FACAMP
FACULDADES DE CAMPINAS

Linguagens de Programação

Aula 3

Sintaxe e semântica. Linguagens e reconhecedores. Métodos formais.

2º semestre de 2019
Prof José Martins Junior

Sintaxe e semântica

- Sintaxe
  - Descreve a forma ou estrutura de expressões, comandos e unidades de programa
  - Conjunto de regras que determinam quais construções são corretas
- Semântica
  - Descreve o significado das expressões, comandos e unidades de programa
  - Como as construções da linguagem devem ser interpretadas e executadas
- Exemplo: IF na linguagem C:
  - Sintaxe: `if (<expressão>) <instrução>`
  - Semântica: se o valor atual da expressão for verdadeiro, a instrução incorporada será selecionada para execução

Exemplo: programa C com erros

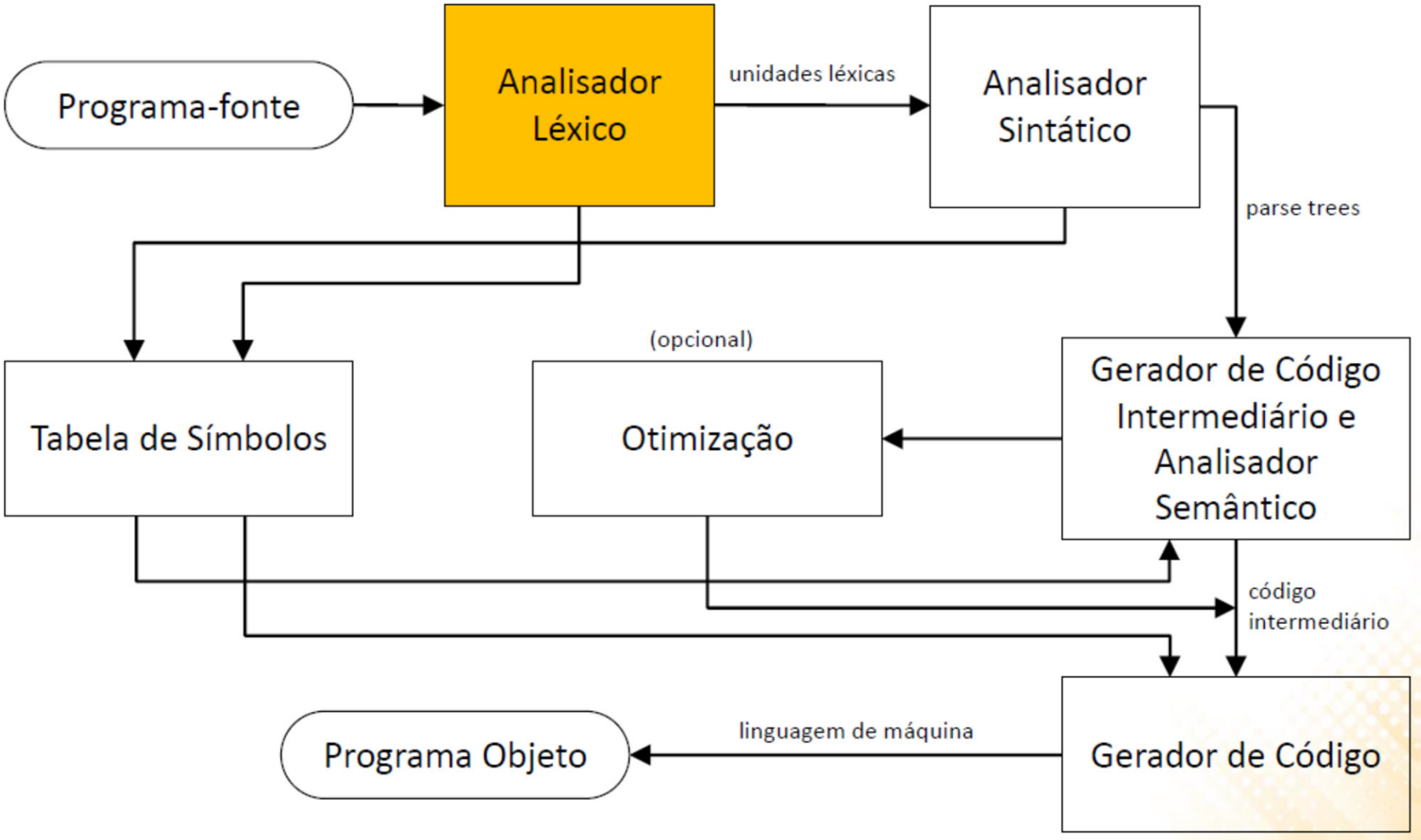
```
int j=0, conta, V[10];
float i@;
conta = '0'
for (j=0, j<10; j++
{
  V[j] = conta++;
}
```

