UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
CIÊNCIA DA COMPUTAÇÃO

Formalismos

Teoria de Linguagens

Bárbara Boechat

São João del-Rei
Março de 2021

Sumário

1 Introdução

Neste trabalho serão expostos os formalismos usados para a obtenção de um autômato finito determinístico a partir de uma expressão regular, na seguinte ordem: Expressão Regular (ER); Autômato Finito Não Determinístico com movimentos vazios (AFN-e); Autômato Finito Não Determinístico (AFN); Autômato Finito Determinístico (AFD); Gramática Linear. Além disso, também será exposta a implementação de um AFD e a iteração com sua interface gráfica.

2 Expressão Regular

Toda linguagem regular pode ser descrita por uma ER, pois ela é um formalismo denotacional (gerador) e é definida a partir de conjuntos (linguagens) básicos, concatenação e união.

Em compiladores, por exemplo, ao projetar linguagens de programação, as ER são úteis no processo de análise sintática. Além de serem adequadas para a comunicação entre humanos ou de humanos com as máquinas.

2.1 Definição Formal

- $\emptyset$ é Expressão Regular e denota a linguagem vazia;
- ε é Expressão Regular e denota a linguagem ε;
- x é Expressão Regular para qualquer x que pertencente ao alfabeto (Σ);
- Se R e S são linguagens regulares, então as linguagens formadas pela União (R + S); Concatenação (RS); Concatenação Sucessiva (R*) é Expressão Regular e denotam respectivamente as linguagens R $\cup$ S, RS = {uv — u $\in$ R e v $\in$ S}, R'*.

A seguir, na tabela 1 exemplos de linguagens geradas por expressões regulares.

ER	Linguagem Gerada
aa	Somente a palavra aa
ba*	Palavras iniciadas em b, seguido por zero ou mais a
(a + b)*	Todas as palavras sobre { a, b }
(a + ε)(b + ba)*	Todas as palavras que não possuem dois a consecutivos

Tabela 1: Linguagens Geradas por ER

Neste trabalho foi adotada a expressão regular **7(0b + 9a)+ 7**, que expressa palavras começadas por 7 concatenadas de um ou mais 0b ou 9a e terminadas por 7, a seguir na seção 4 será definido um AFN-e a partir desta ER.

3 Gramática Linear

Uma gramática regular aplica restrições nas regras de produção e existe mais de uma forma de restringi-las, como por exemplo Gramática Linear à Direita (GLD) em que as produções são da forma A→wB ou A→w Gramática Linear à Esquerda (GLE) em que suas produções são da forma A→Bw ou A→w, e as Gramáticas Lineares(GL), as quais tem no máximo um símbolo não-terminal no lado direito de suas produções, como mostrada em 2 uma linguagem livre de contexto para a ER apresentada.

Variáveis	**Produções**
q0	*q1*7
q1	*q2*9a / *q2*0b
q2	*q1* / *qf*7
qf	ε

Tabela 2: GL para a ER definida

4 AFN-e

Um autômato finito não determinístico com movimentos vazios é definido por M = (Σ, Q, δ, q0, F). O autômato definido para este trabalho M = ({7, 0b, 9a}, {q0, q1, q2, qf}, δ, {qf})

- Σ - Alfabeto (de símbolos de entrada)
- Q - Conjunto de Estados possíveis
- δ - Função programa
- q0 - Estado inicial
- F - Conjunto de Estados Finais

A computação de um AFN-e é análoga à de um AFN, assim, ele é também não-determinístico assumindo simultaneamente mais de um estado. No caso do processamento de uma transição vazia ele assume ao mesmo tempo os estados destino e origem, e a sua leitura não consome nenhum simbolo da fita.

4.1 AFN-e a partir de uma ER

Neste trabalho o AFN-e foi construído seguindo as definições da figura 1 para cada parte da ER definida por **7(0b + 9a)+ 7**. Porém o AFN-e da figura 2 também poderia ter sido gerado pela GL definida na tabela 2.

