Curso: Engenharia de Computação

Linguagens Formais e Compiladores

Prof. Clayton J A Silva, MSc
clayton.silva@professores.ibmec.edu.br

ibmec

Análise Sintática

Recuperação de erros

- **Reconhecedor**: analisador sintático que efetua apenas a determinação se um programa está ou não sintaticamente correto.
- Além desse comportamento mínimo, o analisador sintático pode apresentar diversos níveis de resposta a erros:
  - determinação do local onde ocorreu o erro
  - tentar alguma forma de correção de erros – normalmente limitada aos casos mais simples

Recuperação de erros

- **Considerações importantes**:
  - O analisador deve tentar determinar a ocorrência do erro tão logo quanto possível
  - O analisador deve escolher um ponto para encerrar o processo, tentando analisar o máximo de código possível
  - O analisador deve tentar evitar que um erro gere uma sequência de mensagens espúrias – cascata de erros
  - O analisador deve evitar que a cascata de mensagens sem consumir novas entradas

Recuperação de erros método de ressincronização

1. Uma vez encontrado um erro, o tratamento de erros não deve ser afetado e possa continuar a analisar a entrada.
2. Ao detectar um erro o método deve **ressincronizar a entrada com o não-terminal que espera reconhecer** para prosseguir com a análise.
3. O **mecanismo básico** é associar cada procedimento recursivo a um parâmetro adicional composto por um conjunto de **tokens de ressincronização**.
4. Ao encontrar um erro, o **analisador varre à frente**, descartando as tokens que gerariam erros desnecessários

Recuperação de erros método de ressincronização

5. Uma vez encontrado um erro, o tratamento de erros não deve ser afetado e possa continuar a analisar a entrada.
6. As decisões importantes são quais tokens de ressincronização acrescentar a cada ponto da análise.
7. Geralmente os **conjuntos de Sequência** são candidatos importantes. Os **conjuntos Primeiros** também são importantes.

Conjunto Primeiro

- Seja X um símbolo gramatical (terminal ou não terminal), então o **conjunto composto de terminais** Primeiro(X) é definido por:
  1. Se X for terminal ou vazio então Primeiro(X) = {X}
  2. Se X for não terminal, de derivação $X \rightarrow X_1\ X_2\ \ldots\ X_n$, então Primeiro(X) **contém** Primeiro(X_1) menos vazio. Se para algum i<n, todos os conjuntos i Primeiro contiverem vazio, Primeiro(X) conterá Primeiro(X_{i+1}) menos vazio. Se todos os conjuntos Primeiro(X_i) contiverem vazio, então Primeiro(X) conterá também vazio.

Conjunto de Sequência

- Dado um não terminal A, o **conjunto** Sequência(A) **composto por terminais** e **$** é definido por:
  1. Se A for símbolo inicial, então $ pertence à Sequência(A)
  2. Se houver derivação B $\rightarrow$ α A γ então Primeiro(γ) sem vazio pertence à Sequência (A)
  3. Se houver derivação B $\rightarrow$ α A γ tal que vazio pertença a Primeiro(γ), então Sequência(A) contém Sequência(B)

Exemplo

Seja a gramática:

expr $\rightarrow$ *expr soma termo | termo*

soma $\rightarrow$ *+ | -*

termo $\rightarrow$ *termo mult fator | fator*

mult $\rightarrow$ ***

fator $\rightarrow$ *(expr) | número*

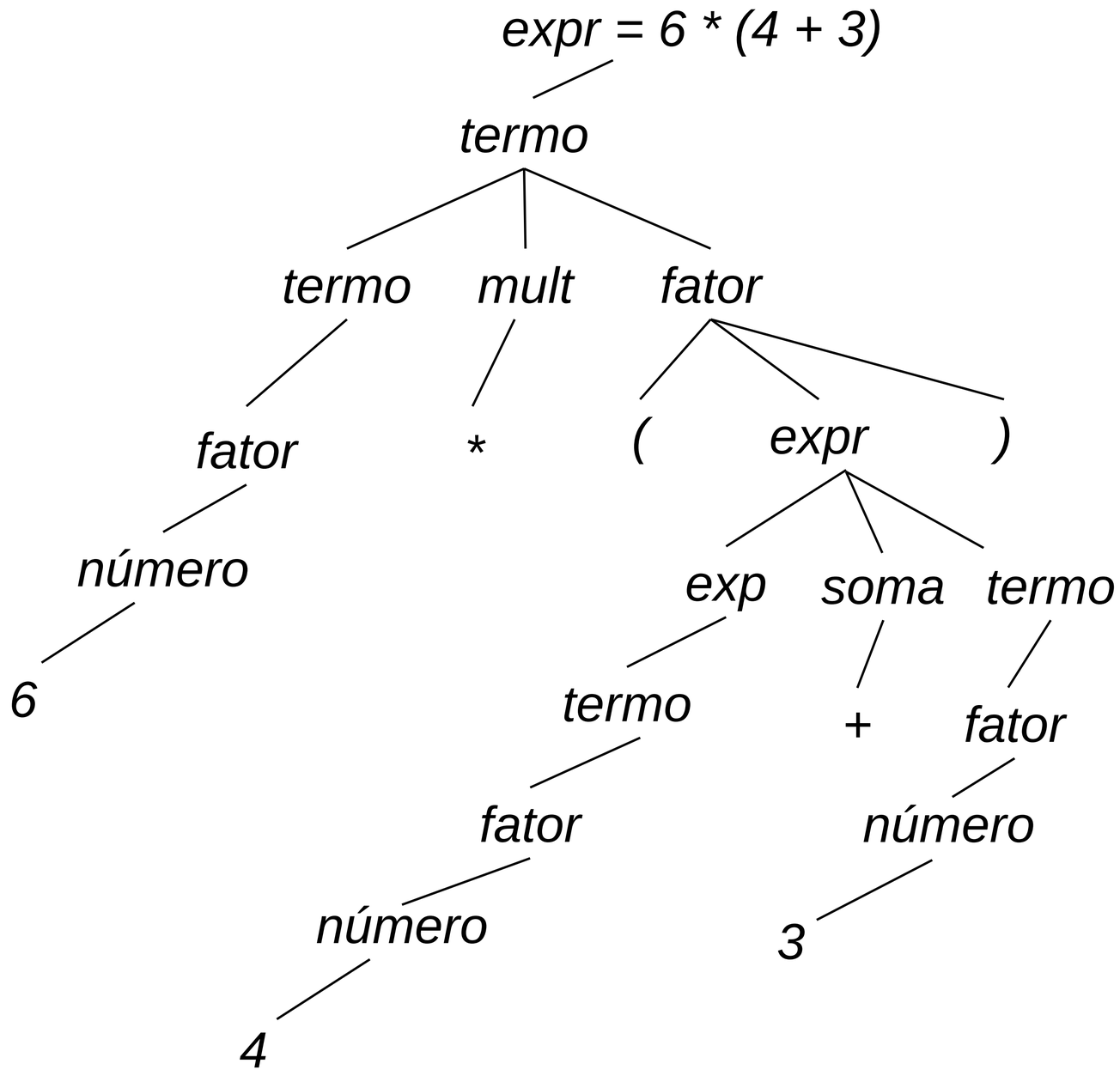
número $\rightarrow$ *0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Exemplo

Produções ou derivações enumeradas:

1. *expr → expr soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Dada a sentença:



Conjuntos Primeiro

1. *expr → expr soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Primeiro(expr) = { **(** , **número** }

Primeiro(termo) = { **(** , **número** }

Primeiro(fator) = { **(** , **número** }

Primeiro(soma) = { **+** , **-** }

Primeiro(mult) = { * }

Conjuntos de Sequência

1. *expr → expr soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Sequência(expr) = { **$** , }

Sequência(termo) = { **$** , }

...

Conjuntos de Sequência

1. *expr → expr soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Sequência(expr) = { **$** , **+** , **–** }

Sequência(soma) = { **(** , **número** }

Sequência(termo) = { **$** , + , – }

... Primeiro(soma) é acrescentado à Sequência(expr); Primeiro(termo) é acrescentado à Sequência(soma); na substituição de (2) em (1), Primeiro(soma) é acrescentada à Sequência(termo).

