

第四章 语法分析

# 自顶向下分析概述

哈尔滨工业大学 陈鄞



## 自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
- 可以看成是从文法开始符号 $S$ 推导出词串 $w$ 的过程

➤ 例

文法

①  $E \rightarrow E + E$

②  $E \rightarrow E * E$

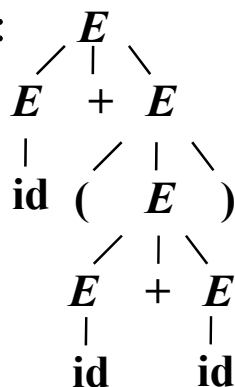
③  $E \rightarrow ( E )$

④  $E \rightarrow \text{id}$

输入

$\text{id} + ( \text{id} + \text{id} )$

分析树:



推导过程:  $E \Rightarrow E + E$

$\Rightarrow E + ( E )$

$\Rightarrow E + ( E + E )$

$\Rightarrow E + ( \text{id} + E )$

$\Rightarrow \text{id} + ( \text{id} + E )$

$\Rightarrow \text{id} + ( \text{id} + \text{id} )$

- 每一步推导中，都需要做两个选择
  - 替换当前句型中的哪个非终结符
  - 用该非终结符的哪个候选式进行替换

## 最左推导 (Left-most Derivation)

➤ 在**最左推导**中，总是选择每个句型的最左非终结符进行替换

➤ 例

文法
① $E \rightarrow E + E$
② $E \rightarrow E * E$
③ $E \rightarrow ( E )$
④ $E \rightarrow id$
输入
$id + ( id + id )$

推导过程

最左推导 ↓

$$\begin{aligned} E &\Rightarrow_{lm} E + E \\ &\Rightarrow_{lm} id + E \\ &\Rightarrow_{lm} id + ( E ) \\ &\Rightarrow_{lm} id + ( E + E ) \\ &\Rightarrow_{lm} id + ( id + E ) \\ &\Rightarrow_{lm} id + ( id + id ) \end{aligned}$$

↑ 最右归约

➤ 如果  $S \Rightarrow_{lm}^* \alpha$ ，则称  $\alpha$  是当前文法的最左句型 (left-sentential form)

## 最右推导 (Right-most Derivation)

➤ 在**最右推导**中，总是选择每个句型的最右非终结符进行替换

➤ 例

文法
① $E \rightarrow E + E$
② $E \rightarrow E * E$
③ $E \rightarrow ( E )$
④ $E \rightarrow \text{id}$
输入
$\text{id} + (\text{id} + \text{id})$

	推导过程	
	$E \Rightarrow_{rm} E + E$	
最右推导 ↓	$\Rightarrow_{rm} E + (E)$	最左归约 ↑
	$\Rightarrow_{rm} E + (E + E)$	
	$\Rightarrow_{rm} E + (E + \text{id})$	
	$\Rightarrow_{rm} E + (\text{id} + \text{id})$	
	$\Rightarrow_{rm} \text{id} + (\text{id} + \text{id})$	

➤ 在自底向上的分析中，总是采用最左归约的方式，因此把**最左归约**称为**规范归约**，而**最右推导**相应地称为**规范推导**

## 最左推导和最右推导的唯一性

$$E \Rightarrow E + E$$

$$\Rightarrow E + (E)$$

$$\Rightarrow E + (E + E)$$

$$\Rightarrow E + (id + E)$$

$$\Rightarrow id + (id + E)$$

$$\Rightarrow id + (id + id)$$

$$E \Rightarrow E + E$$

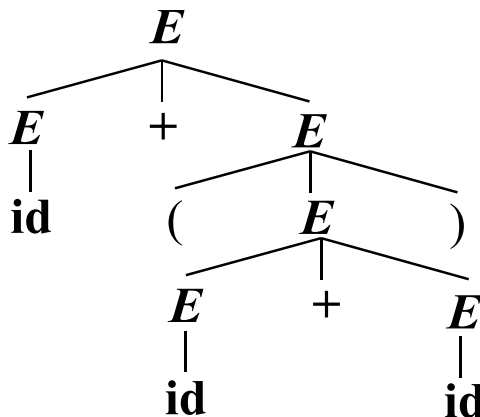
$$\Rightarrow id + E$$

$$\Rightarrow id + (E)$$

$$\Rightarrow id + (E + E)$$

$$\Rightarrow id + (E + id)$$

$$\Rightarrow id + (id + id)$$



$$E \Rightarrow_{lm} E + E$$

$$\Rightarrow_{lm} id + E$$

$$\Rightarrow_{lm} id + (E)$$

$$\Rightarrow_{lm} id + (E + E)$$

$$\Rightarrow_{lm} id + (id + E)$$

$$\Rightarrow_{lm} id + (id + id)$$

$$E \Rightarrow_{rm} E + E$$

$$\Rightarrow_{rm} E + (E)$$

$$\Rightarrow_{rm} E + (E + E)$$

$$\Rightarrow_{rm} E + (E + id)$$

$$\Rightarrow_{rm} E + (id + id)$$

$$\Rightarrow_{rm} id + (id + id)$$

## 自顶向下的语法分析采用最左推导方式

LL parsing.

