INE5416
Paradigmas de Programação

Ricardo Azambuja Silveira
INE-CTC-UFSC
E-Mail: silveira@inf.ufsc.br
URL: www.inf.ufsc.br/~silveira

Conceitos

- ***Léxica- estudo dos símbolos que compõem uma linguagem***
- ***Sintaxe*** **- a forma ou estrutura das expressões, instruções e unidades de programas.**
- ***Semântica*** **– o significado das expressões, instruções e unidades de programas.**

- **Quem deve usar uma definição de linguagem?**
  - **Outros projetistas**
  - **Implementadores**
  - **Programadores (os usuários da linguagem)**

- **Uma *sentença* é uma cadeia de caracteres sobre algum alfabeto**
- **Uma *linguagem* é um conjunto de sentenças**
- **Um *lexema* é a unidade de mais baixo nível sintáticos de uma linguagem. Incluem:**
  - **Identificadores**
  - **Literais**
  - **Operadores**
  - **Palavras reservadas**
- **Um *token* (símbolo)n é uma categoria de lexemas (e.g., identifier)**

Abordagens formais para descrever LPs

- **Sintaxe:**
  - **Reconhecedores – (máquinas de estados) usada em compiladores**
  - **Geradores – formalismo usado para gerar sentenças na linguagem**
    - **Gramáticas**
    - **Expressões**

Gramática livre de contexto

- Desenvolvida por Noam Chomsky em meados dos anos 50 como um gerador de linguagens com o propósito de descrever a sintaxe das linguagens naturais
- Define uma classe de linguagens denominada *linguagens livres de contexto*
- Backus Naur Form - BNF
  - Metalinguagem inventada em 1959 por John Backus para descrever a linguagem Algol 58 e aperfeiçoada por Peter Naur em 1960
  - Uma *metalinguagem* é uma linguagem usada para descrever outra linguagem.
  - A BNF é equivalente a gramática livre de contexto
  - Em BNF, *abstrações* são usadas para representar classes de estruturas sintáticas, na forma

    <abstração> -> descrição da abstração
  - Que funcionam como variáveis sintáticas (também chamadas *símbolos não-terminais*) que derivam dos lexemas (também chamadas *símbolos terminais*)
  - Exemplos:
    - <atribuição> -> <variável> = <expressão>
    - Isto é uma regra, que descreve a estrutura de um comando de atribuição

BNF

- Uma regra tem um lado esquerdo (left-hand side - LHS) e um lado direito (right-hand side - RHS), e consiste em símbolos *terminais* e *não-terminais*
- Uma *gramática* é um conjunto finito e não vazio de regras
- Uma abstração (ou símbolo não-terminal) pode ter mais que um RHS

  <stmt> -> <single_stmt>

  | begin <stmt_list> end
- Uma lista sintatica é descrita em BNF usando recursão

  <ident_list> -> ident

  | ident, <ident_list>
- uma *derivação* é a aplicação repetida de regras, a partir do símbolo de início e terminando com uma sentença formada apenas com símbolos terminais