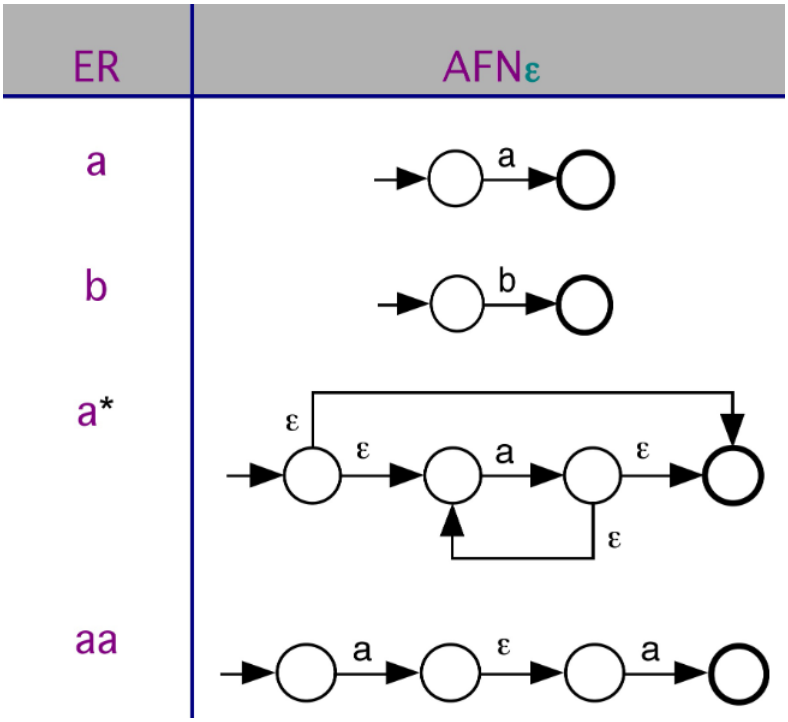

Figura 1: Autômato Implementado

Porém é preciso destacar que algumas das transições vazias foram simplificadas para que o autômato consequentemente não gerasse um AFN muito grande na próxima etapa descrita na 5.

Figura 2: AFN-e Gerado

As etapas do AFN-e que foram simplificadas foram a que define as concatenações sucessivas de um ou mais símbolos e sua união com o prefixo e o sufixo tiveram suas transições vazias simplificados.

5 AFN

Comparando 4 com um autômato finito não determinístico, o AFN, é possível destacar que a diferença entre ambos é que o segundo não pode realizar movimentos vazios, ou seja, não existe a transição vazia de um estado para o outro na função programa do AFN. Contudo, é possível construir um AFN a partir de um AFN-e, de maneira simplificada as seguintes etapas foram seguidas:

1. Quais serão os estados finais?
    - O único estado final será 'qf'.
2. Fecho vazio de cada estado?
    - O único estado com transição vazia é o estado 'q2', assim seu fecho vazio corresponde a (q2, ε) $\cup$ ((q2, ε), {9a, 0b, 7, ε}) = {q1, q2}
3. Substituição das transições vazias
    - De acordo a 2, o estado q2 lendo os símbolos {9a, 0b} tem como destino o próprio q2 e o estado q1, assim a substituição da transição vazia resulta em um self-loop para q2, e uma aresta para q1, como mostrado por MND = ({7, 0b, 9a}, {q0, q1, q2, qf}, δ, {qf}) em 3.

Figura 3: AFN Gerado

6 AFD

A partir de um AFN qualquer é possível construir um AFD que realiza as mesmas combinações, fazendo com que o segundo simule o primeiro. Os estados do AFD simulam combinações de estados alternativos do AFD, assim, é possível verificar através da tabela 3 as combinações para que o autômato da figura 4 seja gerado. Além disso, é preciso destacar que diferentemente do AFN, o AFD não assume simultaneamente mais de um estado.

