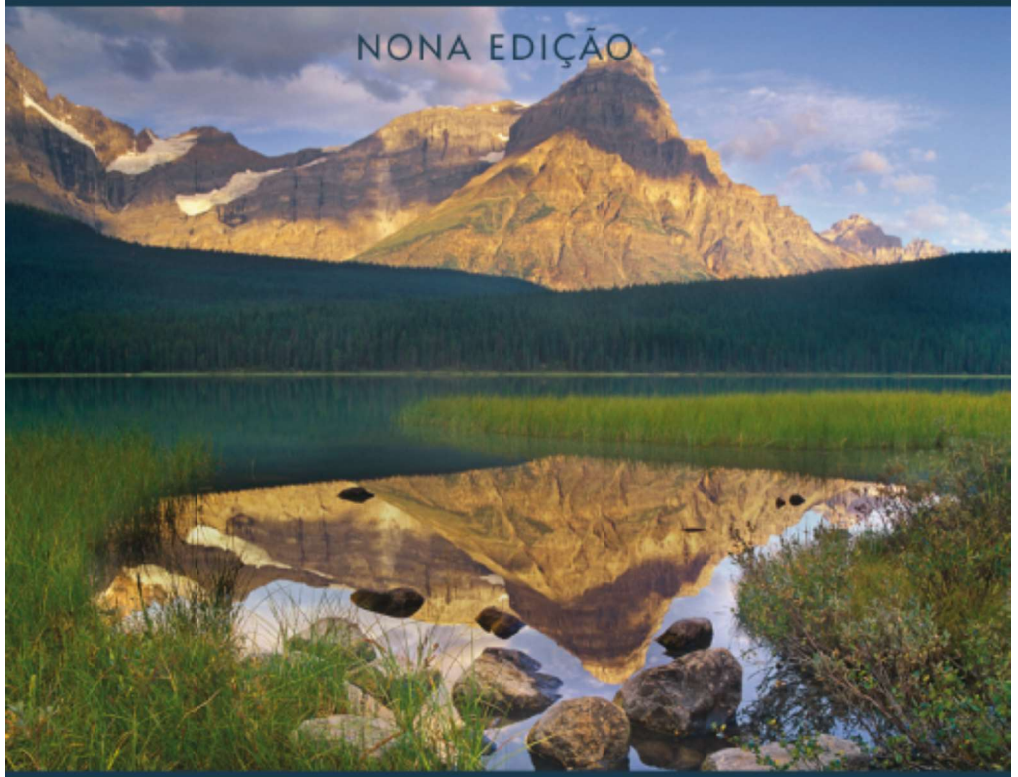
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

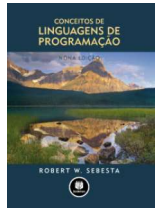


ROBERT W. SEBESTA

bookman

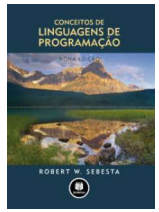
Capítulo 3

Descrevendo Sintaxe e Semântica



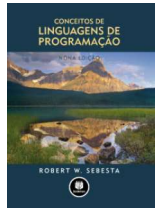
Tópicos do Capítulo 3

- Introdução
- O problema geral de descrever sintaxe
- Métodos formais para descrever sintaxe
- Gramáticas de atributos
- Descrevendo o significado de programas: semântica dinâmica



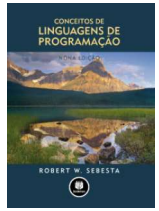
Introdução

- **Sintaxe:** a forma de suas expressões, sentenças e unidades de programas
- **Semântica:** o significado dessas expressões, sentenças e unidades de programas
- Sintaxe e semântica fornecem uma definição da linguagem
  - Usuários de uma definição de linguagem
    - Outros desenvolvedores de linguagem
    - Implementadores
    - Programadores (os usuários da linguagem)



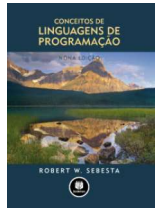
O problema geral de descrever sintaxe: terminologia

