

# FUNDAMENTOS TEÓRICOS DA COMPUTAÇÃO

--- LINGUAGEM LIVRE DE  
CONTEXTO ---

Gramáticas Livres de Contexto - Definição

# Aumentar o arsenal



# Hierarquia de Chomsky

Hierarquia de Chomsky	Gramática	Linguagem	Reconhecedor
Tipo 0	Irrestrita	Recursivamente Enumeráveis	Máquina de Turing
Tipo 1	Sensível ao Contexto	Sensíveis ao Contexto	Máquina de Turing com Fita Limitada
Tipo 2	Livre de Contexto	Livres de Contexto	Autômato de Pilha
Tipo 3	Linear	Regulares	Autômato Finito

# Linguagens Livres de Contexto

As linguagens livres de contexto (LLCs) podem ser geradas por **gramáticas livres de contexto** (GLCs) e reconhecidas por **autômatos com pilha** (APs)

As LLCs são usadas em **compiladores** e interpretadores para realizar **análise sintática** (parsing) de código-fonte. O processo de análise sintática verifica se uma **sequência de símbolos** segue a **gramática especificada pela LLC** e constrói uma árvore de derivação que representa a estrutura sintática do código

A **Gramática Livre de Contexto** consiste em um conjunto de regras de produção que definem **como** as cadeias de símbolos podem ser formadas. As regras de produção geralmente têm a forma  $A \rightarrow \omega$ , onde a variável  $A$  é **substituída** pela cadeia de símbolos  $\omega$

# Gramáticas Livres de Contexto

As GLCs foram primeiramente utilizadas no estudo de **linguagens humanas**, por ser uma maneira de entender o relacionamento dos termos (sujeito, verbo, predicado, preposição, etc) o que leva a uma **recursão natural**

$\langle \text{SENTENCE} \rangle \rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle$

$\langle \text{NOUN-PHRASE} \rangle \rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHASE} \rangle$

$\langle \text{VERB-PHRASE} \rangle \rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHASE} \rangle$

$\langle \text{PREP-PHASE} \rangle \rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle$

$\langle \text{CMPLX-NOUN} \rangle \rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle$

$\langle \text{CMPLX-VERB} \rangle \rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHASE} \rangle$

$\langle \text{ARTICLE} \rangle \rightarrow \text{a} \mid \text{the}$

$\langle \text{NOUN} \rangle \rightarrow \text{boy} \mid \text{girl} \mid \text{flower}$

$\langle \text{VERB} \rangle \rightarrow \text{touches} \mid \text{likes} \mid \text{sees}$

$\langle \text{PREP} \rangle \rightarrow \text{with}$

# Gramáticas Livres de Contexto

Derivação da cadeia “**a boy sees**”

⟨SENTENCE⟩ ⇒ ⟨NOUN-PHRASE⟩⟨VERB-PHRASE⟩  
⇒ ⟨CMPLX-NOUN⟩⟨VERB-PHRASE⟩  
⇒ ⟨ARTICLE⟩⟨NOUN⟩⟨VERB-PHRASE⟩  
⇒ a ⟨NOUN⟩⟨VERB-PHRASE⟩  
⇒ a boy ⟨VERB-PHRASE⟩  
⇒ a boy ⟨CMPLX-VERB⟩  
⇒ a boy ⟨VERB⟩  
⇒ a boy sees

# Gramáticas livres de contexto

As gramáticas livres de contexto (GLC) são as **geradoras** das linguagens livres de contexto (LLC)

Compreende um universo **mais amplo** do que as linguagens regulares

Aplicações práticas

Analisadores sintáticos de compiladores

Tradutores de linguagens e

Processadores de textos em geral

# GLC em linguagem de programação

São usadas para definir a sintaxe de linguagens de programação

```
⟨PROGRAM⟩ → begin program ⟨COMMAND-LIST⟩ end program
⟨COMMAND-LIST⟩ → ⟨COMMAND⟩⟨COMMAND-LIST⟩ | ⟨COMMAND⟩
    ⟨COMMAND⟩ → ⟨ASSIGN-COMMAND⟩ | ⟨IF-THEN-ELSE-COMMAND⟩ |
                  ⟨WHILE-COMMAND⟩ | ...
    ⟨ASSIGN-COMMAND⟩ → ⟨VARIABLE⟩ = ⟨EXPRESSION⟩
    ⟨IF-THEN-ELSE-COMMAND⟩ → if ⟨BOOLEAN-EXPRESSION⟩ then ⟨COMMAND-LIST⟩
                                else ⟨COMMAND-LIST⟩ end if
    ⟨WHILE-COMMAND⟩ → while ⟨BOOLEAN-EXPRESSION⟩ do ⟨COMMAND-LIST⟩ end while
                      ... → ...
    ⟨VARIABLE⟩ → ⟨LETTER⟩⟨VARIABLE-TAIL⟩ | ⟨LETTER⟩
    ⟨VARIABLE-TAIL⟩ → ⟨LETTER⟩⟨VARIABLE-TAIL⟩ | ⟨DIGIT⟩⟨VARIABLE-TAIL⟩ | ⟨LETTER⟩ | ⟨DIGIT⟩
    ⟨LETTER⟩ → A | B | ... | Z | a | b | ... | z
    ⟨DIGIT⟩ → 0 | 1 | ... | 9
```

