Universidade Federal de São João Del-Rei
Departamento de Ciência da Computação

Alex de Andrade Soares

Trabalho prático - I

Teoria de Linguages

São João del-Rei
2020

Sumário

1. Introdução

O TP proposto consiste em implementar um autômato finito determinístico para a sua linguagem. Conforme mencionado, os detalhes de implementação devem ser descritos no relatório que deve acompanhar o programa.

O relatório deve constar ainda de detalhes sobre cada formalismo usado para as linguagens regulares, a saber: AFD, AFN, AFNε, ER e gramáticas lineares.

Para chegar ao AFD a ser implementado, será seguida a sequência de simulações:

1) ER
2) AFN_{ε}
3) AFN
4) AFD

2. Expressão Regular

Considerando a matrícula *d1d2d3d4d5d6d7d8d9* e as três primeiras letras do nome *l1l2l3,* a ER é definida por x1(d2l1 + d9l2)$^{+}$x2, sendo x da forma:

1. x1 é o número de letras do primeiro nome
2. x2 é o número de letras do segundo nome

Dado que minha matrícula é *"182050080"* e meu nome completo *"Alex de Andrade Soares",* temos a ER 4(8a + 0l)$^{+}$7.

3. O AFN_{ε}

O AFNε ou Autômato Finito Não-Determinístico com Movimentos Vazios, permite que o autômato esteja em mais de um estado, muito parecido com o AFN, porém faz o uso de movimentações vazias fazendo com que o autômato assuma

simultaneamente o estado de origem e destino (Pode ser entendido como um caminho alternativo).

O AFN_{ε} não aumenta o poder de reconhecimento de linguagens, apenas torna mais fáceis algumas construções.

3.1. Construção do AFN$_{\varepsilon}$ a partir de uma ER

A ER é uma linguagem regular pelo fato de ser possível criar um autômato finito que a represente.

A tabela de sentenças da ER e seus respectivos autômatos, gerada a partir da linguagem regular já conhecida, pode ser visualizada abaixo:

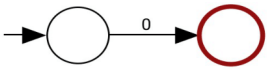
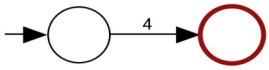
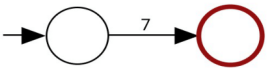
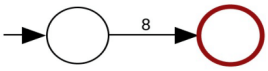
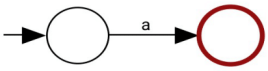
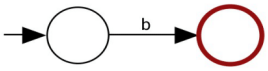
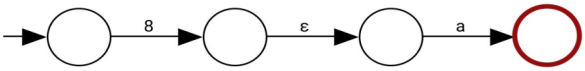
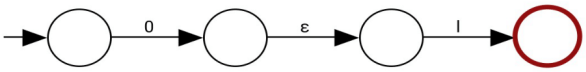
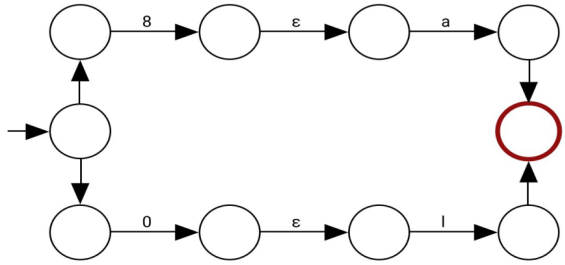
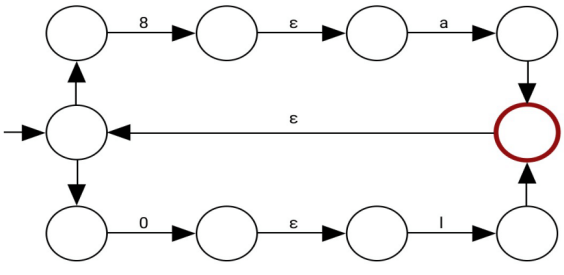
ER	AFNε
0	
4	
7	
8	
a	
l	
8a	
0l	
8a + 0l	
(8a + 0l)+	

Tabela 1: Expressões unitárias da ER e seus respectivos autômatos

A linguagem utilizada pode ser dividida em três partes: M_1, M_2, M_3.

