

Aula 17 - Análise Sintática Top-Down

Ferramentas

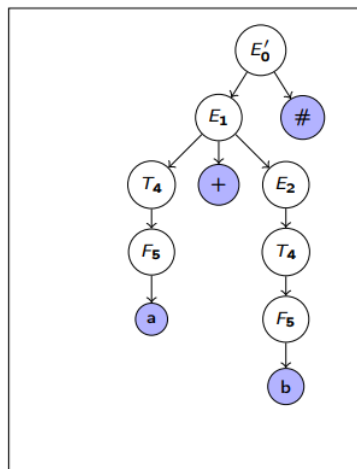
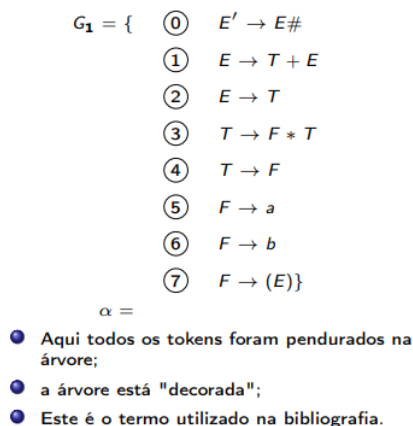
- Compilador é composto por várias ferramentas, porém é notável que a ferramenta que rege todo o processo é o analisador sintático
- Assim, dada uma gramática "basta" construir o analisador sintático para reconhecer se uma sentença pertence ou não a aquela gramática.
- Analisador semântico e gerador de código são implementados como "ações semânticas" dentro do analisador sintático

Analísadores Sintáticos Top-Down - Geral

- Esta categoria de analisadores sintáticos tem as seguintes características: lêem a entrada da esquerda para a direita e constroem a árvore de derivação de cima para baixo substituindo sempre o terminal mais à esquerda
- Para escrever um programa capaz de fazer a análise sintática Top-Down, existem várias abordagens: algoritmo da força bruta (complexidade combinatorial) ou adaptar a gramática para utilizar um Analisador Sintático preditivo que tem complexidade linear

Analísador Sintático com Retrocessos (backtracking) - Força Bruta

- Tenta construir todas as árvores possíveis onde os símbolos de entrada consigam ser pendurados
- Algoritmo recursivo:
 - Coloque S como raiz da árvore de derivação
 - Seja X a variável mais à esquerda
 - Selecione uma produção do tipo $X \rightarrow ABC$
 - Se houverem alguma, pendure
 - Se não houver, descarte a última produção
 - Reaplique o algoritmo



Analísadores Sintáticos Top-Down - Preditivo

- Escolhem uma produção baseada somente em duas informações: o token corrente e a árvore já construída
- A gramática tem que ser modificada para um determinado formato denominado LL(1) - (Left to right parsing, Leftmost derivation e examinam um único token à frente)
- Para isso, são necessárias mudanças na gramática: eliminar retrocessos, fatoração e eliminar recursão esquerda
- Eliminar retrocessos
 - Escolher a derivação baseado no token corrente, para isso usa-se um mecanismo formal para calcular o primeiro símbolo
 - Algoritmo Primeiro (First)
 - Definição Informal: Primeiro(A) é o conjunto de todos os terminais que começam qualquer sequência derivável de A
 - Definição Formal: se $A \Rightarrow^* x$ então $x \in \text{Primeiro}(A)$
 - O primeiro símbolo terminal (ou só "Primeiro") de cada variável pode ser obtido com o seguinte algoritmo:
 1. Desenhe uma tabela com quatro colunas;

2. Preencha a primeira coluna de cada linha com uma variável;
3. Preencha a segunda coluna de cada linha com os terminais ou variáveis que podem ser obtidos em uma derivação.
4. Preencha a terceira coluna com o fecho transitivo da segunda;
5. Copie os terminais da terceira coluna para a quarta coluna.
6. A quarta coluna contém os terminais válidos de cada variável.

