

# Gramáticas - 3

Árvores de Derivação para GLC

Ambigüidade nas GLC

Precedência, Prioridade e Associatividade de operadores

# Árvores de Derivação para GLC

- GLC são as mais importantes para a definição de linguagens de programação e seus compiladores
  - pois podem especificar a maioria das estruturas sintáticas das linguagens de programação.
- Durante a compilação podemos usar a **estrutura sintática** de um **programa** (uma **sentença da linguagem**)
  - para ajudar a produzir a sua tradução para linguagem de máquina.

- A estrutura sintática de uma sentença de entrada pode ser determinada
  - a partir da seqüência de produções usadas para derivar aquela sentença.
- Podemos entender o **analisador sintático (parser)** de um compilador como
  - um dispositivo que **tenta determinar se existe uma derivação da sentença de entrada de acordo com alguma GLC.**

- Em uma gramática é possível ter **várias derivações equivalentes** (usam as **mesmas produções MAS** em **ordem diferente**).
- Para gramáticas irrestritas é difícil definir quando 2 derivações são equivalentes MAS
  - para GLC temos uma representação gráfica que representa uma **classe de equivalências** chamada **Árvore de Derivação**.
- Uma **árvore de derivação** para uma GLC  $G = (V_n, V_t, P, S)$ 
  - é uma **árvore rotulada ordenada** em que cada nó é rotulado por um símbolo de  $V_n \cup V_t \cup \lambda$ .
- Se um nó interior é rotulado com A e seus descendentes diretos são rotulados com  $X_1, X_2, \dots, X_n$  então  $A \rightarrow X_1X_2\dots X_n$  é uma produção de P.

Uma **árvore rotulada ordenada**  $D$  é uma **árvore de derivação** para uma GLC  $G(A) = (V_n, V_t, P, A)$  se

- (1) A raiz de  $D$  é rotulada com  $A$
- (2) Se  $D_1, \dots, D_k$  são as subárvores de descendentes diretos da raiz e a raiz de  $D_i$  é rotulada com  $X_i$ ,  
**então**  $A \rightarrow X_1, \dots, X_k$  é uma produção em  $P$ .  
 $D_i$  deve ser a árvore de derivação para  $G(X_i) = (V_n, V_t, P, X_i)$  se  $X_i$  é não-terminal e  $D_i$  é um nó simples rotulado com  $X_i$  se  $X_i$  é terminal.
- (3) Se  $D_1$  é a única subárvore da raiz  $D$  e a raiz de  $D_1$  é rotulada com  $\lambda$  então  $A \rightarrow \lambda$  é uma produção de  $P$ .

**Vértices interiores são não-terminais e vértices folhas podem ser não-terminais, terminais ou  $\lambda$ .**

**Quando fazemos a árvore de derivação de uma sentença, os vértices folhas são sempre terminais ou  $\lambda$ .**

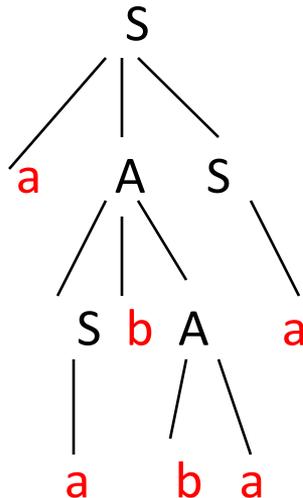
Uma **sentença** está representada na árvore de derivação fazendo-se a **leitura das folhas** (nós sem descendentes) **da esquerda para direita**.

# Exemplos

- Árvore de derivação para a sentença **aabbaa** da GLC  $G = (\{S,A\}, \{a,b\}, P, S)$

$P = \{ S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid ba \mid SS \}$



# Derivações mais à esq ou mais à dir

Temos uma derivação mais à esquerda  
(direita)

quando a cada passo no processo de derivação  
de uma sentença escolhemos a variável mais à  
esquerda (direita)

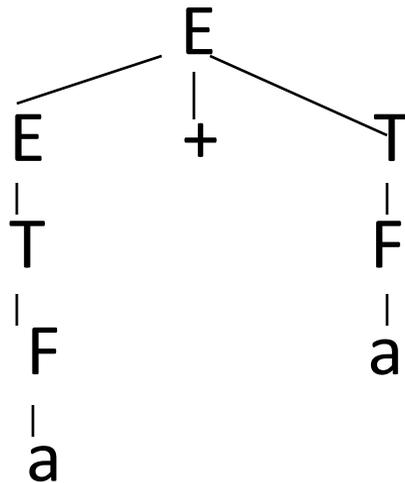
- Seja  $G = (\{E, T, F\}, \{+, *, (, ), a\}, P, E)$

$$P = \{ E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a \}$$

- 1) Quantas derivações equivalentes da sentença “a + a” a árvore de derivação abaixo representa?
- 2) Mostre a derivação **mais à esq** e a **mais à dir**.