- $M_1 = 4$
- $M_2 = (8a + 0l)^+$
- $M_3 = 7$

Assim, é possível representar a linguagem da seguinte forma:

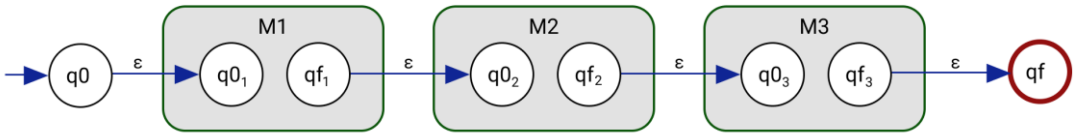


Figura 1: Representação da linguagem em termos de M_1, M_2 e M_3

Finalmente, para obter o AFN_ε para a linguagem basta substituir M_1, M_2, e M_3 pelos autômatos obtidos na Tabela 1.

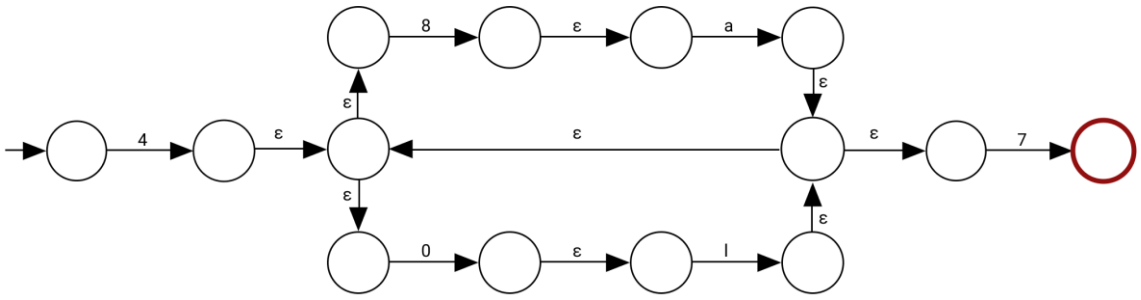


Figura 2: AFN_ε gerado através da ER

1. O AFN

O que difere um AFN de um AFD é o fato de que em um AFN, a aplicação da função programa retorna não apenas um estado, mas um conjunto.

Assim como o AFN_ε, o AFN não tem seu poder computacional aumentado.

Por teorema, a classe dos AFD é equivalente à classe dos AFN.

1.1. Construção do AFN a partir de um AFN$_\varepsilon$

Dando nome aos estados do AFN$_\varepsilon$ obtido na seção 3.1, temos como resultado:

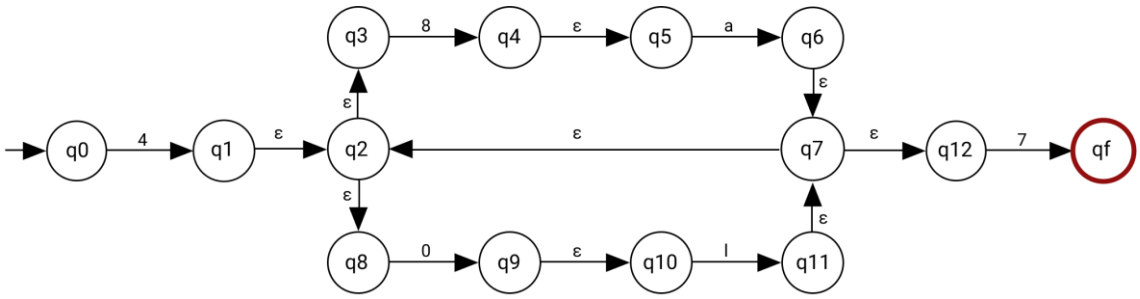


Figura 3: AFNε da seção 3.1 com seus estados nomeados

Com os estados nomeados, é possível encontrar o fecho vazio de cada estado aplicando a função $\delta_\varepsilon^*(q_n) = \delta^*(q_n,\ \varepsilon)$;

.