# GLC em linguagem de programação

lysator.liu.se/c/ANSI-C-grammar-y.html

```
%token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF
%token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EO_OP NE_OP
%token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN TYPE_NAME

%token TYPEDEF EXTERN STATIC AUTO REGISTER
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token STRUCT UNION ENUM ELLIPSIS

%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN

%start translation_unit
%%

primary_expression
: IDENTIFIER
| CONSTANT
| STRING_LITERAL
| '(' expression ')'
;

postfix_expression
: primary_expression
| postfix_expression '[' expression ']'
| postfix_expression '(' ')'
| postfix_expression '(' argument_expression_list ')'
| postfix_expression '.' IDENTIFIER
| postfix_expression PTR_OP IDENTIFIER
| postfix_expression INC_OP
| postfix_expression DEC_OP
;
argument_expression_list
: assignment_expression
| argument_expression_list ',' assignment_expression
;
unary_expression
: postfix_expression
| INC_OP unary_expression
| DEC_OP unary_expression
| unary_operator cast_expression
| SIZEOF unary_expression
| SIZEOF '(' type_name ')'
;
```

# Definição

Uma GLC é uma quádrupla  $(V, \Sigma, P, S)$ :

- $V$  é um conjunto de símbolos não terminais (**variáveis**). Cada variável representa um tipo diferente de cláusula na sentença. As variáveis são também chamados de **categorias sintáticas** (ou classes)
- $\Sigma$  é um conjunto de símbolos **terminais** ( $\Sigma \cap V = \emptyset$ ). Compõem o conteúdo real da sentença, isto é, o conjunto de terminais é o alfabeto da linguagem
- $P$  é o conjunto de **regras de produção** (ou de substituição ou de reescrita) na forma  $V \rightarrow (V \cup \Sigma)^*$ ,
- $S$  é o **símbolo não terminal inicial**, onde  $S \in V$ , que é usado para representar a frase inteira (ou programa)

# GRs e GLCs

## Gramáticas Regulares

Deve ser linear unitária à esquerda ou à direita

$$A \rightarrow aB \mid a$$

$$A \rightarrow Ba \mid a$$

$$A \rightarrow baA \text{ (**não pode**)}$$

$$A \rightarrow ABa \text{ (**não pode**)}$$

## Gramáticas Livres de Contexto

As regras possuem somente uma variável do lado esquerdo

Não pode ter terminal do lado esquerdo

$$A \rightarrow BaC$$

$$Aa \rightarrow BaC \text{ (**não pode**)}$$

# Gramáticas livres de contexto

Tem boa aplicabilidade em linguagens de programação

Estruturas aninhadas

```
begin begin end end  
{ { { } { } } }  
( () () )
```

Blocos e estruturas de controle

```
if <cond> then  
    if <cond> then  
        ...  
    else  
        ...  
    else
```

# Por quê livre de contexto?

- Sendo as nossas produções do tipo  $A \rightarrow \alpha$ , em uma derivação qualquer,  $A$  vai derivar  $\alpha$  **sem depender** de qualquer **análise**, **sem depender** de qualquer **símbolo** que antecede ou sucede  $A$ , que seria o contexto de  $A$
- Ou seja, do lado esquerdo das regras só podem ter **uma variável** e nenhum terminal (do lado esquerdo)
- Como as regras não **dependem** do “contexto” elas são chamadas “**livres**” de contexto
- Regras **com contexto** podem ser, por exemplo,  $aAb \rightarrow BaC$ , isto é,  $A$  vem **depois** do  $a$  e **antes** do  $b$ . Note que regras desse tipo não podem em GLC

# Gramáticas livres de contexto

Considere a linguagem  $L_1 = \{a^n b^n : n \geq 0\}$

Já sabemos que essa linguagem **não é regular**

Se considerarmos  $a = ($  e  $b = )$  este seria o mesmo problema do aninhamento de parêntesis

$aaabbb \leftrightarrow ((( )))$

Qual seria a GLC que gera a linguagem  $L$ ?

$S \rightarrow aSb \mid \epsilon$

Derivação da cadeia  $aabb$

$S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb$

# Gramáticas livres de contexto

Considere a linguagem  $L_2 = \{\text{expressões aritméticas com parêntesis平衡ados, operador de soma e multiplicação e um operando}\}$

$G = (\{E\}, \{+, *\}, \{(), x\}, P, E)$  sendo  $P$  definido por:

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow x$$

Derivação da cadeia  $(x + x) * x$

$$E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + E) * E \Rightarrow (x + E) * E$$

$$\Rightarrow (x + x) * E \Rightarrow (x + x) * x$$

# Derivação por árvore

Raiz é o símbolo inicial

**Vértices interiores** são variáveis

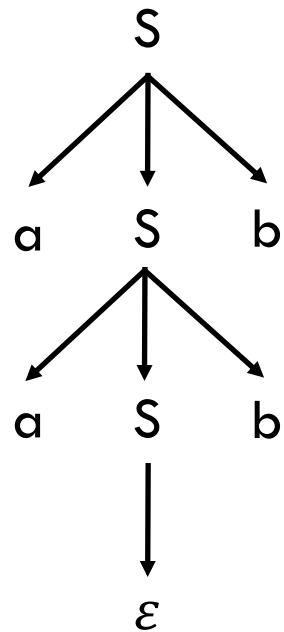
Se  $A$  é um vértice interior e  $X_1, X_2, \dots, X_n$  são os filhos de  $A$

$A \rightarrow X_1X_2 \dots X_n$  é uma produção da GLC

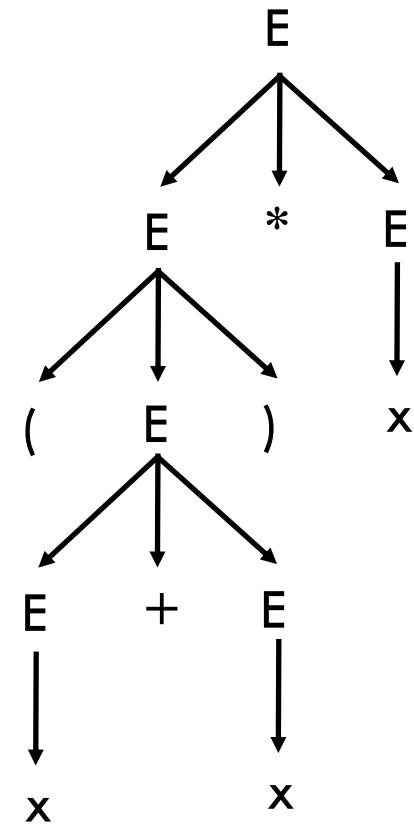
**Vértices folhas** são símbolos terminais ou  $\epsilon$

# Derivação em árvore

$L_1$ : Derivação de  $aabb$



$L_2$ : Derivação de  $(x + x) * x$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

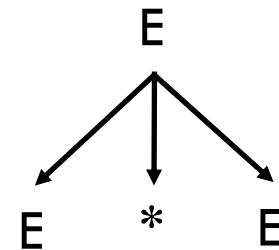
$E$

$E$

# Derivação mais à direita

Derivação da cadeia  $x + x * x$

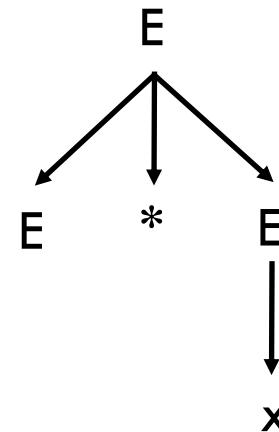
$E$   
 $\Rightarrow E * E$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

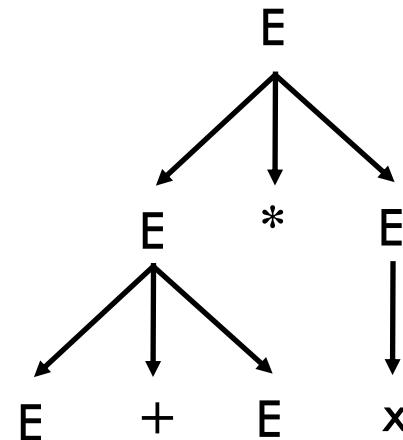
$E$   
 $\Rightarrow E * E$   
 $\Rightarrow E * x$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

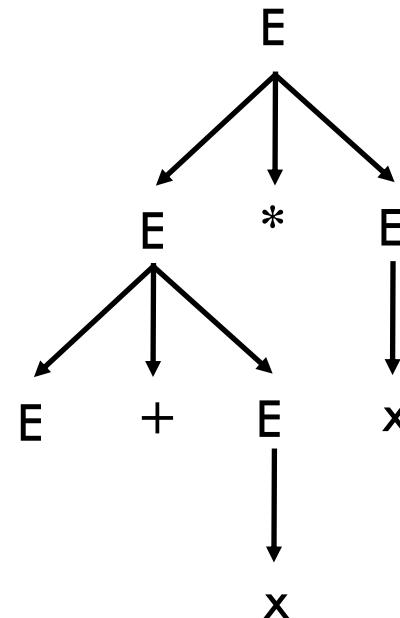
$E$   
 $\Rightarrow E * E$   
 $\Rightarrow E * x$   
 $\Rightarrow E + E * x$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

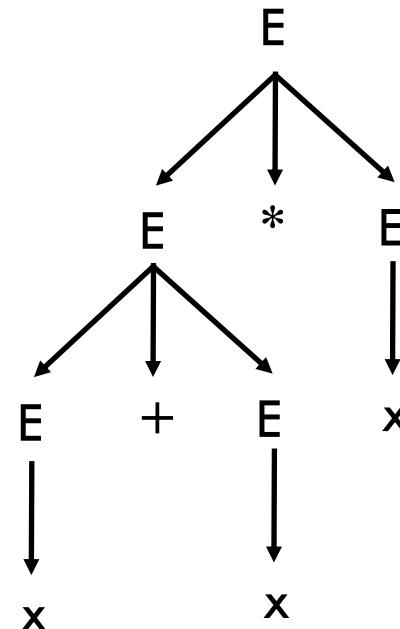
$E$   
 $\Rightarrow E * E$   
 $\Rightarrow E * x$   
 $\Rightarrow E + E * x$   
 $\Rightarrow E + x * x$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

