

# Linguagens Formais e Autômatos (LFA)

Aula de 28/10/2013

**Propriedades Formais Relevantes das  
LLC's; Lema do Bombeamento; Linguagens  
LL e LR; Gramáticas LC Determinísticas;  
Implementação em Ruby**

## De aula anterior

Fonte: <http://www.univasf.edu.br/~marcus.ramos/lfa-2008-1/Apostila.pdf>

Considerando a definição em [Sipser 8.1], uma linguagem é dita estritamente livre de contexto se ela for livre de contexto, porém não-regular. A característica desse tipo de linguagens é que elas são geradas apenas por gramáticas que possuam pelo menos um símbolo não-terminal que seja auto-recursivo central e essencial. Assim, é possível gerar, nas formas sentenciais geradas pela gramática, subcadeias da forma:

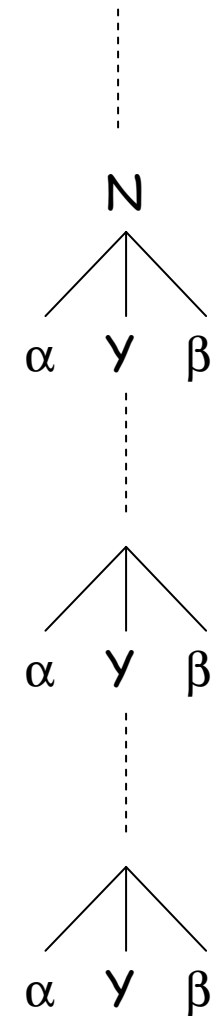
$$Y \Rightarrow^* \alpha Y \beta, \quad \text{com } \alpha, \beta \in \Sigma^+$$

as quais são responsáveis pelo balanceamento dos termos  $\alpha$  e  $\beta$  nas sentenças da linguagem.

De fato, gramáticas lineares à direita não permitem a definição de símbolos não-terminais com essa propriedade. Ao contrário, gramáticas lineares à direita geram apenas formas sentenciais da seguinte forma:

$$Y \Rightarrow^* \alpha Y, \quad \text{com } \alpha \in \Sigma^+$$

em que, obviamente, não há balanceamento de termos nem, portanto, aninhamentos sintáticos. A existência de termos balanceados (ou aninhados) é, por isso, o fator que diferencia uma linguagem estritamente livre de contexto de uma linguagem regular.



## Lema do Bombeamento para LLC's

**Teorema 4.17** (“Pumping Lemma” para linguagens livres de contexto) *Seja  $L$  uma linguagem livre de contexto, com  $\epsilon \notin L$ . Então, existe uma constante inteira  $n$ , dependente apenas de  $L$ , que satisfaz às seguintes condições: (i)  $\forall \gamma \in L, |\gamma| \geq n, \gamma = uvwxy$ ; ; (ii)  $|vwx| \leq n$ ; (iii)  $|vx| \geq 1$ ; (iv)  $\forall i \geq 0, uv^iwx^iy \in L$ .*

Condição (i): para qualquer cadeia  $\gamma$  pertencente a  $L$ , cujo tamanho é no mínimo igual a uma constante  $n$  que depende apenas de  $L$  é possível particioná-la em 5 subcadeias  $uvwxy$  (sendo  $u$  um “prefixo” e  $y$  um “sufixo” de  $\gamma$ ).

Condição (ii): o tamanho da cadeia central  $vwx$  é no máximo igual a  $n$  (i.e. prefixo e/ou sufixo podem ser nulos).

Condição (iii): a concatenação  $v.x$ , da cadeia central, não pode resultar em  $\epsilon$ .