$\delta_\varepsilon^*(q_n)$	$\mathcal{E}$
q0	{ q0 }
q1	{ q1,q2,q3,q8 }
q2	{ q2,q3,q8 }
q3	{ q3 }
q4	{ q4,q5 }
q5	{ q5 }
q6	{ q6,q7,q2,q3,q8,q12 }
q7	{ q7,q2,q3,q8,q12 }
q8	{ q8 }
q9	{ q9,q10 }
q10	{ q10 }
q11	{ q11,q7,q2,q3,q8,q12 }
q12	{ q12 }
qf	{ qf }

Tabela 2: Tabela de fecho vazio de cada estado do AFNε

Conhecendo o fecho vazio de cada estado, podemos então criar a tabela de transição do AFN$_\varepsilon$ aplicando a função programa definida por:

$$\boldsymbol{\delta_N(q_n, x) = \delta_N^*(q_n, x) = \delta_\varepsilon(\{r \mid r \in \delta(s, x);\ s \in \delta_\varepsilon^*(q_n)\}), onde\ x \in \Sigma}$$

Aplicando a função de transição no estado inicial, lendo "4" como exemplo, temos:

$\boldsymbol{\delta_N(q_0, 4) = \delta_N^*(\{q_0\}, 4) = \delta_\varepsilon(\{r \mid r \in \delta(s, 4);\ s \in \delta_\varepsilon^*(q_n)\})}$;

$\boldsymbol{\delta_\varepsilon^*(q_0) = \{q_0\}\ e\ \delta(q_0, 4) = q_1}$;

$\boldsymbol{\delta_\varepsilon(q_1) = \{q_1, q_2, q_3, q_8\}}$.

Agora aplicando a função de transição no estado q_1 lendo "8", temos:

$\boldsymbol{\delta_N(q_1, 8) = \delta_N^*(\{q_1\}, 8) = \delta_\varepsilon(\{r \mid r \in \delta(s, 8);\ s \in \delta_\varepsilon^*(q_n)\})}$;

$\boldsymbol{\delta_\varepsilon^*(q_1) = \{q_1, q_2, q_3, q_8\}}$;

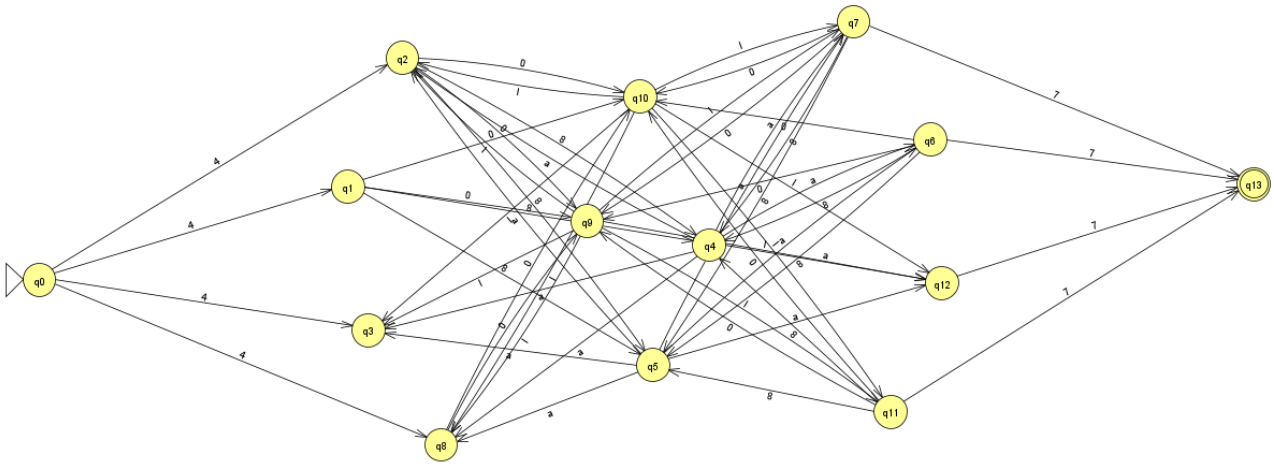
$\boldsymbol{\delta(q_1, 8) = \{\emptyset\},\ \delta(q_2, 8) = \{\emptyset\},\ \delta(q_3, 8) = \{q_4\},\ \delta(q_8, 8) = \{\emptyset\}}$,

então $\boldsymbol{\delta_\varepsilon(q_4) = \{q_4, q_5\}}$.