Um exemplo de gramática

<program> -> begin <stmts> end

<stmts> -> <stmt> | <stmt> ; <stmts>

<stmt> -> <var> = <expr>

<var> -> a | b | c | d

<expr> -> <term> + <term> | <term> - <term>

<term> -> <var> | const

Um exemplo de derivação

<program> => <stmts> => <stmt>

=> <var> = <expr> => a = <expr>

=> a = <term> + <term>

=> a = <var> + <term>

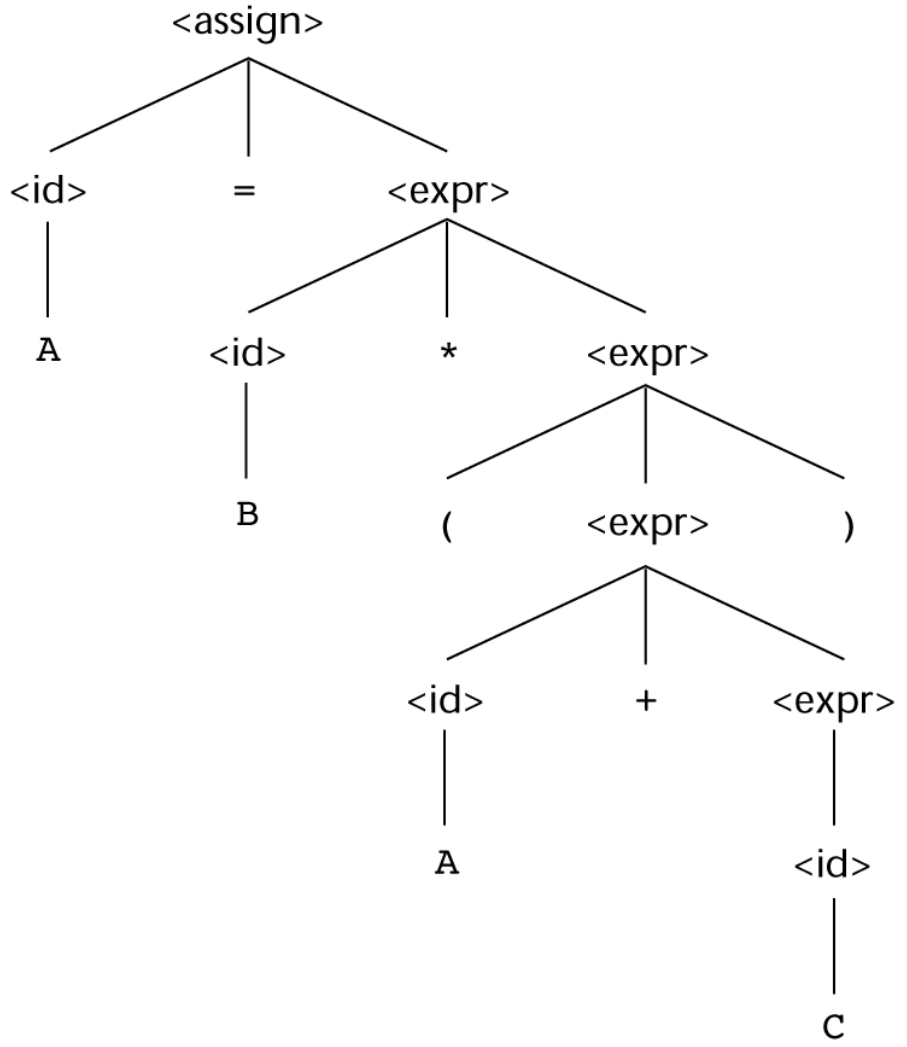
=> a = b + <term>

=> a = b + const

- Cada cadeia de símbolos na derivação é uma *forma sentencial*
- Uma *sentença* é a forma sentencial que tem apenas símbolos terminais
- uma *derivação a esquerda* é aquela em que o símbolo não-terminal mais a esquerda em cada forma sentencial é escolhida para expansão
- Uma derivação pode ser mais a esquerda, mais a direita ou mixta

Árvores de análise

- Uma árvore de análise (parse tree) é uma representação hierárquica de uma derivação
-



BNF extendida (EBNF)

Serve apenas para abreviar a notação da BNF

- **Partes opcionais são colocadas entre colchetes ([])**

```
<proc_call> -> ident [ ( <expr_list>)]
```

- **Partes alternativas das RHSs entre parênteses e separadas por barras verticais**

```
<term> -> <term> (+ | -) const
```

- **Repetições (0 or mais) entre chaves ({})**

```
<ident> -> letter {letter | digit}
```

BNF:

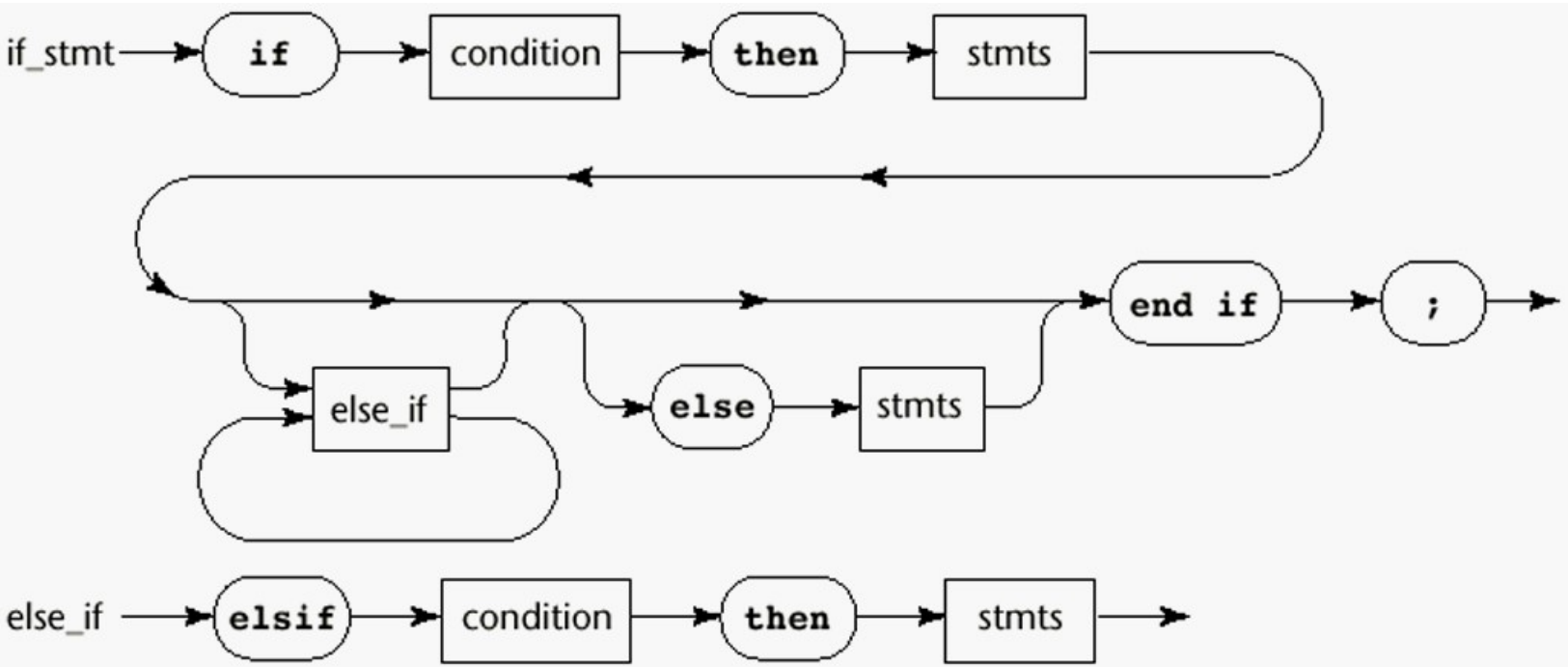
```
<expr> -> <expr> + <term>
        | <expr> - <term>
        | <term>
<term> -> <term> * <factor>
        | <term> / <factor>
        | <factor>
```

EBNF:

```
<expr> -> <termo> {(+ | -) <term>}
<term> -> <fator> {(* | /) <factor>}
```

Grafos de sintaxe

Os grafos de sintaxe e a descrição EBNF do comando if



<if_stmt> ⟶ **if** <condition> **then** <stmts> {<else_if>}

[**else** <stmts>] **end if**

<else_if> ⟶ **elsif** <condition> **then** <stmts>

Análise semântica

- Semântica estática
  - Verificação em tempo de compilação
    - Verificação de tipos de variáveis em expressões
    - Verificação de escôpo de variáveis
    - Verificação de parâmetros
- Semântica dinâmica
  - Especificação formal do significado das construções linguísticas

