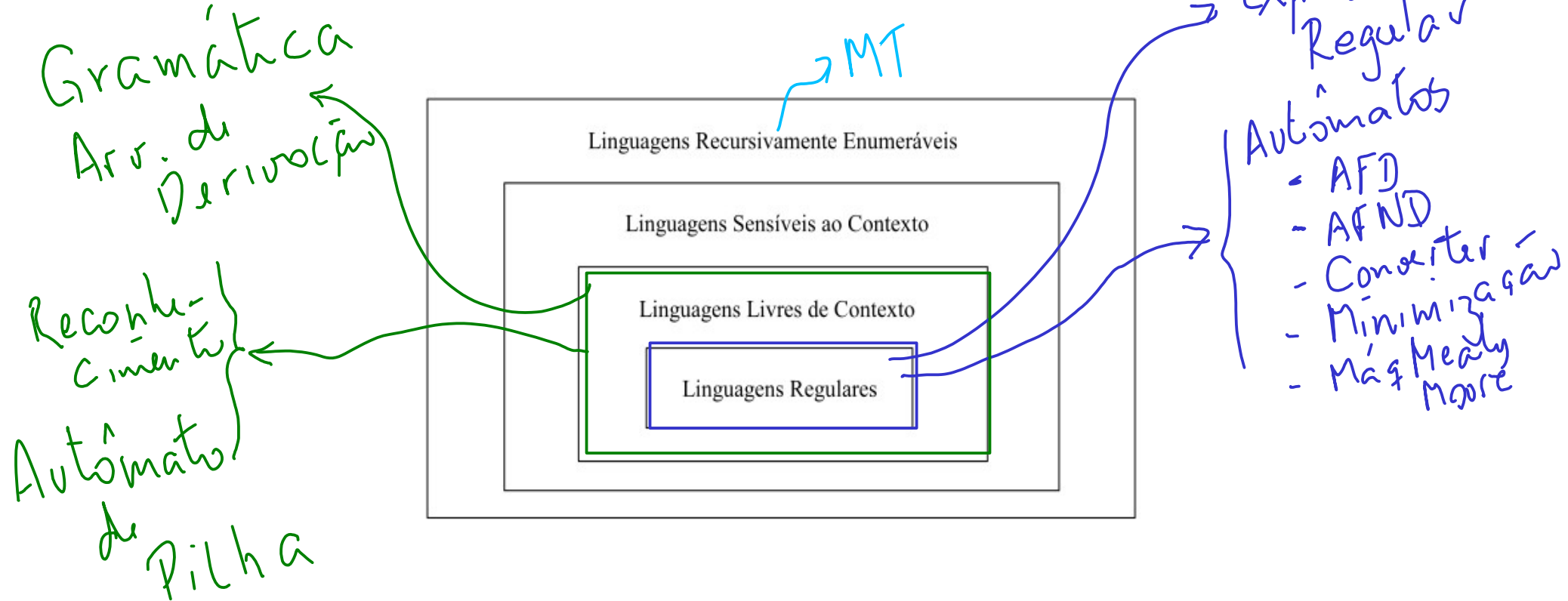


# **Linguagens Livres de Contexto**

# Introdução

- Hierarquia de Chomsky



# Linguagens Livres de Contexto

- Formalização sintática das linguagens de programação de alto nível
- Representação de construções aninhadas
  - na construção de expressões aritméticas
  - na estruturação do fluxo de controle
  - na estruturação do programa

Ordem

if ✓  
for ✓  
while ✓

if for x = while  
✓ ✓ ✓ ✓ ✓

Léxica = OK  
Sintática = Erro

if ( Expr ) { corpo }

# Definição

- Quádrupla  $(V, \Sigma, P, S)$  com os seguintes componentes
  - $V$ : conjunto (finito e não-vazio) dos símbolos terminais e não-terminais
  - $\Sigma$ : conjunto (finito e não-vazio) dos símbolos terminais; corresponde ao alfabeto da linguagem definida pela gramática
  - $P$ : conjunto (finito e não-vazio) das regras de produção, todas no formato  $\alpha \rightarrow \beta$ , com  $\alpha \in (V - \Sigma)$  e  $\beta \in V^*$ ;
  - $S$ : raiz da gramática,  $S \in (V - \Sigma)$