■ Exemplo

$$\begin{aligned} S &\rightarrow AS|BA \\ A &\rightarrow aB|C \\ B &\rightarrow bA|d \\ C &\rightarrow c \end{aligned}$$

- Passo 3: Fecho transitivo
- Linhas B e C só tem terminais: copia

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABadbc	adbc
A	aC	aCc	ac
B	bd	bd	bd
C	c	c	c

• Fatoração

- A gramática pode ser escrita fatorando o primeiro termo comum, resultando em regras com duas notações possíveis

- $S \rightarrow a(AS|SA)$
- $S \rightarrow aX$
- $X \rightarrow AS|SA$

- Exemplo

- Fatore a gramática G_1 abaixo.

$$\begin{aligned} G_1 = \{ & E \rightarrow T + E | T \\ & T \rightarrow F * T | F \\ & F \rightarrow a|(E) \} \end{aligned}$$

•

$$\begin{aligned} G_2 = \{ & E \rightarrow T[+E|\epsilon] \\ & T \rightarrow F[*T|\epsilon] \\ & F \rightarrow a|(E) \} \end{aligned} \quad \begin{aligned} G_3 = \{ & E \rightarrow TE_1 \\ & E_1 \rightarrow +E|\epsilon \\ & T \rightarrow FT_1 \\ & T_1 \rightarrow *T|\epsilon \\ & F \rightarrow a|(E) \} \end{aligned}$$

- O símbolo ϵ (epsilon) indica a cadeia vazia. Não é um símbolo de entrada, e deve ser entendido como “qualquer outra coisa” e não consome tokens.

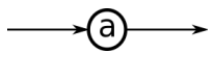
• Eliminar recursão esquerda

- Produções com recursão à esquerda seguem o seguinte formato: $A \rightarrow A\alpha|\beta$
- Modificar para: $A \rightarrow \beta\{\alpha\}$, onde $\{$ e $\}$ indicam repetição de zero a n vezes
- Exemplos

- $A \rightarrow Aa|b|c$
- $A \rightarrow (b|c)\{a\}$
- $E \rightarrow E + T_1 | T_2 | x$
- $E \rightarrow (T_2 | x)\{+T_1\}$
- $E \rightarrow E + T_1 | E - T_2 | T_3 | k$
- $E \rightarrow E(+T_1 | -T_2) | T_3 | k$
- $E \rightarrow (T_3 | k)\{(+T_1 | -T_2)\}$

• Escrever o programa

- Uma vez obtida uma gramática no formato LL(1), é possível converter as regras em um programa que pode ser implementado em praticamente todas as linguagens de programação.
- Terminais

- Diagrama sintático: 


- Programa:

```

1  procedimento E() {
2      Se token != "a" então
3          ErroFatal("Esperado 'a', recebido %s", token)
4      FimSe
5      token := Proximo(); (* consome token, pendura na árvore *)
6  }

```

○ Variáveis

- Diagrama sintático: 


- Programa:

```

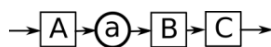
1  procedimento E() {
2      Se token <> Primeiro(A) então
3          ErroFatal("Esperados %s, recebido %s", Primeiro(A), token)
4      FimSe
5      A();
6  }

```

○ Sequências

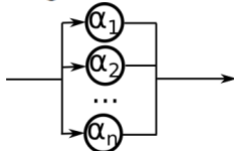
- Diagrama sintático: 

- Programa para $E \rightarrow AaBC$



○ Alternativos

- Diagrama sintático:



- Diagrama para $E \rightarrow A|a|B|C$

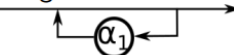
```

1  procedimento E() {
2      Caso token
3          ∈ Primeiro(A): A();
4          "a" : token := Proximo(); (* consome token, pendura na árvore *)
5          ∈ Primeiro(B): B();
6          ∈ Primeiro(C): C();
7          outros: ErroFatal("...");
8      FimCaso
9  }

```

○ Repetitivos

- Diagrama sintático:



- Programa para $E \rightarrow A\{, A\}$

```

1  procedimento E() {
2      Se token <> Primeiro(A) então
3          ErroFatal("...");
4      FimSe
5      enquanto (token == ",")
6          token := Proximo(); (* consome token, pendura na árvore *)
7          Se token <> Primeiro(A) então
8              ErroFatal("...");
9          FimSe
10         A();
11     FimEnquanto
12 }