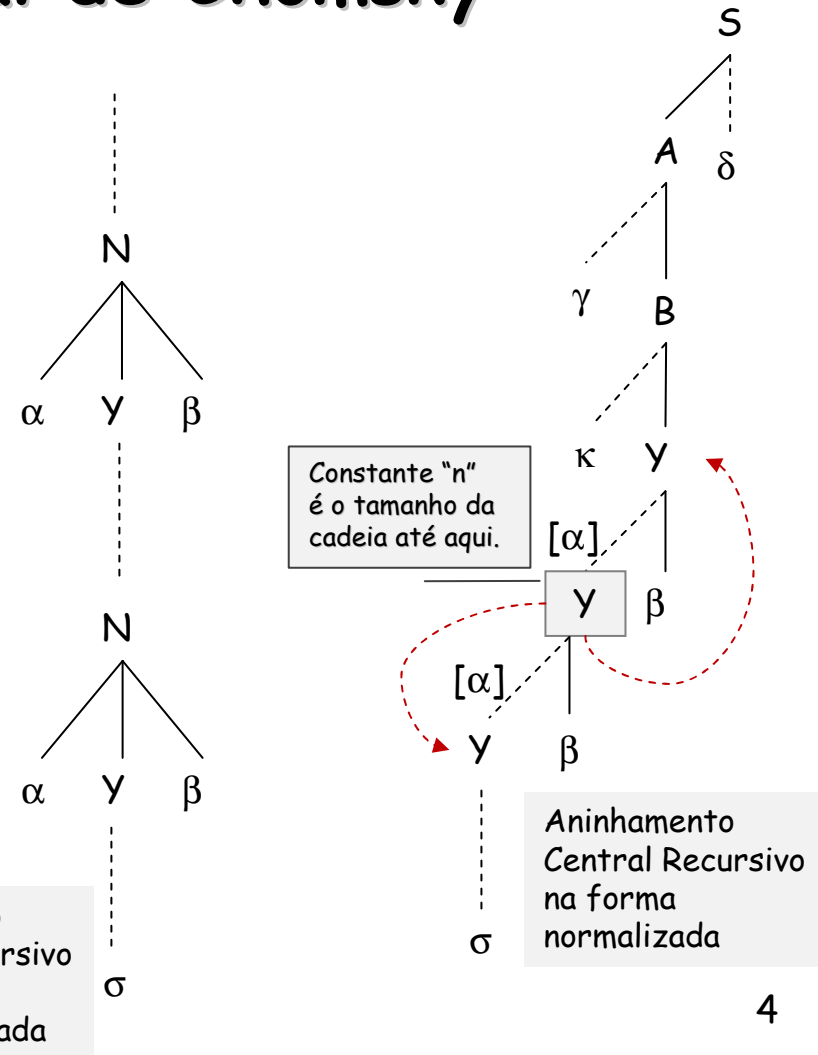
Condição (iv): qualquer cadeia resultante de  $uv^iwx^iy$  para  $i \geq 0$  pertence a  $L$ .

Resumindo: se  $uvwxy \in L$  então  $uv^*wx^*y \in L$ , respeitadas as condições i-iv.

# O Lema e a Forma Normal de Chomsky

Como já visto, toda GLC pode ser normalizada para que suas estruturas internas (i.e. desconsiderado o último passo de derivação) sejam árvores binárias.

O que o Lema do Bombeamento está "dizendo", como no caso das linguagens regulares, é que, se uma  $L$  não possui cadeias que podem ser "bombeadas" nas condições apresentadas anteriormente, então ela não é uma LLC.



## Exercícios

As linguagens caracterizadas a seguir são livres de contexto? Por quê?

1. A Linguagem  $L$  cujas cadeias têm a forma  $a^n b^n c^n$
2. A Linguagem  $L$  cujas cadeias têm a forma  $\{wcw \mid w \in \{a, b\}^+\}$
3. A Linguagem  $L$  cujas cadeias têm a forma  $\{ww \mid w \in \{a, b\}^*\}$
4. A Linguagem  $L$  cujas cadeias têm a forma  $\{a^n b^n a^m \mid n \geq 0, m \geq 0 \text{ e } n \neq m\}$
5. A Linguagem  $L$  cujas cadeias têm a forma  $\{a^k \mid k \geq 1 \text{ é um número primo}\}$

## Linguagens Livres de Contexto Não Determinísticas

As linguagens livres de contexto são reconhecidas por autômatos de pilha (matéria que será detalhada na próxima aula).

