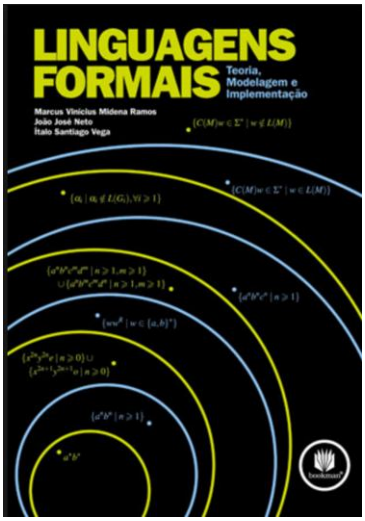
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
Linguagens Formais e Autômatos – Aula 09 – 1º SEMESTRE/2016
Prof. Luciano Silva

TEORIA: LINGUAGENS LIVRES DE CONTEXTO (I)

Nossos **objetivos** nesta aula são:

- conhecer o conceito de linguagens livres de contexto e gramáticas livres de contexto (GLC)
- praticar com construção de GLCs e árvores sintáticas



Para esta semana, usamos como referência as **Seções 4.1 (Gramáticas Livres de Contexto) até 4.4 (Ambigüidade)** do nosso livro da referência básica:

RAMOS, M.V.M., JOSÉ NETO, J., VEJA, I.S. **Linguagens Formais: Teoria, Modelagem e Implementação**. Porto Alegre: Bookman, 2009.

Não deixem de ler estas seções depois desta aula!

LINGUAGENS LIVRES DE CONTEXTO (GRAMÁTICAS LIVRES DE CONTEXTO)

- Uma **gramática livre de contexto (GLC)** é uma quádrupla G=(T,V,P,S) formada de:
  - um conjunto de símbolos terminais T
  - um conjunto de símbolos não-terminais V
  - um conjunto de produções (regras) $\alpha \rightarrow \beta$, onde $\alpha \in V$
  - um símbolo inicial $S \in V$

Exemplo: *T = {num,+}*
V = {expr}
P = { expr → expr + expr , expr → num }
S = expr

- Uma **derivação** em uma GLC é uma sequência de aplicações das regras de G a partir do símbolo inicial. Em uma derivação aplicamos uma única regra em cada passo da sequência.

  Exemplo*: expr* $\Rightarrow$ *expr + expr* $\Rightarrow$ *expr + num* $\Rightarrow$ *expr + expr + num* $\Rightarrow$ *expr + num + num* $\Rightarrow$
  $\Rightarrow$ *num + num + num*
  Notação: *expr* $\Rightarrow$** *num + num + num*

  Note que existem outras derivações capazes de gerar a palavra *num + num + num.* Por exemplo:
  expr $\Rightarrow$ *expr+expr* $\Rightarrow$ *num+expr* $\Rightarrow$ *num + expr + expr* $\Rightarrow$ *num + num + expr* $\Rightarrow$ *num +num +num*

- A **linguagem gerada por uma GLC G**, denotada por **L(G)**, é o conjunto de todas as palavras $\omega \in T^*$, tais que $S \Rightarrow^* \omega$.

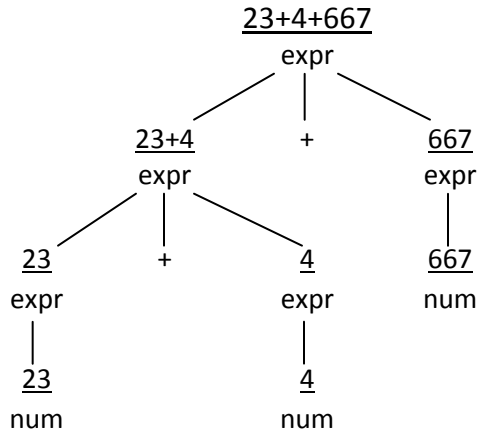
  Exemplo: *a palavra num + num é gerada pela gramática do primeiro exemplo, pois temos expr* $\Rightarrow$ *expr + expr* $\Rightarrow$ *expr + num* $\Rightarrow$ *num + num.*

- Uma linguagem $L \subseteq \Sigma^*$ é chamada de **linguagem livre de contexto** se existir uma GLC $G = (\Sigma, V, P, S)$ tal que L = L(G).