Conjunto de Sequência

1. *expr → expr soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Sequência(expr) = { **$** , **+** , **–** }

Sequência(soma) = { **(** , **número** }

Sequência(termo) = { **$** , + , – , ***** }

Sequência(mult) = { **(** , **número** }

Sequência(fator) = { ***** }

... Primeiro(mult) é acrescentado à Sequência(termo); Primeiro(fator) é acrescentado à Sequência(mult); e na substituição de (6) em (5), Primeiro(mult) é acrescentado à Sequência(fator).

Conjunto de Sequência

1. *expr → exp soma termo*
2. *expr → termo*
3. *soma → +*
4. *soma → -*
5. *termo → termo mult fator*
6. *termo → fator*
7. *mult → **
8. *fator → (expr)*
9. *fator → número*
10. *número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

Sequência(expr) = { **$** , **+** , **-** , **)** }

Sequência(soma) = { **(** , **número** }

Sequência(termo) = { **$** , + , - , ***** }

Sequência(mult) = { **(** , **número** }

Sequência(fator) = { ***** , **$** , **+** , **-** , **)**}

... Sequência(expr) é acrescentada à Sequência(fator); Terminal) é acrescentado à Sequência(fator) e à Sequência(expr)

Exemplo

Seja a gramática:

expr → *expr soma termo* | *termo*

soma → + | -

termo → *termo mult fator* | *fator*

mult → *

fator → (*expr*) | *número*

número → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Exemplo

Análise recursiva dependente

- AnaliseExpr

1. Em cada procedure de abstração, são definidos os terminais do **conjuntoPrimeiro** e os terminais do **conjuntoSequência** – N, M são definidas como variáveis globais, pois precisam ser chamadas:

```
65
66  void expr(){
67      char *tokenPrimeiro[] = {"(","0","1","2","3","4","5","6","7","8","9"};
68      N = 11;
69      char *tokenSequencia[] = {"$","+","-",")"};
70      M = 4;
71      verificaEntrada(tokenPrimeiro, tokenSequencia);
72      if (testaPertence(registro[tokenProximo].stringvalor,tokenPrimeiro,N)){
73          termo();
74          if (testaPertence(registro[tokenProximo].stringvalor,opsoma,2)) {
75              soma();
76              termo();
77          }
78      }
79      //verificaEntrada(tokenSequencia,tokenPrimeiro);
80  }
```

Exemplo
Análise recursiva dependente

- AnaliseExpr

2. Criação de uma procedure **`verificaEntrada()`** que verifica se o token de entrada à frente pertence ao conjuntoPrimeiro – em caso positivo, segue a análise sintática, em caso contrário, configura-se um **erro – chamada à procedure `erro()`**

```
166 void verificaEntrada(char *conjuntoPrimeiro[], char *conjuntoSequencia[]){
167     if (!(testaPertence(registro[tokenProximo].stringvalor,conjuntoPrimeiro,N))){
168         erro();
169         tokenProximo++;
170         //varreParaFrente(conjuntoPrimeiro , conjuntoSequencia);
171     }
172 }
```

