN(2)

AFN(2) = AFNϵ(2).

4.2.3 AFNϵ(3) $->$ AFN(3)

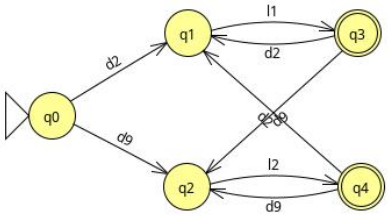

Figure 4: Transformação do AFNϵ(3) em AFN(3)

4.3 AFN $->$ AFD

Agora os AFNs são transformados em AFDs.

4.3.1 AFN(1) $->$ AFD(1)

AFD(1) = AFN(1).

4.3.2 AFN(2) $->$ AFD(2)

AFD(2) = AFN(2).

4.3.3 AFN(3) $->$ AFD(3)

AFD(3) = AFN(3).

4.4 ER $->$ AFD

Agora, para transformar os AFDs em um único AFD que corresponde a ER, os estados finais de um AFD são conectados ao estado inicial do próximo AFD, na seguinte ordem: AFD(1)AFD(3)AFD(2).

A máquina resultante da concatenação dos AFDs, é a máquina que aceita/rejeita palavras da ER x1(d2l1 + d9l2)$^+$x2, que é vista logo em seguida:

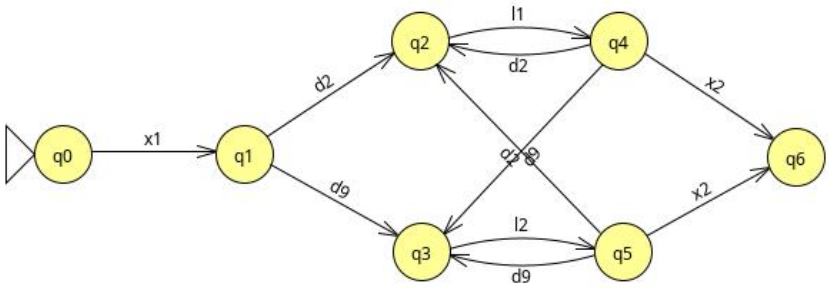

Figure 5: Transformação da ER em AFD

5 Implementação

Em algumas ocasiões, o autômato AFD não seria válido para algumas linguagens, então, o autômato implementado foi um AFN, pois possui a possibilidade de ir para mais de um estado a partir de um estado e uma entrada.

A implementação se deu utilizando a linguagem C e os editores de texto Nano e Kate.

5.1 Entrada

A entrada das informações(nomes e matrícula) se dá pela leitura de um arquivo init.txt na pasta raiz do projeto. A entrada das palavras a serem analisadas depende da escolha do tipo de execução.

Existem duas possibilidades para a execução:

- palavras digitadas à mão na CLI;
- palavras lidas a partir de um arquivo.

Caso a pessoa opte pela entrada tendo as palavras lidas de um arquivo, essas palavras devem ser escritas no arquivo init.txt, pulando uma linha após as informações de entrada, e com uma linha entre cada uma delas.

5.2 Saída

A máquina analisa a palavra letra a letra, e existem 3 (três) possíveis saídas:

- Palavra aceita: após a leitura de toda a palavra, existe algum estado final ativo;
- Palavra rejeitada por indefinição: a partir de um estado e uma entrada, não existe(m) estado(s) futuro(s) para prosseguir;
- Palavra rejeitada após lida: após a leitura de toda a palavra, não existem estados finais ativos.

De acordo com a linguagem que tem como entrada os nomes "lucas", "romulo", e a matrícula "172500036", temos alguns exemplos de palavras:

- Aceitas: 57l6, 56u6, 57l7l6, 57l6u6, 56u7l6, 56u6u6, 57l7l7l6, 56u6u6u6, 57l7l6u6, 56u6u7l6;
- Rejeitadas por indefinição: a, b, c, d, e;
- Rejeitadas por não ter estado final após ler a palavra: 57l, 56l, 57, 56, 5.