# Ambigüidade nas GLC

- Um requisito importante de uma LP é que ela **não seja ambígua** ou, no mínimo, que qq ambigüidade na linguagem seja evidente ou facilmente evitada.
- O mais famoso caso de ambigüidade é o **else pendente**, presente na especificação de muitas LP.

- Seja a gramática:

C -> if b then C else C

C -> if b then C

C -> s

Ela é ambígua desde que a cadeia

**If b then if b then s else s**

Pode ser interpretada como

(i) If b then (if b then s else s)

Ou

(ii) If b then (if b then s) else s

A primeira é a preferida em LP's

pois utilizam a regra informal **“case o else com o if mais próximo”**  
que remove a ambiguidade

- Podemos reescrever a gramática acima com 2 não-terminais C1 e C2:

C1 -> if b then C1 | if b then C2 else C1 | s

C2 -> if b then C2 else C2 | s

O fato que somente C2 precede o else

garante que entre par then-else gerado por qq uma das produções deve aparecer ou um s ou outro else. Assim, (ii) nunca ocorre.

(ii) If b then (if b then s) else s

Vamos definir exatamente o significado de ambiguidade:

Seja  $G1 = (\{E,T,F,U\}, \{0,1,2,\dots,9\}, P, E)$

$$P = \{ E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow U \mid (E) \\ U \rightarrow 0 \mid 1 \mid \dots \mid 9 \}$$

uma gramática para expressões aritméticas para inteiros de 0..9.

Por que as LP não usam a gramática G2:

$$P = \{ E \rightarrow E + E \mid E * E \mid (E) \mid F \\ F \rightarrow 0 \mid 1 \mid \dots \mid 9 \}$$

que contém um menor número de símbolos e produções???

A razão é que:

(1) G2 é ambígua e G1 não é.

(2) Em G2 + e \* tem a mesma prioridade de resolução enquanto que na matemática eles não tem.

Dada a expressão  $2 + 3 * 5$

Ela é interpretada como?