Tal como já visto para os autômatos finitos, reconhecedores de linguagens regulares, também os autômatos de pilha podem ser determinísticos ou não determinísticos.

Porém, ao contrário do que se passa com as linguagens regulares - que SEMPRE podem ser reconhecidas por um autômato determinístico - há linguagens livres de contexto que só podem ser reconhecidas por autômatos não determinísticos.

"A linguagem para a qual se pode provar a inexistência de quaisquer autômatos de pilha determinísticos que a reconheçam, e, simultaneamente, a existência de pelo menos um autômato de pilha não-determinístico que a reconheça, é denominada linguagem livre de contexto não-determinística."



## Linguagens LL(k) e LR(k)

As cadeias  $w$  pertencentes a Linguagens Livres de Contexto resultam de um processo de derivação que aplica sucessivamente regras de produção que ligam o símbolo raiz  $S$  de uma gramática livre de contexto até as cadeias  $w$ . ( $S \Rightarrow^* w$ )

As etapas intermediárias da derivação  $S \Rightarrow^* w$  determinam a "estrutura sintática" de  $w$ , a qual pode ser única (para gramáticas não ambíguas) ou múltipla (para gramáticas ambíguas).

A produção de uma estrutura sintática para a cadeia  $w$  é chamado de "parsing" (ou "análise sintática"). Os algoritmos de parsing podem: (a) privilegiar derivações à esquerda; (b) privilegiar derivações à direita; ou (c) misturar (a) e (b), seguindo um controle mecânico ou manual, estabelecido por um usuário (ou outro programa).

Continua.

## Linguagens LL(k) e LR(k)

### Continuação

O **parsing** de  $w$  pode tornar-se mais eficiente se no processo de derivação (à esquerda ou direita), o algoritmo de análise puder **usar um "oráculo"** (ou informação do que vem a seguir, considerando como símbolo de entrada uma **subcadeia  $x$  de  $w$**  tal que  $|x| > 1$ ). Esta "predição" pode evitar que sejam seguidos caminhos alternativos improdutivos, o torna o **parsing determinístico**. Este mecanismo é chamado de "**lookahead**".

Uma linguagem é chamada de LR(k) se:

- a análise da cadeia de entrada é feita da esquerda (**Left**) para a direita
- o "parsing" segue processo de derivação à direita (**Right**) e
- usa como "oráculo" a inspeção de  **$k$**  símbolos da cadeia de entrada na análise. Se  $k=1$ , o parsing não necessita de oráculo para ser determinístico.



## Linguagens LL(k) e LR(k)

### Continuação

Uma linguagem é chamada de LL(k) se:

- a análise da cadeia de entrada é feita da esquerda (**L**eft) para a direita
- o "parsing" segue processo de derivação à esquerda (**R**ight) e
- usa como "oráculo" a inspeção de k símbolos da cadeia de entrada na análise. Se  $k=1$ , o parsing não necessita de oráculo para ser determinístico.

É fácil perceber que toda linguagem determinística com  $k = n$  continuará determinística com  $k > n$ , embora o inverso não seja verdadeiro (uma linguagem determinística com  $k > n$ , não necessariamente o é para  $k = n$ ).