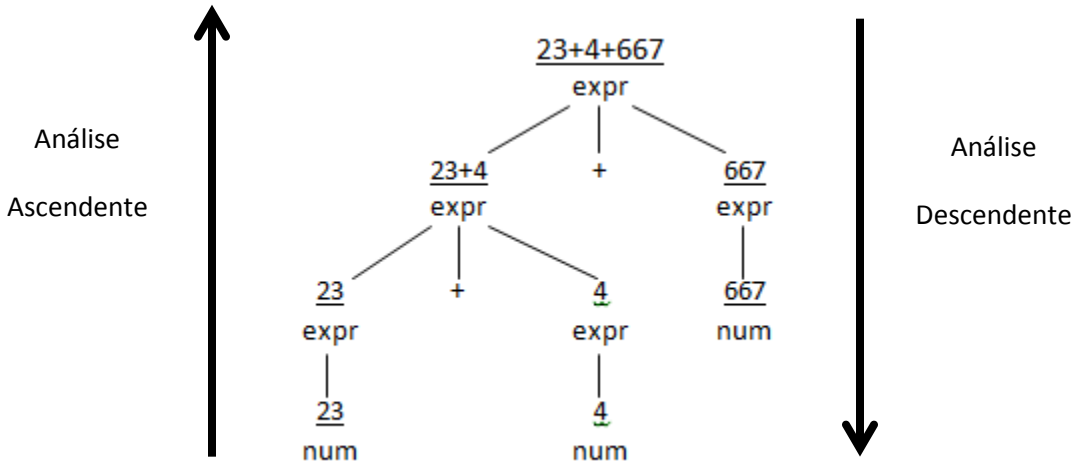
  Exemplo: *a linguagem L = { todas as expressões aritméticas contendo somente números e somas } é uma linguagem livre de contexto, pois basta considerar a gramática do primeiro exemplo como geradora desta linguagem L.*

- Uma derivação também pode ser representada através de uma **árvore sintática**.

  Exemplo: a palavra 23+4+667 pode ser reconhecida pela gramática do primeiro exemplo.



- O processo de geração da árvore sintática é chamado de **análise sintática** e pode ser realizada de duas maneiras: de cima para baixo (**análise descendente ou *top-down***) ou de baixo para cima (**análise ascendente ou *bottom-up***) .



EXERCÍCIO TUTORIADO

(a) Construa uma GLC para reconhecer a linguagem de todas as expressões aritméticas que envolvam somas e multiplicações.

(b) Construa uma árvore sintática relativa à derivação da palavra 23+5*6, utilizando a gramática do item (a).

(a) Construa uma GLC para reconhecer a linguagem de todas as expressões aritméticas que envolvam somas, subtrações, multiplicações, divisões e parênteses.

(b) Construa uma árvore sintática para representar a derivação da expressão 2+3*(5-6/8).

PROBLEMA

A principal aplicação de linguagens livres de contexto é na construção de analisadores sintáticos, uma das principais ferramentas dos processos de compilação e interpretação. Uma grande parte das linguagens de programação atuais (C, C++, Java, *Scheme*, PROLOG, dentre outras) possuem gramáticas livres de contexto como geradoras da linguagem. Para termos um exemplo disto, vamos considerar uma pequena gramática (escrita na notação BNF – *Backus-Naur Form*) para definição de variáveis e comandos de atribuição com expressões aritméticas:

<programa> ::= <declarações> <comandos>
<declarações> ::= <declaração> | <declaração> <declarações >
<declaração> ::= tipo id ;
<comandos> ::= <comando> | <comando> <comandos>
<comando> ::= <atribuição>
<atribuição> ::= id = <expr> ;
<expr> ::= <termo> + <termo> | <termo> - <termo> | <termo>
<termo> ::= <fator> * <fator> | <fator> / <fator> | <fator>
<fator> ::= num | (<expr>) | <expr>

A partir desta gramática, produza uma árvore sintática relativa ao processo de análise sintática do seguinte programa:

```
int x; int soma;
x=2+3;
soma = 3*8;
```

EXERCÍCIOS EXTRA-CLASSE

1. Altere a GLC do PROBLEMA para incluir expressões envolvendo variáveis.

   Exemplo: *soma = **x** + 3*(5+6);*

2. Altere a GLC do PROBLEMA para incluir chamadas de funções numéricas com um parâmetro:

   Exemplo: *soma = x + **seno(5)***3/(7+8);*

3. Uma gramática é chamada de **ambígua** se existir alguma palavra para a qual existirem duas derivações diferentes que gerem árvores sintáticas diferentes. Mostre que a gramática abaixo é ambígua:

   expr → expr + expr | expr * expr
   expr → num

   Para isto, você deve descobrir uma palavra ω tal que: seja possível obter uma derivação que gere ω e, a partir desta derivação, construir uma árvore sintática A_1 e, também, obter uma outra derivação que gere ω e, a partir desta nova derivação, construir uma árvore sintática A_2 que seja diferente de A_1.

4. Altere a gramática da questão (3) para remover a ambiguidade.