Relatório 01 – CTC-34

José Otávio Vidal e Luiz Felipe Schiaveto
Instituto Tecnológico de Aeronáutica. São José dos Campos – SP –Brasil.

E-mail: jose.otavio.vidal@gmail.com
luizfschiaveto@gmail.com

INTRODUÇÃO

Os algoritmos implementados em cada uma das questões foram orientados pelas sugestões dadas nos enunciados das próprias questões. O grupo utilizou a linguagem Python, e o IDE "PyCharm". Pode-se associar as questões aos seguintes códigos fonte:

- Questão 01
  - **q01.py**; neste código, é digitado manualmente a expressão regular correspondente, de forma que, a partir dela, é feito o respectivo e-AFN, que é gerado automaticamente na tela.
- Questão 02
  - **q02.py**; neste código, digitamos manualmente as especificações do autômato (arcos, estados finais, estados, alfabeto) e, ao rodar o código, é gerada automaticamente a imagem do autômato sem as e-transições.
- Questão 03
  - **q03.py;** neste código, deve ser digitado manualmente o tipo de entrada que será colocada no algoritmo (expressão regular, e-AFN ou AFN) e, então, a partir desses, é gerado o tratamento adequada e automaticamente se configura na tela os resultados das cadeias pedidas.

Pode-se relevar a estrutura utilizada para descrição dos autômatos:

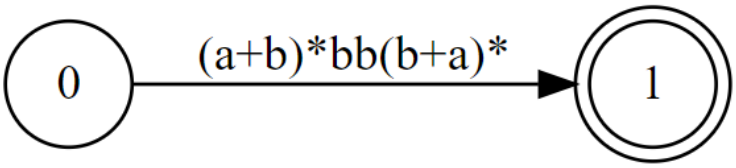
- Existe uma lista de tuplas que contém três elementos cada. O primeiro elemento da tupla é o estado inicial, o segundo elemento é o elemento do alfabeto que garante a transição, ou ainda a transição '&', e o terceiro elemento é o estado final da transição a ser descrita.
  - [(...,...,...), (...,...,...), (...,...,...)]

RESULTADOS

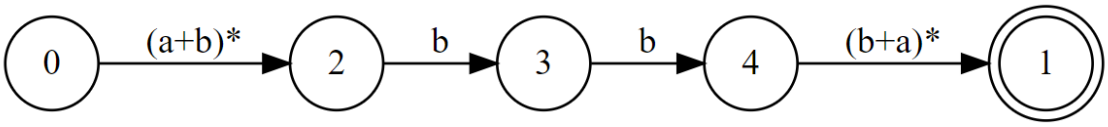
1 – Implementar o seguinte algoritmo, obtendo como entrada uma expressão regular e produzindo um autômato e-AFN.
I – Inicia o autômato como um nó inicial e um nó final com transição especificando a expressão regular.
II – verificar se é uma união de linguagens (expressa como A+B), separar a união em arcos.
III- verificar se é uma concatenação de linguagens, separar utilizando um novo nó.
IV – se a expressão for um fecho de Kleene, construir um loop em novo nó separado por □-Transições.
V – se a expressão estiver entre parênteses, remover os parênteses.
VI – repete II, III, IV, V e VI para cada arco até que todo arco tenha apenas um símbolo ou □ (repr &).

