

# Análise Sintática

Profa. Heloise Manica Paris Teixeira

Parte 1

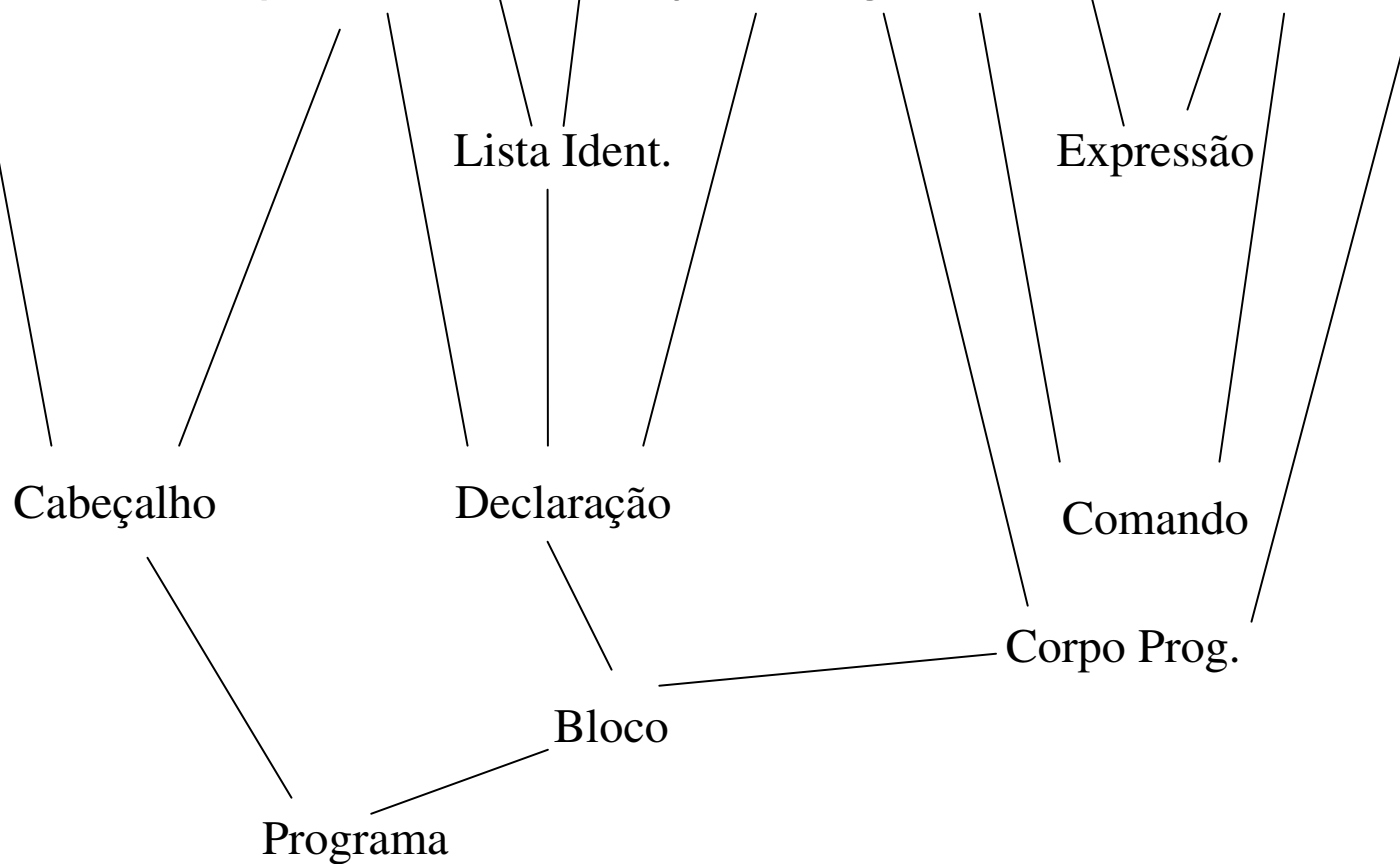
Parte destes slides foram cedidos pela Prof. Valéria Feltrin (DIN-UEM)

# Analizador sintático

- Analizador sintático ou ***parser***:
  - processo principal do compilador
    - Coordena as outras etapas
- O parser é um algoritmo que, recebendo como entrada uma cadeia  $\alpha$ , emite como saída:
  - Aceitação de  $\alpha$ , se  $\alpha$  pertence à linguagem, ou
  - ERRO, se  $\alpha$  não pertence à linguagem.
- Funções:
  - A análise sintática deve reconhecer a estrutura global do programa, por exemplo, verificando se **comandos, declarações, expressões**, etc. **têm as regras de composição respeitadas**.
    - Verificar a boa formação do programa: quais cadeias pertencem à linguagem
    - Construção da árvore sintática do programa
  - Tratar erros

# Analizador sintático

*Program Exemplo; Var A, B: byte; begin A := B + 0,5; end.*



# Análise sintática

- Dada uma *gramática livre de contexto* (G), queremos descobrir se uma dada cadeia  $x$  pertence ou não à linguagem da gramática ( $L(G)$ )
- No caso afirmativo, queremos adicionalmente descobrir a maneira pela qual a cadeia pode ser derivada seguindo as regras da gramática

# Análise sintática

- As regras abaixo definem o comando **WHILE** do **Pascal**, as palavras em maiúsculo representam terminais e minúsculo os não-terminais:

```
comando          → comandoWhile
                  | comandoIf
                  | comandoAtrib,
                  | ...
comandoWhile     → WHILE expr_bool DO comando;
expr_bool        → expr_arit < expr_arit
                  | expr_arit > expr_arit
                  | ...
expr_arit        → expr_arit * termo
                  | termo
                  | ...
termo            → expr_arit
                  | NÚMERO
                  | IDENTIFICADOR
```