Aplicando a função programa a cada estado, lendo cada símbolo do alfabeto, chegaremos à tabela de transição abaixo:

$\delta_N(q_n)$	0	4	8	7	a	l	ε
q0	-	q1, q2, q3,q8	-	-	-	-	-
q1	q9,q10	-	q4,q5	-	-	-	-
q2	q9,q10	-	q4,q5	-	-	-	-
q3	-	-	q4,q5	-	-	-	-
q4	-	-	-	-	q6,q7,q12,q2,q3,q8	-	-
q5	-	-	-	-	q6,q7,q12,q2,q3,q8	-	-
q6	q9,q10	-	q4,q5	qf	-	-	-
q7	q9,q10	-	q4,q5	qf	-	-	-
q8	q9,q10	-	-	-	-	-	-
q9	-	-	-	-	-	q11,q7,q2,q12,q3,q8	-
q10	-	-	-	-	-	q11,q7,q12,q2,q3,q8	-
q11	q9,q10	-	q4,q5	qf	-	-	-
q12	-	-	-	qf	-	-	-
qf	-	-	-	-	-	-	-

Gerando o grafo através da tabela de transição obtida, chegamos ao grafo



Como pode ser notado, a leitura, montagem e compreensão de um AFN pode ser muito complexa. A simulação do mesmo autômato, porém na classe AFD, pode ficar muito mais fácil de se compreender.

2. O AFD

A montagem da tabela de transição do AFD será feita iniciando pelo estado incial “q0” e cada símbolo do AFN terá uma coluna na tabela.

As transições serão montadas de acordo com a tabela do AFN e será o conjunto união de todo estado possível a partir do estado corrente, lendo cada um dos símbolos.

Cada novo estado gerado, terá como destino o conjunto união de cada um de seus elementos para cada símbolo do alfabeto. A análise é feita para cada novo estado gerado e todos os estados que possuírem o estado final do AFN, também será um estado final.

Exemplo:

$\boldsymbol{\delta_N}(q0,\ 4) = \{\ q1,\ q2,\ q3,\ q8\ \}$

Assim:

$\boldsymbol{\delta}(q0,\ 4) = q1q2q3q7$

O estado q1q2q3q8 terá como destino a união dos estados obtidos através da leitura para cada símbolo:

$\boldsymbol{\delta_N}(q1,\ 0) = \{\ q9,\ q10\ \}$ e $\boldsymbol{\delta_N}(q1,\ 8) = \{\ q4,\ q5\ \}$;

$\boldsymbol{\delta_N}(q2,\ 0) = \{\ q9,\ q10\ \}$ e $\boldsymbol{\delta_N}(q2,\ 8) = \{\ q4,\ q5\}$;

$\boldsymbol{\delta_N}(q3,\ 8) = \{\ q4,\ q5\ \}$;

$\boldsymbol{\delta_N}(q8,\ 0) = \{\ q9,\ q10\ \}$.

Assim:

$\boldsymbol{\delta}(q1q2q3q8,\ 0) = q9q10$ e $\boldsymbol{\delta_N}(q1q2q3q8,\ 8) = q4q5$

O mesmo procedimento é aplicado nos estados q9q10 e q4q5 e também nos estados que ainda serão gerados a partir deles até que todos tenham sido analisados.

Por conveniência, os novos estados serão chamados p_0, p_1, p_2, p_3, p_4, p_5, p_f sendo:

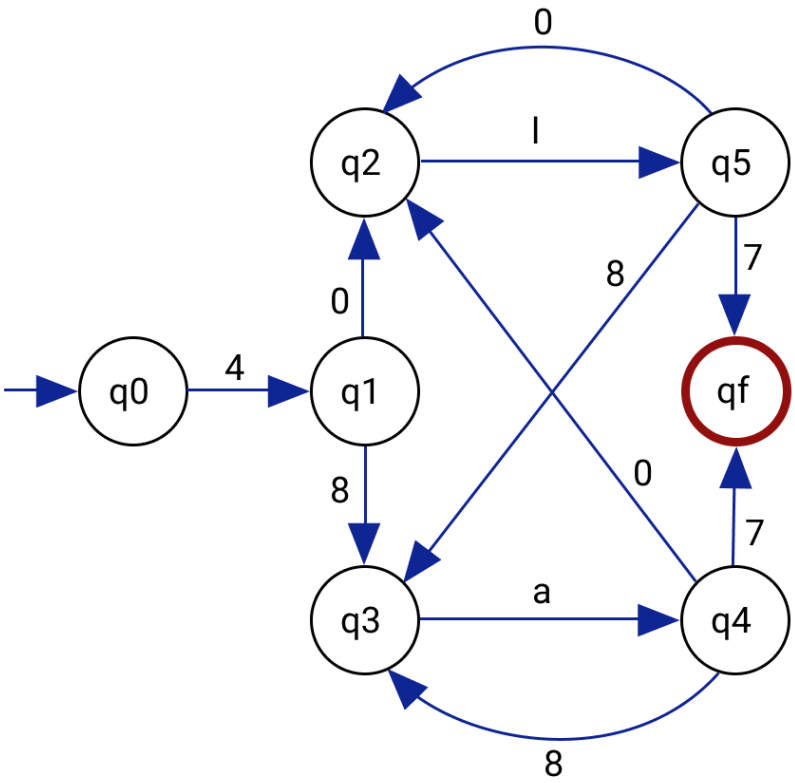
- $p_0 = q_0$;
- p1 = q1q2q3q8;
- p2 = q9q10;
- p3 = q4q5;
- p4 = q2q3q6q7q8q12
- p5 = q2q3q7q8q11q12;
- pf = qf