- Uma *sentença* é uma cadeia de caracteres formada a partir do alfabeto da linguagem
- Uma *linguagem* é uma cadeia de sentenças
- Um *lexema* é a unidade sintática de mais baixo nível de uma linguagem (por exemplo, `*`, `sum`, `begin`)
- Um *token* é uma categoria de lexemas (por exemplo, identificador)



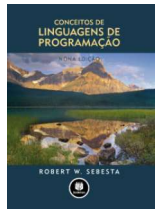
Definição formal de linguagens

- **Reconhecedores**
  - Um dispositivo de reconhecimento lê cadeias de caracteres a partir do alfabeto da linguagem e indica se a cadeia pertence ou não à linguagem
  - Exemplo: parte de análise sintática de um compilador
    - A discussão detalhada sobre análise sintática aparece no Capítulo 4 e será abordada em Compiladores
- **Geradores**
  - Um dispositivo que gera sentenças de uma linguagem
  - Pode determinar se a sintaxe de uma sentença em particular está sintaticamente correta a comparando à estrutura do gerador



BNF e gramáticas livres de contexto

- Gramáticas livres de contexto
  - Desenvolvido por Noam Chomsky no meio dos anos 1950
  - Geradores de linguagem, feitos para descrever a sintaxe de linguagens naturais
  - Define uma classe de linguagens chamadas de livres de contexto
- Forma de *Backus-Naur* (1959)
  - Inventada por John Backus para descriver Algol 58
  - BNF é equivalente às gramáticas livres de contexto

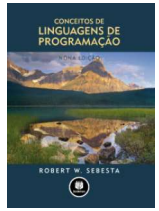


Fundamentos de BNF

- Em BNF, abstrações são usadas para representar classes de estruturas sintáticas – elas agem como variáveis sintáticas (também chamadas de símbolos não terminais, ou simplesmente não terminais)
- Terminais são lexemas ou tokens
- Uma regra tem um lado esquerdo (LHS), que é um não terminal, e um lado direito (RHS), que é uma cadeia de terminais e não terminais
  - Exemplos das regras de BNF:

    `<ident_list> → identifier | identifier, <ident_list>`

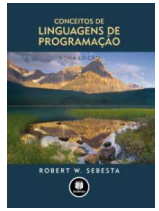
    `<if_stmt> → if <logic_expr> then <stmt>`
- Gramática: coleção de regras
- Um símbolo inicial é um elemento especial dos não terminais de uma gramática



Regras de BNF

- Uma abstração (ou símbolo não terminal) pode ter mais de um RHS

```
<stmt> → <single_stmt>
        | begin <stmt_list> end
```

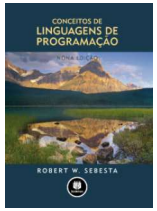


Descrevendo listas

- Listas sintáticas são descritas usando recursão

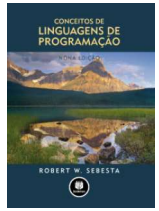
```
<ident_list> → ident
             | ident, <ident_list>
```

- Uma derivação é uma aplicação de regras repetida, começando com um símbolo inicial e terminando com uma sentença (todos símbolos terminais)



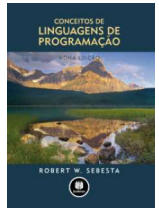
Um exemplo de gramática

```
<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | const
```



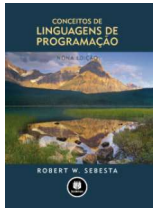
Um exemplo de derivação

```
<program> => <stmts>  => <stmt>
                      => <var> = <expr>
                      => a = <expr>
                      => a = <term> + <term>
                      => a = <var> + <term>
                      => a = b + <term>
                      => a = b + const
```