⇒  
GRAMÁTICA

# Árvores de Derivação

- Melhor visualização da estrutura das sentenças da linguagem
  - facilitando a análise das mesmas
- Facilita a representação interna
  - nos compiladores e interpretadores

GRAMÁTICA  $\xRightarrow{+ \text{ cadeia}}$  Árvore de Derivação

Não Terminal =  $\{E, T, F\}$

## Árvores de Derivação

$a * (a + a)$

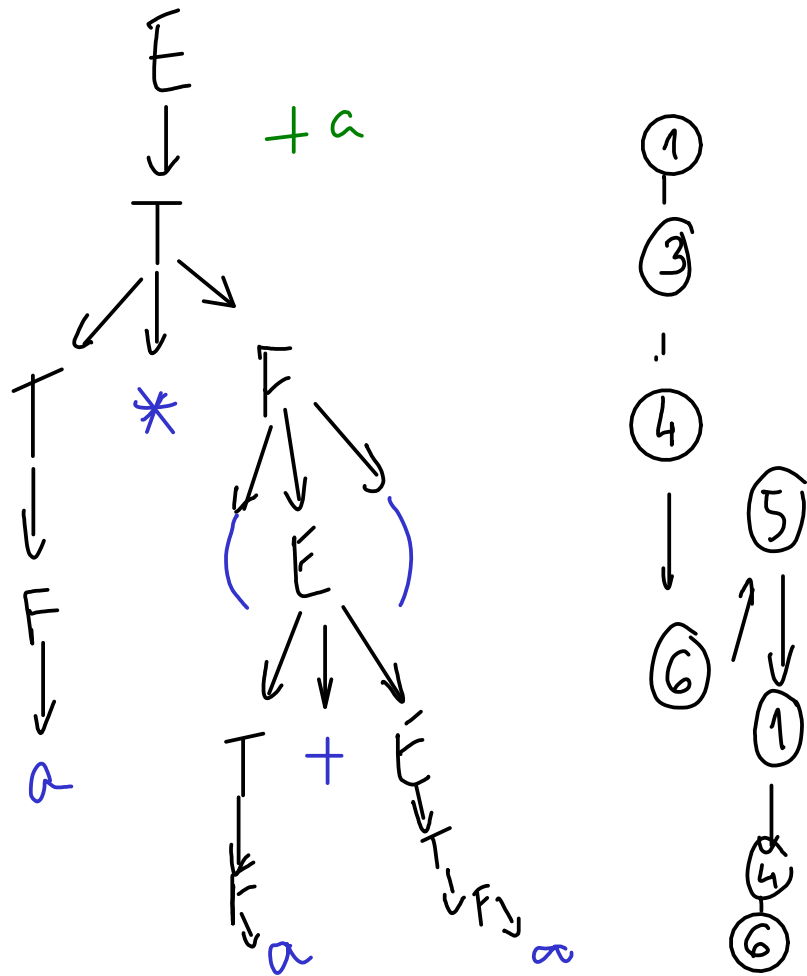
Terminais =  $\{+, *, (, ), a\}$

- Exemplo:

- Sentença:  $a * (a + a)$

- 1 -  $\{E\} \rightarrow \boxed{T + E,}$   
2 -  $E \rightarrow T,$   
3 -  $T \rightarrow F * T,$   
4 -  $T \rightarrow F,$   
5 -  $F \rightarrow (E),$   
6 -  $F \rightarrow a\}$

