Linguagens Formais e Autômatos (LFA)

Slides complementares da aula de 06/11/2013

Dicas para o trabalho de implementação em Ruby – Conversão para a Forma Normal de Greibach

Algoritmos de conversão de uma GLC qualquer para a forma normal de Greibach

- Eliminação de:
  - Símbolos inúteis
  - Símbolos inacessíveis
  - Produções unitárias
  - Transições em vazio
  - Recursões à esquerda

- Conversão para a FN de Greibach
  - Permite o uso da classe GramaticaGreibach, que espera um conjunto de produções já convertidas

Interface de implementação

```
class ConversorGLC

    def self.EliminarSimbolosInuteis( glc )

    def self.EliminarSimbolosInacessiveis( glc )

    def self.EliminarProducoesEmVazio( glc )

    def self.EliminarProducoesUnitarias( glc )

    def self.EliminarRecursoesAEsquerda( glc )

    def self.ConverterFNGreibach( glc )

end
```

Todos os métodos recebem e retornam um objeto da classe **GramaticaLivreContexto**

Exemplo 1 (eliminação de símbolos inúteis)

```
require "glc/ConversorGLC"

glc1 = GramaticaLivreContexto.new()

glc1.adicionarProducao({ "<S>" => ["<A>", "<B>"] })
glc1.adicionarProducao({ "<A>" => ["a<B>", "b<S>", "b"] })
glc1.adicionarProducao({ "<B>" => ["<A><B>", "<B>a"] })
glc1.adicionarProducao({ "<C>" => ["<A><S>", "b"] })

glc2 = ConversorGLC::EliminarSimbolosInuteis( glc1 )
puts glc2.listarProducoes
```

Não terminais delimitados por '<' e '>', como na notação BNF

Resultado:

```
<A> => b<S> | b
<C> => <A><S> | b
<S> => <A>
```

Exemplo 2 (eliminação de produções em vazio)

```
require "glc/ConversorGLC"

glc1 = GramaticaLivreContexto.new()
glc1.adicionarProducao({ "<S>" => ["<A><B><C>"] })
glc1.adicionarProducao({ "<A>" => ["<B><B>", ""] })
glc1.adicionarProducao({ "<B>" => ["<C><C>", "a"] })
glc1.adicionarProducao({ "<C>" => ["<A><A>", "b"] })
glc2 = ConversorGLC::EliminarProducoesEmVazio( glc1 )
puts glc2.listarProducoes
```

Resultado:

```
<S> => <A><B><C> | <B><C> | <A><C> | <A><B> | <C> | <B> | <A>
<A> => <B><B> | <B>
<B> => <C><C> | a | <C>
<C> => <A><A> | b | <A>
<S'> => <S> | ''
```

Exemplo 3 (conversão para FN Greibach)

```
require "glc/ConversorGLC"
glc1 = GramaticaLivreContexto.new()
glc1.adicionarProducao({ "<E'>" => ["+<T>", "+<T><E'>"] })
glc1.adicionarProducao({ "<T'>" => ["*<F>", "*<F><T'>"] })
glc1.adicionarProducao({ "<E>" => ["<T>", "<T><E'>"] })
glc1.adicionarProducao({ "<T>" => ["<F>", "<F><T'>"] })
glc1.adicionarProducao({ "<F>" => ["(<E>)", "a"] })
glc1_linha = ConversorGLC::ConverterFNGreibach( glc1 )
puts glc1_linha.listarProducoes
```

Resultado:

```
<F> => (<E><)'> | a
<T> => (<E><)'> | a | (<E><)'><T'> | a<T'>
<E> => (<E><)'> | a | (<E><)'><T'> | a<T'> | (<E><)'><E'> | a<E'>
       | (<E><)'><T'><E'> | a<T'><E'>
<T'> => *<F> | *<F><T'>
<E'> => +<T> | +<T><E'>
<)'> => )
```

Rotinas auxiliares

ConversorGLC::Simbolos(w)	Retorna um array com os símbolos de w. Exemplo: `Simbolos('<A>b') == ['<A>','b']`
ConversorGLC::Simbolo(w, i)	Retorna o i-ésimo símbolo de w. Exemplo: `Simbolos('<A>b',1)=='b'`
ConversorGLC::Terminal?(s)	Indica se s é símbolo terminal
ConversorGLC::NaoTerminal?(s)	Idem, para não terminal
GramaticaLivreContexto.listarProducoes	Apresenta o conteúdo das produções da GLC
GramaticaLivreContexto.naoTerminais	Retorna um array contendo os símbolos não terminais da GLC
GramaticaLivreContexto.producoes	Retorna um hash contendo as produções da GLC

Dicas para a implementação em Ruby

- Estudar os principais métodos da classe **Array**, entre eles:
  - each, each_index
  - include?
  - index
  - join
  - all?
  - select
  - operadores `|`, `&`, `<<`, `+`, `-`
- Lembrar que arrays iniciam com índice zero
- Últimos elementos podem ser indexados com índice -1, -2, etc.
- Ranges são representados por x..y.

Exemplos

```
a = [ 2, 3, 4, 5 ]

puts a[1..-1].inspect  # [3, 4, 5]
puts a.include?(3)     # true
puts a.index(3)     # 1
puts a.join         # 2345
puts a.all?{ |x| x < 10 } # true

puts a.select{|x| x.modulo(2)==0}.inspect  # [2, 4]

puts (a | [5,6]).inspect  # [2, 3, 4, 5, 6]
puts (a & [5,6]).inspect  # [5]
puts (a << 5).inspect  # [2, 3, 4, 5, 5]
puts (a + [5]).inspect # [2, 3, 4, 5, 5, 5]
```

Note que, neste caso, o array a foi alterado (ver linha seguinte)

Tutoriais de Ruby e orientação a objetos

- https://www.ruby-lang.org/pt/documentation/
- http://www.tutorialspoint.com/ruby/ruby_object_oriented.htm
- http://www.techotopia.com/index.php/Ruby_Object_Oriented_Programming