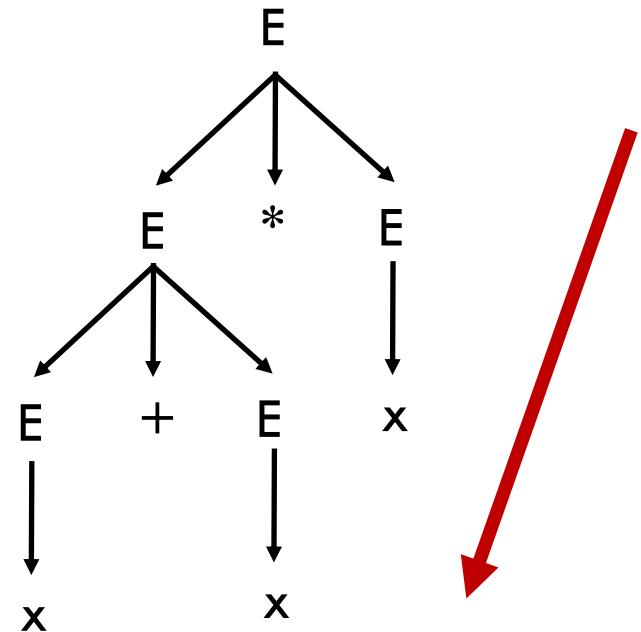
$E$   
 $\Rightarrow E * E$   
 $\Rightarrow E * x$   
 $\Rightarrow E + E * x$   
 $\Rightarrow E + x * x$   
 $\Rightarrow x + x * x$



# Derivação mais à direita

Derivação da cadeia  $x + x * x$

$E$   
 $\Rightarrow E * E$   
 $\Rightarrow E * x$   
 $\Rightarrow E + E * x$   
 $\Rightarrow E + x * x$   
 $\Rightarrow x + x * x$



A cadeia é reconhecida pela gramática  
Árvore pende mais para a esquerda

# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

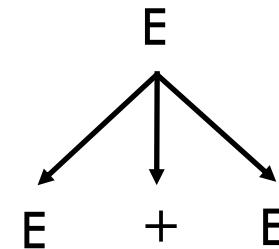
$E$

$E$

# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

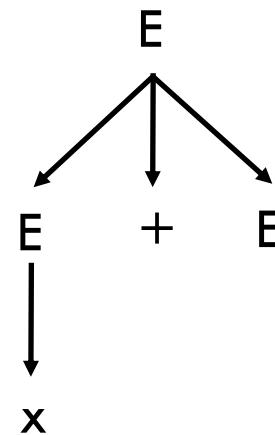
$E$   
 $\Rightarrow E + E$



# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

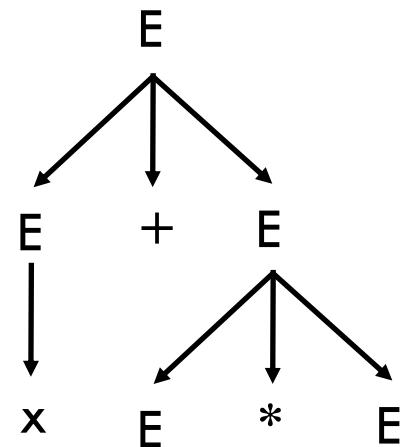
$E$   
 $\Rightarrow E + E$   
 $\Rightarrow x + E$



# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

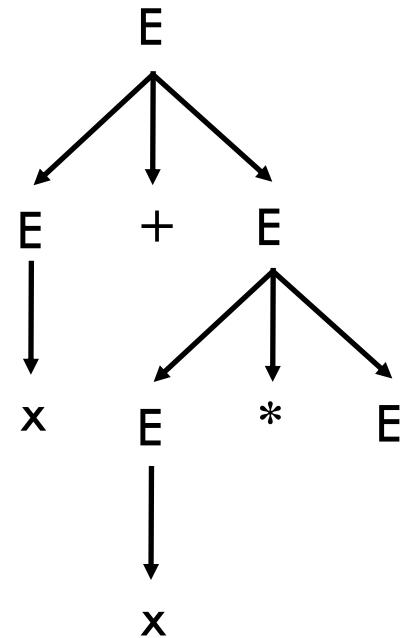
$E$   
 $\Rightarrow E + E$   
 $\Rightarrow x + E$   
 $\Rightarrow x + E * E$



# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

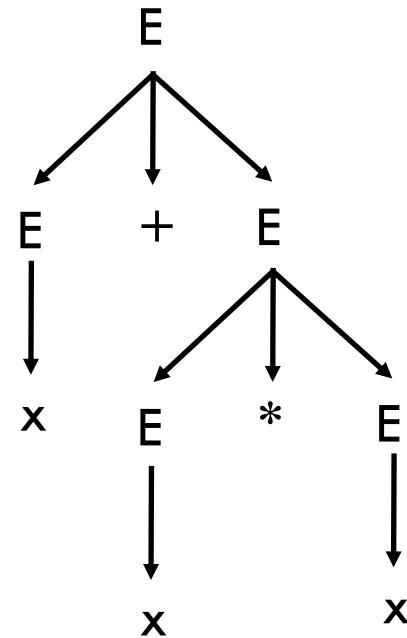
$E$   
 $\Rightarrow E + E$   
 $\Rightarrow x + E$   
 $\Rightarrow x + E * E$   
 $\Rightarrow x + x * E$



# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

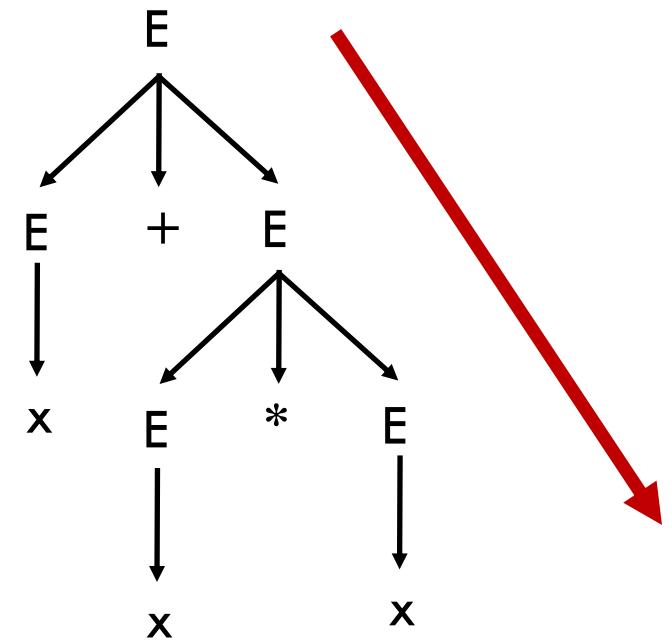
$E$   
 $\Rightarrow E + E$   
 $\Rightarrow x + E$   
 $\Rightarrow x + E * E$   
 $\Rightarrow x + x * E$   
 $\Rightarrow x + x * x$



# Derivação mais à esquerda

Derivação da cadeia  $x + x * x$

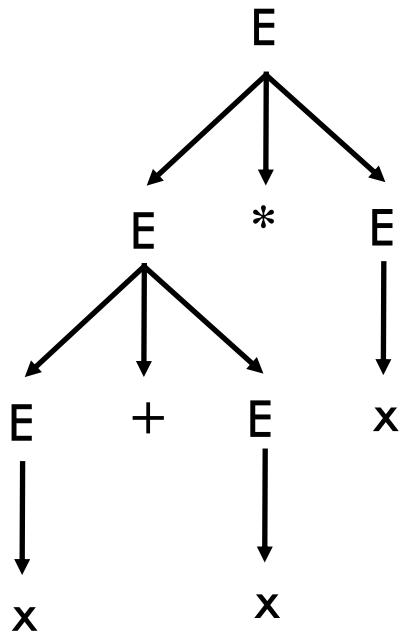
$E$   
 $\Rightarrow E + E$   
 $\Rightarrow x + E$   
 $\Rightarrow x + E * E$   
 $\Rightarrow x + x * E$   
 $\Rightarrow x + x * x$



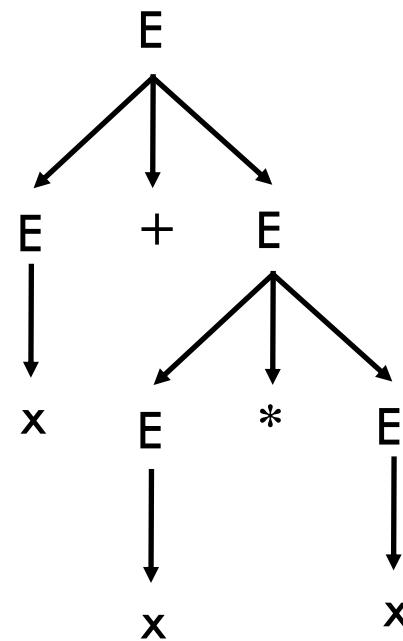
A cadeia é reconhecida pela gramática  
Árvore pende mais para a direita

# Ambiguidade

Derivação Direita



Derivação Esquerda



Uma gramática é ambígua se para uma palavra há **duas ou mais** árvores de derivação

# Ambiguidade

Se, a partir de uma mesma gramática, eu gero **duas** (ou mais) árvores de derivação **diferentes**, a gramática é **ambígua**

**Não existe** um algoritmo formal para identificar ambiguidade

Basta mostrar **duas árvores de derivação diferentes** para a mesma cadeia (fazer na mão, usando a técnica tentativa e erro)

Qual o **problema** da ambiguidade?