A tabela de transição gerada a partir dos estados gerados ficou da seguinte forma:

δ	0	4	8	7	a	l
p0	-	p1	-	-	-	-
p1	p2	-	p3	-	-	-
p2	-	-	-	-	-	p5
p3	-	-	-	-	p4	-
p4	p2	-	p3	pf	-	-
p5	p2	-	p3	pf	-	-
pf	-	-	-	-	-	-

Pela tabela de transição, chegamos finalmente ao AFN.

Obs.: Por conveniência, os nomes dos estados foram alterados de p_n para q_n.



3. Implementação do AFD

O autômato foi implementado utilizando a linguagem C por motivos de familiaridade com a linguagem para conclusão de trabalhos práticos em outras disciplinas.

3.1. Entrada

O programa faz o uso de dois arquivos, sendo eles "*automaton.txt*" e "*words.txt*" ambos devendo estar no diretório raiz.

É importante salientar que o primeiro arquivo citado é indispensável para a correta execução do programa pois, é através dele é definido o autômato usado para o reconhecimento.

A escolha de utilizar um arquivo para representação do autômato foi feita pois dá a possibilidade de alterar o autômato do programa, permitindo o reconhecimento de outras linguagens regulares de forma fácil.

Já segundo, é determinado pelo usuário durante a execução do programa, portanto, pode ser nomeado de outra forma.

1. Exemplo de entrada – automaton.txt

automaton.txt

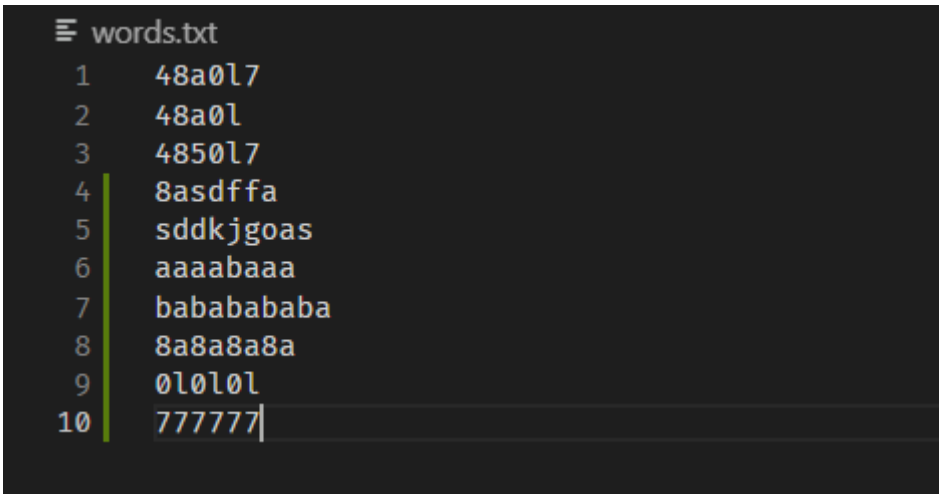
```
7

0
4,1

1
0,2
8,3

2
l,5

3
a,4

4
8,3
0,2
7,6

5
0,2
8,3
7,6

6
-,-2
```

Para uma correta representação de um AFD, o arquivo deve seguir um padrão.