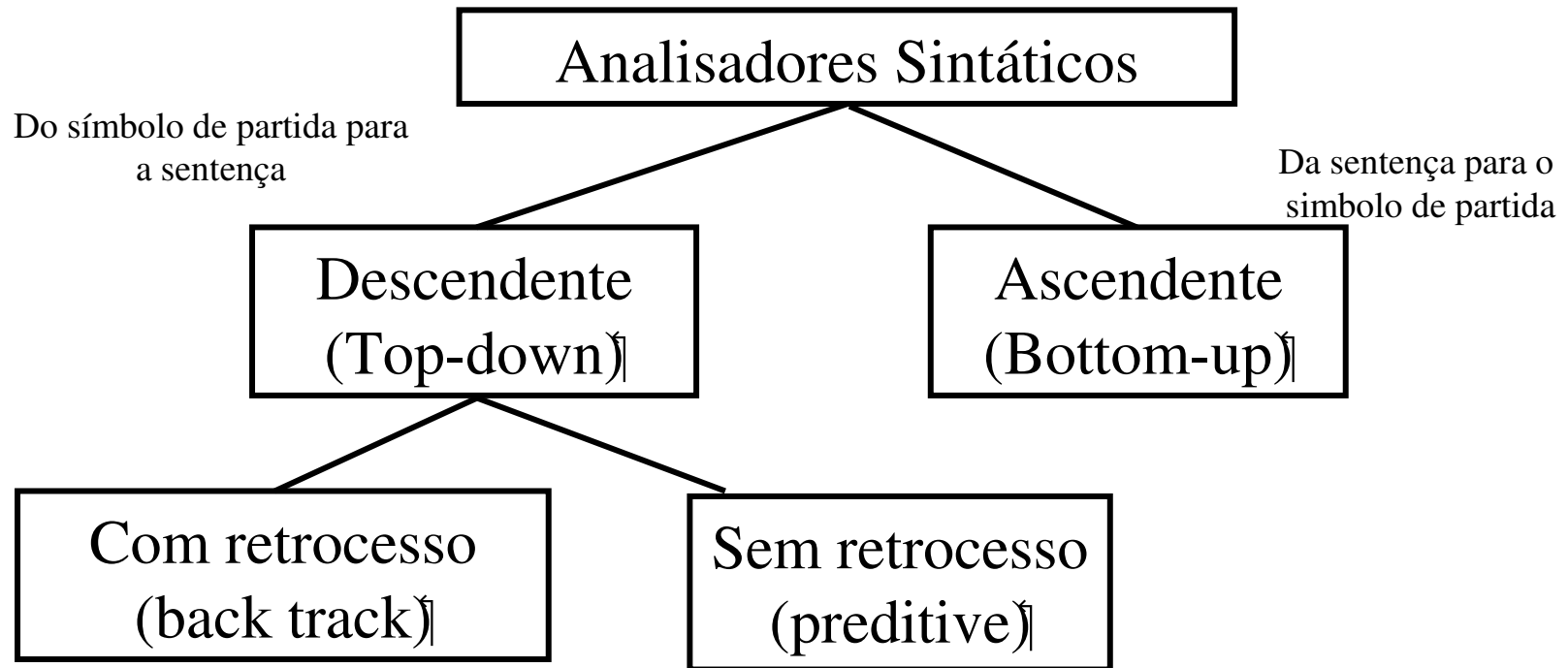
# Análise sintática

- Análise sintática (método descendente) top-down:
  - é uma análise onde se procura, a partir do símbolo inicial da gramática, chegar à cadeia que está sendo analisada **progredindo** nas regras de produção;
  - Esta análise equivale à seqüência dos números das regras de produção utilizadas  $S \Rightarrow \alpha$  através de derivações mais à esquerda.

# Análise sintática

- Análise sintática (método ascendente) bottom-up:
  - é uma análise onde se procura, a partir da cadeia que está sendo analisada, chegar ao símbolo inicial da gramática **regredindo** nas regras de produção;
  - Esta análise equivale à seqüência invertida dos números das regras de produção utilizadas  $S \Rightarrow \alpha$  através de derivações mais à direita.

# Análise sintática





# Análise sintática

- Tanto os métodos descendentes como os ascendentes constroem a árvore *da esquerda para direita*
  - A escolha das regras deve se basear na cadeia a ser reconhecida, que é lida da esquerda para a direita

Imaginem um compilador que começasse a partir do fim do código-fonte!