- Autômato 1

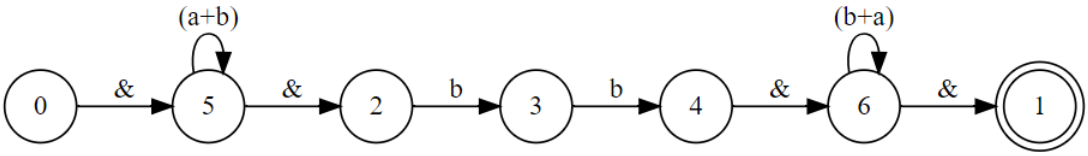
1. Criar o estado inicial e o final



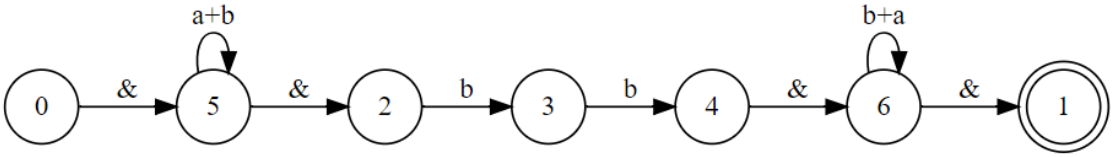
2. Separar as linguagens concatenadas



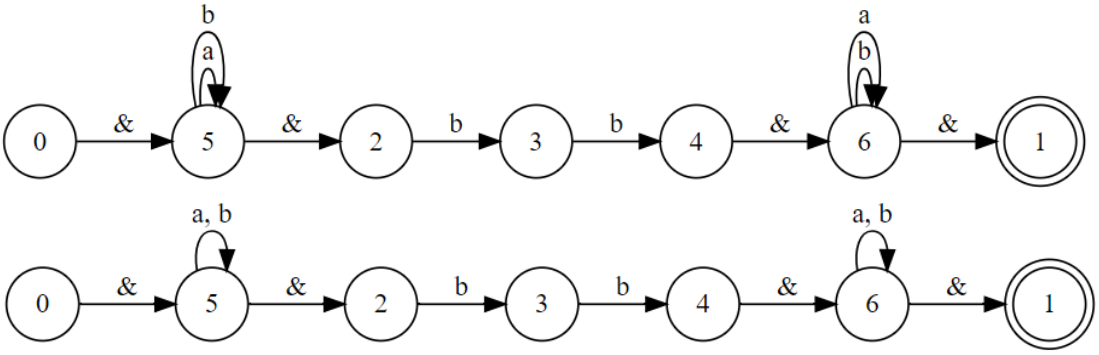
3. Eliminar o fecho de Kleene



4. Eliminar os parênteses

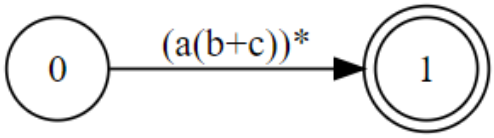


