Teoria de Linguagens - Trabalho Prático 1

Carlos Magno Geraldo Barbosa

Novembro de 2020

1 Introdução

Teoria de linguagens formais, nasceu na década de 1950, com o proposito de desenvolver teorias relacionadas as linguagens naturais. Porém, logo se observou que esta teoria poderia permitir a evolução dos estudos relacionados a linguagens artificiais, em especial para linguagens originárias da computação. Desde então, essa evolução resultou no desenvolvimento de diversas áreas, com destaque para aplicações em análises léxicas e sintática de linguagens de programação, além de aplicações em modelagem de circuitos lógicos, sistemas biológicos e recentemente animações e tratamento de linguagens não lineares [1].

Portanto, ciente de toda a importância deste estudo para a ciência da computação e diversas áreas, este trabalho apresenta uma breve exposição do conteúdo visto em sala de aula (Capítulos de linguagens regulares e propriedades de linguagens regulares), a partir de anotações, slides e do livro linguagens formais e autômatos [1]. Além da exposição do conteúdo é apresentado a estrutura básica da implementação de um simulador de um Autômato Finito Determinístico (AFD), desenvolvido na linguagem python, contando com uma interface WEB para sua utilização. O automato apresentado neste trabalho é equivalente a expressão regular(ER) personalizada apresentado em 2. O processo de transformação da ER no AFD implementado foi realizado seguindo a equação em 1.

$$ER \rightarrow AFN\varepsilon \rightarrow AFN \rightarrow AFD \rightarrow AFD_{minimo} \tag{1}$$

$$C1(09 + AL)^*3 \tag{2}$$

Um conceito inicial de grande importância para este trabalho é o conceito de linguagens formais. No qual, temos atrelado os conceitos de alfabeto e palavra. Podemos definir alfabeto como um conjunto finito de símbolos ou caracteres. Temos como exemplo de alfabeto o conjunto de letras do alfabeto brasileiro, alfabeto binário e o conjunto vazio. O conjunto de números naturais e primos são alguns exemplos de conjuntos que não podem ser considerados como alfabetos, uma vez que não são finitos. Por sua vez, o conceito de palavra pode ser apresentado como uma cadeia finita de caracteres ou símbolos justapostos. Em linguagens artificiais como a linguagem de programação python, uma palavra é um programa. Ciente destas definições, podemos apresentar que a linguagem formal é um conjunto de todas as palavras sobre um alfabeto. Considerando uma linguagem de programação temos que a linguagem formal é o conjunto de todos os programas da linguagem.

A definição que uma linguagem de programação é um conjunto de todos os programas (palavras) de uma linguagem não se mostram adequados para implementação em um computador. Deste modo,esse conjunto pode ser limitado através da utilização de um formalismo gramática. **Gramática** pode ser definida como um conjunto finito de regras, que quanto aplicadas sucessivamente sobre um alfabeto resulta em palavras. Sendo representada como uma quadrupla ordenada $G = (V, T, P, S)$, no qual temos que o seguinte significado para cada elemento desta quádrupla ordenada: **V** Conjunto finito de símbolos variáveis não terminai; **T** Conjunto finito de símbolos terminais disjunto de V; **P** Regras finitas de produção; e **S** elemento distinguido de V: símbolo inicial ou variável inicial.

Outro importante conceito para o estudo das linguagens formais é o conceito de linguagens regulares ou tipo 3. Sendo este, de acordo com a hierarquia de Chomsky a classe mais simples das linguagens. Sendo apresentada com os formalismos:

- **Autômato finito** é um formalismo operacional reconhecedor, sendo basicamente um sistema finito de estados. No qual cada estado somente conhece a informação necessária para a transição para o próximo estado, sendo que o mesmo sempre para. Um formalismo reconhecedor pode ser determinístico (AFD), não determinístico (AFN), e com movimentos vazios(AFN-ε). Um automato é composto por uma fita, que contém as informações a serem processadas; Uma unidade de controle que reflete o estado atual da maquina, o qual se desloca para direita, acessando um elemento da fita por vez e por ultimo a função programa, que comanda a leitura e define os estados da máquina.

- **Expressão regular** é um formalismo denotacional e gerador, pois se pode inferir como construir as palavras de uma linguagem. Uma expressão regular é definida a partir de conjuntos(linguagens) básicos e operações de concatenação e de união. Toda linguagem regular pode ser descrita por uma expressão denominada expressão regular.

- **Gramatica Regular** é um formalismo gerador, como anteriormente apresentado, mas com restrições quanto a geração.