$$(2 + (3 * 5)) = 17$$

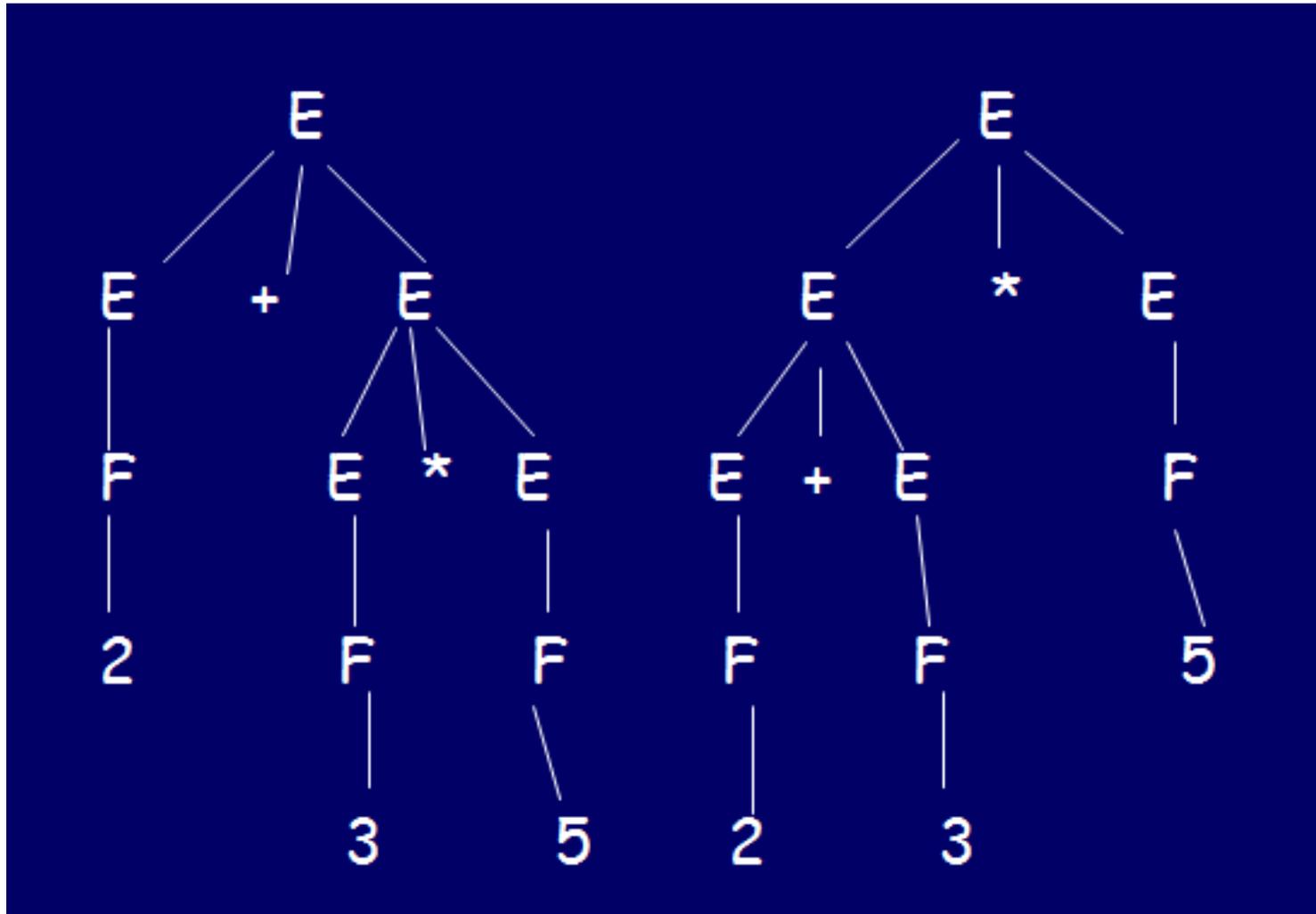
Ou

$$((2 + 3) * 5) = 25$$

# Como mostrar que uma gramática é ambígua?

- Com árvores de derivação.
- **Def1:** Uma GLC (G) é ambígua se há pelo menos **uma cadeia** pertencente à  $L(G)$  **com mais de uma árvore de derivação** para representá-la.
- **Def2:** Existência de **uma sentença** com **duas ou mais derivações mais à esq** (ou **mais à dir**).

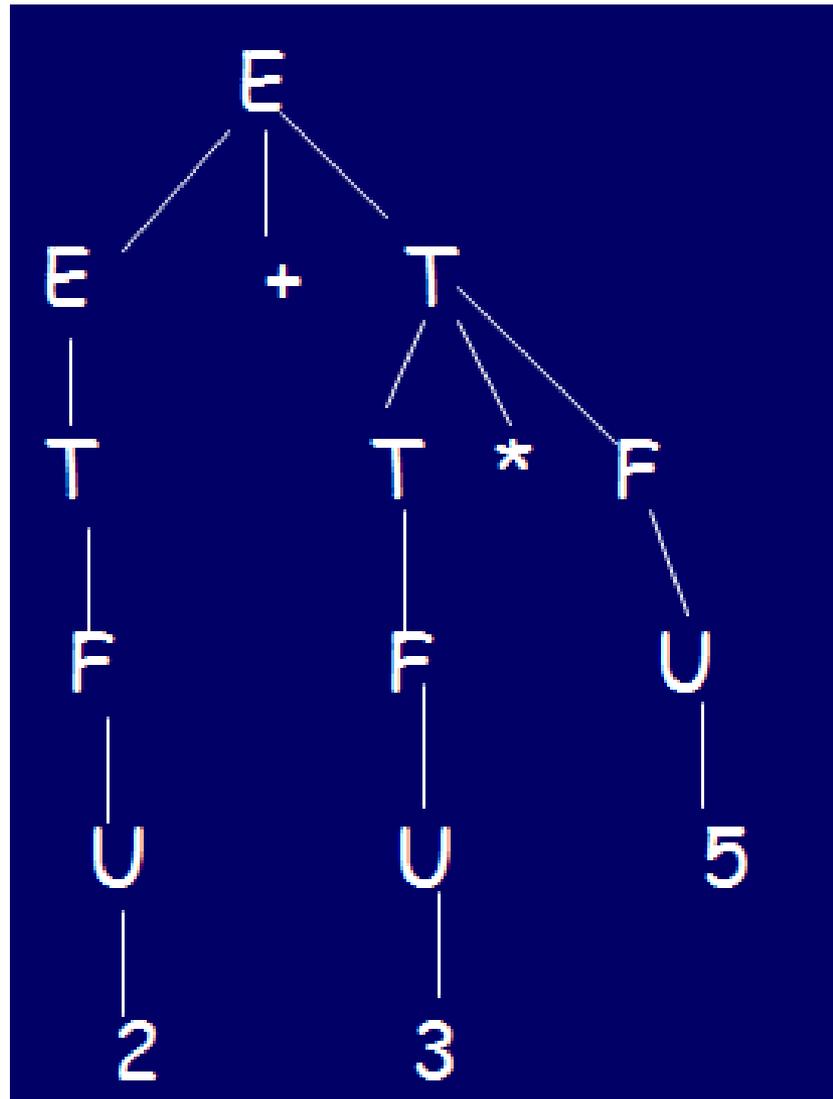
# Árvores Derivação de $2 + 3 * 5$ em G2



