ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Graduação em Engenharia de Computação

PCS 3850 - Lógica computacional

Exercício Programa 2 - Relatório

Nome	**NUSP**
Arthur Pires da Fonseca	10773096
Breno Loscher Rocha	09784439

São Paulo - SP

13/03/2022

Sumário

Introdução

Ao longo das aulas, foi visto sobre como cada linguagem possui um conjunto de símbolos e um conjunto de gramáticas.

Com base nessas regras, pode-se descrever sistemas de linguagem arbitrariamente complexos, o que rapidamente cria um problema: como saber se uma sequência pertence ou não a uma linguagem ?

Dessa forma, o objetivo deste trabalho é criar um programa que, usando linguagens funcionais, consiga identificar se uma determinada cadeia faz parte da linguagem.

Enunciado

O objetivo do exercício é implementar o algoritmo de reconhecimento de cadeias geradas por uma gramática de estrutura de frase *recursiva*, que foi definido em sala de aula - algoritmo para reconhecer cadeias a partir de gramáticas.

No algoritmo um dos argumentos é a cadeia ω a ser verificada, e o outro a gramática.

Então, produz-se iterativamente um conjunto Ti contendo todas as formas sentenciais da gramática recursiva cujo comprimento ℓ seja $\ell \leq |\omega|$, até que o próximo conjunto $T_{i+1} = T_i$. Se $\omega \in T_{i+1}$ então a cadeia é aceita (isto é, foi gerada pela gramática), senão é rejeitada.

Interpretação do enunciado

Dada uma gramática, estruturada por um conjunto de símbolos, recebidos numa lista, e um conjunto de estrutura de frase recebidos como uma matriz Nx2, deve-se, de forma recursiva, gerar todas as possíveis cadeias para um determinado tamanho.

Por fim, procura-se determinada cadeia, nessa lista de possibilidades, para que seja identificado se ela faz parte ou não da linguagem.

Desenvolvimento

O primeiro item pedido no EP foi ignorado, pois o grupo não conseguiu identificar como “uma função [...] que permita percorrer um conjunto recursivamente [...] e, a cada chamada recursiva da sua função, retorne um dos elementos do conjunto” poderia encaixar neste exercício.

Em suma, o código pode ser dividido em duas partes: a parte principal gera as sequências, e a parte de busca faz uso delas para identificar se uma certa cadeia faz parte da linguagem em questão.

Para esclarecer as partes do arquivo principal, a seguir uma descrição do que cada função faz no código:

- **cadeia_maiusc:** identifica se **qualquer** cadeia na lista de palavras tem caracteres não-terminais.
- **tem_maiusc:** identifica se uma variável do tipo String possui **alguma** letra maíuscula em sua composição.
- **generate_partial_vocab:** em cada palavra intermediária do processo de geração da linguagem, substitui os símbolos “à esquerda” nas regras da gramática pelos símbolos “à direita” na mesma; para fazer isso, “recorta” a palavra original em várias partes (String.split), para depois “colá-las” substituindo cada uma das subcadeias originais pelas cadeias ditadas pela gramática; para que a função devolva um conjunto apenas com cadeias de caracteres, é necessário haver alguns ajustes operacionais, utilizando união de conjuntos (MapSet.union) e o método “reduce”, que tem justamente a propriedade de **reduzir** um conjunto de dados a um só.
- **remove_too_big:** remove da lista de palavras as cadeias que são maiores que um tamanho pré-estabelecido; como as gramáticas utilizadas sempre aumentam e, portanto, nunca diminuem as cadeias geradas, esta função serve para limitar a quantidade de palavras geradas pelo programa, garantindo que sua execução irá parar em algum momento.
- **remove_uppercase:** remove da lista de cadeias todas as palavras que têm **algum** caractere maiúsculo.
- **generate_vocab:** gera a linguagem definida por uma gramática, começando com o símbolo não-terminal inicial “S”.
- **find_word_on_vocab:** utiliza a função “generate_vocab” para saber todas as palavras que compõem a linguagem em questão, em seguida identifica se a palavra procurada pode ser encontrada no conjunto de palavras gerado.

