Gustavo Ziyu Wang
8988790

EP 3

PCS3556 - Lógica Computacional
17 de Março de 2019

1 Introdução

O objetivo deste exercício programa é a simulação de autômatos utilizando a linguagem de programação funcional *Elixir*. Serão construídos dois simuladores:

1. Simulador de autômato finito determinístico
2. Simulador de autômato finito não determinístico

2 Implementação

No arquivo *ep3.exs* é apresentado o código do programa. Todas as funções referentes ao simulador estão contidas no módulo **Automata**, enquanto que os testes estão contidos no módulo **AutomataTest**.

Para utilizar o programa, basta iniciá-lo com o *Elixir*. Quando o código é executado, ele realiza automaticamente os testes, que serão apresentados na seção 3.

As funções que simulam os autômatos são **runFDA** e **runFNDA**. Ambas recebem os seguintes parâmetros de **entrada**:

- **input**: cadeia a ser analisada, no formato *Lista* de *Strings*. Ela é lida da esquerda para a direita. Ex.: input = [“a”, “b”, “a”, “b”]
- **state**: estado inicial da máquina, no formato *String*. Ex.: state = “q0”
- **finalStates**: lista de estados finais de aceitação, no formato *Lista* de *Strings*. Ex.: finalStates = [“q0”, “q1”]
- **transitions**: parâmetro que detalha as transições possíveis do autômato, no formato *Lista* de *Tuplas*. Ex.: transitions = [{“a”, “q0”, “q1”}, {“a”, “q1”, “q1”}, {“b”, “q1”, “q0”}]

Colocando corretamente os parâmetros de entrada na função, a simulação pode ser feita, como mostra a figura 1.

O programa retorna como **saída**:

- **true** se a cadeia é aceita pelo autômato. Isto é, o estado final é de aceitação.
- **false** se a cadeia não é aceita pelo autômato.

```
iex(6)> finalStates = ["q0"]
["q0"]
iex(7)> input = ["a", "b", "a", "b"]
["a", "b", "a", "b"]
iex(8)> state = "q0"
"q0"
iex(9)> transitions = [{"a", "q0", "q1"}, {"a", "q1", "q1"}, {"b", "q1", "q0"}]
[{"a", "q0", "q1"}, {"a", "q1", "q1"}, {"b", "q1", "q0"}]
iex(10)> Automata.runFDA(input, state, finalStates, transitions)
true
```

Figura 1: Exemplo de execução do programa.

2.1 Autômatos Finitos Determinísticos

Para implementar a primeira parte deste exercício programa, foram desenvolvidas duas funções auxiliares:

1. **nextState(input, state, [transitionsHead | transitionsTail])**: retorna o próximo estado, a partir da lista de transições fornecida.
2. **acceptState?(state, [h | t])**: verifica se o estado *state* está contido na lista de estados de aceitação.

Com essas funções, foi possível escrever a função **runFDA**, que é apresentada a seguir.

Algorithm 1: Simulador de Autômatos Finitos Determinísticos

```
def runFDA([inputHead | inputTail], state, finalStates, transitions) do
            cond do
            nextState(inputHead, state, transitions) != nil ->
                    runFDA(inputTail, nextState(inputHead,
                    state, transitions), finalStates, transitions)
            true -> false # rejeita pois nao ha transicao para inputHead
        end
end

def runFDA([], state, finalStates, _) do
            acceptState?(state, finalStates)
end
```

2.2 Autômatos Finitos Não Determinísticos

Para verificar se a cadeia deve ou não ser aceita pelo autômato não determinístico, o programa analisa todas os estados finais possíveis em que o autômato pode terminar a partir da entrada recebida. Caso algum destes estados seja um estado de aceitação, a cadeia será aceita.