## Definição Formal (Ramos, 2009)

Suponha-se que  $G = (V, \Sigma, P, S)$  seja uma gramática livre de contexto. Então:

- Diz-se que  $G$  é uma gramática  $LL(k)$  se for possível analisar deterministicamente as cadeias de  $L(G)$  da seguinte forma: quaisquer que sejam a cadeia de entrada  $w \in L(G)$  e a forma sentencial  $\alpha X \beta$ , obtida por meio da aplicação exclusiva de derivações mais à esquerda, e tal que  $S \Rightarrow^* \alpha X \beta$ , com  $\alpha \in \Sigma^*$ ,  $X \in N$  e  $\beta \in V^*$ , se a escolha da produção  $X \rightarrow \mu$  puder ser feita de forma unívoca, inspecionando-se no máximo os  $k$  primeiros símbolos de  $\gamma$ , onde  $\alpha X \beta \Rightarrow \alpha \mu \beta \Rightarrow^* \alpha \gamma$ ,  $\gamma \in \Sigma^*$ . A aplicação da produção escolhida faz com que  $\alpha X \beta \Rightarrow \alpha \mu \beta$  e, eventualmente,  $\alpha \mu \beta \Rightarrow^* w$ .
- Diz-se que  $G$  é uma gramática  $LR(k)$  se for possível analisar deterministicamente as cadeias de  $L(G)$  da seguinte forma: quaisquer que sejam a cadeia de entrada  $w \in L(G)$  e a forma sentencial  $\alpha \mu \beta$ , obtida por meio da aplicação exclusiva de reduções<sup>3</sup> mais à esquerda, e tal que  $w \Rightarrow^* \alpha \gamma \Rightarrow^* \alpha \mu \beta$ , com  $\alpha, \mu \in V^*$  e  $\beta, \gamma \in \Sigma^*$ , se a escolha da produção  $X \rightarrow \mu$  puder ser feita de forma unívoca, inspecionando-se no máximo os  $k$  primeiros símbolos de  $\gamma$ . A aplicação da produção escolhida faz com que  $\alpha \mu \beta \Rightarrow \alpha X \beta$  e, eventualmente,  $\alpha X \beta \Rightarrow^* S$ .

<sup>3</sup>Uma redução corresponde ao inverso de uma derivação, ou seja, à substituição do lado direito de uma produção, em uma forma sentencial, pelo lado esquerdo correspondente. Por exemplo, se  $X \rightarrow \mu$  é uma produção, então  $\alpha X \beta \Rightarrow \alpha \mu \beta$  denota uma derivação e  $\alpha \mu \beta \Rightarrow \alpha X \beta$  denota uma redução. Uma sequência de reduções mais à esquerda corresponde à ordem inversa da sequência de derivações mais à direita que gera a mesma cadeia.

Exercício em duplas

Mostre que/se

$G_2 = (V, \Sigma, P, S)$   
 $V = \{S, X, a, b, c\}$   
 $\Sigma = \{a, b, c\}$   
 $P = \{S \rightarrow Xc,$   
 $\quad X \rightarrow aXb \mid \varepsilon\}$

é  $LL(1)$ .

