EP2

Marcos Tan Chi Chen
NUSP: 9833065

Fevereiro 2021

Conteúdo

1 Objetivos

O principal objetivo deste exercício é compreender o funcionamento da funcional e implementar o algoritmo de reconhecimento de cadeias com base na gramática. Este algoritmo será implementado utilizando a linguagem funcional Elixir.

2 Visão geral do algoritmo

O algoritmo receberá como entrada dois parâmetros: uma cadeia w e uma gramática. A sua principal tarefa é reconhecer se esta cadeia w pertence à gramática fornecida. Para este fim, o algoritmo irá implementar algumas funções auxiliares que contribuirão para esta verificação. O algoritmo também utiliza tipo de dados (structs) para facilitar a definição dos parâmetros dos objetivos como será demonstrado abaixo.

2.1 Tipos de dados

Neste algoritmo serão utilizados dois tipos de estrutura de dados para facilitar o entendimento do algoritmo. São elas:

2.1.1 Gramatica

O tipo de dado "Gramatica"será a estrutura que irá guardar os parâmetros de uma gramática da forma G = (N, T, P, S) onde N é a lista de símbolos não terminais, T lista de símbolos terminais, P lista de leis de formações e S é o simbolo inicial da gramática.

```
defmodule Gramatica do
    defstruct naoTerminais: "SBC",
              terminais: "abc",
              leisDeFormacoes: [%LeiDeFormacao{alfa: "S", beta: "aBC"}],
              simboloInicio: "S"
end
```

2.1.2 Lei De Formação

O tipo de dado "LeiDeFormacao"representa as lei de formação da gramática. Cada lei de formação possui o seu alfa e o seu beta correspondente.

```
defmodule LeiDeFormacao do
    defstruct alfa: "a", beta: "abc"
end
```

3 Funções auxiliares

Para a operação do algoritmo, será construído algumas funções auxiliares que serão responsáveis por tarefas específicas do algoritmo para melhor modularização do código.

3.1 Função Procura Lei De Acordo O Alfa

Esta função tem como responsabilidade retornar a da lei de formação que contem um determinado alfa selecionado. Ela recebe como parâmetros: o conjunto de Leis de formações em que se quer realizar a procura e o valor do alfa no qual se quer procurar a lei de formação equivalente. Na imagem abaixo pode-se ver um exemplo de teste desta função.

```
test "procuraLeiDeAcordoComOAlfa" do

  leisDeFormacoes = [
    %LeiDeFormacao{alfa: "S", beta: "aSBC"},
    %LeiDeFormacao{alfa: "CB", beta: "BC"},
    %LeiDeFormacao{alfa: "aB", beta: "ab"},
    %LeiDeFormacao{alfa: "bB", beta: "bb"},
    %LeiDeFormacao{alfa: "bC", beta: "bc"},
    %LeiDeFormacao{alfa: "S", beta: "aBC"},
    %LeiDeFormacao{alfa: "cC", beta: "CC"}
  ]

  simboloInicial = "S"

  assert Ep2.procuraLeiDeAcordoComOAlfa(leisDeFormacoes, simboloInicial) ==  %LeiDeFormacao{alfa: "S", beta: "aSBC"}
end
```

Figura 1: Teste função procure lei de acordo com alfa

Como podemos observar abaixo, a função será feita de forma recursiva, com isso, teremos duas declarações da função. A condição de parada é quando o alfa da recursão for igual ao alfa procurado.

```
1  def procuraLeiDeAcordoComOAlfa([head | tail], alfa) do
2      if(head.alfa == alfa) do
3        head
4      else
5        procuraLeiDeAcordoComOAlfa(tail, alfa)
6      end
7  end
8
9  def procuraLeiDeAcordoComOAlfa([], alfa) do
10 end
```

3.2 Função Verifica Apenas Terminais

Já a função verifica apenas terminais tem a responsabilidade de validar se uma dada cadeia possui apenas caracteres terminais. Desta forma, o algoritmo saberá quando a cadeia em análise já é considerada como uma derivação final. Esta função recebe como parâmetros a lista de caracteres da cadeia a verificar, e a lista de caracteres terminais da gramática.

```
def verificaSeApenasTerminais([head | tail], listLetrasTerminais) do
    # if head pertence aos terminais, continua ate acabar, se acabar eh true
    pertence =
      Enum.find(listLetrasTerminais, fn terminal ->
        head == terminal
      end)

    if(pertence != nil) do
      verificaSeApenasTerminais(tail, listLetrasTerminais)
    else
      false
    end
end

def verificaSeApenasTerminais([], listLetrasTerminais) do
    true
end
```

```
test "verificaSeApenasTerminais" do
  beta = ["a","a","b","b","c","c"]
  terminais = ["a","b","c"]

  assert Ep2.verificaSeApenasTerminais(beta, terminais) == true
end
```

Figura 2: Teste função verifica se apenas terminais

3.3 Função Verifica Cadeia Pertence Derivações

Esta função será a ultima verificação realizada pelo algoritmo. Uma vez que todas as derivações possíveis da gramática forem listadas, esta função irá verificar se a cadeia a ser verificada pertence a todas as derivações geradas da gramática.

```
def verificaCadeiaPertenceDerivacoes([head | tail], cadeiaAVerificar) do
    if(head == cadeiaAVerificar) do
      true
```

```
    else
      verificaCadeiaPertenceDerivacoes(tail, cadeiaAVerificar)
    end
end

def verificaCadeiaPertenceDerivacoes([], cadeiaAVerificar) do
    false
end
```

```
test "verificaCadeiaPertenceDerivacoes" do
  listaDerivacoes = ["abc", "ac", "bc"]
  cadeiaAVerificar = "bc"

  assert Ep2.verificaCadeiaPertenceDerivacoes(listaDerivacoes, cadeiaAVerificar) == true
end
```

Figura 3: Teste função verifica se apenas terminais

3.4 Função Gerar Derivações

Esta é a função principal do algoritmo. Será a responsável por receber a gramática como parâmetro e gerar todas as derivações possíveis da gramática. Para o funcionamento desta função, o critério de parada da recursão será quando a cadeia gerada das derivações possuir apenas caracteres terminais ou caso contrário continuar realizando as derivações possíveis. Caso a cadeia gerada for maior que o comprimento da cadeia a verificar, já será considerado que a cadeia gerada não é mais derivável. Outro caso de parada será quando a cadeia derivada ainda possuir valores terminais, contudo não há mais leis de formações que possam realizar a derivação.

```
def geraDerivacoes([head | tail], simboloInicial, terminais, tamanhoCadeia,
    derivacoes) do
    if(String.length(head.beta) == tamanhoCadeia) do
    end

    listLetrasBeta = String.codepoints(head.beta)
    listLetrasTerminais = String.codepoints(terminais)

    #Verifico se o beta em analise apresenta apenas terminais
    isApenasTerminais = verificaSeApenasTerminais(listLetrasBeta,
    listLetrasTerminais)

    if(isApenasTerminais) do
      geraDerivacoes(tail, simboloInicial, terminais, tamanhoCadeia,
    derivacoes ++ [head.beta])
```

```
    else
      geraDerivacoes(tail, simboloInicial, terminais, tamanhoCadeia,
    derivacoes)
    end
end

# acabou, retorna a lista derivacoes
def geraDerivacoes([], simboloInicial, terminais, tamanhoCadeia, derivacoes)
    do
    derivacoes
end
```

4 Conclusão

Com isso, será concluído o segundo exercício de programação da matéria com a boa compreensão do algoritmo de reconhecimento de cadeias dentro de uma determinada gramática.