# Derivações esquerdas de $2 + 3 * 5$ em $G_2$

- $E \Rightarrow E + E \Rightarrow F + E \Rightarrow 2 + E \Rightarrow 2 + E * E \Rightarrow 2 + F * E \dots$
- $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow F + E * E \Rightarrow 2 + E * E \Rightarrow 2 + F * E$   
...

# Árvore Derivação de $2 + 3 * 5$ em G1



Primeiro resolvemos  $3 * 5$  para depois somar com  $2$ .

Dizemos que  $*$  tem maior prioridade de resolução, embora  $+$  tenha maior precedência (escopo).

# 2 causas da ambiguidade

- A prioridade não é respeitada
  - Precisamos forçar somente uma forma de estruturar o uso de vários operadores na árvore de derivação
- Uma seqüência de operadores idênticos podem se agrupar a partir da esquerda ou da direita.
  - Precisamos forçar somente uma forma de se associar os operadores idênticos

- Para retirar a ambiguidade:
  - introduzimos várias variáveis e estratificamos as regras, quando temos operadores com várias prioridades de resolução.
  - Para resolver a ambiguidade vinda do uso de vários operadores idênticos, forçamos o uso da recursão para esquerda ou direita.

# Regras de Precedência, Prioridade e Associatividade

- Ajudam na decisão da interpretação correta de expressões nas LP
- Def: Um operador é **associado à esquerda** se os operandos são agrupados da esq para dir e **associado à direita** se são agrupados da dir para esq.
- Está relacionada com a **posição da recursão nas regras** que permitem um operador ser aplicado mais de uma vez. Por convenção  $a+a+a = (a+a) + a$

# Pascal

- 1) Expressões parentizadas são resolvidas primeiro
- 2) Not                      associados a direita
- 3) \* / div mod and      associados a esquerda
- 4) + - or                      “                      “
- 5) < > <> >= <= =                      ocorrem uma vez

EXP -> ES <> ES | ES (uma vez)