Foram desenvolvidas as seguintes funções para auxiliar a implementação:

1. **getPossibleFinalStates([reachableStatesHead | reachableStatesTail], [transitionsHead | transitionsTail], newReachableStates, transitions, endStates)**: retorna todos os estados em que o autômato pode terminar, a partir da entrada recebida. Esta função recebe como entrada:
   - **[reachableStatesHead | reachableStatesTail]**: *Lista* de *Tuplas*, compostas por uma cadeia (*Lista* de *Strings*) e um estado (*String*). Representa o conjunto de estados e entradas que ainda devem ser simulados. Ex.: [{[“a”, “b”], “q0”}, {[“a”], “q1”}]
   - **[transitionsHead | transitionsTail]**: parâmetro que detalha as transições a serem analisadas do autômato, no formato *Lista* de *Tuplas*.
   - **newReachableStates**: novos conjuntos de cadeias e estados obtidos a partir das transições realizadas, no formato *Lista* de *Tuplas*.
   - **transitions**: *Lista* de *Tuplas* com todas as transições possíveis do autômato.
   - **endStates**: *Lista* de *Strings* com os estados finais atingidos.
2. **join(list, [h | t])**: realiza a junção de duas listas sem repetição.
3. **removeHead([_ | t])**: remove a cabeça da lista.
4. **getHead([h | _])**: retorna a cabeça da lista.
5. **checkIfContainsAnyElement(finalStates, [h | t])**: verifica se a lista de estados de aceitação contém algum dos elementos da lista *[h | t]*.

Com estas funções, a implementação da função que simula o autômato não determinístico, **runFNDA** é simples:

Algorithm 2: Simulador de Autômatos Finitos Não Determinísticos

```
def runFNDA(input, state, finalStates, transitions) do
        checkIfContainsAnyElement(finalStates,
        getPossibleFinalStates([{input, state}],
        transitions, [], transitions, []))
end
```

3 Testes

Os primeiros casos de teste (**FDA1** a **FDA4**) verificam que o programa funciona corretamente para um autômato simples, verificando, por exemplo, o caso no qual o programa recebe uma entrada que não está presente na lista de transições (**FDA3**).

Em seguida, os casos de teste simulam os autômatos apresentados em aula (*Busch, C.*).

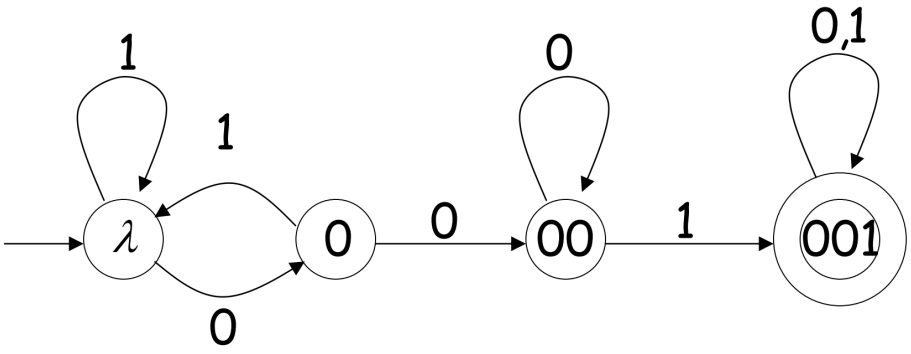

Figura 2: Autômato dos casos de teste FDA5 e FDA6.

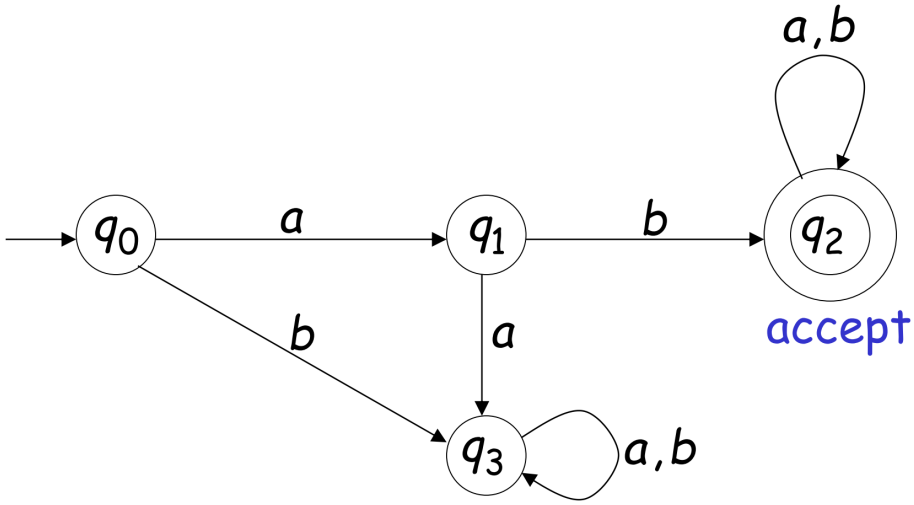

Figura 3: Autômato dos casos de teste FDA7 e FDA8.