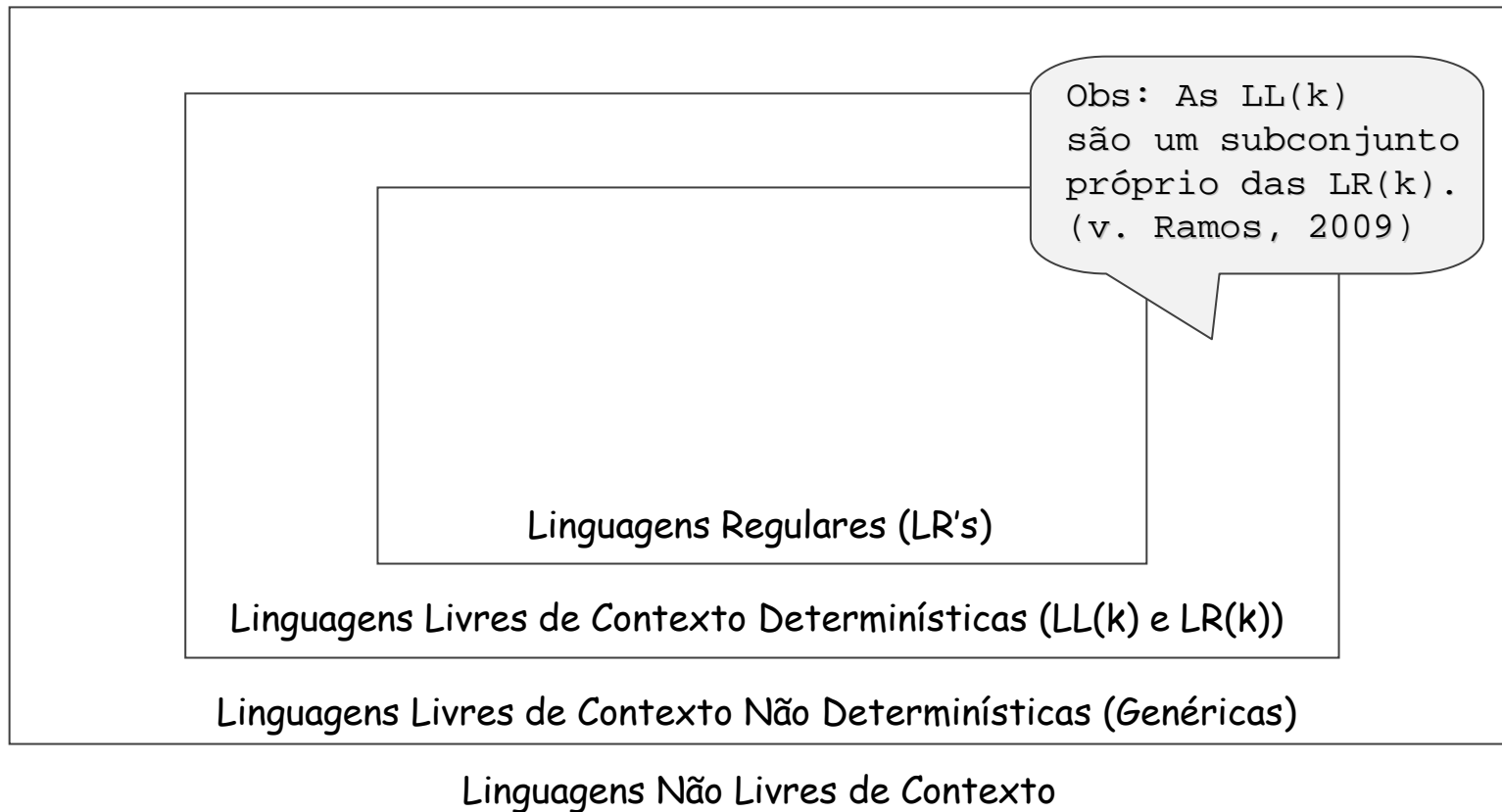
Exercício para casa

Mostrar que/se

$G_2 = (V, \Sigma, P, S)$   
 $V = \{S, X, a, b, c\}$   
 $\Sigma = \{a, b, c\}$   
 $P = \{S \rightarrow Xc,$   
 $\quad X \rightarrow aXb \mid \varepsilon\}$

é  $LR(1)$ .