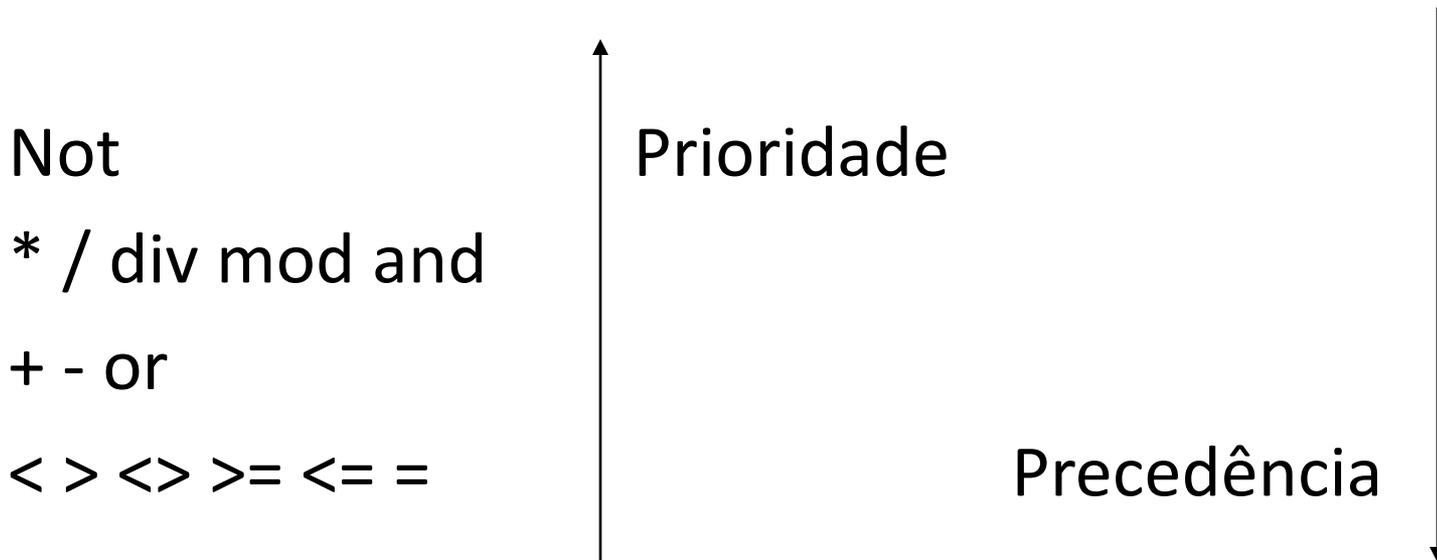
ES -> ES + T | T      (esq)

T -> T \* F | F      (esq)

F -> not F | U      (dir)

U -> <inteiro> | <real> | (EXP) { as duas últimas  
regras geralmente  
aparecem juntas na  
gramática do Pascal }

- Os níveis de **prioridade** indicam a quais operadores são permitidos agrupar seus operandos primeiro (resolver primeiro).



- Quanto maior a **precedência** maior o escopo (abrangência na árvore de derivação).

# Linguagens Inerentemente ambíguas

É simples encontrar um exemplo de GLC ambígua.

Na gramática abaixo para a sentença “a” temos  
2 árvores

$S \rightarrow A \mid B$

$A \rightarrow a$

$B \rightarrow a$

Mas não é tão simples exibir uma **LLC**  
**inerentemente ambígua**

Def ☹️: Uma LLC é inerentemente ambígua se toda  
gramática que a gera é ambígua

$L = \{a^i b^j c^k \mid (i,j,k \geq 1) \text{ e } ((i = j) \text{ OU } (j = k))\}$