```

○ Exemplo completo

1. Aumentar a gramática

$$\begin{aligned}
 G_1 = \{ & E \rightarrow E + T | E - T | T \\
 & T \rightarrow T * F | F \\
 & F \rightarrow a|(E) \}
 \end{aligned}$$

- Aumentar gramática:

$$\begin{aligned}
 G'_1 = \{ & E' \rightarrow E \# \\
 & E \rightarrow E + T | E - T | T \\
 & T \rightarrow T * F | F \\
 & F \rightarrow a|(E) \}
 \end{aligned}$$

2. Fazer a tabela de Primeiro

$$\begin{aligned}
G'_1 = \{ & E' \rightarrow E\# \\
& E \rightarrow E + T \mid E - T \mid T \\
& T \rightarrow T * F \mid F \\
& F \rightarrow a \mid (E) \}
\end{aligned}$$

• Tabela Primeiro:

$$\begin{aligned}
\text{Primeiro}(E') &= \{a, (\} \\
\text{Primeiro}(E) &= \{a, (\} \\
\text{Primeiro}(T) &= \{a, (\} \\
\text{Primeiro}(F) &= \{a, (\}
\end{aligned}$$

3. Fatorar

$$\begin{aligned}
G'_1 = \{ & E' \rightarrow E\# \\
& E \rightarrow E + T \mid E - T \mid T \\
& T \rightarrow T * F \mid F \\
& F \rightarrow a \mid (E) \}
\end{aligned}$$

$$\begin{aligned}
G''_1 = \{ & E' \rightarrow E\# \\
& E \rightarrow E(+ T \mid - T) \mid T \\
& T \rightarrow T * F \mid F \\
& F \rightarrow a \mid (E) \}
\end{aligned}$$

4. Eliminar Recursão Esquerda

$$\begin{aligned}
G''_1 = \{ & E' \rightarrow E\# \\
& E \rightarrow E(+ T \mid - T) \mid T \\
& T \rightarrow T * F \mid F \\
& F \rightarrow a \mid (E) \}
\end{aligned}$$

$$\begin{aligned}
G'''_1 = \{ & E' \rightarrow E\# \\
& E \rightarrow T \{ (+ T \mid - T) \} \\
& T \rightarrow F \{ * F \} \\
& F \rightarrow a \mid (E) \}
\end{aligned}$$

○ Código do exemplo

1. $F \rightarrow a \mid (E)$

```

1  procedimento F() {
2      caso token for:
3          "a":
4              token := Proximo(); (* consome token, pendura na árvore *)
5          "(":
6              token := Proximo(); (* consome token, pendura na árvore *)
7              E();
8              Se token != ")" então
9                  ErroFatal("...");
10             FimSe
11             token := Proximo(); (* consome token, pendura na árvore *)
12         outros:
13             ErroFatal("...");
14     FimCaso
15 }
```

2. $T \rightarrow F \{ * F \}$

```

1  procedimento T() {
2      F();
3      enquanto token == "*" faça:
4          token := Proximo(); (* consome token, pendura na árvore *)
5          F();
6      FimEnquanto
7  }
```

3. $E \rightarrow T \{ (+ T \mid - T) \}$

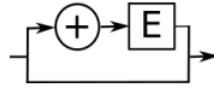
```

1  procedimento E() {
2      T();
3      enquanto token ∈ { " + ", "- " } faça:
4          token := Proximo(); (* consome token, pendura na árvore *)
5          T();
6      FimEnquanto
7  }
```

4. $E' \rightarrow E\#$

```
1  procedimento E'() {  
2      E();  
3      Se token != <fim de arquivo> então  
4          ErroFatal("...");  
5      FimSe  
6  }
```

5. $E_1 \rightarrow +E|$



```
1  procedimento E1() {  
2      Se token == "+" então  
3          token := Proximo(); (* consome token, pendura na árvore *)  
4          E();  
5      Senão  
6          (* ε, não faz nada! *)  
7      FimSe  
8  }
```