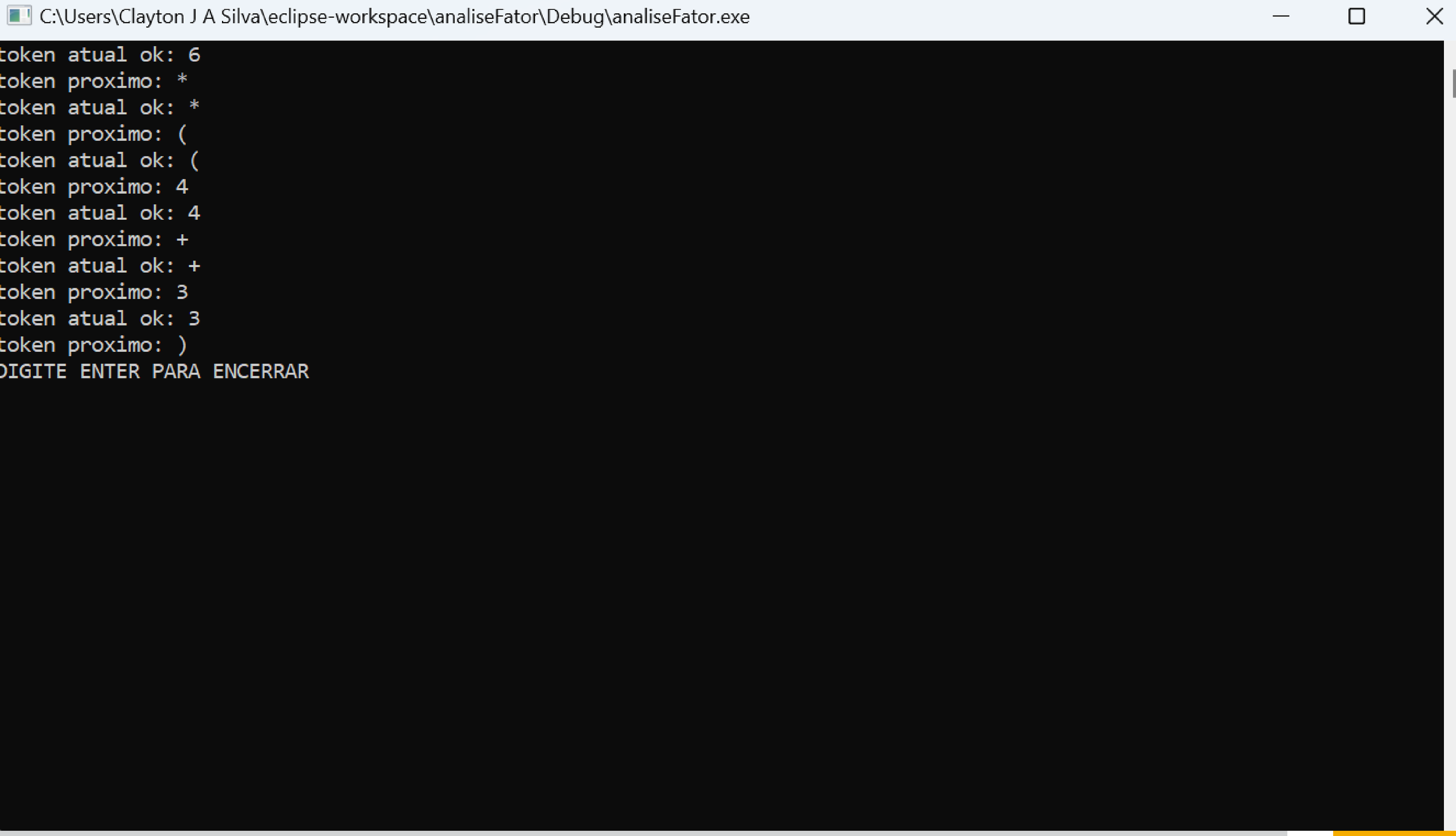
Exemplo

Análise recursiva dependente

- AnaliseExpr

3. Criação de uma procedure **`varreParaFrente()`** que **descarta os tokens desnecessários** até que se encontre o token da próxima produção ou derivação no conjuntoSequência.

```
173
174 void varreParaFrente(char *conjuntoPrimeiro[], char *conjuntoSequencia[]){
175     while (!(testaPertence(registro[tokenProximo].stringvalor,conjuntoPrimeiro,N)))
176         while (!(testaPertence(registro[tokenProximo].stringvalor,conjuntoSequencia,M)))
177             tokenProximo++;
178 }
179
```


C:\Users\Clayton J A Silva\eclipse-workspace\analiseFator\Debug\analiseFator.exe
token atual ok: 6
token proximo: *
token atual ok: *
token proximo: (
token atual ok: (
token proximo: 4
token atual ok: 4
token proximo: +
token atual ok: +
token proximo: 3
token atual ok: 3
token proximo:)
DIGITE ENTER PARA ENCERRAR

IBMEC.BR

/IBMEC

IBMEC

@IBMEC_OFICIAL

@IBMEC

Ibmec