$S \rightarrow abc \mid aRbI \mid YbWc$

$I \rightarrow Ic \mid c$

$R \rightarrow ab \mid aRb$

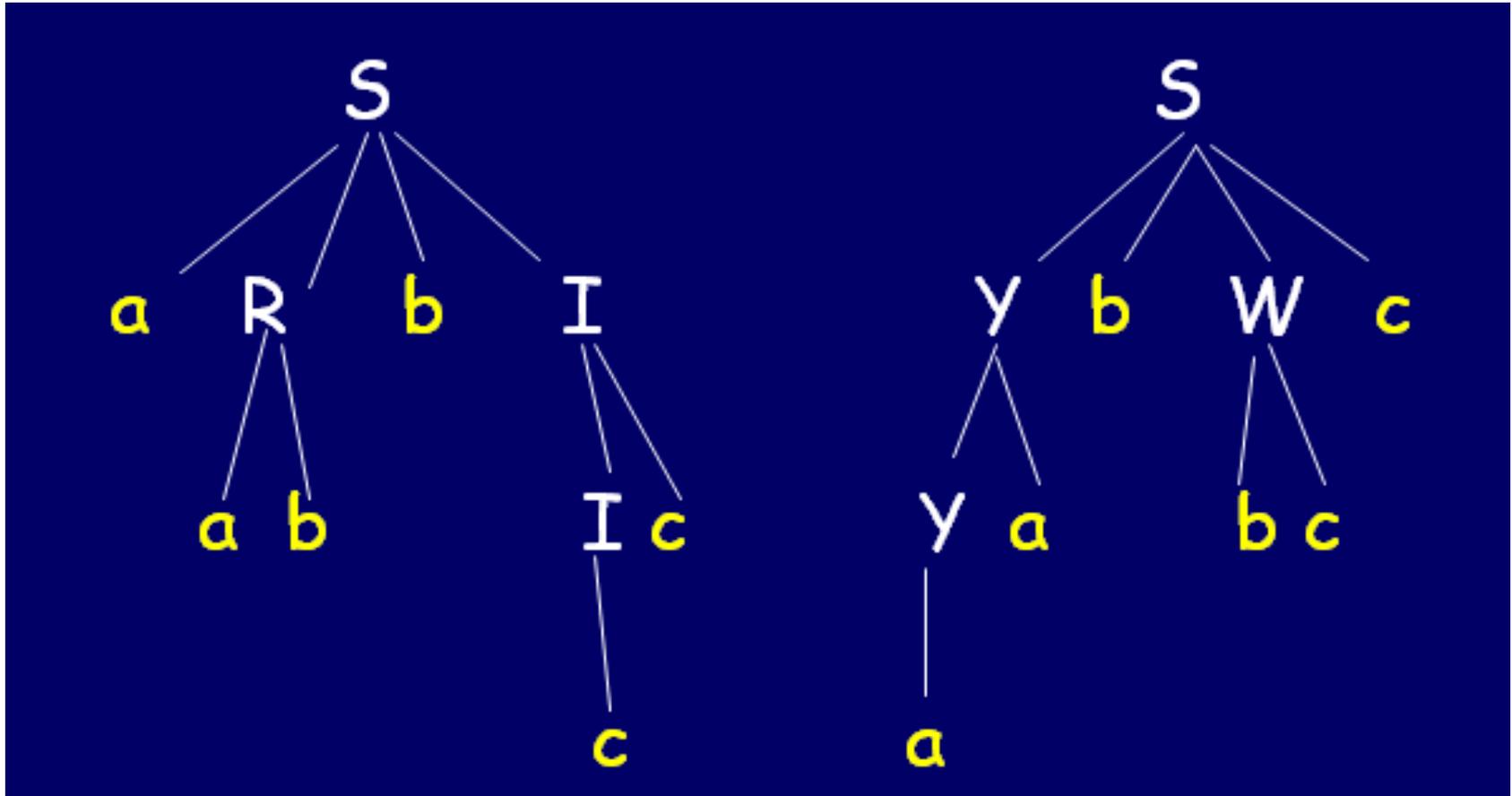
$Y \rightarrow Ya \mid a$

$W \rightarrow bc \mid bWc$

O fato de ser inerentemente ambígua é que toda gramática que gera  $L$  gera palavras para  $i=j$  por um processo diferente do qual usa para gerar as palavras para  $j=k$ .

É impossível não gerar algumas palavras para as quais  $i=j=k$  por ambos os processos.

- Exemplo: aabbcc



Felizmente problemas de ambiguidade inerente  
não aparecem em LP!

# GLC e problemas Indecidíveis

**Teo** ☹️: É indecidível se uma dada GLC é ambígua. **Não** existe um algoritmo **genérico** que dada uma GLC qq sempre retorne a resposta (sim/não) para ela.

## MAS

- ☺️ em casos particulares nós podemos reconhecer a ambiguidade e remover “a mão”
- ☺️ E podemos utilizar classes mais restritas de GLC que por definição não são ambíguas como as LL(K)

Este problema (decidir se uma GLC é ambígua) é **parcialmente decidível**, pois para determinadas gramática o procedimento pára e diz sim MAS pode rodar indefinidamente para outros casos.

# Exemplos de produções ambíguas

1.  $A \rightarrow AA$
2.  $A \rightarrow A \alpha A$
3.  $A \rightarrow \alpha A \mid A\beta$
4.  $A \rightarrow \alpha A \mid \alpha A\beta A$

# Quais Gramáticas são ambíguas?

(1)  $S \rightarrow bA \mid aB$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

(2) A gramática usada para gerar expressões aritméticas na notação **posfix** na linguagem APL:

$S \rightarrow SS + \mid SS - \mid SS * \mid x \mid y$

ou na notação **prefix**:

$S \rightarrow +SS \mid -SS \mid *SS \mid x \mid y$

(3) A gramática para gerar parênteses aninhados:

$S \rightarrow (S) \mid ( ) \mid SS$

(4) A gramática que define os operadores lógicos and e or

$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid (E) \mid a$