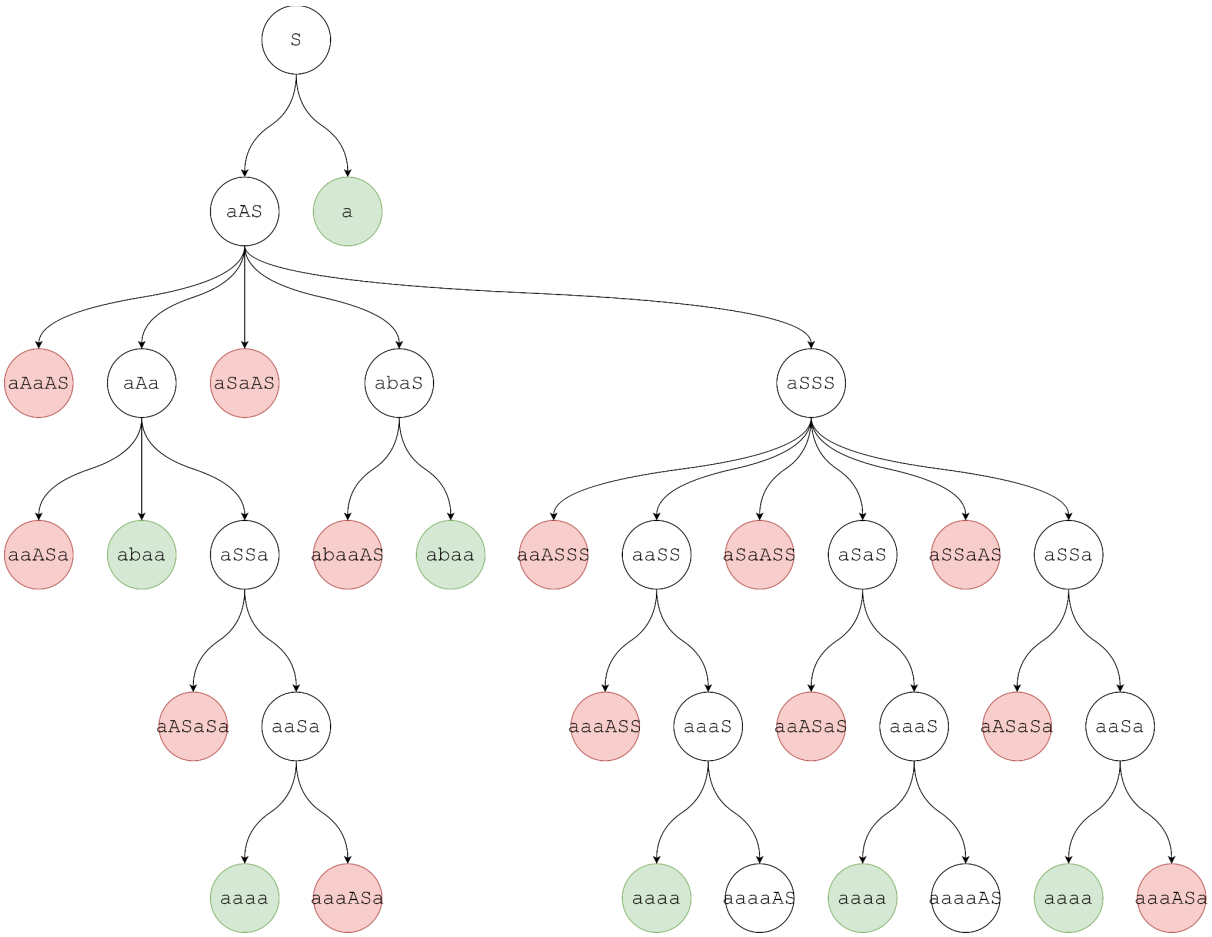
Dessa forma, o código cumpre os outros dois itens propostos no EP:

- Gerar a linguagem a partir de uma gramática:
    - Gerar sequências, baseado nas transições de um único símbolo:
        - A ideia é substituir todas as regras da gramática nas palavras geradas parcialmente a cada iteração do algoritmo.
        - As palavras geradas não deverão exceder o tamanho “max_size” definido.
        - Assim que a lista de palavras gerada pela gramática sobre as cadeias parciais até então compiladas, o algoritmo para e remove todas as palavras que ainda possuem símbolos não-terminais.
        - As palavras que sobrarem estão contidas na linguagem gerada pela gramática fornecida.
- Verificar se uma certa palavra pertence à linguagem:
    - Utilizando o processo descrito no item anterior, deve-se verificar se a palavra está contida no conjunto final gerado.

Um exemplo do funcionamento do código pode ser visto abaixo:

Caracter inicial: "S"
Gramatica: [["S", "aAS"],["S", "a"],["A", "SbA"], ["A", "ba"], ["A", "SS"]]
Sequencia: “abaa”

Primeira parte, achar as sequências:



Em vermelho estão as sequência que são maiores que a que deve ser procurada em verde as sequências terminais.

Depois procuramos a sequencia "abaa" na lista de resultados:

Vocabulário: "abaa" ∈ ["a","abaa","aaaa"] → Verdadeiro

Conclusão

Neste projeto, a dupla pôde exercitar um pouco mais dos conhecimentos de linguagens funcionais, com um uso de construções mais complexas em Elixir.

Além disso, o conhecimento e o entendimento sobre a construção de um vocabulário foi expandido.

Ao longo do programa, muitas dúvidas surgiram, e por sua vez consultas aos materiais em aula, dessa forma os conhecimentos da disciplina foram exercitados, e um pouco mais de suas possíveis aplicações entendidas.

Apêndice

O código fonte deste EP pode ser encontrado neste repositório: https://github.com/Brenolr/LogicaComputacional/tree/main/Ep2/ep2.