# Exemplo

- Considere a seguinte gramática

1.  $E \rightarrow E + T$

2.  $E \rightarrow T$

3.  $T \rightarrow T * F$

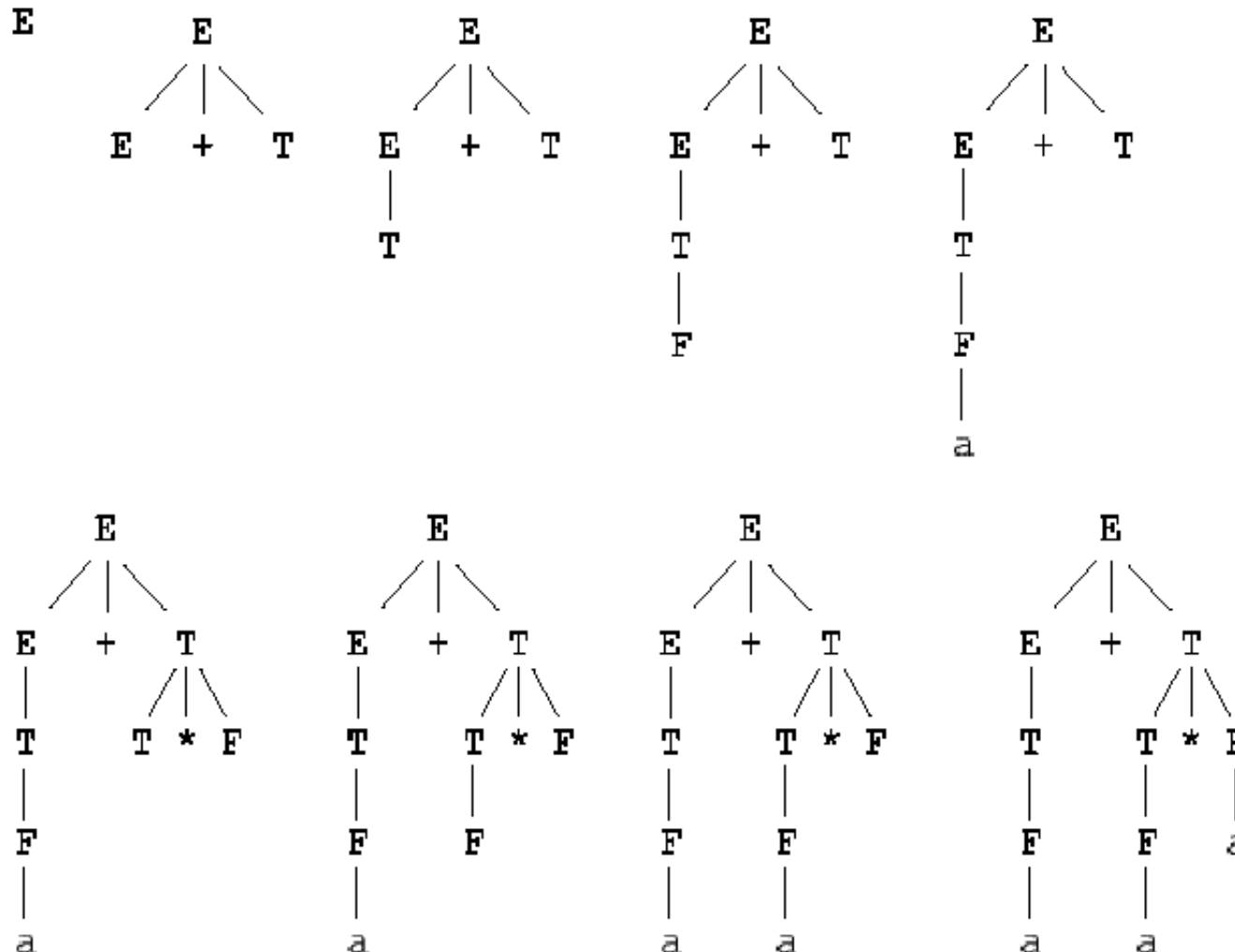
4.  $T \rightarrow F$

5.  $F \rightarrow ( E )$

6.  $F \rightarrow a$

e a cadeia  $x = \mathbf{a+a*a}$

# Construção da árvore de derivação de $a+a*a$ usando-se um método **descendente**



$E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow ( E )$   
 $F \rightarrow a$

Note que as regras são consideradas na mesma ordem em que as regras seriam usadas em uma derivação esquerda

		$\begin{array}{c} T \\   \\ F \\   \\ a + a * a \end{array}$
$\begin{array}{c} a + a * a \\ \\ E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} F \\   \\ a + a * a \\ \\ E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} T \\   \\ F \\   \\ a + a * a \end{array}$
$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$
$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$	$\begin{array}{c} E \\   \\ T \\   \\ F \\   \\ a + a * a \end{array}$

Construção da  
árvore de  
derivação de  
 $a+a*a$  usando  
um método  
**ascendente**

$E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow ( E )$   
 $F \rightarrow a$

Nesse caso, a ordem  
das regras  
corresponde à  
derivação invertida

# Análise Sintática

- Embora a árvore de derivação seja usada para descrever os métodos de análise, na prática ela não é efetivamente construída
- A única estrutura de dados necessária para o processo de análise sintática é uma **pilha**
  - Guarda informação sobre os nós da árvore de derivação relevantes em cada fase do processo

# Análise Descendente

- A representação do processo será feita através de configurações  $(\alpha, \gamma)$ 
  - $\alpha$ : conteúdo da pilha
  - $\gamma$ : *resto* da entrada ainda não analisada
- Por convenção, vamos supor que o topo da pilha fica à esquerda
  - o primeiro símbolo de  $\alpha$  é o símbolo do topo da pilha

# Análise descendente

- Duas formas de transição de uma configuração para outra:
  - ***expansão de um não-terminal pela regra  $A \rightarrow \beta$*** 
    - permite passar da configuração  $(A\alpha, y)$  para a configuração  $(\beta\alpha, y)$
  - ***verificação de um terminal  $a$*** 
    - permite passar da configuração  $(a\alpha, ay)$  para a configuração  $(\alpha, y)$ 
      - serve para retirar terminais do topo da pilha e expor no topo da pilha o próximo não-terminal a ser expandido
  - **Configuração inicial:**  $(S, x)$ 
    - $S$  o símbolo inicial da gramática e  $x$  é cadeia a ser analisada
  - **Configuração final:**  $(\epsilon, \epsilon)$ 
    - pilha vazia e a entrada/cadeia toda considerada
  - Por enquanto, nada foi dito sobre a forma de escolha da regra a ser aplicada