- A primeira linha deve conter a quantidade de estados do autômato seguida de uma linha vazia;
- A representação de cada estado se inicia com o índice do estado em uma linha sozinho;
- O autômato deve conter apenas um estado inicial e este deve ser o primeiro a ser representado;
- A representação das transições deve estar na linha imediatamente após o índice do estado;
- Cada transição é representada no formato x,y onde, x é um símbolo que pertence ao alfabeto e y é o índice do estado que o autômato assume lendo x;
- O autômato pode conter mais de um estado final e eles pode contar com transições para outros estados, porém, todos devem contar com a transição "-,-2".

Com as regras acima podemos notar que o autômato possui 7 estados sendo 0 o estado inicial e apenas o 6 como estado final.

2. Exemplo de entrada – words.txt

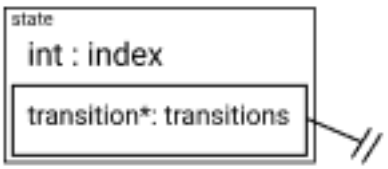


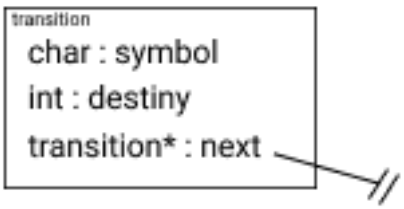
Diferentemente do arquivo *"automaton.txt"*, possui apenas uma regra.

- As palavras devem estar dispostas uma em cada linha;

3.2. Tipos Abstratos de Dados

Para a implementação do AFD foram criados dois tipos abstratos de dados, para os estados e para as transições como podem ser visualizadas nos diagramas abaixo:





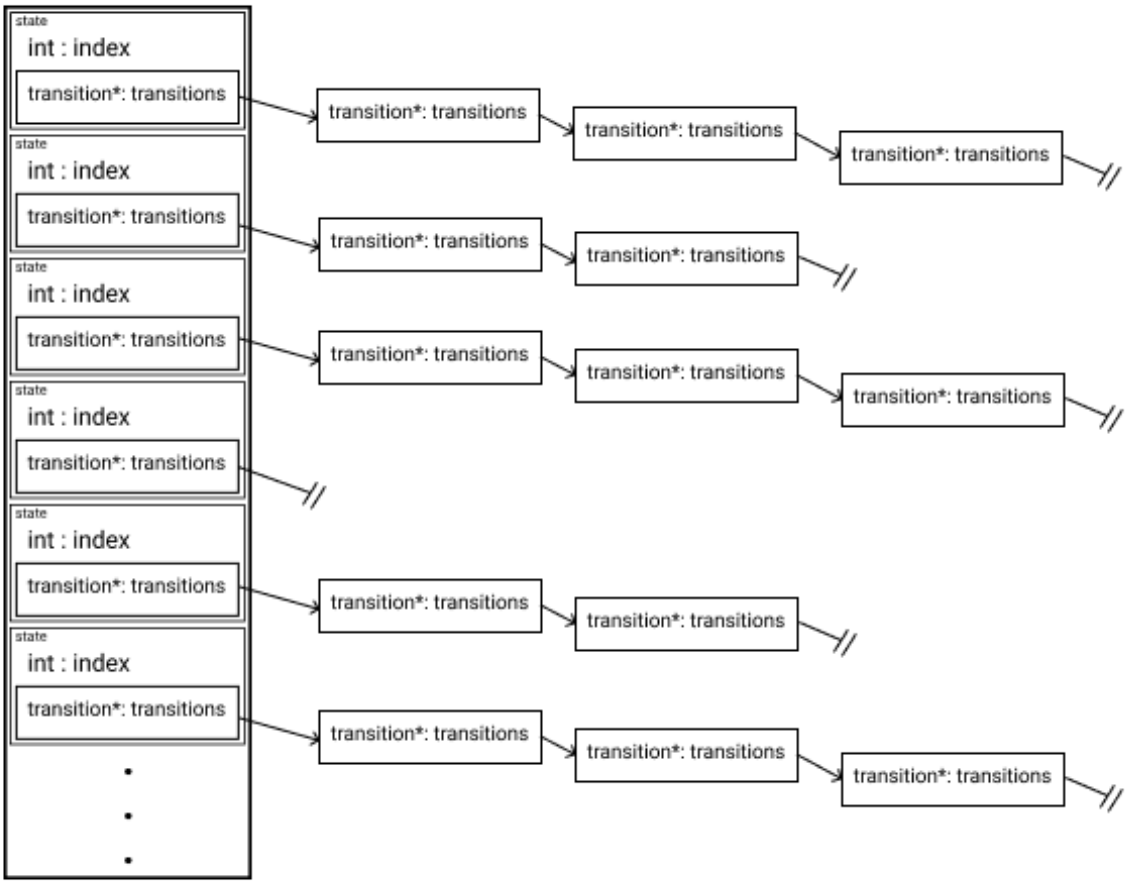
state: Representa um estado do autômato, armazenando seu index e a cabeça para uma lista de transição para o estado.