Uma das versões finais do código principal e do arquivo de testes encontram-se a seguir:

cadeias.ex

```
defmodule Cadeias do

 def cadeia_maiusc(lista) do
   list_has_uppercase = Enum.map(lista, fn el -> tem_maiusc(el) end)

   has_uppercase = Enum.reduce(list_has_uppercase, fn el, acc ->  el
or acc end)
   has_uppercase
 end

 def tem_maiusc(palavra) do
   palavra != String.downcase(palavra)
 end

 def generate_partial_vocab(word_set, grammar) do

   l = Enum.map(word_set,
     fn word ->
       l2 = Enum.map(grammar,
         fn rule ->
           [pattern, replacement] = rule
           split_word = String.split(word, pattern)

           n = Enum.count(split_word) - 2
           partial_vocab = if n >= 0 do
             for i <- 0 .. n do
               part1 = Enum.slice(split_word, 0, i)
               part2 = Enum.slice(split_word, i, 2)
               part3 = Enum.slice(split_word, i + 2, n - i)
```

```
              first = Enum.join(part1, pattern)
              middle = Enum.join(part2, replacement)
              last = Enum.join(part3, pattern)

              if Enum.count(part1) == 0 do
                if Enum.count(part3) == 0 do
                  middle
                else
                  Enum.join([middle, last], pattern)
                end
              else
                if Enum.count(part3) == 0 do
                  Enum.join([first, middle], pattern)
                else
                  Enum.join([first, middle, last], pattern)
                end
              end
            end
          else
            MapSet.new([word]) # mantain lowcase words
          end
          partial_vocab
        end
      )
      s2 = MapSet.new(l2)

      s2
    end
  )

  foo = fn l, acc ->
    if not is_nil(l) do
      if not is_nil(acc) do
        MapSet.union( MapSet.new(l), MapSet.new(acc) )
      else
        MapSet.union( MapSet.new(l), MapSet.new([]) )
      end
    end
  end

  new_wordlist_set = Enum.reduce(l, fn s, acc -> MapSet.union(s,
acc) end)
```

```
    new_word_set = Enum.reduce(new_wordlist_set, foo)

    new_word_set
  end

  def remove_too_big(word_set, max_size) do
    Enum.filter(word_set, fn el -> (String.length(el) <= max_size)
end)
  end

  def remove_uppercase(word_set) do
    Enum.filter(word_set, fn el -> not tem_maiusc(el) end)
  end

  # word_set precisa começar com um elemento
  def generate_vocab(word_set, grammar, max_size, last_clean_size) do

    clean_word_set = remove_too_big(word_set, max_size)

    if cadeia_maiusc(clean_word_set) do
      new_word_set = generate_partial_vocab(clean_word_set, grammar)

      new_clean_size = Enum.count(clean_word_set)

      if last_clean_size != new_clean_size do
        generate_vocab(new_word_set, grammar, max_size,
new_clean_size)
      else
        remove_uppercase(new_word_set)
      end
    else
      remove_uppercase(clean_word_set)
    end
  end

  def find_word_on_vocab(init, grammar, word) do
    vocab = Cadeias.generate_vocab(MapSet.new(init), grammar,
String.length(word), 0)
    # IO.inspect(vocab)
    Enum.member?(vocab, word)
  end
```

```
end
```

cadeias_test.exs

```
defmodule CadeiasTest do
 use ExUnit.Case
 doctest Cadeias

 test "Caso de Testes 01" do
   grammar = [["S", "aAS"],["S", "a"],["A", "SbA"], ["A", "ba"],
["A", "SS"]]
   inicial = ["S"]
   sequence = "abaa"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
true
 end
 test "Caso de Testes 02" do
   grammar = [["S", "aAS"],["S", "a"],["A", "SbA"], ["A", "ba"],
["A", "SS"]]
   inicial = ["S"]
   sequence = "a"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
true
 end
 test "Caso de Testes 03" do
   grammar = [["S", "aAS"],["S", "a"],["A", "SbA"], ["A", "ba"],
["A", "SS"]]
   inicial = ["S"]
   sequence = "abbb"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
false
 end
 test "Caso de Testes 04" do
   grammar = [["A", "Aa"],["B", "b"],["C", "c"], ["A", "a"]]
   inicial = ["A"]
   sequence = "aaa"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
```

```
true
 end
 test "Caso de Testes 05" do
   grammar = [["A", "Aa"],["B", "b"],["C", "c"], ["A", "a"]]
   inicial = ["A"]
   sequence = "aaaaaa"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
true
 end
 test "Caso de Testes 06" do
   grammar = [["A", "Aa"],["B", "b"],["C", "c"], ["A", "a"]]
   inicial = ["A"]
   sequence = "abbb"
   assert Cadeias.find_word_on_vocab(inicial,grammar,sequence) ==
false
 end
end
```