# Configurações sucessivas de um analisador descendente para a cadeia x

	pilha	(resto da) entrada	derivação esquerda
	E	a+a*a	E
— <i>Topo</i>	E+T	a+a*a	$\Rightarrow$ E+T
— <i>da pilha</i>	T+T	a+a*a	$\Rightarrow$ T+T
	F+T	a+a*a	$\Rightarrow$ F+T
	a+T	a+a*a	$\Rightarrow$ a+T
	+T	+a*a	
	T	a*a	
	T*F	a*a	$\Rightarrow$ a+T*F
	F*F	a*a	$\Rightarrow$ a+F*F
	a*F	a*a	$\Rightarrow$ a+a*F
	*F	*a	
	F	a	
	a	a	=
	$\epsilon$	$\epsilon$	

$E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow ( E )$   
 $F \rightarrow a$




# Análise ascendente

- A representação do processo também será feita através de configurações  $(\alpha, \mathbf{y})$ 
  - $\alpha$  e  $\mathbf{y}$  representam, respectivamente, o conteúdo da pilha e o *resto* da entrada ainda não analisada
- Entretanto, a convenção sobre o topo da pilha é invertida: fica à direita
  - o último símbolo de  $\alpha$  é o símbolo do topo

# Análise ascendente

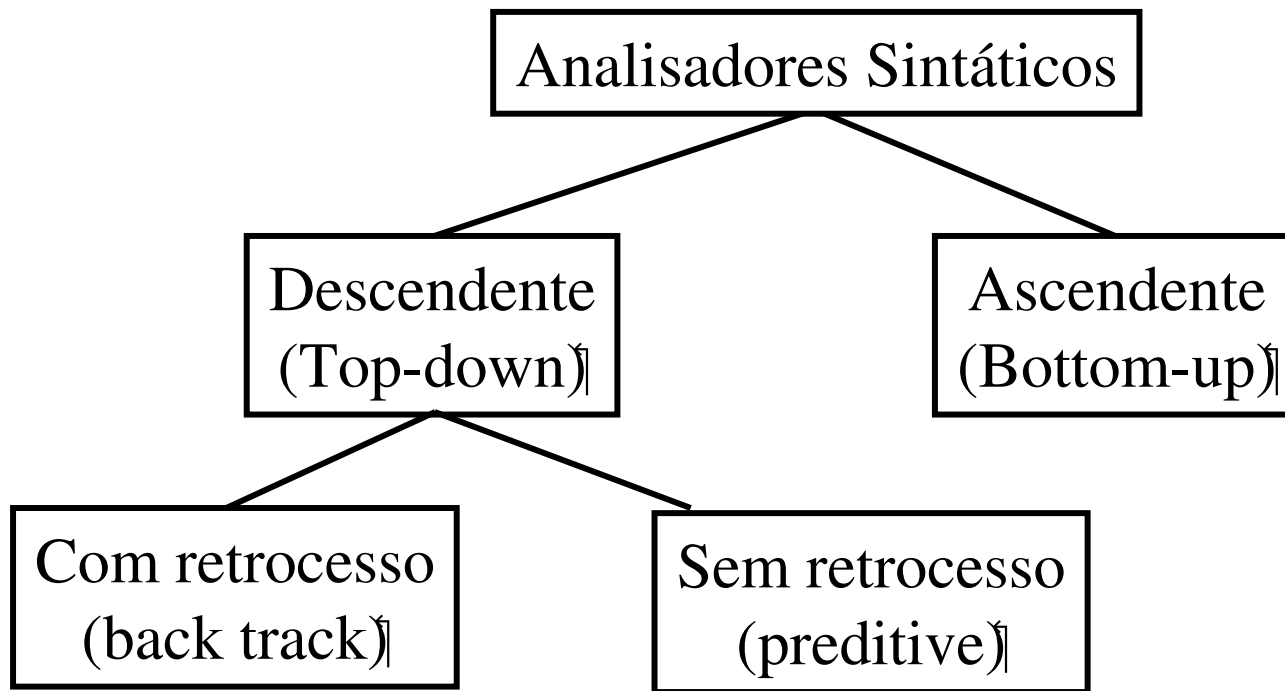
- Duas formas de transição de uma configuração para outra:
  - **redução regra  $A \rightarrow \beta$** 
    - permite passar da configuração  $(\beta\alpha, y)$  para a configuração  $(A\alpha, y)$
  - **empilhamento ou deslocamentos de um terminal  $a$** 
    - permite passar da configuração  $(\alpha, ay)$  para a configuração  $(a\alpha, y)$ 
      - serve para retirar terminais da entrada e colocá-lo no topo da pilha
  - **Configuração inicial:  $(\epsilon, x)$** 
    - a pilha está vazia e  $x$  é cadeia a ser analisada
  - **Configuração final:  $(S, \epsilon)$** 
    - cadeia toda considerada é reduzida para  $S$