# LLC's Não Determinísticas, Determinísticas e LR's



## Ambiguidade e Determinismo (Ramos, 2009)

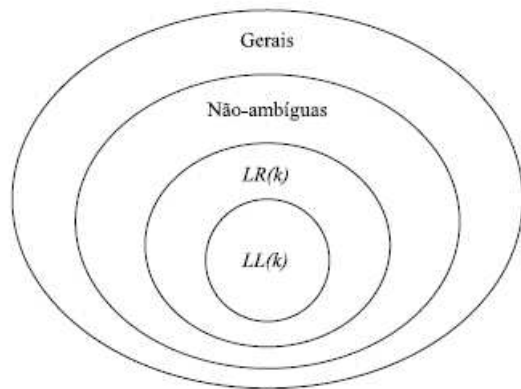


Figura 4.24: Hierarquia das gramáticas livres de contexto

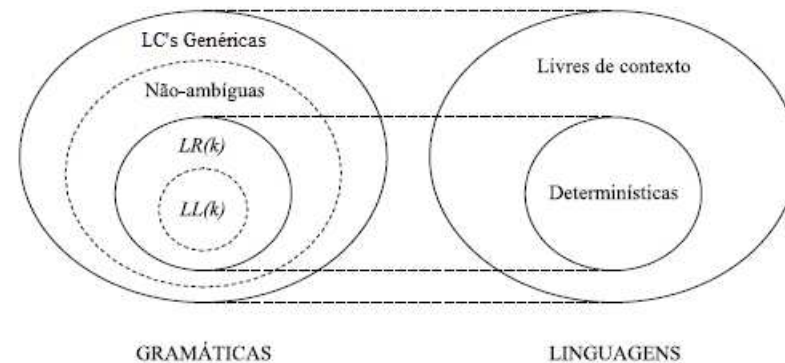


Figura 4.25: Correspondência entre classes de gramáticas e classes de linguagens livres de contexto

Diz-se que uma gramática livre de contexto é não-ambígua se, para toda e qualquer cadeia pertencente à linguagem por ela gerada, existir uma única seqüência de derivações mais à esquerda e uma única seqüência de derivações mais à direita que a geram.

---

## **Auto-Aprendizado: Propriedades de Fechamento das LC's**

Livro de Ramos (2009), seção 4.13 do capítulo sobre Linguagens Livres de Contexto.

## Implementação de gramáticas em Ruby (livro Ramos(2009), p.403)

- Novas classes
  - `GramaticaLivreContexto`
  - `GramaticaGreibach`
  - `Greibach`



## Arquivo glc/GramaticaLivreContexto.rb (1)

```
class GramaticaLivreContexto

  def initialize()
    @producoes = {}
  end

  def adicionarProducao( producao )
    @producoes.update( producao )
  end

end
```

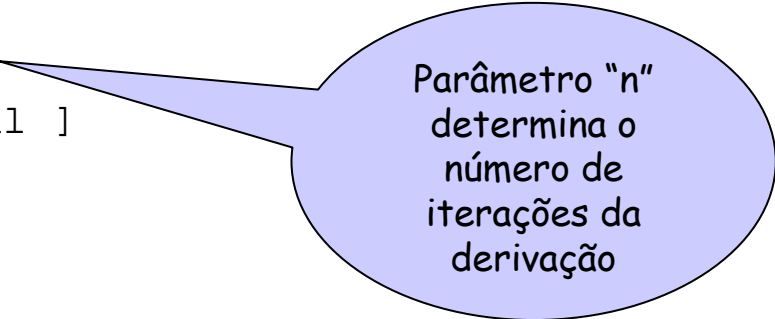
Parâmetro "produção"  
deve ser um hash.

Exemplo:

`{"E" => ["T+E", "T"]}`

## Arquivo glc/GramaticaLivreContexto.rb (2)

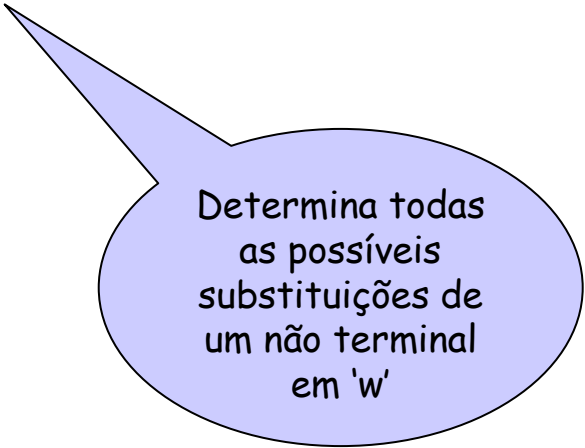
```
def derivar( formaInicial, n )  
  formasSentenciais = [ formaInicial ]  
  derivacoes = []  
  (1..n).each do |i|  
    formasSeguintes = []  
    formasSentenciais.each do |w|  
      formasDeW = aplicarProducoesEm( w )  
      derivacoes << { w => formasDeW }  
      formasSeguintes |= formasDeW  
    end  
    formasSentenciais = formasSeguintes  
  end  
  return derivacoes  
end
```