Os formalismos apresentados são equivalentes, como apresentados na figura 1.

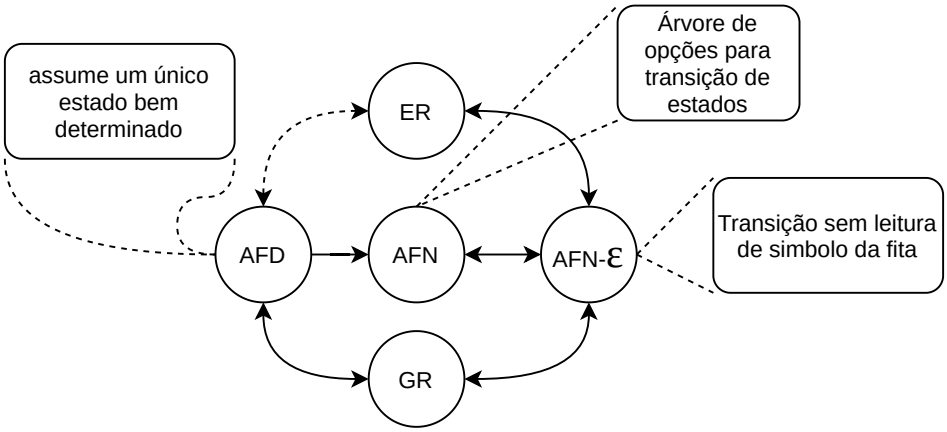


Figure 1: Equivalência

As linguagens regulares são representadas por formalismo de baixa complexidade, sendo fácil definir linguagens não regulares. Portanto se faz necessários modos de verificar se determinada linguagem é regular. Para mostrar que uma linguagem é regular, basta representa-la usando algum dos formalismos apresentados, como destacados na figura 1. Para provar que uma linguagem não é regular se faz necessário o desenvolvimento da prova para cada caso. Para este fim podemos utilizar o lema do bombeamento: Se L é uma linguagem regular, então: existe uma constante n tal que, para qualquer palavra w de L onde $|w| \geq n$, w pode ser definida como $w = uvz$ onde:

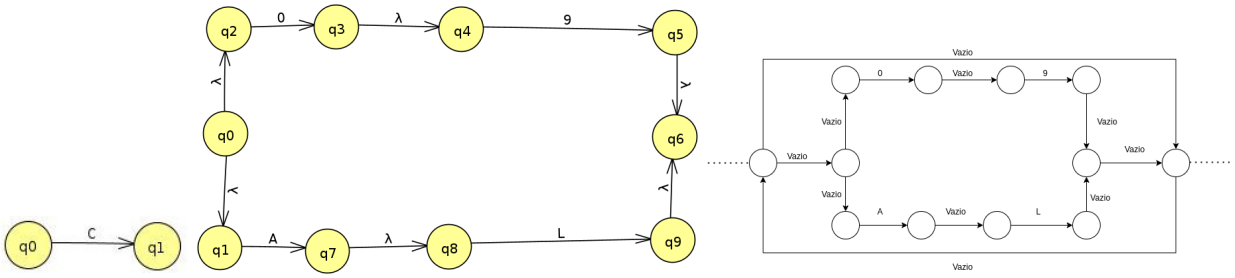
$$[uv] \leq n, [v] \geq 1 \tag{3}$$

sendo que, $i \leq 0, uv^i z$ é palavra de L.

1.1 Transformação ER para AFN-ε

Um AFN-ε é um automato no qual a mudança de estados é encapsulada, permitindo alterar entre estados do autômato sem um consumo de simbolo.

O processo de transformação de ER 2 para automato finito não determinístico com movimentos vazios(AFN-ε) é realizado de acordo com os passos apresentados na figura 2, no qual cada componente da ER é transformado em um automato equivalente, com movimentos vazios:



(a) ER C (b) ER (09+AL) (c) ER(09+AL)*

Figure 2: Geração do AFN com movimentos vazios

Nesse processo cada termo da expressão regular é transformado em um componente do (AFN-ε), resultando no autômato apresentado em 3.