Fases de análise da compilação

- Análise Léxica
  - Considera as unidades léxicas (tokens) do programa
  - `int, j, =, 0, conta, [, for, (, <, int, ++, {`
  - Erro: `i@` (caractere especial em nome de variável)
- Análise Sintática
  - Faz a combinação de tokens que formam o programa
  - comando_for → for (expr1; expr2; expr3) {comandos}
  - Erros: `;` `for(j=0, ...` `)`
- Análise Semântica
  - Verifica a adequação do uso
  - Tipos semelhantes em comandos (atribuição x igualdade, por exemplo), uso de identificadores declarados, entre outros
  - Erro: conta = **'0'** (atribuirá valor 48, em vez de 0)

Processo geral de compilação

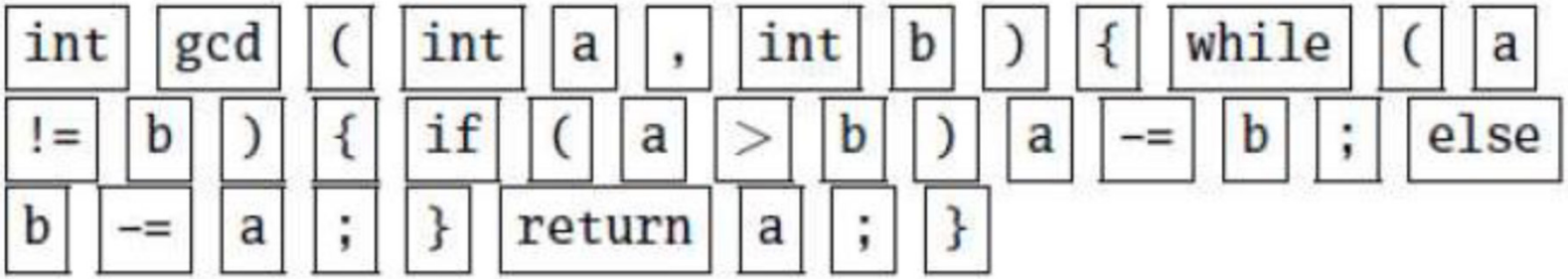
- Primeira etapa, o analisador léxico



Analisador léxico

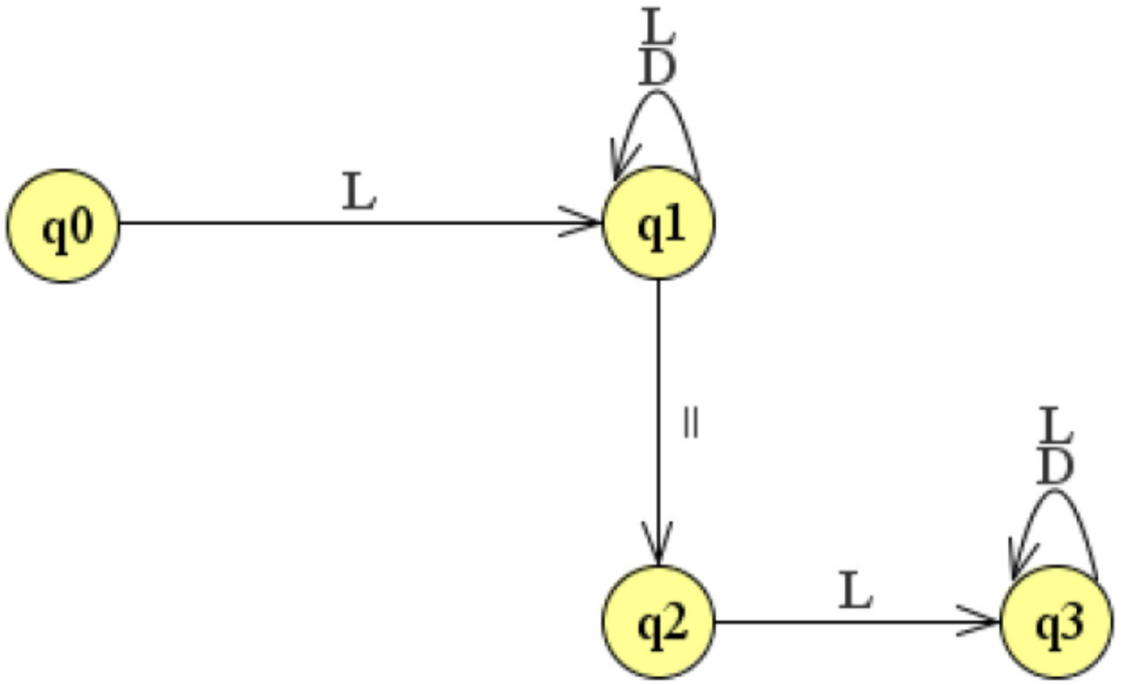
- Divide o código fonte em partes (tokens) ou átomos
  - Pode ser feita através de autômatos finitos ou expressões regulares
    - Um autômato finito é uma máquina de estados finitos formada por um conjunto de estados (um estado inicial e um ou mais estados finais)

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```



Exemplo de autômato finito

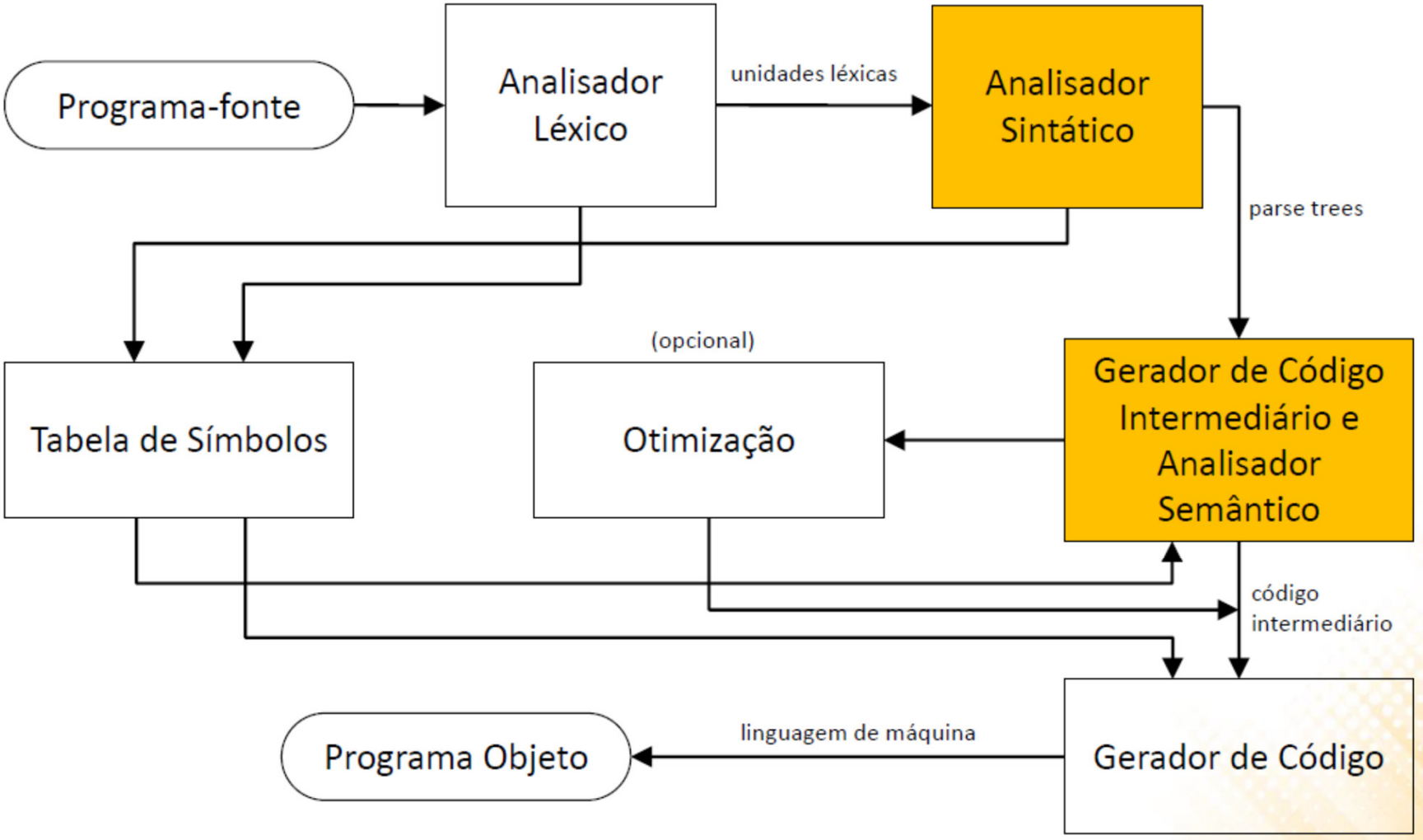
- Exemplo de AF para reconhecer atribuições entre variáveis