O passo a passo da máquina, e a saída, são apresentadas ao usuário através da CLI, de forma organizada, palavra por palavra. A leitura da situação atual do programa é essencial para a utilização.

5.3 Código

5.3.1 Fluxo de execução

O programa faz as leituras e verificações necessárias para a execução. Caso tudo esteja em ordem, um AFN é criado a partir das informações de entrada.

Após a criação do AFN, o programa entra em uma repetição onde lê as palavras até que a palavra seja um NULL. A palavra atual é analisada, e a saída é retornada ao usuário.

Após a leitura das palavras, o programa é finalizado.

5.3.2 O tipo abstrato de dados AFN

No arquivo AFN.h são declaradas algumas estruturas e o tipo abstrato de dados. Sabendo que o formato de uma máquina é M=(Q,Σ,δ,q0,qf), os dados que foram utilizados tentam se assemelhar o máximo possível a esse formato.

A estrutura q_t representa o Q, e cada q_t tem uma flag para que indica se é um estado final. q0 e qf fazer parte de Q.

Sigma_t representa Σ, e cada sigma possui um nome e um valor x1, 5, p.e.

Delta_t é um grafo que liga cada Q a uma entrada de Σ, representando assim as regras de produção δ.

O TAD AFN possui um vetor de q_t, representando o Q, q1, qf; um vetor de Sigma_t que representa o alfabeto Σ; um grafo que representa as regras δ; e dois inteiros q_size e sigma_size, que representam o tamando de Q e Σ, respectivamente.

O TAD é constituído também de funções relacionadas diretamente ao AFN, e procedimentos auxiliares para o funcionamento das funções princiais. Essas funções e procedimentos estão listados aqui em seguida.

5.3.3 Funções

O TAD constitui de 3 funções básicas:

- Criação de um AFN(new_automata): recebe como parâmetro as informações (nomes e matrícula) lidas do arquivo e cria um AFN baseado nessas informações;
- Exibição de um AFN(print_AFN): apresentação visual mais "humana" do AFN para o usuário;

- Verificação de uma palavra(verify_word): recebe uma palavra como entrada, a mesma é analisada pelo AFN e é apresentada a saída relacionada àquela palavra.

5.3.4 Procedimentos auxiliares

- Cria Q(new_q): cria um vetor de q_t e marca o estado q6 como final;
- Cria Σ(new_sigma): extrai as informações x1, x2, d2, d9, l1, l2 e as salva em um vetor, nessa ordem;
- Cria δ(new_delta): cria o grafo que relaciona QxΣ. Dá o valor -1 a todas as arestas do grafo, e depois preenche os valores relativos as regras de δ. As arestas que ficam com valor -1 representam as indefinições;
- Verifica entrada(verify_entry): verifica se a entrada pertence ao alfabeto Σ antes de tentar usá-la. Caso a entrada não pertença a Σ, o programa já retorna uma indefinição.

6 Execução

Nenhum binário acompanha o projeto, então é necessária a criação do mesmo. É utilizado um makefile para automatizar a tarefa de criação do binário, para isso, basta seguir os passos:

- cd pasta/do/projeto
- make
- make install

O binário é criado na pasta ./bin/, então, para facilitar, a pasta pode ser acessada pelo comando *cd ./bin/*. A execução também pode ser feita a partir da pasta raiz do projeto, utilizando *./bin/automata <flag>*. Como existem 2 tipos de execução (arquivo/entrada padrão), são usadas flags para selecionar o tipo de execução. As flags:

- Utilizando arquivo(**F**ile): **./automata f**, as palavras são lidas a partir do arquivo init.txt;
- Utilizando a entrada padrão(C**LI**): **./automata l**, as palavras são digitadas na CLI.

7 Conclusão

O propósito desse trabalho foi apresentar o passo a passo para a transformação de uma ER em um AFD, e um exeplo de implementação para o AFD; e pode-se perceber que as linguagens são diferentes de acordo com as entradas (nomes e matrícula), mesmo que a ER seja a mesma; inclusive, para algumas linguagens a máquina funciona como um AFD, e para outras, como um AFN.