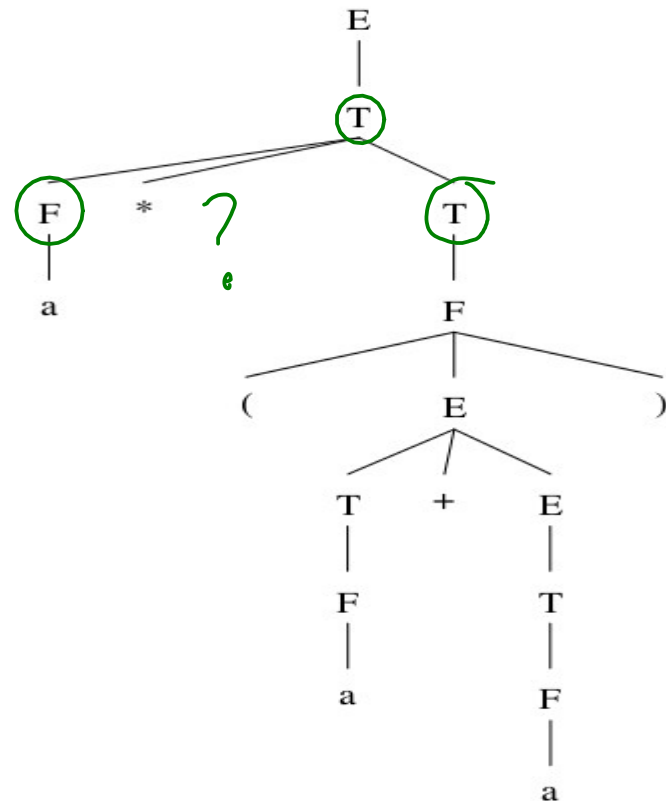
GRAMÁTICA



# Árvores de Derivação

- Exemplo:
  - Sentença:  $a * (a + a)$

$$\begin{aligned} \{ & E \rightarrow T + E, \\ & E \rightarrow T, \\ & T \rightarrow F * T, \\ & T \rightarrow F, \\ & F \rightarrow (E), \\ & F \rightarrow a \} \end{aligned}$$



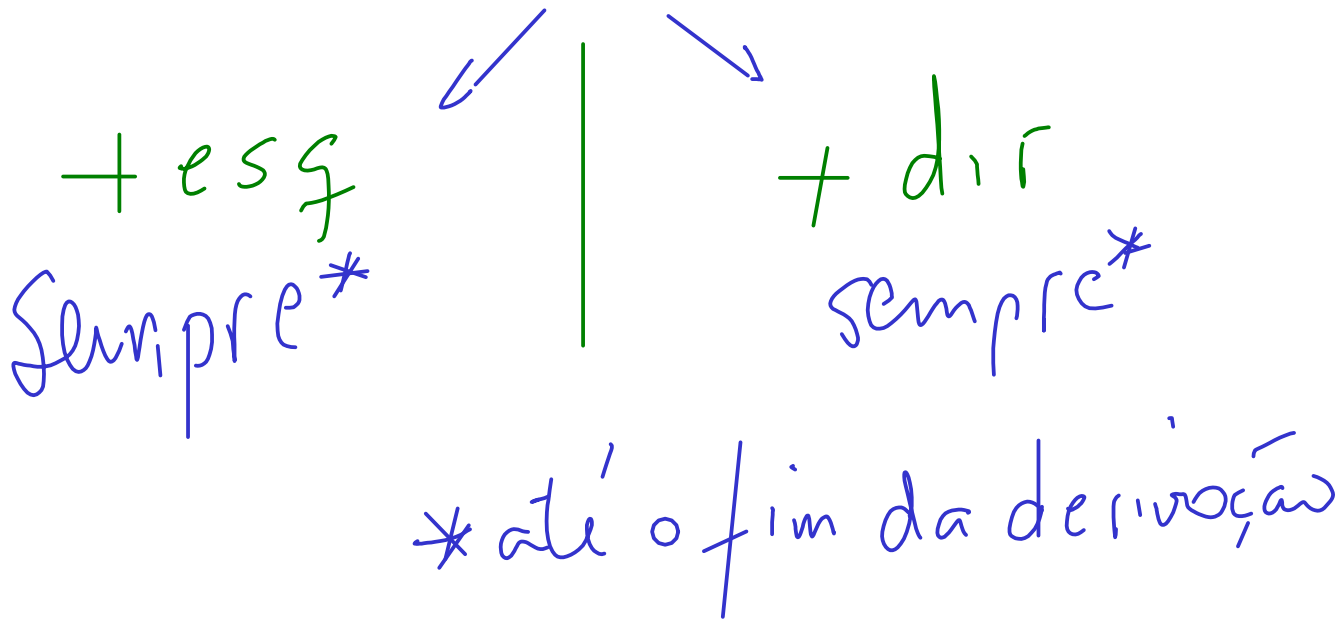
# Árvores de Derivação

- Árvores de derivação não contêm informação sobre a sequência em que foram aplicadas as produções
  - elas informam apenas quais foram as produções aplicadas, mas não em que ordem



# Gramática não-ambígua

- Se para toda e qualquer cadeia pertencente à linguagem, existir uma única sequência de **derivações mais à esquerda** e uma única sequência de **derivações mais à direita** que a geram