L → a..z
D → 0..9
= → atribuição

Processo geral de compilação

- Próximas etapas, os analisadores sintático e semântico



Descrição da sintaxe

- Linguagens (naturais ou artificiais) são conjuntos de sequências de caracteres de algum alfabeto, onde
  - Uma sentença é uma sequência de caracteres sobre um alfabeto
  - Uma linguagem é um conjunto de sentenças
  - Um lexema é a unidade sintática de menor nível em uma linguagem (exemplo: *, sum, begin)
  - Um token é uma categoria de lexemas (exemplo: identificador, números, caracteres, etc.)
- Sintaticamente, um programa é uma sequência de lexemas

Exemplo de Sintaxe

```
index = 2 * count + 17;
```

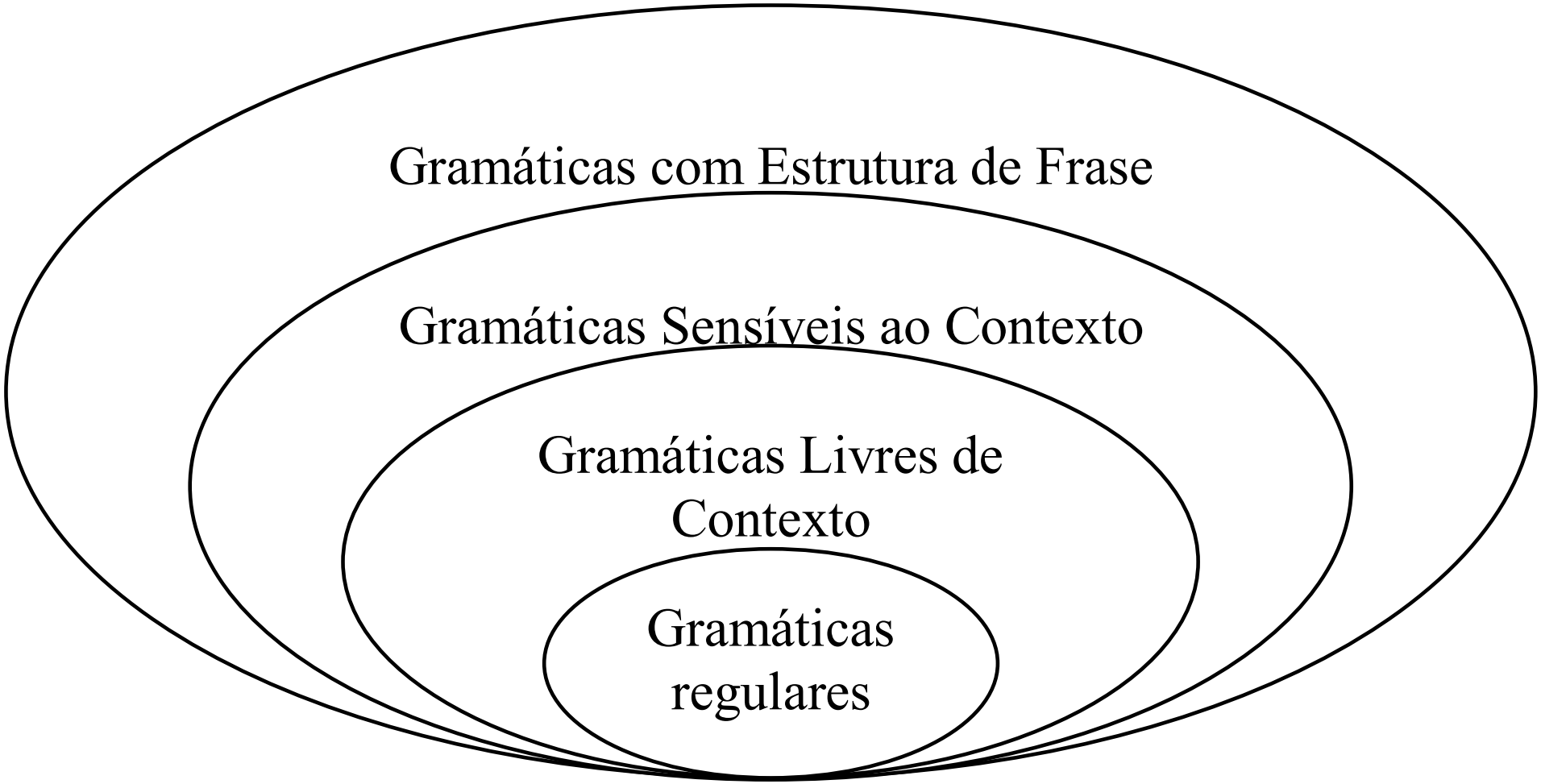
Lexemas	**Tokens**
index	identificador
=	sinal_atribuicao
2	int_literal
*	mult_op
cont	identificador
+	soma_op
17	int_literal
;	ponto_e_virgula

Métodos formais

- Reconhecedores
  - Dispositivo que lê uma cadeia de entrada de uma linguagem e decide se esta pertence ou não à linguagem
- Geradores
  - Dispositivo que gera uma sentença da linguagem sempre que acionado
- Sintaxe
  - Definida formalmente através de uma gramática
- Gramática
  - Conjunto de definições que especificam uma sequência válida de caracteres
- Duas classes de gramáticas aplicáveis a LPs
  - Gramáticas livres de contexto
  - Gramáticas regulares

Hierarquia de Chomsky

- Quanto mais externa, mais irrestrita e ambígua



Forma de Backus-Naur (BNF)

- Criada por John Backus (1959) para descrever o Algol 58
  - Modificada por Peter Naur para descrever o Algol 60
  - Considerada uma gramática livre de contexto
  - É capaz de descrever maioria das sintaxes de LPs
- Formato geral de uma abstração

  `<atribuição>` → `<var> = <expressão>`

  LHS RHS
- É definida através de uma regra ou produção formada por
  - Lado esquerdo (LHS)
    - Abstração a ser definida (símbolo não-terminal)
  - Lado direito (RHS)
    - Definição da abstração, composta por símbolos, lexemas e referências a outras abstrações
  - Símbolos e lexemas são denominados símbolos terminais.