5. Eliminar a união de linguagens

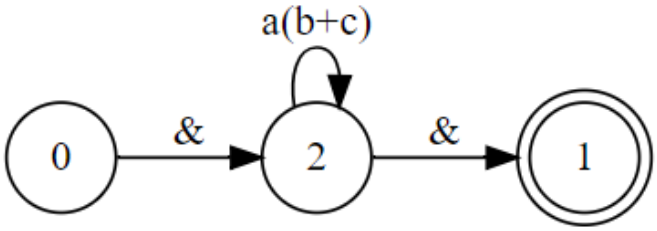


- <u>Autômato 2</u>

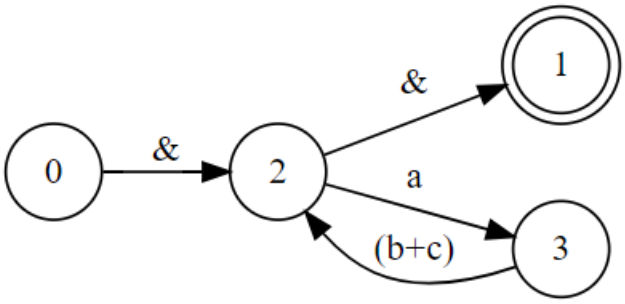
1. Escrever o estado inicial e final



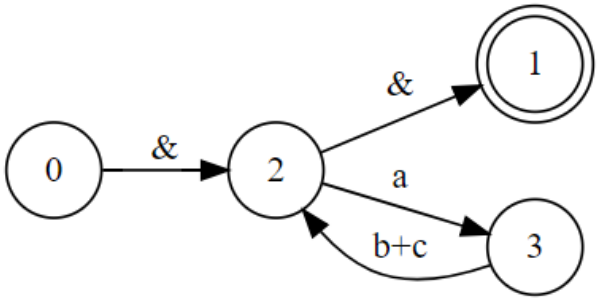
2. Eliminar o fecho de Kleene



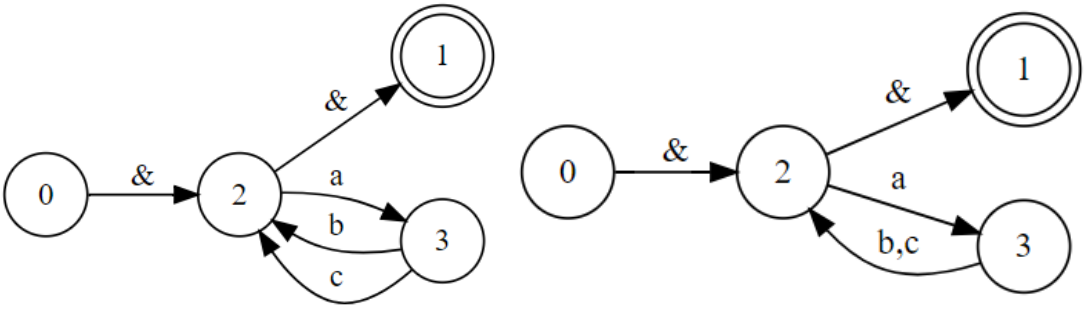
3. Eliminar a concatenação de linguagens



