ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

PCS3556 – Lógica Computacional

CAIO VINICIUS SOARES AMARAL - NUSP: 10706083

THALES AUGUSTO SOUTO RODRIGUEZ - NUSP: 10706110

GUSTAVO PRIETO – NUSP 4581945

PROF. DR. Ricardo Rocha

Exercício de Programação 3

São Paulo

Introdução

O objetivo deste exercício programa é permitir que os alunos programem em elixir tanto um autômato determinístico finito (DFA) quanto um autômato não-determinístico finito (NFA), conforme visto em aula.

Para tal, o exercício consiste na implementação de um código que seja capaz de verificar se uma determinada sequência de símbolos input é aceita ou rejeitada pelo autômato, a partir dos estados de aceitação e das tabelas de transição definidos.

O repositório do projeto pode ser encontrado em: https://github.com/master0022/ep3

Algoritmo

Primeiramente, a implementação de autômatos consiste em utilizar o princípio que todo DFA é um NFA, e começamos implementando um NFA.

Antes de tudo, é importante definir o que é um autômato no código desenvolvido. Um autômato consiste de três elementos:

1. Uma lista com os estados de aceitação
2. Uma tabela de transições
3. O estado atual

Esses três elementos permitem representar e simular um autômato. Entrando em detalhes, a tabela de transições é descrita no seguinte formato:

```
transitions =  %{
   q0: [ ["0",:qR], ["1",:q1] ],
   q1: [ ["0",:q2], ["1",:q1] ],
   q2: [ ["0",:q2], ["1",:q1] ],
   qR: [ ["0",:qR], ["1",:qR] ]
 }
```

Assim, cada estado possui uma lista contendo um “input”, e o próximo estado correspondente a esse input

automata.transitions[:q0] == [["0",:qR], ["1",:q1]]

É uma linha da tabela.

Existem apenas duas funções relevantes para um autômato NFA no código,

next_states (transitions_row,input)

Esta função receba uma *linha* da tabela de transições, e retorna uma lista com todos os “próximos estados” possíveis.

Em uma DFA, esta função é trivial e retorna sempre um único elemento, o próximo estado dado um input.

Mas em uma NFA, essa função é importante para obter todos os próximos estados (Por exemplo, o input “0” no estado q0 pode levar tanto a q1, quanto q2. Então é retornado [q1,q2])

automata_simulation(automata,inputlist,input_idx\\1)

Esta função é o núcleo do algoritmo para simular NFAs. Ela funciona da seguinte forma:

1. Verifica se chegamos do input
    a. Caso sim, verifica se o estado atual do autômato é um estado de aceitação e retorna que a cadeia foi aceita ou rejeitada.
    b. Caso não, continue
2. Dado o input atual, calcule quais são os próximos estados possíveis com next_state.
    a. Caso não haja nenhum próximo estado, isso significa que o input não está na tabela de transições e é rejeitado.
3. Para *cada* próximo estado, inicie uma “nova simulação” automata_simulation, configurado para o próximo estado e recebendo o próximo input.
4. Verifica se algum dos autômatos do passo 3 aceitaram o restante do input. Caso sim, retorna que a cadeia foi aceita.

Agora, para simular um DFA, foi implementado **simulate_DFA**
Esta função verifica se alguma linha da tabela de transição possui um input que leva a dois estados. Caso exista, retorna um erro :invalid_transitions. Caso aceito, simula o DFA como se ele fosse um NFA.

Existem algumas outras funções auxiliares no código, como **recursive_has_duplicates(list)** que usada para verificar se um autômato é um DFA ou NFA, e **parse(string)** que recebe uma string de cadeia e retorna uma lista com os caracteres (por exemplo "abcdefg" -> ["","a","b","c","d","e","f","g",""])

Casos de teste

Para rodar os casos de teste criados, basta entrar na pasta do projeto elixir e executar

mix test

Ou, caso seja desejado usar as funções com casos novos, elas estão disponíveis na pasta /lib, no arquivo **automata.ex**

Os casos de teste gerados foram os seguintes, e podem ser encontrados em /test/**automata_test.exs**:

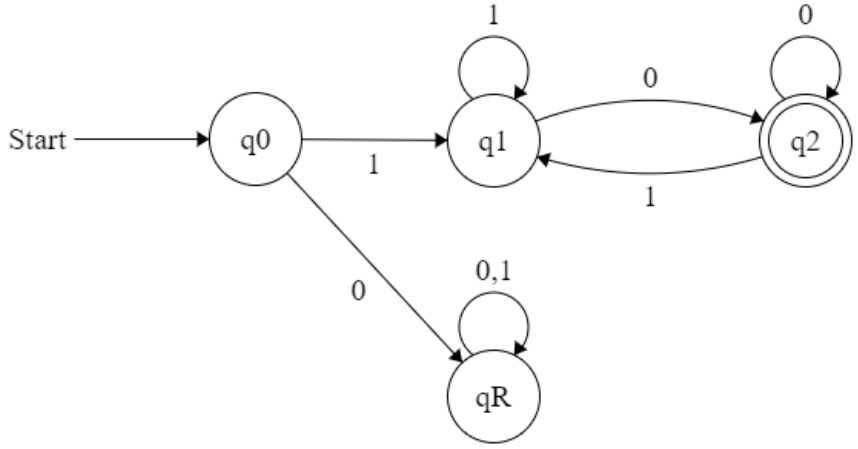
Teste 1

Autômato testado:

E um autômato simples, com um estado de rejeição qR e que começa no q0. Ele aceita strings do tipo (1)(0|1)*(0), ou seja, que começam com 1 e terminam com 0.

Estado\input	“0”	“1”
q0	qR	q1
q1	q2	q1
q2	q2	q1
qR - Rejeição	qR	qR

Aceitação = Q2



Cadeias testadas / Output esperado

```
input1 = "0001011010101110"    # REJEITA, comeca com 0
input2 = "0011111111"          # REJEITA, comeca com 0
input3 = "1011111111"          # REJEITA, termina com 1
input4 = "1011111110"          # TRUE, comeca com 1 e termina com 0. Aceito.
input5 = "0123456789"          # REJEITA, caracteres não reconhecidos
```

Teste 2 – DFA vs NFA

Esse teste tem o propósito de observar se, quando simulamos um DFA, ele rejeita simular um NFA.

São testados dois autômatos equivalentes ao autômato anterior, porém um é um NFA e o outro é um DFA.

Autômatos testados:

NFA:

Estado\input	“0”	“1”
q0	qR	q1
q1	q2	q1, q0
q2	q2	q1
qR - Rejeição	qR	qR

DFA:

Estado\input	“0”	“1”
q0	qR	q1
q1	q2	q1
q2	q2	q1
qR - Rejeição	qR	qR

Aceitação = Q2

Cadeias testadas / Output esperado

input1 = "0001011010101110" # REJEITA, comeca com 0

Esperado que:

A simulação (de DFA) do autômato DFA rejeite a cadeia

A simulação (de DFA) do autômato NFA retorne um erro, :invalid_transitions, e não simule

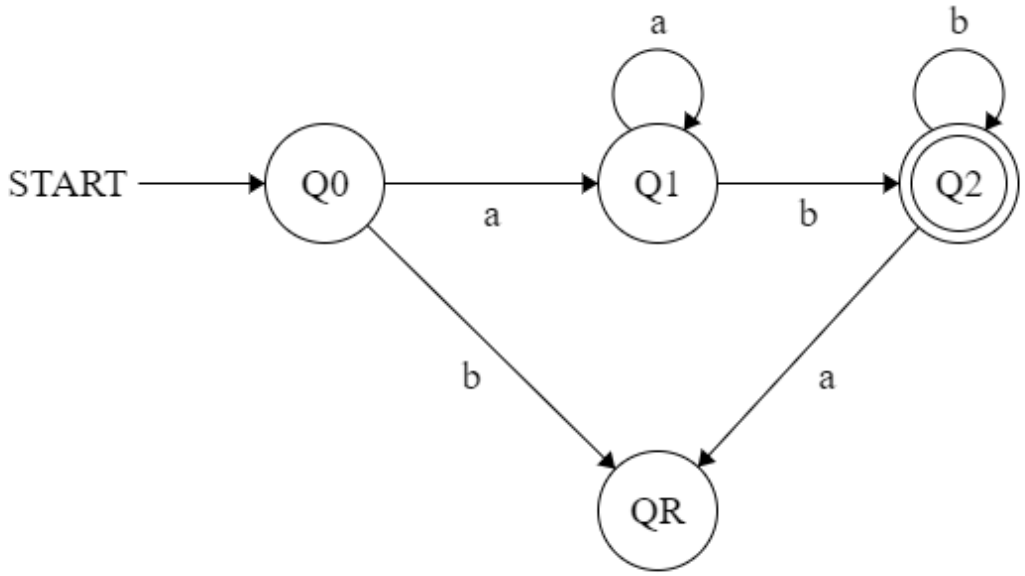
Teste 3

Autômato testado:
Este autômato aceita cadeias do tipo (a)*(b)*, ou seja, que começa com uma sequência de “a”, e termina com uma sequência de “b”