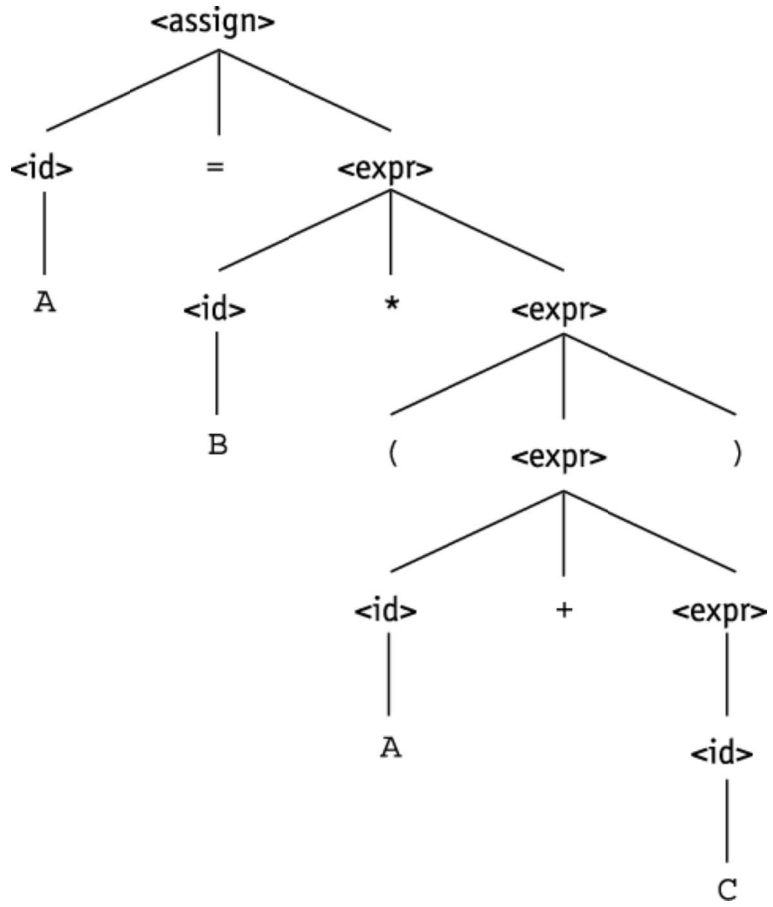
Derivações

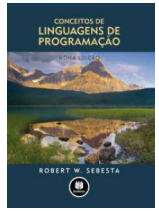
- Cada cadeia de símbolos em uma derivação é chamada de *forma sentencial*
- Uma *sentença* é uma forma sentencial que tem apenas símbolos terminais
- Uma *derivação mais à esquerda* é uma em que o não terminal mais à esquerda em cada forma sentencial é expandido
- Uma derivação pode ser nem mais à esquerda nem mais à direita



Árvore de análise sintática

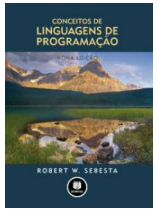
- Representação hierárquica de uma derivação