# Configurações sucessivas de um analisador ascendente para a cadeia x

Pilha	(resto da) entrada	derivação direita (invertida)
$\varepsilon$	a+a*a	a+a*a
a	+a*a	
F	+a*a	$\Leftarrow F+a*a$
T	+a*a	$\Leftarrow T+a*a$
E	+a*a	$\Leftarrow E+a*a$
E+  $\leftarrow$ <i>Topo</i>	a*a	
E+a	*a	
E+F	*a	$\Leftarrow E+F*a$
E+T	*a	$\Leftarrow E+T*a$
E+T*	a	
E+T*a	$\varepsilon$	
E+T*F	$\varepsilon$	$\Leftarrow E+T*F$
E+T	$\varepsilon$	$\Leftarrow E+T$
E	$\varepsilon$	$\Leftarrow E$

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

# Análise Sintática Descendente (ASD) com retrocesso



# Análise Sintática Descendente (ASD) com retrocesso

- Quando a gramática permite, em um determinado estágio da derivação, a aplicação de mais de uma regra
  - Isso ocorre quando o mesmo símbolo terminal aparece no início do lado direito de mais de uma regra de produção. Exemplo:
    - $A \rightarrow a\alpha$
    - $A \rightarrow a\beta$
- Características:
  - Método de tentativa e erro: tenta todas as possibilidades
  - Ineficiente, em geral **não** é usado no reconhecimento de linguagens de programação
  - Desvantagem: dificuldade de se restaurar a situação no ponto de escolha e o atraso que isto provoca
- Funcionamento:
  - A cada passo, escolhe uma regra e aplica
  - Se falhar em algum ponto, retrocede e escolhe uma outra regra
  - O processo termina quando a cadeia é reconhecida ou quando as regras se esgotaram e a cadeia não foi reconhecida