Para os casos de teste **FDA5** e **FDA6** utilizou-se o autômato da figura 2, que aceita cadeias que contenham a sequência de caracteres *001*.

Em seguida, nos casos **FDA7** e **FDA8**, foi testado como entrada o autômato da figura 3, que aceita cadeias que começam com a sequência *ab*.

Para os autômatos não determinísticos, inicialmente foram feitos testes para verificar o bom funcionamento da função **getPossibleFinalStates**. No caso **Epsilon1**, a função retorna corretamente as saídas esperadas para o autômato que possui uma transição do tipo ϵ. Já o caso **Multiple1** verifica que a função retorna os estados corretos para um autômato que possui múltiplas transições com a mesma entrada.

Nos casos **FNDA1** a **FNDA4**, foi testado como entrada o autômato da figura 4, que aceita cadeias do tipo $\{ab\}\{ab\}^*$.

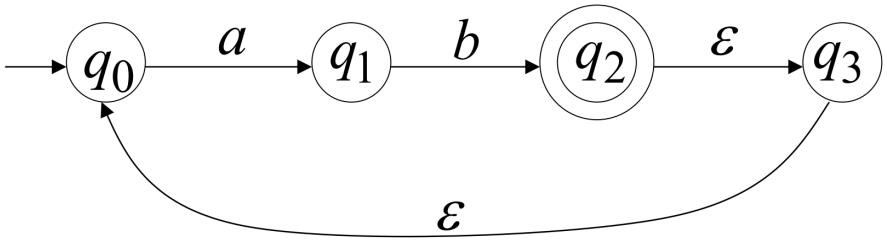

Figura 4: Autômato dos casos de teste FNDA1 a FNDA4.

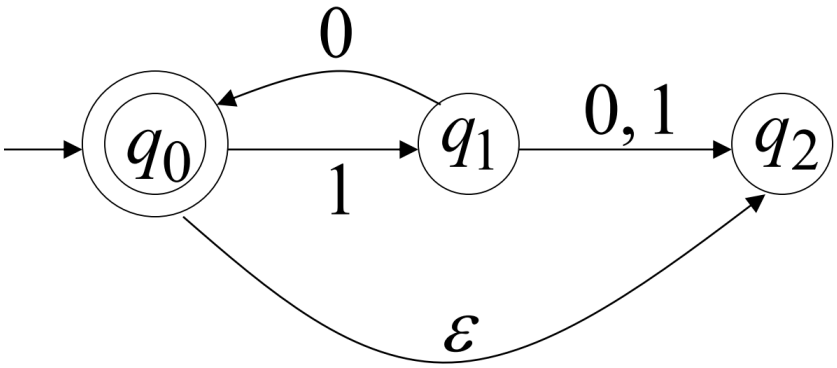

Figura 5: Autômato dos casos de teste FNDA5 a FNDA7.

Finalmente, os casos **FNDA5** a **FNDA7** testaram como entrada o autômato da figura 5, que aceita cadeias do tipo $\{10\}^*$.

Figura 6: Resultado dos testes.

Pode-se observar pela figura 6 que os resultados obtidos estão de acordo com o esperado.

4 Conclusão

Este terceiro exercício programa permitiu desenvolver ainda mais os conhecimentos em linguagem funcional e recursão utilizando a linguagem *Elixir*. Foi possível verificar que a implementação de um autômato finito determinístico é muito mais simples do que um não determinístico. Uma outra maneira de realizar a implementação deste último seria convertê-lo, como foi visto em aula, num autômato determinístico. Entretanto, durante a implementação do programa, verificou-se que este método seria muito mais complexo em relação à maneira que foi escolhida.

Referências

[1] Wagenknecht, C., Friedman, P.D.: Teaching Nondeterministic and Universal Automata Using SCHEME. Computer Science Education, vol. 8, Issue 3, 1998, p.197-227. (download em: http://www.informaworld.com/smpp/content content=a714015965)

[2] Busch, C.: Deterministic Finite Automata And Regular Languages, Louisiana State University.

[3] Busch, C.: Non-Deterministic Finite Automata, Louisiana State University.