Parâmetro "n"  
determina o  
número de  
iterações da  
derivação

## Arquivo glc/GramaticaLivreContexto.rb (3)

```
def aplicarProducoesEm( w )  
  
  formasSentenciais = []  
  
  @producoes.each do |ladoEsquerdo, ladosDireitos|  
    formasSentenciais |=  
      calcularFormasSentenciaisDe( w,  
        [ladoEsquerdo, ladosDireitos] )  
  end  
  
  return formasSentenciais  
end
```



Determina todas  
as possíveis  
substituições de  
um não terminal  
em 'w'

## Arquivo glc/GramaticaLivreContexto.rb (4)

```
def calcularFormasSentenciaisDe( w, producoes )
  ladoEsquerdo, ladosDireitos = producoes
  formasSentenciais = []
  ladosDireitos.each do |ladoDireito|
    formaSentencial = w.sub(ladoEsquerdo, ladoDireito)
    if( formaSentencial != w )
      formasSentenciais |= [formaSentencial]
    end
  end
  return formasSentenciais
end
```

## Arquivo glc/CasoUso1.rb

```
glc1 = GramaticaLivreContexto.new()

glc1.adicionarProducao({ "E" => [ "T+E", "T" ] })
glc1.adicionarProducao({ "T" => [ "F*T", "F" ] })
glc1.adicionarProducao({ "F" => [ "(E)", "a" ] })


derivacoes = glc1.derivar( "E", 2 )
derivacoes.each do |sentencas|
  sentencas.each do |w,substituicoes|
    puts "#{w.inspect()}=>#{substituicoes.inspect()}"
  end
end
```

## Arquivo greibach/Greibach.rb

```
class Greibach
  def initialize()
    @glc = GramaticaGreibach.new()
  end
  def adicionarProducao( producao )
    @glc.adicionarProducao( producao )
  end

  def pertence?( w )
    @substituicoes = calcularSubstituicoes( w )
    puts @substituicoes.inspect()
    return true if( @substituicoes.include?(w) )
    return false
  end

  def calcularSubstituicoes( w )
    return @glc.calcularSubstituicoes( w )
  end
end
```



Assume que as  
produções estão  
na FNG



## Arquivo greibach/GramaticaGreibach.rb (1)

```
class GramaticaGreibach < GramaticaLivreContexto
  def calcularSubstituicoes( w )
    substituicoes = @producoes["S"]

    n = w.length()
    (1..(n-1)).each do |i|
      formasSeguintes = []
      substituicoes.each do |formaSentencial|
        formasDeW = aplicarProducoesEm( formaSentencial,
                                          formaSentencial[i,1] )
        formasSeguintes |= formasDeW
      end
      substituicoes = formasSeguintes
    end

    return substituicoes
  end
end
```

Obriga que o  
símbolo inicial  
seja "S"

## Arquivo greibach/GramaticaGreibach.rb (2)

```
def aplicarProducoesEm( w, ladoEsquerdo )
  return [] if( @producoes[ ladoEsquerdo ].nil? )
  ladosDireitos = @producoes[ ladoEsquerdo ]
  return calcularFormasSentenciaisDe( w,
                                       [ladoEsquerdo, ladosDireitos] )
end
end
```

## Arquivo greibach/Cenario.rb

```
require "greibach/Greibach"

g = Greibach.new()
g.adicionarProducao({ "S" => [ "aBC", "bBC" ] })
g.adicionarProducao({ "B" => [ "bB", "b" ] })
g.adicionarProducao({ "C" => [ "c" ] })

puts g.pertence?( "abbc" )

puts g.pertence?( "bcbc" )
```