Relatório - EP3

Tiago Koji Castro Shibata - 8988730
Escola Politécnica
Universidade de São Paulo
tiago.shibata@usp.br

I. Introdução

Esse relatório acompanha o terceiro exercício programa (EP3) da disciplina PCS3556 - Lógica Computacional. Nesse exercício programa, é implementado um simulador de autômato finito determinístico e não determinístico em Elixir.

26 de Março de 2018

II. Tarefa

A tarefa consiste em implementar um algoritmo de simulação de autômato determinístico e não determinístico em Elixir, experimentando com conceitos vistos em aula.

Dado um autômato, estado inicial e cadeia, o algoritmo deve retornar se é possível que o autômato aceite a cadeia (ou seja, há uma sequência de transições iniciando no estado inicial e acabando em estado de aceitação que gere a cadeia desejada).

Como vimos em aula, o estudo de autômatos é bastante importante. Linguagens reconhecidas por autômatos são regulares, e toda linguagem regular pode ser representada por um autômato. Autômatos determinísticos (DFA) e não determinísticos (NFA) são equivalentes (é possível converter qualquer NFA em DFA equivalente, que aceita e rejeita as mesmas cadeias).

Na hierarquia de Chomsky, autômatos finitos estão na classe de linguagens regulares:

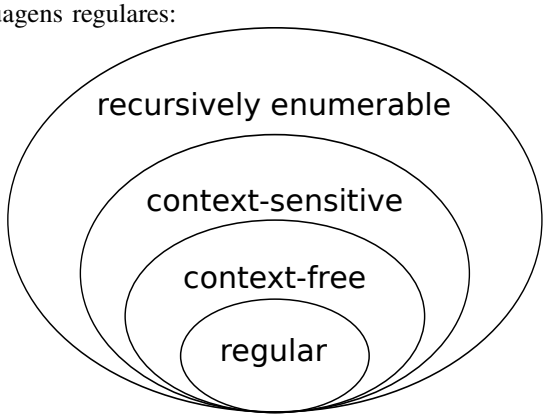

Fig. 1: Hierarquia de Chomsky

Na minha implementação, o autômato é dado como uma lista de transições (lista de tuplas do tipo $\{estado, caractere, próximo\_estado\}$). Os estados de aceitação são dados como uma lista de estados e a cadeia desejada é dada como uma lista.

III. Algoritmo

O algoritmo deve suportar ... (ou seja, pela hierarquia de Chomsky, ...):

Conforme especifiado, a implementação é recursiva e feita em Elixir, uma linguagem funcional. Conforme sugerido no enunciado, ...

IV. Estruturas de dados

Em alguns locais, estruturas de conjunto fornecidas pelo Elixir ($MapSet$) foram usadas tendo em mente performance e facilidade: o uso de conjunto evita que varramos a lista toda para buscar um elemento, e o conjunto permite operações fáceis e rápidas de união ou diferença quando necessário.

V. Código e testes

A função $example(arg1, arg2)$ recebe uma ... Ela gera ...

A regra é dada como uma tupla $\{cadeia\ inicial, cadeia\ a\ ser\ colocada\}$. Foram escritos testes para essa função:

Os testes foram essenciais no desenvolvimento, já que essa função apresenta muitos *corner cases*. Por exemplo, ...

A função foi implementada buscando a primeira correspondência da condição da regra na forma sentencial com a função *String.split...*:

References

[1] Friedel Ziegelmayer. *Elixir ExDoc*. https://hexdocs.pm/elixir/, acessado em 11/02/2018