- 总是选择每个句型的最左非终结符进行替换
- 根据输入流中的下一个终结符，选择最左非终结符的一个候选式

## 例

### ➤ 文法

$$\textcircled{1} E \rightarrow T E'$$

$$\textcircled{2} E' \rightarrow + T E' \mid \varepsilon$$

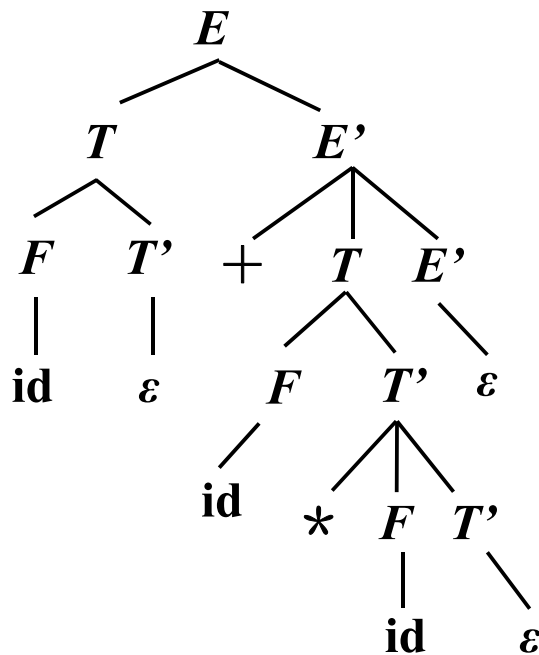
$$\textcircled{3} T \rightarrow F T'$$

$$\textcircled{4} T' \rightarrow * F T' \mid \varepsilon$$

$$\textcircled{5} F \rightarrow ( E ) \mid \text{id}$$

### ➤ 输入

id + id \* id  
↑ ↑ ↑ ↑ ↑ ↑



## 自顶向下语法分析的通用形式

- 递归下降分析 (*Recursive-Descent Parsing*)
  - 由一组过程组成，每个过程对应一个非终结符
  - 从文法开始符号 $S$ 对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果 $S$ 对应的过程体恰好扫描了整个输入串，则成功完成语法分析


```
void A() {  
1)  选择一个 $A$ 产生式,  $A \rightarrow X_1 X_2 \dots X_k$  ;  
2)  for (  $i = 1$  to  $k$  ) {  
3)      if (  $X_i$  是一个非终结符号 )  
4)          调用过程  $X_i()$  ;  
5)      else if (  $X_i$  等于当前的输入符号  $a$  )  
6)          读入下一个输入符号 ;  
7)      else /* 发生了一个错误 */ ;  
    }  
}
```

可能需要回溯(*backtracking*),  
导致效率较低



## 预测分析 (*Predictive Parsing*)

- 预测分析是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来选择正确的 $A$ -产生式。
- 可以对某些文法构造出向前看 $k$ 个输入符号的预测分析器，该类文法有时也称为 $LL(k)$  文法类
- 预测分析不需要回溯，是一种确定的自顶向下分析方法




第四章 语法分析

# 自顶向下分析概述

哈尔滨工业大学 陈鄞





## 第四章 语法分析

# 文法转换

哈尔滨工业大学 陈鄞



## 问题1

➤ 例

➤ 文法  $G$

$$S \rightarrow aAd \mid aBe$$

$$A \rightarrow c$$

$$B \rightarrow b$$

➤ 输入

$a b c$



同一非终结符的多个候选式存在  
共同前缀，将导致回溯现象

## 问题2

### ➤ 例

#### ➤ 文法 $G$

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

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

$$F \rightarrow ( E ) \mid \text{id}$$

$$E \Rightarrow E + T$$

$$\Rightarrow E + T + T$$

$$\Rightarrow E + T + T + T$$

$$\Rightarrow \dots$$

#### ➤ 输入

id + id \* id



左递归文法会使递归下降分析器陷入无限循环

含有  $A \rightarrow A\alpha$  形式产生式的文法称为是直接左递归的 (immediate left recursive)

如果一个文法中有一个非终结符  $A$  使得对某个串  $\alpha$  存在一个推导  $A \Rightarrow^+ A\alpha$ ，那么这个文法就是左递归的