# ASD com retrocesso

- Exemplo: Considere a gramática

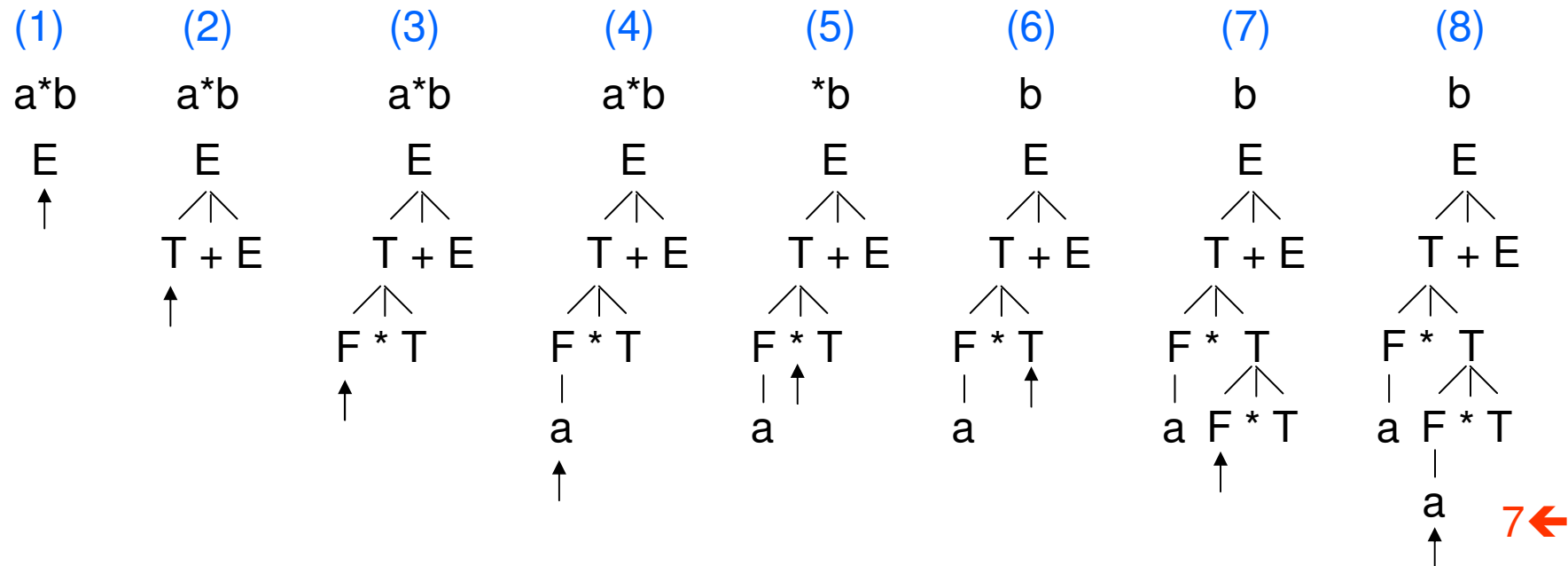
1.  $E \rightarrow T + E \mid T$

2.  $T \rightarrow F^* T \mid F$

3.  $F \rightarrow a \mid b \mid (E)$

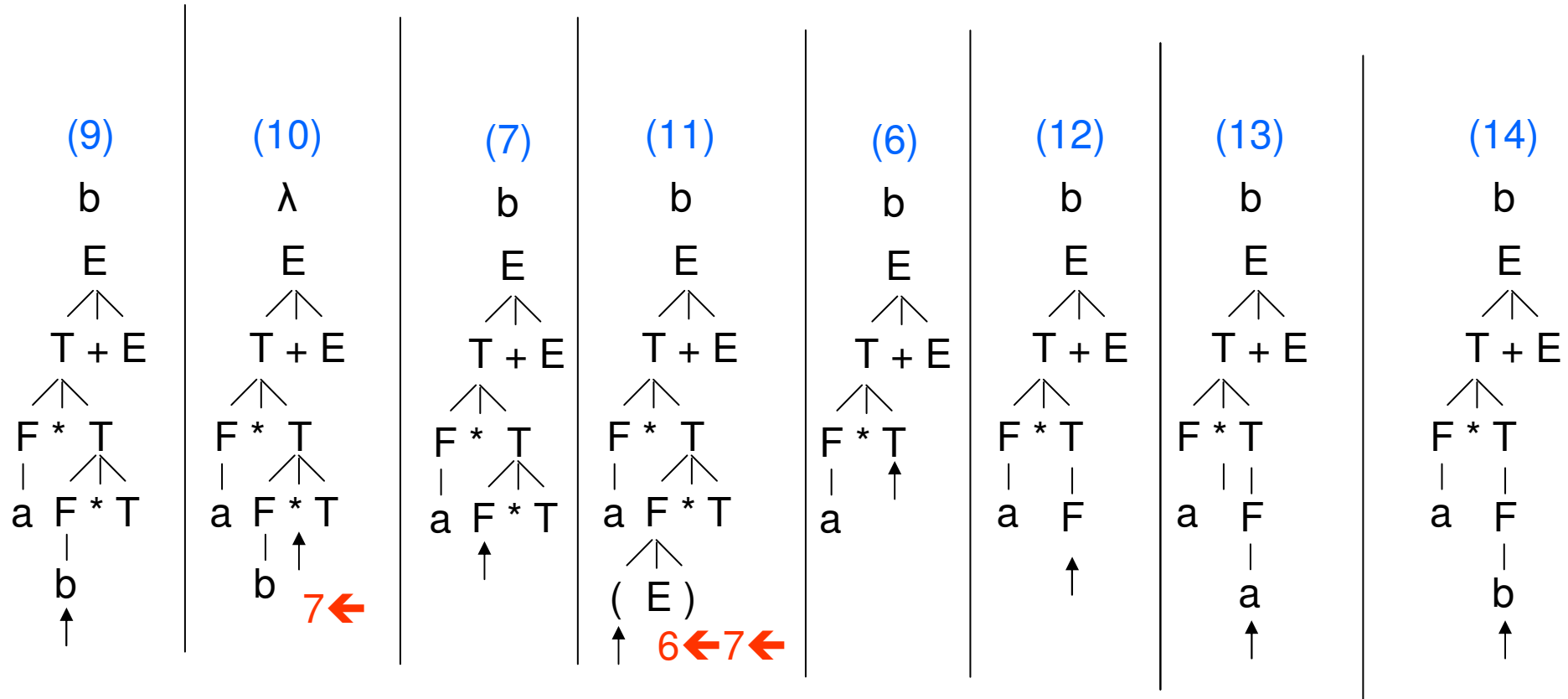
Reconhecer a cadeia  $a^*b$

# ASD com retrocesso



1.  $E \rightarrow T + E \mid T$
2.  $T \rightarrow F * T \mid F$
3.  $F \rightarrow a \mid b \mid (E)$

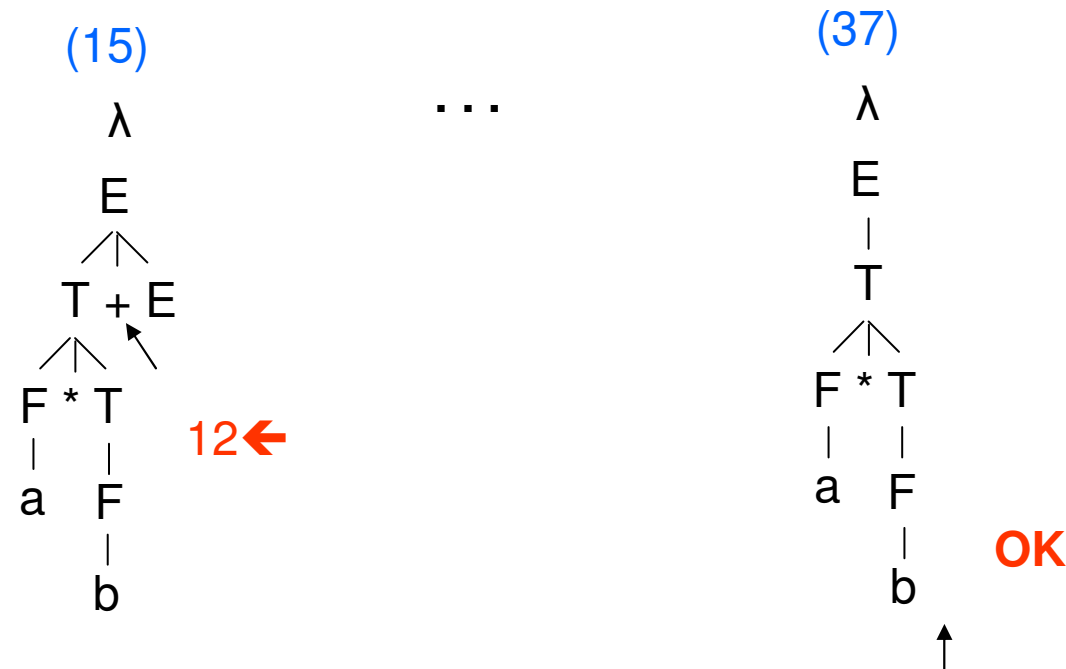
# ASD com retrocesso



1.  $E \rightarrow T + E \mid T$
2.  $T \rightarrow F * T \mid F$
3.  $F \rightarrow a \mid b \mid (E)$



# ASD com retrocesso



- O número de derivações pode ser uma função exponencial do tamanho da cadeia

# **Coletando informação sobre Gramáticas**

# Coletando informação sobre Gramáticas

- Para decidir quais as regras a serem usadas durante o processo da análise sintática algumas informações serão necessárias.
    - três tipos de informação sobre símbolos não terminais em gramáticas livres de contexto.
1. se  $A$  gera ou não a cadeia vazia  $\epsilon$ .
  2. quais são os símbolos terminais iniciadores das cadeias geradas a partir de  $A$ : se  $A \Rightarrow^* \alpha$ , que terminais podem aparecer como primeiro símbolo de  $\alpha$ .
  3. quais são os símbolos terminais seguidores de  $A$ : ou seja, se  $S \Rightarrow^* \alpha A \beta$ , que terminais podem aparecer como primeiro símbolo de  $\beta$ .