4. Eliminar os parênteses

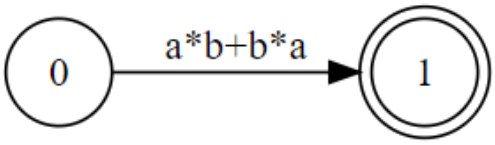


5. Eliminar a união de linguagens

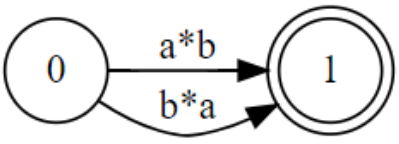


- <u>Autômato 3</u>

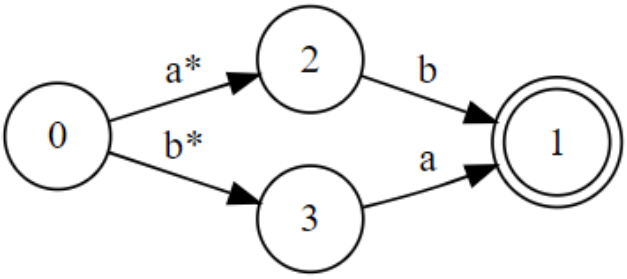
1. Criar estado inicial e final



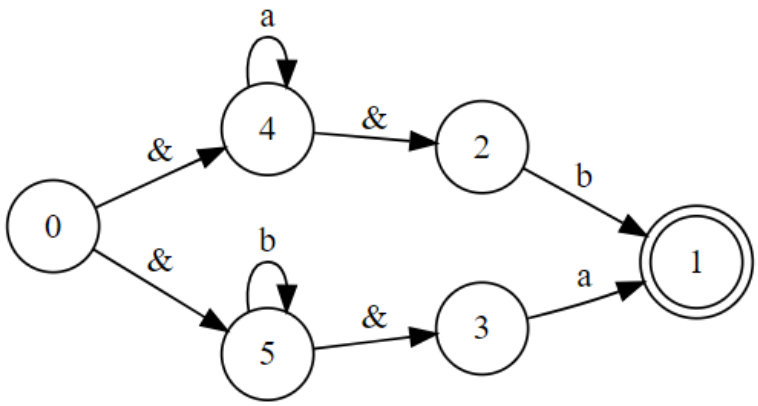
2. Eliminar a união de linguagens



3. Eliminar a concatenação de linguagens

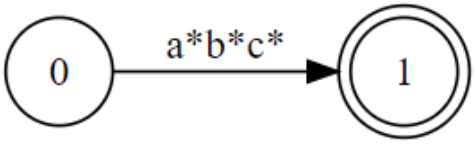


4. Eliminar o fecho de Kleene

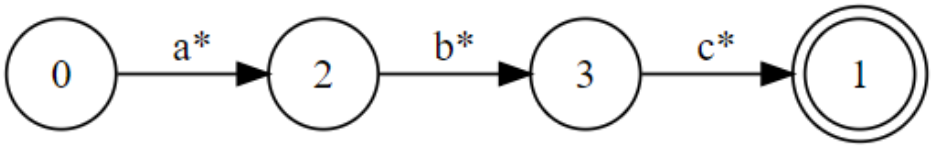


- Autômato 4

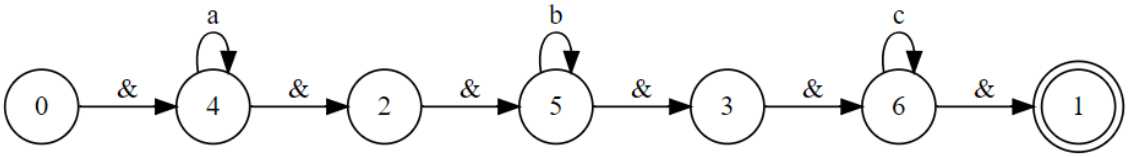
1. Criar o estado inicial e o final



2. Eliminar a concatenação de linguagens



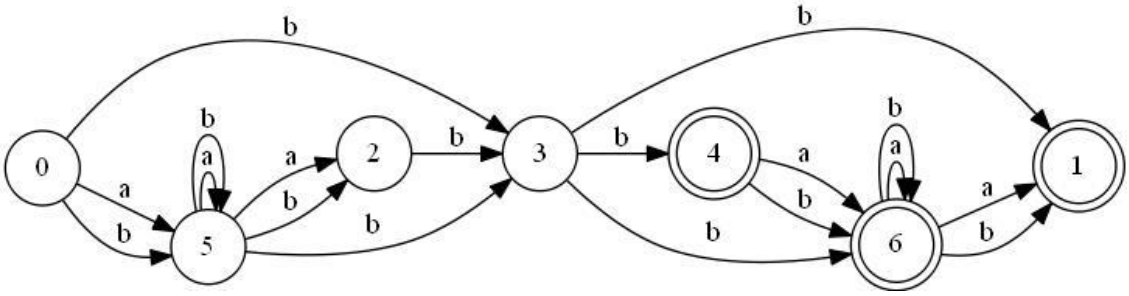
3. Eliminar o fecho de Kleene



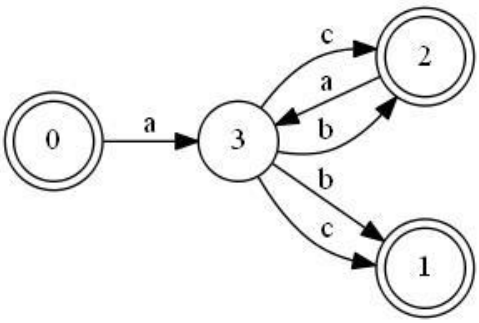
---

2 – Implemente a remoção das □-transições de um autômato (isto pode resultar em múltiplos estados finais). Pode utilizar o seguinte procedimento:
I – computar o □-fecho de cada estado
II – todo arco de A em X gera um arco de A em Y para cada Y no □-fecho(X)
III – todo arco de Y em A para qualquer Y no □-fecho(X) gera um arco de X para A.
IV – X é estado final se algum Y no □-fecho(X) for final.