transition: Representa a transição de para um estado de destino, armazenando um símbolo que pode ser lido pelo autômato no estado atual, o index do próximo estado e um ponteiro para a próxima transição da lista.

3.3. Estrutura de dados

A estrutura de dados do programa é baseada em um vetor do tipo *state* com o tamanho informado no arquivo "*automaton.txt*".

O diagrama abaixa oferece uma melhor visão da estrutura de dados.



Obs.: A quantidade de itens na lista de transição de cada estado varia de acordo com o AFD utilizado.

4. Funções implementadas

Para o funcionamento do programa, foram implementadas as seguintes funções:

- *init_states(int automaton_size):*

  automaton_ size: tamanho do autômato;

Aloca e cria na memória um vetor do tipo *state* onde serão armazenados todos os estados com seus índices e uma lista de transições já iniciada.

- *get_destiny(char* line):*

  line: vetor do tipo char contendo uma linha que representa uma transição, lida do arquivo "automaton.txt".

Percorre a string copiando para outra string apenas os valor após a "," e retorna a nova string como um inteiro.

- *init_automaton():*

Realiza a leitura do arquivo *"autômato.txt"* capturando o tamanho do autômato, inicializando seus estados e adicionando as transições em suas listas.

- *add_transition(transition* list_head, char symbol, int destiny):*

  list_ head: nó cabeça da lista de transições;

  symbol: símbolo que compõe a transição;

  destiny: índice do estado de destino da transição.

Adiciona a transição no início da lista de transição recebida.

- *init_transition_list():*

Aloca e cria uma transição sem símbolo e com o ponteiro para a próxima igual a NULL, pois é utilizada apenas como cabeça da lista.

- *show_menu(state *automaton):*

  automaton: Ponteiro para o vetor de estados do autômato.

Exibe o menu dando ao usuário as opções de ler uma palavra individualmente, através de um arquivo ou sair do programa.

O menu é exibido ao fim de uma execução, até que o usuário escolha a opção de sair do programa.

- *execute_automaton(state automaton, int current_state, char *word, int command_line):*

  automaton: Vetor de estados do autômato.

  current_ state: Inicialmento é igual o índice do estado inicial e é atualizado durante a execução da função.

  Word: vetor de char contendo a palavra a ser testada pelo autômato;

  command_ line: Inteiro que assume o valor 1 ou 0. 1 quando o usuário escolhe verificar uma palavra pelo terminal e 0 quando escolher verificar utilizando um arquivo. Sinaliza se deve ou não exibir os passos da verificação.

A função percorre cada caractere da palavra atualizando *current_ state* para o estado em que o autômato se encontra, realiza a verificação se a palavra é aceita ou não e imprime o resultado na tela.

- *read_from_line(state *automaton):*

  automaton: Vetor de estados do autômato.

Solicita que o usuário insira uma palavra e chama a função *execute_ automaton* para a verificação da mesma.

- *read_from_file(state *automaton):*

  automaton: Vetor de estados do autômato.

Solicita que o usuário informe o nome do arquivo em que se encontram as palavras que deseja verificar e chama $execute\_automaton$ para cada uma delas.

- *next_state(transition *transitions, char symbol):*

  transitions: Ponteiro para o nó cabeça da lista de transições de um estado;

  symbol: Símbolo que deve ser verificado o estado de destino quando lido.

Percorre a lista de transições e retorna o índice do estado de destino quando *symbol* é lido.

- *is_final(transition *transitions):*

  transitions: Ponteiro para o nó cabeça da lista de transições de um estado.

Percorre a lista de transições verificando se existe um símbolo que leve ao estado -2. Retorna 1 caso exista, indicando que o estado é um estado final