# Coletando informação sobre Gramáticas

**Algoritmo 1:** Identificação dos nãoterminais que derivam a cadeia vazia  $\epsilon$ .

O algoritmo trabalha com uma lista  $L$  de regras, que vai sendo alterada pela execução. Se a gramática não tem regras e/ou símbolos inúteis, o algoritmo termina com todos os nãoterminais marcados *sim* (nãoterminal gera  $\epsilon$ ), ou *não* (nãoterminal não gera  $\epsilon$ ).

0. Inicialmente a lista  $L$  contém todas as regras de  $G$ , exceto aquelas que contém um símbolo terminal. (Regras cujo lado direito têm um terminal não servem para derivar  $\epsilon$ .)
1. Se existe um nãoterminal  $A$  sem regras (não interessa se  $A$  originalmente não tinha regras, ou se todas as regras foram retiradas de  $L$ ) marque  $A$  com *não*.
2. Se um nãoterminal  $A$  tem uma regra  $A \rightarrow \epsilon$ , retire de  $L$  todas as regras com  $A$  do lado esquerdo, e retire todas as ocorrências de  $A$  do lado direito das regras de  $L$ . Marque  $A$  com *sim*. (Note que uma regra  $B \rightarrow C$  se transforma em  $B \rightarrow \epsilon$ , se a ocorrência de  $C$  do lado direito for retirada.)
3. Se uma regra tem do lado direito um nãoterminal marcado *não*, retire a regra. (Regras cujo lado direito têm um nãoterminal que não deriva  $\epsilon$  não servem para derivar  $\epsilon$ .)
4. Repita os passos 1, 2, 3 até que nenhuma nova informação seja adicionada.

# Coletando informação sobre Gramáticas

## Identificação de não terminais que geram cadeias vazias

**Exemplo** : Seja a gramática  
executar o algoritmo para verificar se P, A, B, C, D geram a cadeia vazia  $\epsilon$ .

- |    |   |    |  |
|----|---|----|--|
| 1. | $P \rightarrow A B C D$   | 5. | <del><math>\rightarrow</math></del> $\text{---} B \rightarrow \text{---} B b \text{---}$ |
| 2. | $A \rightarrow \epsilon$  | 6. | <del><math>\rightarrow</math></del> $\text{---} C \rightarrow \text{---} c \text{---}$   |
| 3. | <del><math>\rightarrow</math></del> $\text{---} A \rightarrow \text{---} a \text{---} A \text{---}$ | 7. | $C \rightarrow A B$  |
| 4. | $B \rightarrow \epsilon$  | 8. | <del><math>\rightarrow</math></del> $\text{---} D \rightarrow \text{---} d \text{---}$   |

Inicialmente, as regras 3, 5, 6 e 8 não são incluídas em L por causa dos terminais do lado direito. L contém inicialmente

1.  $P \rightarrow A B C D$
2.  $A \rightarrow \epsilon$
4.  $B \rightarrow \epsilon$
7.  $C \rightarrow A B$

Inicialmente a lista L contém todas as regras, exceto aquelas que contém um símbolo terminal, pois esta não gera a cadeia vazia!

Passo 1: se existir um não-terminal sem regras, marque como NÃO

Assim, D é marcado como NÃO

# Coletando informação sobre Gramáticas

1.  $P \rightarrow \cancel{A} B C D$   
~~2.  $A \rightarrow \epsilon$~~   
4.  $B \rightarrow \epsilon$   
7.  $C \rightarrow \cancel{A} B$



1.  $P \rightarrow \cancel{B} C D$   
~~4.  $B \rightarrow \epsilon$~~   
7.  $C \rightarrow \cancel{B}$

**Passo 2:** Se um não terminal A tem uma regra  $A \rightarrow \epsilon$ , retire todas as regras com A do lado esquerdo, e retire todas as ocorrências de A do lado direito das regras de L e marque A com SIM.

Assim, A é marcado como SIM e retira-se sua ocorrência das regras em que A esta do lado direito.

Como B tem uma regra com lado direito vazio, pode ser marcado *sim*, pelo passo 2, que também retira a regra 4. As ocorrências de B nas regras 1 e 7 podem ser retiradas, e portanto L contem

1.  $P \rightarrow C D$   
7.  $C \rightarrow \epsilon$

NÃO = D,  
SIM = A, B

# Coletando informação sobre Gramáticas

1.  $P \rightarrow \cancel{C} D$   
~~7.  $C \rightarrow \epsilon$~~

NÃO = D,  
SIM = A, B, C

Como C tem uma regra com lado direito vazio, pode ser marcado *sim*, pelo passo 2, que também retira a regra 7. A ocorrência de C na regra 1 pode ser retirada, e portanto L contém