Forma de Backus-Naur (BNF)

- < > indica um não-terminal (termo que precisa ser expandido)
- Símbolos não cercados por < > são terminais;
  - São representativos. Exs.: if, while, (, =
  - O símbolo → significa “é definido como”
  - Símbolos cercados por {} indicam que o termo pode ser repetido n vezes (ou nenhuma)
  - O símbolo | significa or e é usado para separar alternativas
- BNF é um dispositivo gerativo para definir linguagens
  - Sentenças da linguagem são geradas através de sequências de aplicações das regras
  - Iniciam por um símbolo não terminal chamado de início
  - Uma geração de sentença é denominada derivação

Exemplo de gramática

```
<programa> → begin <lista_inst> end
<lista_inst> → <inst> ; <lista_inst>
                | <inst>
<inst> → <var> = <expressao>
<var> → A | B | C
<expressao> →  <var> + <var>
                | <var> - <var>
                | <var>
```

Exemplo de derivação à esquerda

```
<programa>    => begin <lista_inst> end
              => begin <inst> ; <lista_inst> end
              => begin <var> = <expressão> ; <lista_inst> end
              => begin A = <expressão> ; <lista_inst> end
              => begin A = <var> + <var> ; <lista_inst> end
              => begin A = B + <var> ; <lista_inst> end
              => begin A = B + C ; <lista_inst> end
              => begin A = B + C ; <inst> end
              => begin A = B + C ; <var> = <expressão> end
              => begin A = B + C ; B = <expressão> end
              => begin A = B + C ; B = <var> end
              => begin A = B + C ; B = C end
```

Exemplos BNF

- Crie uma gramática BNF para realizar a análise sintática de
  1. Números binários inteiros como 01100, 111 ou 10101
  2. Uma lista simples da forma A1,B2,A4,C3
  3. Uma expressão limitada a identificadores x,y ou z, operações de adição (+) e subtração (-) e com a possibilidade de parênteses. Exemplo: x+(z-y)

Exemplos BNF

- Crie uma gramática BNF para realizar a análise sintática de
  1. Números binários inteiros como 01100, 111 ou 10101

     ```
     <integer> ::= <digit><integer> | <digit>
     <digit> ::= 0 | 1
     ```

  2. Uma lista simples da forma A1,B2,A4,C3

     ```
     <list> ::= <element>,<list> | <element>
     <element> ::= <letter><digit>
     <letter> ::= A | B | C
     <digit> ::= 1|2|3|4
     ```

  3. Uma expressão limitada a identificadores x,y ou z, operações de adição (+) e subtração (-) e com a possibilidade de parênteses. Exemplo: x+(z-y)

     ```
     <expr> ::= <factor> <opr> <expr> | <factor>
     <factor> ::= x | y | z | <parexp>
     <parexp> ::= (<expr>)
     <opr> ::= +|-
     ```