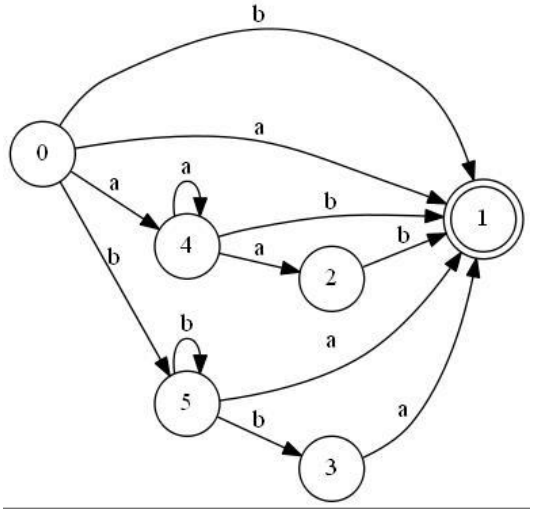
- Autômato 1



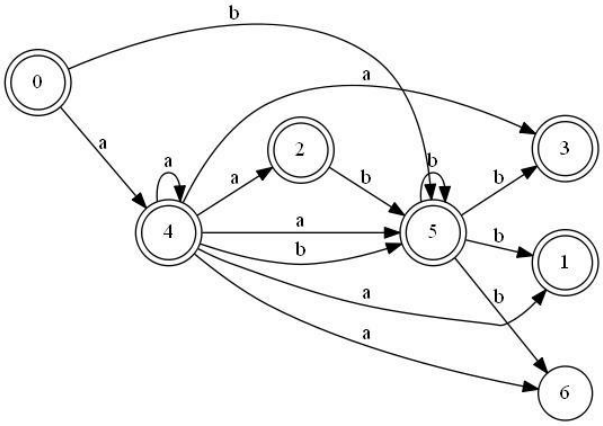
- Autômato 2



- Autômato 3



- Autômato 4



---

3 – Dados: (a) uma expressão regular, um □-AFN ou um AFN conforme os itens acima e (b) uma cadeia de entrada, implemente uma função que encontre todas as sub-cadeias da cadeia do item (b) aceitas pela estrutura do item (a).
Dica: converter (a) para AFN usando (1) e (2), simular o AFN a partir de cada posição da cadeia (b).

- Autômato 1

Após executar o código q03.py, e colocar as devidas informações necessárias em relação ao primeiro autômato, obtivemos as seguintes sub-cadeias; a cadeia em análise foi **baabba.**

- ['b', 'a', 'a', 'b', 'b']

- ['b', 'a', 'a', 'b', 'b', 'a']
- ['a', 'a', 'b', 'b']
- ['a', 'a', 'b', 'b', 'a']
- ['a', 'b', 'b']
- ['a', 'b', 'b', 'a']
- ['b', 'b']
- ['b', 'b', 'a']

- Autômato 2

Após executar o código q03.py, e colocar as devidas informações necessárias em relação ao segundo autômato, obtivemos as seguintes sub-cadeias; a cadeia em análise foi **baabba**.

- []
- ['a', 'b']

E com relação a **abacabc:**

- []
- ['a', 'b']
- ['a', 'b', 'a', 'c']
- ['a', 'b', 'a', 'c', 'a', 'b']
- ['a', 'c']
- ['a', 'c', 'a', 'b']

- Autômato 3

Após executar o código q03.py, e colocar as devidas informações necessárias em relação ao terceiro autômato, obtivemos as seguintes sub-cadeias; a cadeia em análise foi **baabba.**

- ['b']
- ['b', 'a']
- ['a', 'a', 'b']
- ['a']
- ['a', 'b']
- ['b', 'b', 'a']

- Autômato 4

Após executar o código q03.py, e colocar as devidas informações necessárias em relação ao segundo autômato, obtivemos as seguintes sub-cadeias; a cadeia em análise foi **baabba**.

- []
- ['b']
- ['a']
- ['a', 'a']
- ['a', 'a', 'b']
- ['a', 'a', 'b', 'b']
- ['a', 'b']
- ['a', 'b', 'b']
- ['b', 'b']

E com relação a **abacabc:**

- []
- ['a']
- ['a', 'b']
- ['b']
- ['a', 'c']
- ['c']
- ['a', 'b', 'c']
- ['b', 'c']