1.  $P \rightarrow D$

**Passo 3:** Se uma regra tem do lado direito um não terminal marcado como NÃO, retire a regra.

Passo 1: se existir um não-terminal sem regras, marque como NÃO

NÃO = D, P  
SIM = A, B, C

Note que C não deriva  $\epsilon$  diretamente, mas sim em três passos, por exemplo, através da derivação

$C \Rightarrow A B \Rightarrow B \Rightarrow \epsilon.$

# Coletando informação sobre Gramáticas

- Cálculo dos Iniciadores (FIRST)

Formalmente, podemos definir os símbolos iniciadores de um nãoterminal  $A$  através de um conjunto  $\text{First}(A)$

$$\text{First}(A) = \{ a \mid a \text{ é um terminal e } A \Rightarrow^* a\alpha, \text{ para alguma cadeia } \alpha \text{ qualquer} \}$$

- Algoritmo2: Calculo First(x)

1. Se  $x \in V_t$ , então  $\text{FIRST}(x) = \{x\}$

2. Se  $x \in V_n$  e  $x \rightarrow a\alpha \in P$ , então coloque **a** em  $\text{FIRST}(x)$ ; da mesma forma, se

$x \rightarrow \lambda \in P$  coloque  $\lambda$  em  $\text{FIRST}(x)$



# Coletando informação sobre Gramáticas

- Algoritmo2 (continuação)

3. Se  $x \rightarrow y_1 y_2 \dots y_n \in P$ , então para todo  $y_1 y_2 \dots y_i \in V^n$  ( $i \leq n$ ) faça:

- adicione  $\text{FIRST}(y_1) - \{\lambda\}$  em  $\text{FIRST}(x)$ .
- para  $j=2..i$  adicione  $\text{FIRST}(y_j) - \{\lambda\}$  em  $\text{FIRST}(x)$  se  $\text{FIRST}(y_{j-1})$  contiver  $\{\lambda\}$ .
- se  $\lambda \in \text{FIRST}(y_j)$  para  $j=1..n$  então adicione  $\lambda$  em  $\text{FIRST}(x)$ .

**Em outros termos:**

- coloque  $\text{FIRST}(y_1)$ , exceto  $\lambda$ , em  $\text{FIRST}(x)$
- se  $\lambda \in \text{FIRST}(y_1)$  então:
  - coloque  $\text{FIRST}(y_2)$ , exceto  $\lambda$ , em  $\text{FIRST}(x)$
  - se  $\lambda \in \text{FIRST}(y_2)$  então:
    - coloque  $\text{FIRST}(y_3)$ , exceto  $\lambda$ , em  $\text{FIRST}(x)$
    - se  $\lambda \in \text{FIRST}(y_3)$  então:
      - coloque  $\text{FIRST}(y_4)$ , exceto  $\lambda$ , em  $\text{FIRST}(x)$
      - : : : : :
    - até  $i$
    - finalmente, se para todo  $y_j$  ( $j=1..i$ ) o  $\text{FIRST}(y_j)$  contiver  $\lambda$  então adicione  $\lambda$  em  $\text{FIRST}(x)$ .

# Coletando informação sobre Gramáticas

## Exercícios

- 1) Dadas as seguintes regras de gramática, encontre o conjunto *first* para os símbolos não-terminais de cada uma delas:

$$\text{First}(S) = \{a, b, f, g, c, \lambda\}$$

a)  $S \rightarrow A \mid B$

$$\text{First}(A) = \{a, b, f, g, c, \lambda\}$$

$$A \rightarrow aAS \mid BD$$

$$\text{First}(B) = \{b, f, \lambda\}$$

$$B \rightarrow bB \mid fAC \mid \lambda$$

$$C \rightarrow cC \mid BD$$

$$\text{First}(C) = \{c, b, f, g, \lambda\}$$

$$D \rightarrow gD \mid C \mid \lambda$$

$$\text{First}(D) = \{g, c, b, f, \lambda\}$$

# Coletando informação sobre Gramáticas

b)  $S \rightarrow ABd$

$$A \rightarrow aA \mid \lambda$$

$$B \rightarrow bB \mid cA \mid AC$$

$$C \rightarrow cB \mid \lambda$$

$$\text{First}(S) = \{b, c, a, d\}$$

$$\text{First}(A) = \{a, \lambda\}$$

$$\text{First}(B) = \{b, c, a, \lambda\}$$

$$\text{First}(C) = \{c, \lambda\}$$

c)  $S \rightarrow A \mid BC$

$$A \rightarrow aAS \mid D$$

$$B \rightarrow bB \mid fAC \mid \lambda$$

$$C \rightarrow cC$$

$$D \rightarrow gD \mid C \mid \lambda$$

$$\text{First}(S) = \{a, g, c, b, f, \lambda\}$$

$$\text{First}(A) = \{a, g, c, \lambda\}$$

$$\text{First}(B) = \{b, f, \lambda\}$$

$$\text{First}(C) = \{c\}$$

$$\text{First}(D) = \{g, c, \lambda\}$$

# Coletando informação sobre Gramáticas

d)  $S \rightarrow aA \mid bB$

$A \rightarrow aAS \mid BD$

$B \rightarrow bB \mid fAC \mid \lambda$

$C \rightarrow cC \mid Dd$

$D \rightarrow gD \mid C \mid \lambda$

$\text{First}(S) = \{ a, b \}$

$\text{First}(A) = \{ a, b, d, f, g, c, \lambda \}$

$\text{First}(B) = \{ b, f, \lambda \}$

$\text{First}(C) = \{ c, g, d \}$

$\text{First}(D) = \{ g, c, d, \lambda \}$