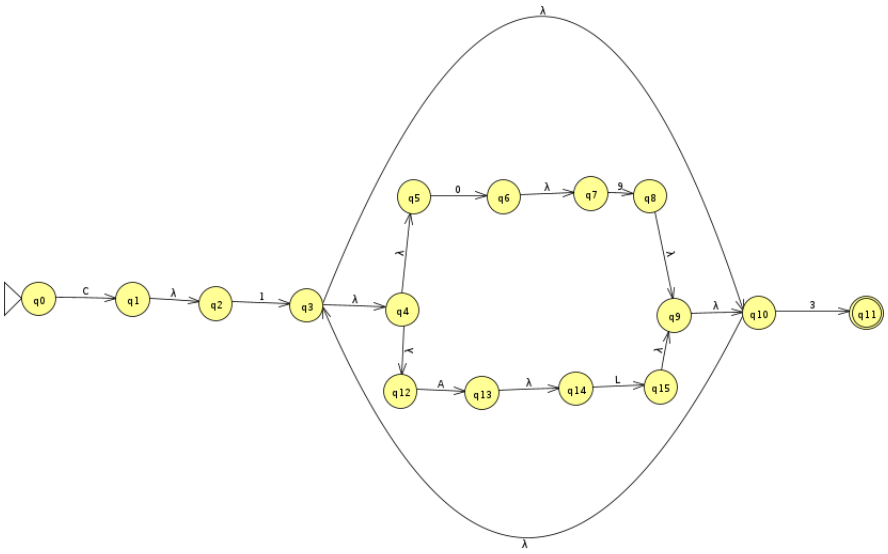


Figure 3: AFN-ε; gerado a partir da ER

1.2 Transformação AFN-ε para AFN

A partir do AFN-ε é gerado um AFN sem movimentos vazios, apresentado na figura 4. O processo de transformação de AFN-ϵ, para AFN, consiste na substituição do estado vazio, por arestas atingindo todos os estados inicialmente atingidos sem consumo de estado, possibilitando atingir todos os estados inicialmente atingidos pela transição vazia.

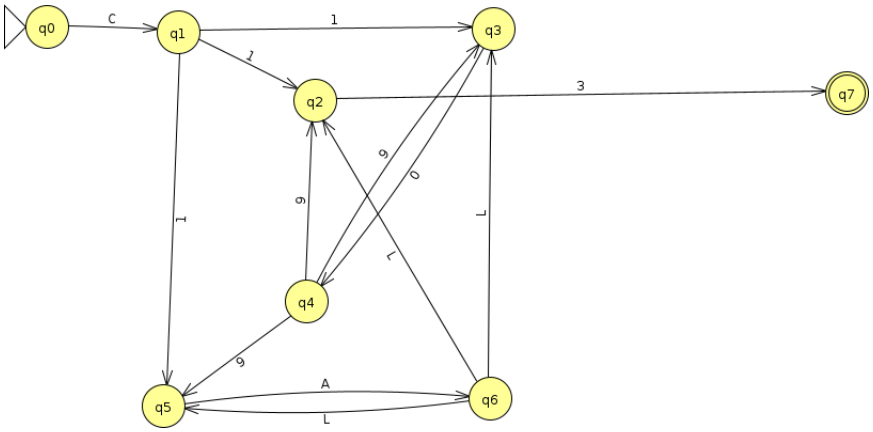


Figure 4: AFN sem movimentos vazios

1.3 Transformação de AFN para AFD

O AFN é convertido para um autômato finito determinístico (AFD), seguindo a ideia geral de combinar os estados do AFN em estados que simulem as diversas combinações de estados alternativos do AFN. A figura 5, apresenta a combinação destes estados, resultando no automato AFD da figura 6.

Estados	C	1	0	9	A	L	3	Renomeado
q0	q1	-	-	-	-	-	-	q0
q1	-	<q2, q3, q5>	-	-	-	-	-	q1
<q2, q3, q5>	-	-	q4	-	q6	-	qf	q2
q4	-	-	-	<q2, q3, q5>	-	-	-	q3
q6	-	-	-	-	-	<q2, q3, q5>	-	q4
qf	-	-	-	-	-	-	-	q5

Figure 5: Transição AFN para AFD

As combinações apresentadas em 5, permite a combinação das árvores de possibilidades, não determinísticas, de cada estado apresentado no AFN. Os estados combinados foram renomeados.

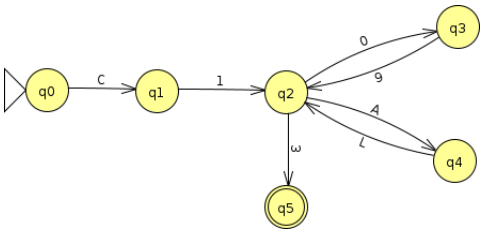


Figure 6: AFD gerado a partir do AFN sem movimentos vazios.