A tabela é gerada a partir do AFN da figura 3 de maneira que devem ser representadas todas as transições existentes nesse autômato, assim, o conjunto de símbolos lidos acumulado em cada célula da tabela vira um novo estado e suas transições também devem ser computadas até que todas as possibilidades tenham sido verificadas.

	Símbolos		
Estados	7	0b	9a
>q0	q1	–	–
q1	–	q2	q2
q2	qf*	q1q2	q1q2
q1q2	qf*	q1q2	q1q2
qf*	–	–	–

Tabela 3: Linguagens Geradas por ER

Após a criação da tabela de transformação, a construção do AFD da figura 4 apenas seguiu as diretivas das transições entre os estados, sempre observando que caso haja um estado final no conjunto de estados de uma célula, o novo estado gerado é também final. Gerando assim MD = ({7, 0b, 9a}, {q0, q1, q2, q1q2, qf}, δ, {qf})

Figura 4: AFD Implementado

6.1 Implementação

A implementação do AFD foi realizada em JavaScript, porém nenhum framework como Vue ou Angular foi utilizado na construção do projeto, assim é necessário apenas abrir em algum navegador o arquivo index.html ou acessar o link gitpages/teoria.

O código do autômato em si é bem simples, sua construção é a partir de uma classe que recebe como parâmetros os estados do autômato, sendo o estado inicial marcado com o simbolo '>' e o estado final com o simbolo '*', seu alfabeto e as transições existentes entre os estados. Assim, essa classe implementa funções de transição entre os estados,

para determinar se a leitura feita tem como destino um estado alcançável, se o estado lido é final e se o simbolo lido pertence ou não ao alfabeto.

Enfim, cada entrada ou fita é executa simbolo a simbolo, determinando o novo estado atual, caso não haja transição a partir do estado atual com o simbolo lido, a ação é considerada indefinição; caso o simbolo lido não faça parte do alfabeto um erro é retornado; caso o simbolo lido seja o último da fita porém o estado atual não seja final a palavra é rejeitada e o código emite um log de estado não final e o estado atual.

6.2 Interface

Só é possível fornecer entradas ao autômato por meio da interface gráfica, assim, foram implementadas duas possibilidades de entrada, por meio de arquivo, ou entradas fornecidas separadamente.

Afim de fornecer entradas válidas os símbolos da fita devem ser separados com vírgula e no caso dos arquivos apenas serão aceitos arquivos que contenham uma palavra por linha. Por exemplo, uma entrada válida seria **7, 0b, 9a, 7**.

6.3 Animação de Leitura

Após a inicialização e durante a leitura da fita o autômato mostrado na 4 terá seus estados pintados a medida que a leitura avança, como nas figuras 5 e 6. Dessa maneira, o caminho de nós e arestas feito até sua finalização ficará pintado por alguns segundos e logo voltará ao estado inicial com todos os nós descoloridos.