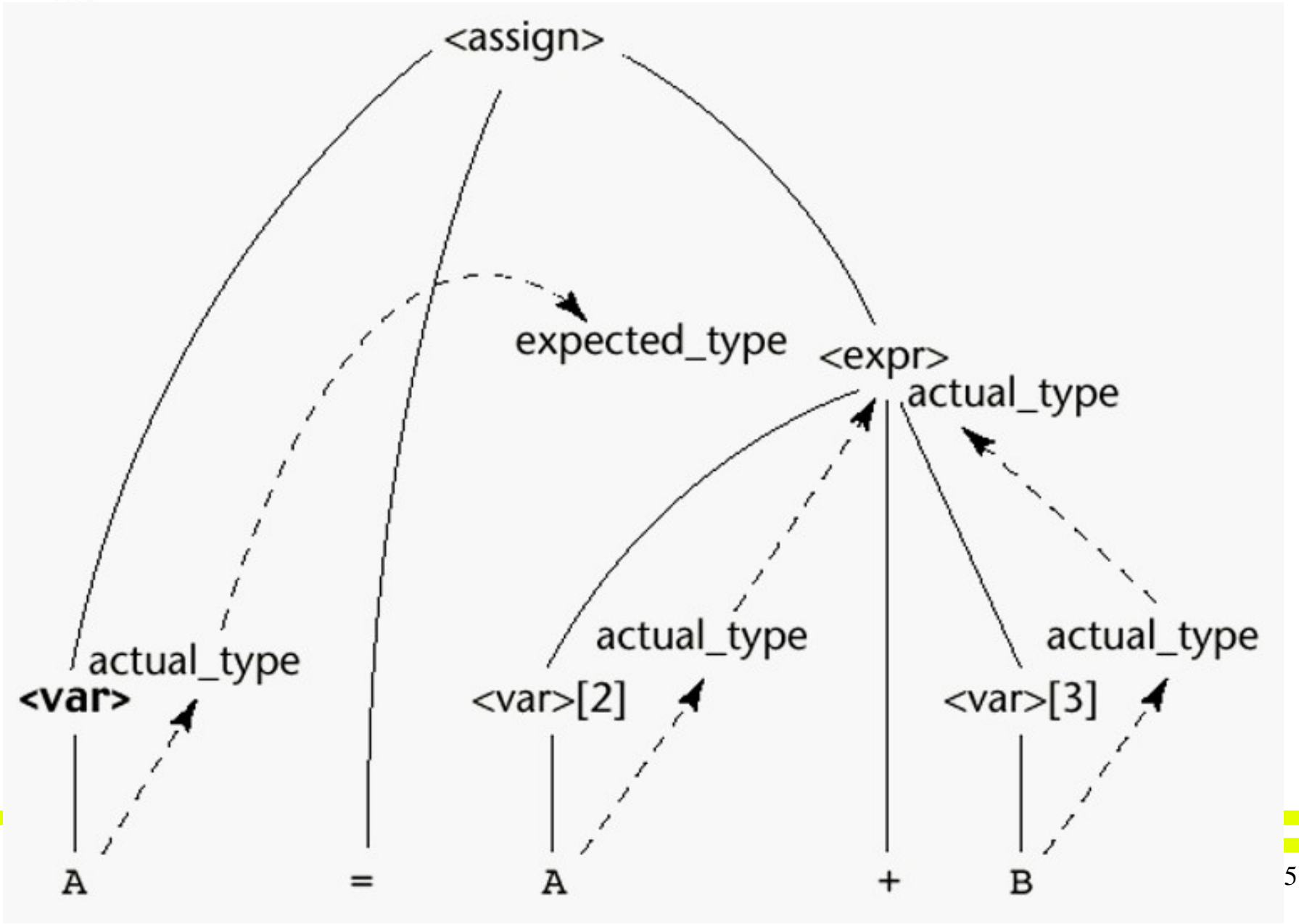
Gramática de atributos

- Desenvolvida por Knuth, 1968
- Gramáticas livre de contexto não tem capacidade para descrever completamente a sintática de linguagens de programação
- Mecanismos adicionados a GLC pata tratar algumas informações semânticas relacionadas as formas legais do programa na construção das árvores de análise
- Valor primário da gramática de atributos:
  - Especificação da semântica estática
  - Projeto de compiladores (verificação da semântica estática)
- Definição:
- Uma *gramática de atributo* é a gramática livre de contexto com as seguintes adições:
  - Para cada símbolo gramatical x há um conjunto A(x) de atributos
  - Cada regra tem um conjunto de funções que definem certos atributos dos símbolos não-terminais em uma regra
  - Cada regre tem um conjunto (possivelmente vazio) de predicados para checar a consistência dos atributos

Gramática de atributos

- *Como os valores dos atributos são computados?*
- 1. Se todos os atributos foram herdados, a árvore é decorada em ordem top-down.
- 2. Se todos os atributos foram sintetizados, a árvore é decorada em ordem bottom-up.
- 3. Em muitos casos, os dois tipos de atributos são usados e uma combinação de top-down e bottom-up é usada.

Gramática de atributos



Semântica dinâmica

- Denota o significado das expressões, das instruções e das unidades de programas
- Nenhuma notação ou formalismo simples para descrição semântica é aceito largamente para descrever a semântica dinâmica das LPs
- Utilidade:
  - Conhecimento da linguagem pelos programadores
  - Construção de compiladores
  - Geração automática de compiladores
  - Prova de exatidão de programas
- Principais métodos:
  - Semântica operacional- baseada na representação por algoritmos
  - Semântica axiomática – baseada em notação lógica formal
  - Semântica denotacional – baseada em notação funcional