1) O analisador **não vai saber** qual das árvores de derivação ele tem que escolher

2) A ordem de **execução das operações** aritméticas pode ser afetada

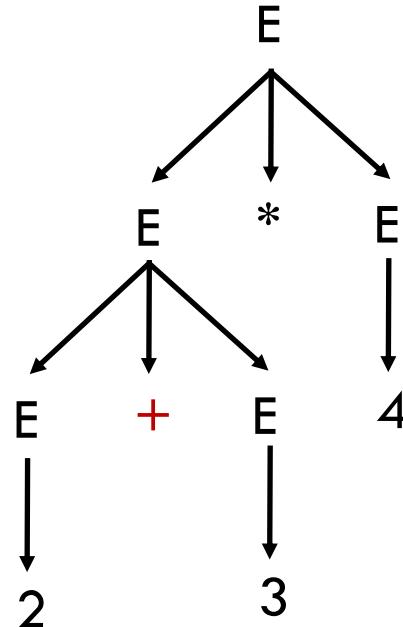
Ambiguidade em gramáticas tem que ser **eliminada**

# Ambiguidade

Considerando a operação:  $2+3*4$ , o resultado deveria ser **14**, porque a multiplicação tem precedência sobre a soma.

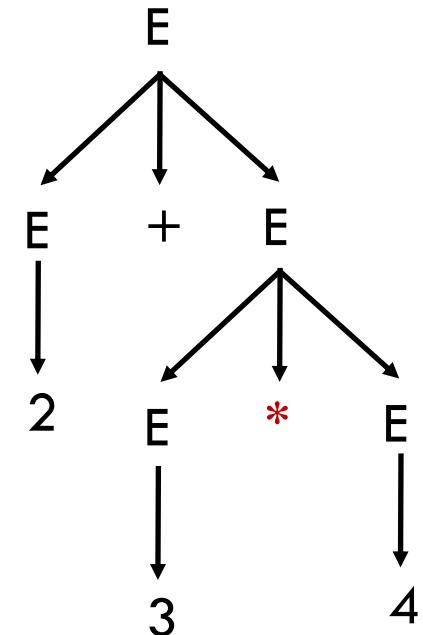
Veja que a ordem das execuções das duas árvores de derivação geram resultados diferentes

Derivação Direita



Resultado: **20**

Derivação Esquerda



Resultado: **14**

# Exercício Resolvido (1)

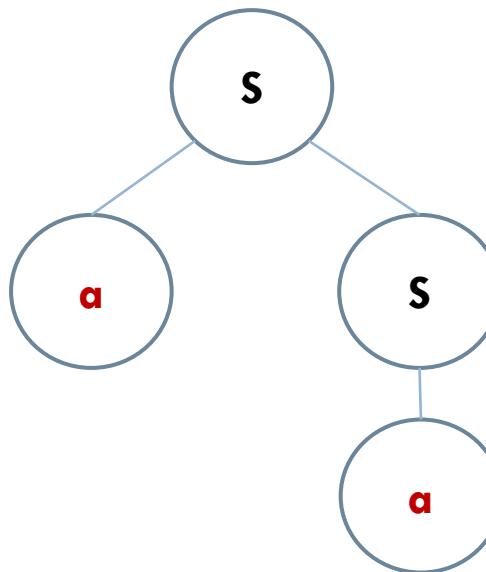
Verifique se a gramática  
é ambígua:

$$S \rightarrow aS \mid Sa \mid a$$

Considere a seguinte  
cadeia de entrada:

$$w = aa$$

Esquerda



# Exercício Resolvido (1)

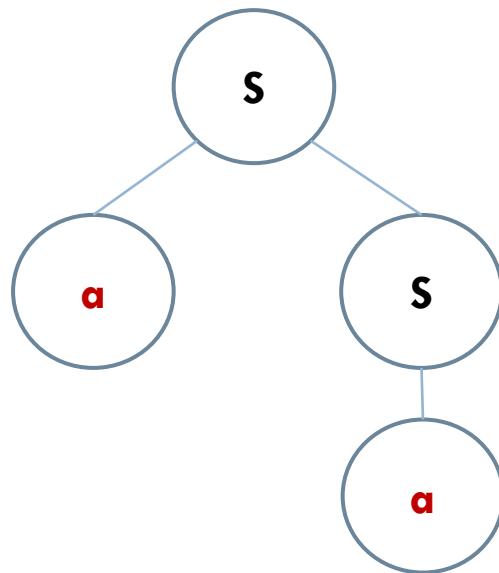
Verifique se a gramática é ambígua:

$$S \rightarrow aS \mid Sa \mid a$$

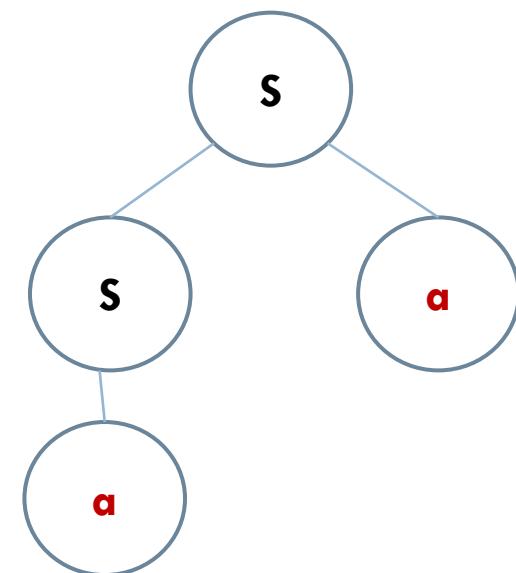
Considere a seguinte cadeia de entrada:

$$w = aa$$

Esquerda



Direita



Isso mostra que a gramática é ambígua

# Exercício Resolvido (2)

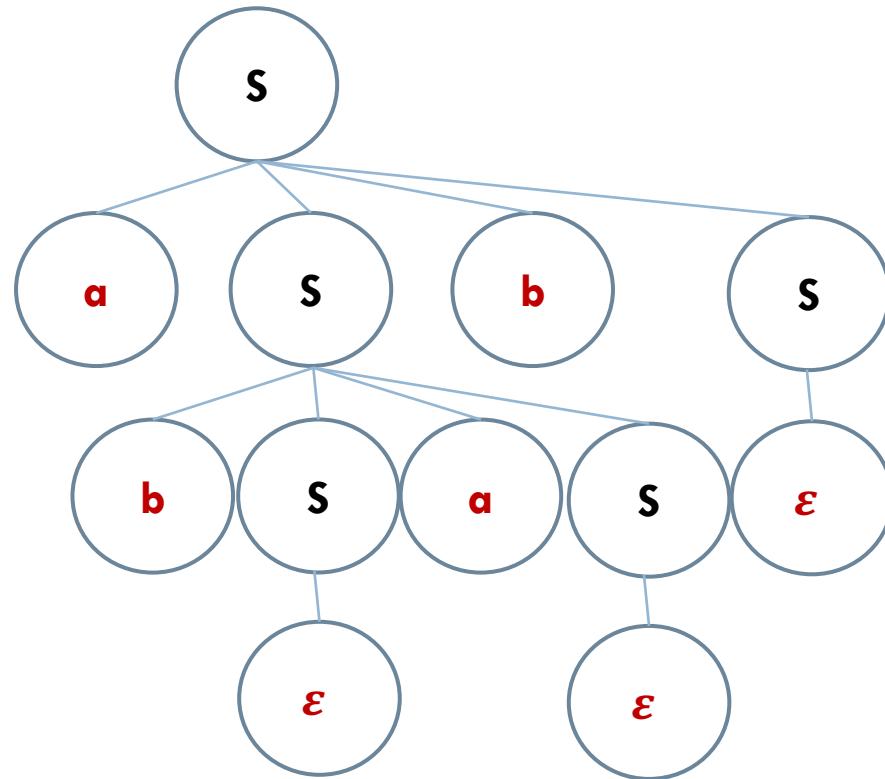
Verifique se a gramática  
é ambígua:

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

Considere a seguinte  
cadeia de entrada:

$$w = abab$$

Derivação mais à  
**esquerda**



# Exercício Resolvido (2)

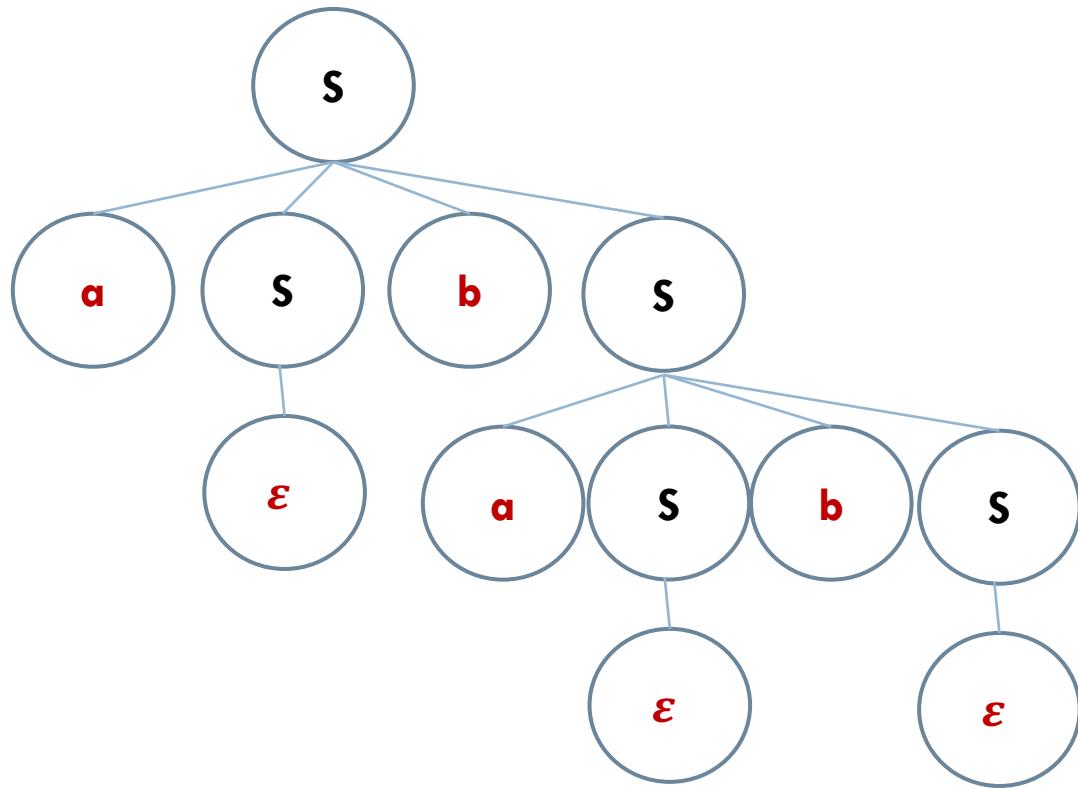
Verifique se a gramática  
é ambígua:

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

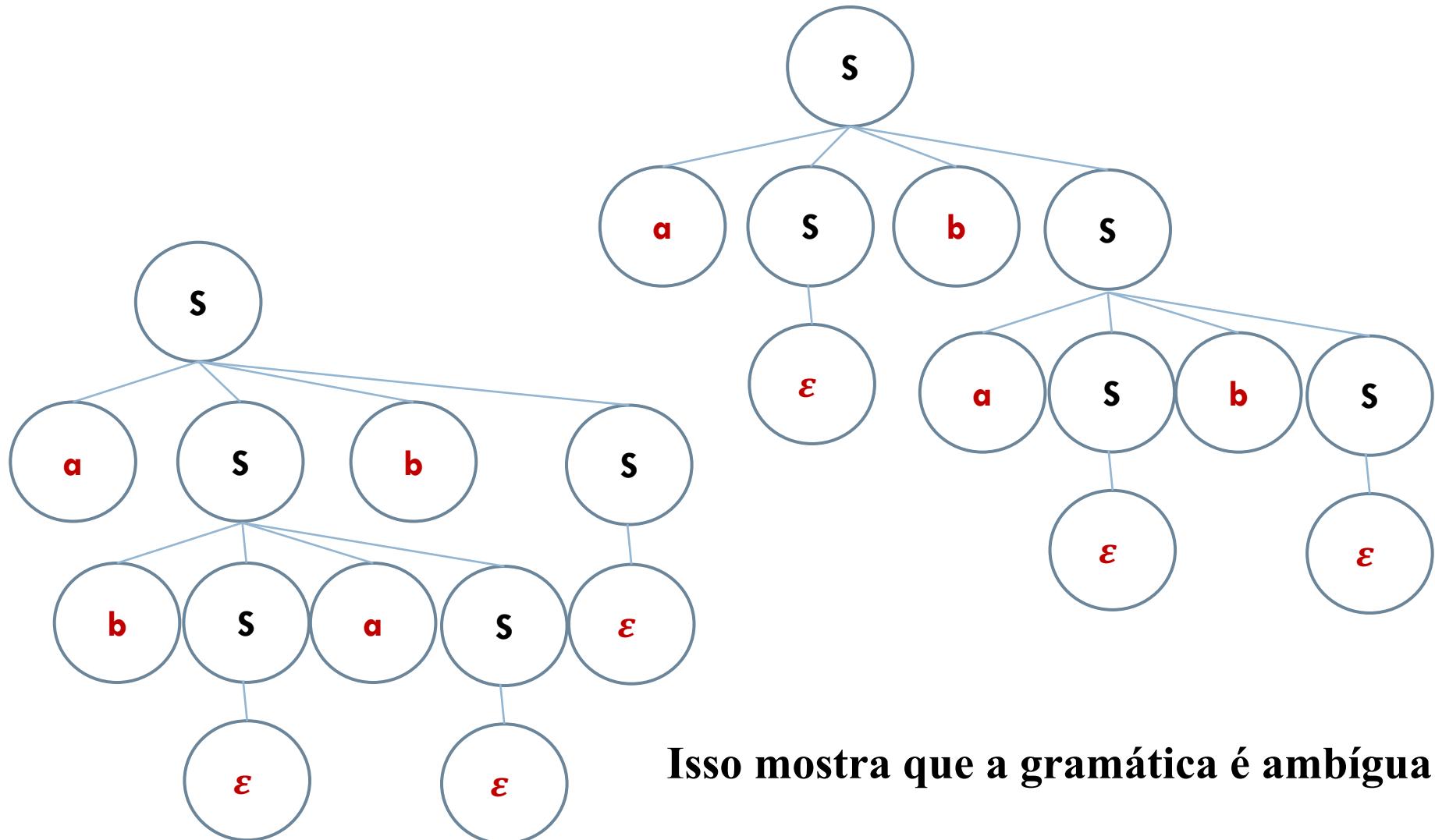
Considere a seguinte  
cadeia de entrada:

$$w = abab$$

Derivação mais à **direita**



# Exercício Resolvido (2)





**Advinha quem vai arrasar na prova de  
gramática livre de contexto?**