# Exemplo 1

$a=2$

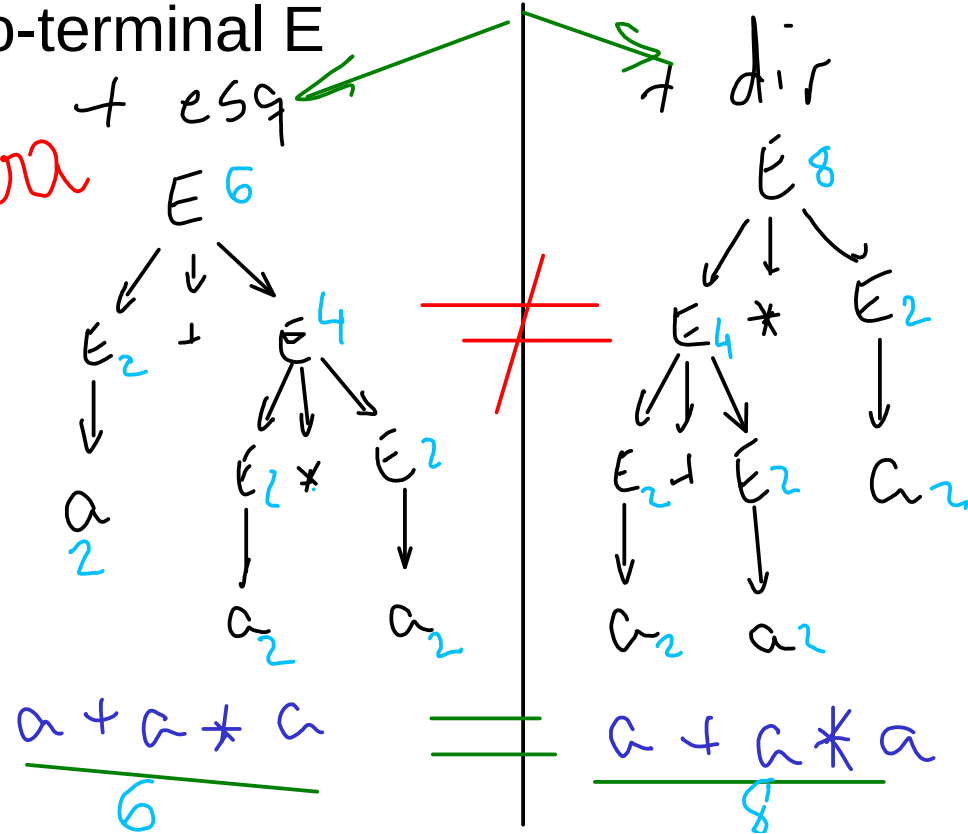
- Linguagem das expressões aritméticas sobre  $\{a, +, *, (, )\}$  com apenas um símbolo não-terminal  $E$

$\{E \rightarrow E + E,$   
 $E \rightarrow E * E,$   
 $E \rightarrow a,$   
 $E \rightarrow (E)\}$

*Prima*

- Cadeia:  $a + a * a$

*GRAMÁTICA  
AMBIGUA*



# Exemplo 1

- Aplicando-se inicialmente:  $E \rightarrow E + E$ 
  - $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$
- Aplicando-se inicialmente:  $E \rightarrow E * E$ 
  - $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

## Exemplo 2

- Ambiguidade

The diagram illustrates the ambiguity of nested if-then-else statements by showing two different ways to parse the same sequence of tokens: `if <exp> then if <exp> then <com> else <com>`.

**Top Parsing (Blue and Green Annotations):**

- The first `<exp>` is circled in green.
- A blue bracket groups the entire expression: `if <exp> then if <exp> then <com> else <com>`.
- A green bracket groups the inner expression: `if <exp> then <com>`.
- A magenta bracket groups the final `<com>`.
- A black bracket is placed under the entire expression.

**Bottom Parsing (Blue Annotations):**

- A blue bracket groups the entire expression: `if <exp> then if <exp> then <com> else <com>`.
- A black bracket is placed under the entire expression.

The ambiguity arises because the same sequence of tokens can be interpreted as either a nested if-then-else (top) or a single if-then-else with a nested if-then (bottom).

if ( F ) then  
    if (     ) then  
        com1  
    else  
        com2

output

com2

if ( F ) then  
    if (     ) then  
        com1  
    else  
        com2

output

∅