O presente AFD encontrado é mínimo, uma vez que com aplicação do algoritmo de minimização não é possível reduzir o número de estados. O automato encontrado não apresenta estados redundantes. O processo de minimização consiste da aplicação de um algoritmo que busca unificar os estados equivalentes, deste modo os estados redundantes são eliminados. Apesar de não possuir estados redundantes, a minimação não resulta em um aumento de eficiência em relação ao tempo de processamento. O ganho principal deste processo esta em uma utilização menor de espaço, para armazenar o autômato.

2 Implementação

O AFD proposto foi implementado na linguagem de programação python utilizando principalmente a estrutura de dados dicionário e lista. Dicionário e um estrutura de chave valor, similar a uma *hashmap*, no qual a partir de determinada chave é possível recuperar um valor armazenado. A fita contendo a palavra a ser verificada pelo automato é simulada através de uma estrutura de lista, no qual cada elemento da palavra é checado de maneira unitária. A unidade de controle por sua vez é representada por meio de um estrutura de repetição, no qual se desloca da esquerda para direita verificando um elemento por vez. Para representar a função programa é utilizado uma estrutura de dicionário como apresentado em 7. O dicionário utilizado para manter os estados das transições validas e apresentado do modo da figura 7.

Figure 7: Estrutura dicionário

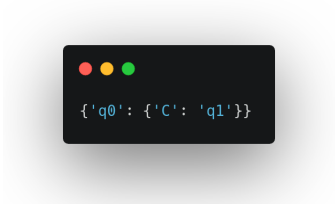


Figure 8: Exemplo de Transição

A figura 8, pode ser apresentada como lendo ‘C‘ no estado ‘q0‘ realize uma transição para o estado ‘q1‘. A leitura de um simbolo distinto do disponível para o estado corrente, resulta em uma transição inválida.

O programa implementado realiza a geração do autômato a partir da leitura de um arquivo no formato 9.

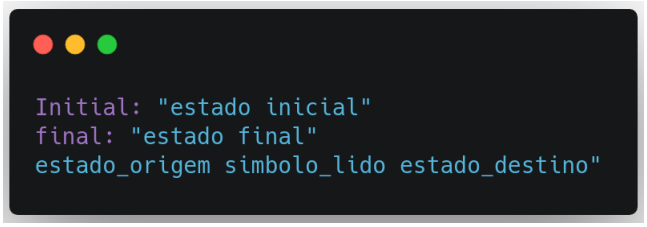
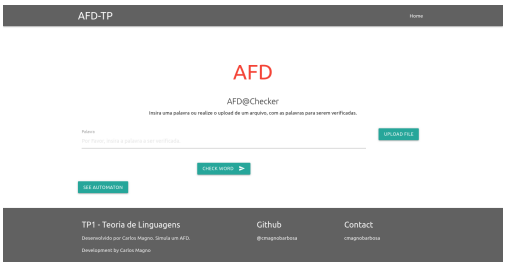


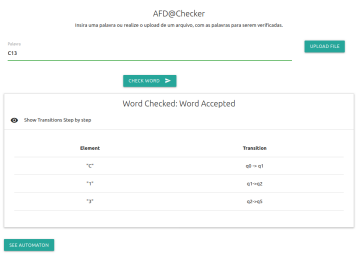
Figure 9: Exemplo da leitura de um automato do arquivo

3 Imagens da Interface

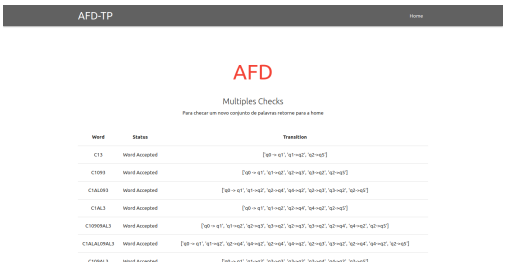
Foi desenvolvido uma interface WEB utilizando Flask e materialize css.



(a) Imagem 1



(b) Imagem 2



(c) Verificando palavras em arquivo.

Figure 10: Interface WEB

4 Manual

Segue em anexo um manual que apresenta um guia de como realizar a instalação e execução da implementação.

Palavra base aceita *C13*. Juntamente com o este trabalho segue um arquivo com palavras aceitas(palavras_aceitas.txt) e palavras negadas(palavras_negadas.txt).

References

[1] Paulo Blauth. Linguagens formais e autômatos. *Artmed Editora SA*, 2010.