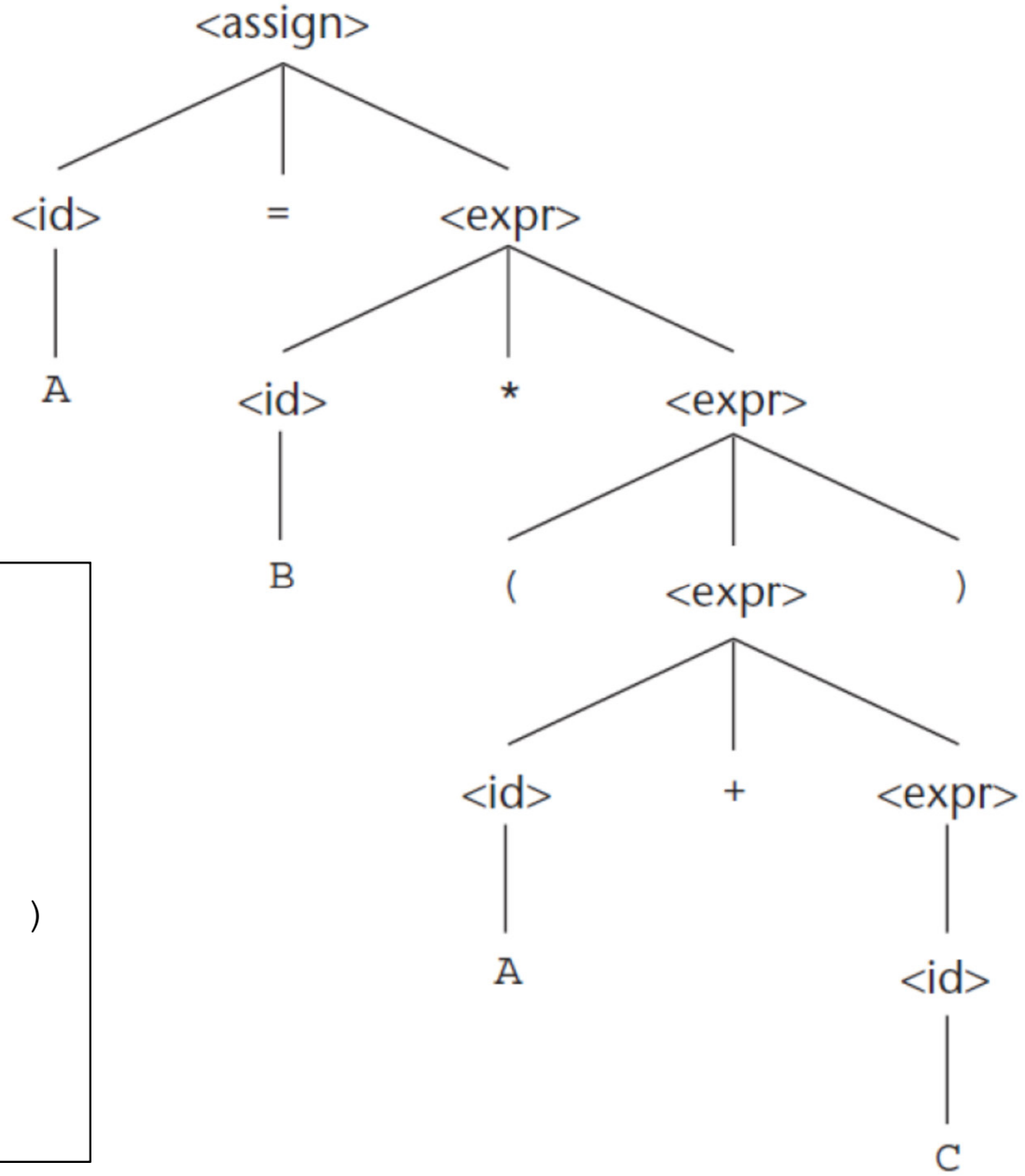
Parse Trees

- Representação em forma de árvore da derivação
  - Todo nó interno da árvore é um símbolo não-terminal
  - Toda folha é rotulada com um símbolo terminal
  - Toda sub-árvore descreve uma instância de uma abstração na sentença
- Usada para gerar código de máquina
  - Compilador explora a árvore e gera código conforme reconhece sentenças
  - Busca em profundidade

Parse Trees – um exemplo

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
        | <id> * <expr>
        | ( <expr> )
        | <id>
```

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
         => A = B * ( <id> + <expr> )
         => A = B * ( A + <expr> )
         => A = B * ( A + <id> )
         => A = B * ( A + C )
```



Árvores sintáticas e semânticas

- Após a análise, os detalhes de derivação não são necessários para as próximas fases do processo de compilação
- O Analisador Semântico remove as produções intermediárias para criar uma árvore sintática abstrata (abstract syntax tree)
  - Operadores são avaliados quando uma sentença é reconhecida
  - As partes mais baixas da parse tree são completadas primeiro
  - Assim um operador gerado mais baixo em uma parse tree é avaliado primeiro, ou seja, tem **precedência** sobre os produzidos acima

Bibliografia

- SEBESTA, R. W. Conceitos de Linguagens de Programação. Porto Alegre: Bookman, 2011. 792p.
- Notas de aulas da disciplina “IPRJ - Conceitos de Linguagens de Programação”, Prof. Edirlei Soares de Lima, PUC-RJ.