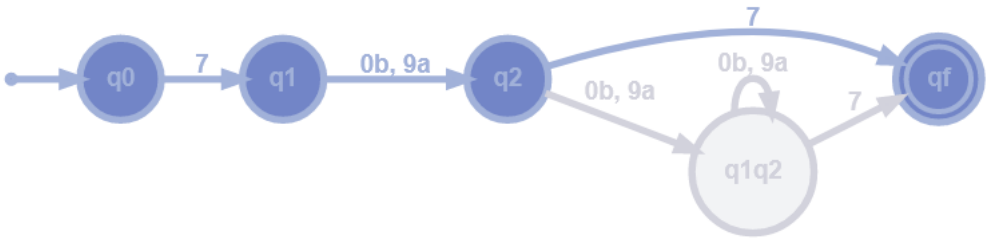

Figura 5: Caminho feito pela entrada **7, 0b, 7**

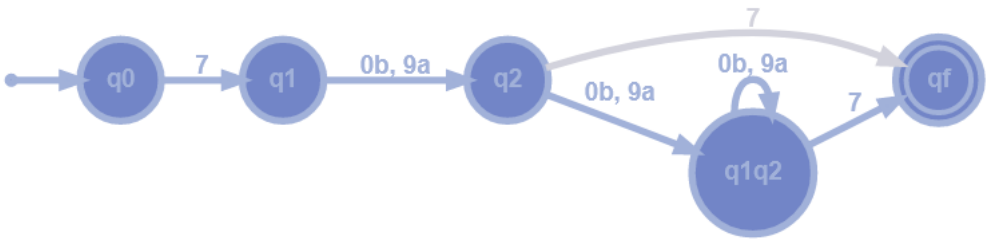

Figura 6: Caminho feito pela entrada **7, 0b, 0b, 9a, 7**

6.4 Log de Leitura

Cada leitura realizada pelo autômato emite um log de leitura correspondente às transições realizadas para que o usuário possa acompanhar a animação e mesmo assim ter registrado os estados e leituras pelos quais o código passou para atingir seu parecer final.

Na figura 7 após a leitura, a palavra é aceita e o estado final neste histórico das transições assume a cor verde claro para indicar o final bem sucedido.

Figura 7: Log de Leitura - Palavra Aceita

Para o caso da fita ter acabado e o autômato ter terminado sua leitura em um estado não final, a figura 8 mostra que o log do estado final assume a cor verde escuro.

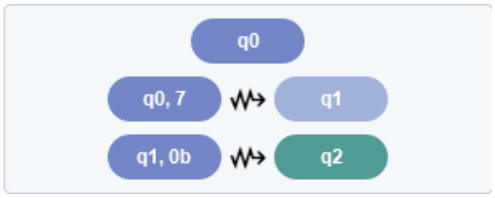

Figura 8: Log de Leitura - Estado Não Final

E por fim, para a indefinição, como mostrado na figura 9, o autômato indica pela palavra Indefinido que a leitura de um simbolo a partir de certo estado levaria a um próximo estado indefinido.

Figura 9: Log de Leitura - Palavra Indefinida

6.5 Log Final

No log final, afim de manter um registro de todas as entradas já lidas e de seus pareceres uma mensagem é emitida ao final de cada leitura, como mostrado na figura 10, são mostradas a entrada, um símbolo positivo caso ela tenha sido aceita; um símbolo negativo caso não e uma mensagem de acordo aos três possíveis casos citados anteriormente.

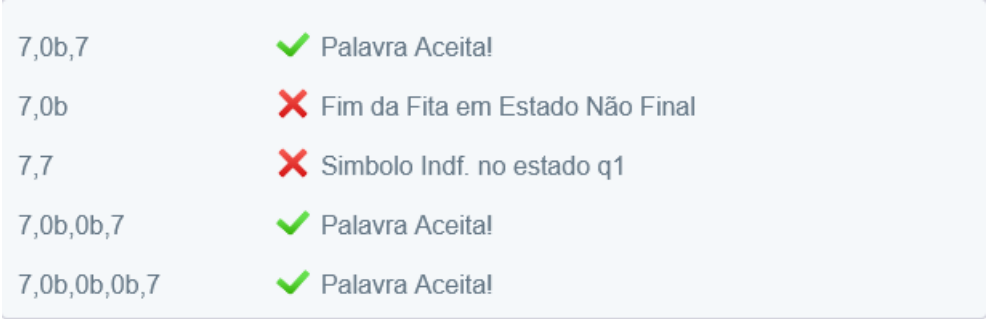

Figura 10: Log Final

7 Conclusão

Defronte os passos apresentados para a construção do AFD, a parte de maior dificuldade encontrada foram as definições para transformar o AFN-e em AFN, pois na prática parece um tanto trivial sua realização, porém transformar em teoria é consideravelmente mais desafiador. Quanto a implementação, as dificuldades foram encontradas na implementação da interface gráfica, já que a linguagem JavaScript trabalha com o conceito de assincronia, foi relativamente difícil conciliar a leitura da fita com os eventos da interface. Contudo, a implementação do AFD em si foi bem tranquila e livre de maiores problemas alcançando o resultado esperado.