

# Compiladores

## Aula 09: Adequação de Gramáticas (Top-Down)

Prof. Aléssio Miranda Júnior  
alessio@cefetmg.br

CEFET-MG - Campus Timóteo  
Dep. Engenharia de Computação

Março de 2026

- 1 O Desafio do Parsing Top-Down
- 2 Eliminação de Recursão à Esquerda
- 3 Fatoração à Esquerda
- 4 Conclusão e Próximos Passos
- 5 Referências

## Revisão: Parsing Descendente (Top-Down)

- O processo começa no Símbolo Inicial ( $S$ ) e tenta derivar a string de entrada construindo a árvore da raiz para as folhas.
- **A Implementação Prática:** Cada Não-Terminal da gramática se torna uma **função recursiva** na linguagem de programação (ex: C, Java).
- Esse método é conhecido como **Parser Descendente Recursivo** (*Recursive Descent Parser*).

**O Problema:** Uma função recursiva mal projetada causa o quê?

## Revisão: Parsing Descendente (Top-Down)

- O processo começa no Símbolo Inicial ( $S$ ) e tenta derivar a string de entrada construindo a árvore da raiz para as folhas.
- **A Implementação Prática:** Cada Não-Terminal da gramática se torna uma **função recursiva** na linguagem de programação (ex: C, Java).
- Esse método é conhecido como **Parser Descendente Recursivo** (*Recursive Descent Parser*).

**O Problema:** Uma função recursiva mal projetada causa o quê?

*Estouro de Pilha (Stack Overflow) devido a loops infinitos!*

Para que possamos programar um analisador Top-Down determinístico (que escolhe a regra certa apenas olhando o próximo token, sem tentar e voltar atrás - *backtracking*), a nossa GLC precisa atender a duas exigências fundamentais:

1. Não pode possuir **Recursão à Esquerda** (causa loop infinito).
2. Os lados direitos das produções de um mesmo não-terminal não podem começar com o mesmo símbolo (exige **Fatoração à Esquerda** para evitar não-determinismo).

## O Que é Recursão à Esquerda?

Uma gramática possui **recursão à esquerda** se existe um não-terminal  $A$  tal que há uma derivação:

$$A \Rightarrow^+ A\alpha$$

onde  $\alpha$  é qualquer cadeia de terminais e/ou não-terminais.

**Recursão à Esquerda Direta:** Ocorre na mesma regra de produção:

$$A \rightarrow A\alpha \mid \beta$$

*(Onde  $\beta$  não começa com  $A$ ).*

# O Efeito Catastrófico no Código

Imagine transformar a regra  $E \rightarrow E + T \mid T$  em código C:

## Tentativa de Parser Recursivo

```
void E() {
    if (lookahead == ???) {
        E();      // <--- LOOP INFINITO AQUI!
        match('+');
        T();
    } else {
        T();
    }
}
```

A função chama a si mesma antes de consumir qualquer token da entrada. O caso base nunca é atingido.

## Algoritmo: Eliminando a Recursão Direta

Podemos transformar a gramática matematicamente para usar **recursão à direita**, que consome tokens antes da chamada recursiva.

Dada a regra com recursão à esquerda:

$$A \rightarrow A\alpha \mid \beta$$

Substituimos por duas novas regras, introduzindo um novo não-terminal ( $A'$ ):

$$\begin{aligned} A &\rightarrow \beta A' \\ A' &\rightarrow \alpha A' \mid \epsilon \end{aligned}$$

## Exemplo Prático: Expressões Aritméticas

Lembram da nossa gramática não-ambígua da última aula?

$$E \rightarrow E + T \mid T$$

Aplicando a fórmula ( $A = E$ ,  $\alpha = +T$ ,  $\beta = T$ ):

### Gramática Transformada (Sem Recursão)

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \epsilon$$

*[Dica de Aula: Ir ao quadro e transformar também a regra do Termo:  $T \rightarrow T * F \mid F$  passo a passo com os alunos].*

## O Problema do Não-Determinismo (Backtracking)

Mesmo sem recursão à esquerda, o parser Top-Down precisa saber qual caminho seguir olhando apenas o próximo token da entrada (o *lookahead*).

Considere o comando `if` em C/Java:

$$\begin{aligned} \text{Comando} &\rightarrow \mathbf{if} ( Expr ) \text{Comando} \\ \text{Comando} &\rightarrow \mathbf{if} ( Expr ) \text{Comando} \mathbf{else} \text{Comando} \end{aligned}$$

Se o próximo token na entrada for `if`, qual das duas regras a função `Comando()` deve escolher? É impossível saber sem ler toda a expressão e o primeiro comando inteiro.

**Fatoração à Esquerda** é uma transformação na gramática útil para produzir parsers determinísticos.

A ideia é adiar a decisão. Quando duas ou mais produções de um mesmo não-terminal começam com o mesmo prefixo  $\alpha$ , nós "fatoramos" esse prefixo comum.

Dada uma regra da forma:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$

(Onde  $\alpha$  é o prefixo comum, não-vazio).

Substituímos introduzindo um novo não-terminal ( $A'$ ):

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta_1 \mid \beta_2 \end{aligned}$$

## Exemplo Prático: Fatorando o Comando `if`

Voltando ao nosso problema do `if-then-else`:

$$C \rightarrow \mathbf{if} (E) C \mid \mathbf{if} (E) C \mathbf{else} C$$

Identificamos o prefixo comum  $\alpha = \mathbf{if} (E) C$ . As partes divergentes são  $\beta_1 = \epsilon$  e  $\beta_2 = \mathbf{else} C$ .

### Gramática Fatorada

$$C \rightarrow \mathbf{if} (E) C C'$$

$$C' \rightarrow \mathbf{else} C \mid \epsilon$$

Agora o parser lê o "if", a expressão e o comando. Só depois ele olha o próximo token para decidir se chama o "else" ou encerra ( $C' \rightarrow \epsilon$ ).

O que alcançamos hoje?

- **Eliminação de Recursão à Esquerda:** Garante que nosso parser não entre em loop infinito.
- **Fatoração à Esquerda:** Garante que não precisaremos fazer *backtracking* (tentativa e erro), tornando o algoritmo eficiente ( $O(n)$ ).

**Na próxima aula:** Com a gramática "limpa" e adequada, formalizaremos os conjuntos **FIRST** e **FOLLOW** para construir automaticamente a Tabela de Análise Sintática Preditiva (A família de parsers LL(1)).

- **AHO, A. V. et al.** *Compiladores: Princípios, Técnicas e Ferramentas*. 2ª Edição. Cap. 4 (Seção 4.3 e 4.4).
- **COOPER, K.; TORCZON, L.** *Engineering a Compiler*. 2ª Edição. Cap. 3 (Seção 3.3).

**Obrigado!**

**Email:** [alessio@cefetmg.br](mailto:alessio@cefetmg.br)

**Web:** [alessiojr.com](http://alessiojr.com)