No estado QR é proposital as transições não estarem marcadas, uma vez que é um estado de rejeição.

Estado\Input	b	a
Q0	QR	Q1
Q1	Q1	Q2
Q2	Q2	QR
QR	-	-

Aceitação = Q2



Cadeias testadas / Output esperado

```
input1 = "aaaaabbbbb"        # ACEITA
input2 = "bbbbbaaaaa"        # REJEITA
input3 = "a"                 # REJEITA
input4 = "baaaabbbb"         # REJEITA
input5 = "aaabbbaaabbbb"     # REJEITA
```

Teste 4

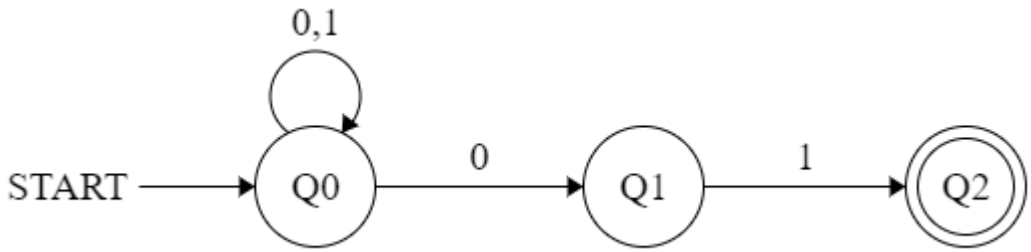
Autômato testado:

Aceita cadeias do tipo (0|1)*01, ou seja, uma sequência de 0s e 1s que termina em 01.

Transições para o estado de rejeição são implícitas neste e no próximo exemplo.

Estado\Input	0	1
Q0	Q0, Q1	Q0
Q1	-	Q2
Q2	-	-

Aceitação = Q2



Cadeias testadas / Output esperado

```
input1 = "010100101110101"      # Aceita
input2 = "01"                   # Aceita
input3 = "111110010"            # REJEITA, termina em 10
input4 = "00000011111"          # REJEITA, termina em 11
input5 = "11110000100"          # REJEITA, termina em 00
```

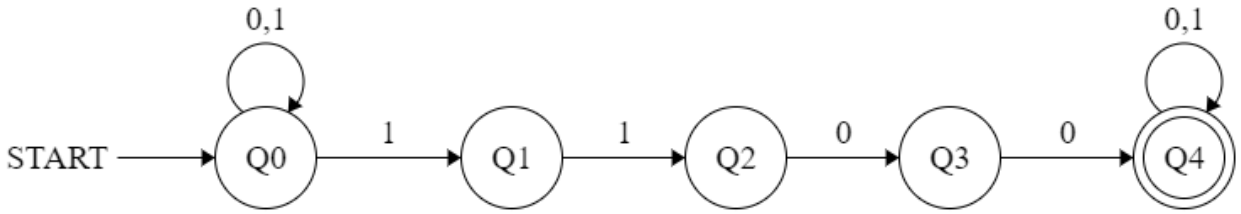
Teste 5

Autômato testado:

Aceita cadeias do tipo (0|1)* 1100 (0|1)*, ou seja, uma sequência de 0s e 1s que possuem 1100 como substring.

Estado\Input	0	1
Q0	Q0	Q0, Q1
Q1	-	Q2
Q2	Q3	-
Q3	Q4	-
Q4	Q4	Q4

Estado de aceitação Q4



Cadeias testadas / Output esperado

```
input1 = "010100101**1100**101"     # Aceita, 010100101(**1100**)101
input2 = "01"                       # REJEITA
input3 = "111**1100**110"           # Aceita, 111(**1100**)110
input4 = "000000111110"             # REJEITA
input5 = "11**1100**00100"          # Aceita, 11(**1100**)00100
```