Ambiguidade em gramáticas

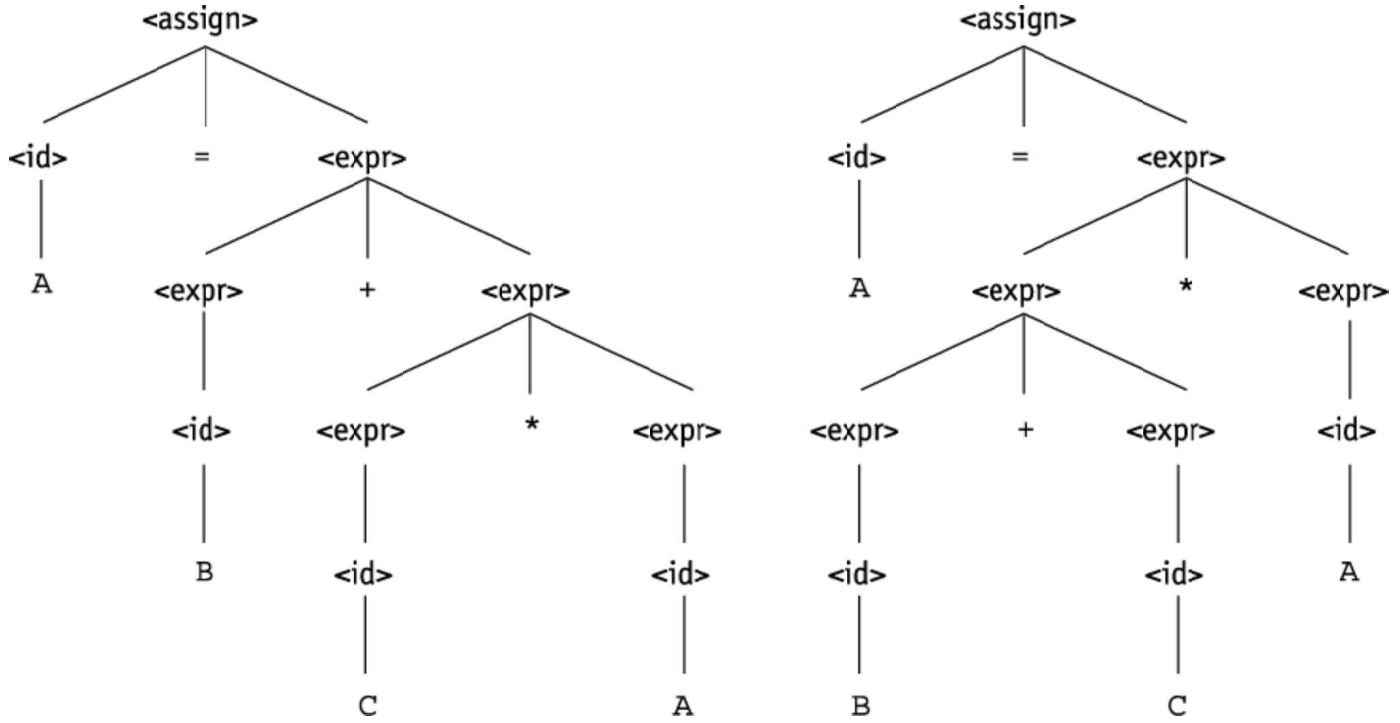
- Uma gramática é *ambígua* se gera uma forma sentencial com duas ou mais árvores de análise sintática distintas

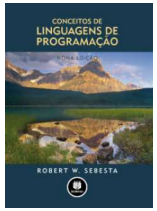


Uma gramática de expressão ambígua

<expr> → <expr> <op> <expr> | const

<op> → / | -

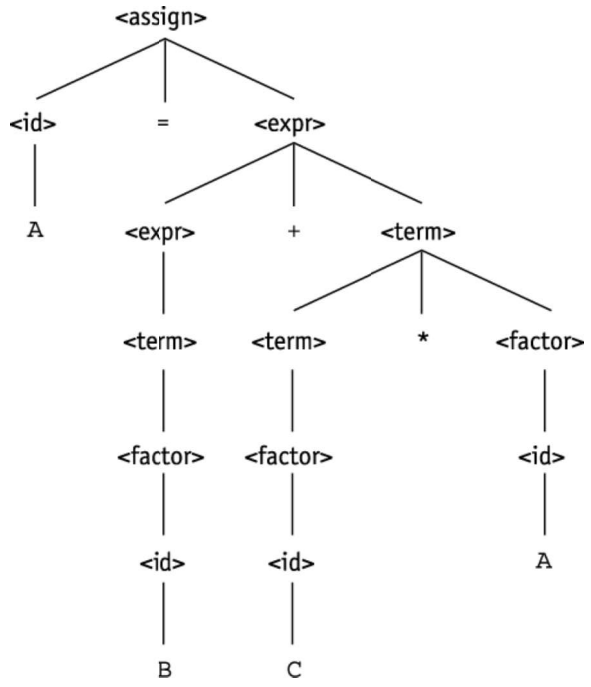


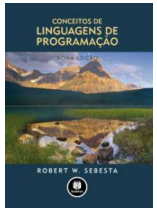


Uma gramática de expressão não ambígua

- Se usarmos a árvore de análise sintática para indicar níveis de precedência dos operadores, não podemos ter a ambiguidade

```
<expr> → <expr> - <term>  |  <term>
<term> → <term> / const| const
```





Associatividade de operadores

- Associatividade de um operador também pode ser indicada por uma gramática

```
<expr> -> <expr> + <expr> |  const  (ambígua)
<expr> -> <expr> + const  |  const  (não ambígua)
```