经过两步或两步以上推导产生的左递归称为是间接左递归的

## 消除直接左递归

$$\begin{array}{l}
 A \rightarrow A\alpha \mid \beta (\alpha \neq \varepsilon, \beta \text{ 不以 } A \text{ 开头}) \quad \boxed{r=\beta\alpha^*} \\
 \downarrow \\
 A \rightarrow \beta A' \\
 A' \rightarrow \alpha A' \mid \varepsilon
 \end{array}
 \quad
 \begin{array}{l}
 A \Rightarrow A\alpha \\
 \Rightarrow A\alpha\alpha \\
 \Rightarrow A\alpha\alpha\alpha \\
 \dots \\
 \Rightarrow A\alpha \dots \alpha \\
 \Rightarrow \beta \alpha \dots \alpha
 \end{array}$$

事实上, 这种消除过程就是把左递归转换成了右递归

➤ 例

$$\begin{array}{l}
 E \rightarrow E + T \mid T \\
 T \rightarrow T * F \mid F \\
 F \rightarrow (E) \mid \text{id}
 \end{array}
 \quad
 \begin{array}{l}
 E \rightarrow T E' \\
 E' \rightarrow + T E' \mid \varepsilon \\
 T \rightarrow F T' \\
 T' \rightarrow * F T' \mid \varepsilon \\
 F \rightarrow (E) \mid \text{id}
 \end{array}
 \quad
 \begin{array}{l}
 A' \Rightarrow \alpha A' \\
 \Rightarrow \alpha\alpha A' \\
 \Rightarrow \alpha\alpha\alpha A' \\
 \dots \\
 \Rightarrow \alpha \dots \alpha A' \\
 \Rightarrow \alpha \dots \alpha
 \end{array}$$

## 消除直接左递归的一般形式

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

$(\alpha_i \neq \varepsilon, \beta_j \text{不以} A \text{开头})$



$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$$

消除左递归是要付出代价的——引进了一些非终结符和 $\varepsilon$ 产生式

## 消除间接左递归

➤ 例

$$S \rightarrow A a \mid b$$

$$\begin{aligned} S &\Rightarrow Aa \\ &\Rightarrow Sda \end{aligned}$$

$$A \rightarrow A c \mid S d \mid \varepsilon$$

➤ 将 $S$ 的定义代入 $A$ -产生式，得：

$$A \rightarrow A c \mid A a d \mid b d \mid \varepsilon$$

➤ 消除 $A$ -产生式的直接左递归，得：

$$A \rightarrow b d A' \mid A'$$

$$A' \rightarrow c A' \mid a d A' \mid \varepsilon$$



## 消除左递归算法

- 输入：不含循环推导（即形如 $A \Rightarrow^+ A$ 的推导）和 $\varepsilon$ -产生式的文法 $G$
- 输出：等价的无左递归文法
- 方法：

- 1) 按照某个顺序将非终结符号排序为 $A_1, A_2, \dots, A_n$  .
- 2) for ( 从1到n的每个 $i$  ) {
- 3)     for ( 从1到 $i-1$ 的每个 $j$  ) {
- 4)         将每个形如 $A_i \rightarrow A_j \gamma$ 的产生式替换为产生式组  $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$  ,  
            其中 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$  , 是所有的 $A_j$ 产生式
- 5)     }
- 6)     消除 $A_i$ 产生式之间的立即左递归
- 7) }

## 提取左公因子 (*Left Factoring*)

➤ 例

➤ 文法  $G$

➤  $S \rightarrow aAd \mid aBe$

➤  $A \rightarrow c$

➤  $B \rightarrow b$



➤ 文法  $G'$

➤  $S \rightarrow aS'$

➤  $S' \rightarrow Ad \mid Be$

➤  $A \rightarrow c$

➤  $B \rightarrow b$

通过改写产生式来推迟决定，  
等读入了足够多的输入，获得  
足够信息后再做出正确的选择

## 提取左公因子算法

- 输入：文法  $G$
- 输出：等价的提取了左公因子的文法
- 方法：

对于每个非终结符  $A$ ，找出它的两个或多个选项之间的最长公共前缀  $\alpha$ 。如果  $\alpha \neq \varepsilon$ ，即存在一个非平凡的 (*nontrivial*) 公共前缀，那么将所有  $A$ -产生式


$$A \rightarrow \alpha \beta_1 | \alpha \beta_2 | \dots | \alpha \beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_m$$

替换为

$$A \rightarrow \alpha A' | \gamma_1 | \gamma_2 | \dots | \gamma_m$$

$$A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

其中， $\gamma_i$  表示所有不以  $\alpha$  开头的产生式体； $A'$  是一个新的非终结符。不断应用这个转换，直到每个非终结符的任意两个产生式体都没有公共前缀为止




## 第四章 语法分析

# 文法转换

哈尔滨工业大学 陈鄞





第四章 语法分析

# LL(1) 文法

哈尔滨工业大学 陈鄞



## S\_文法

假如允许S\_文法包含 $\epsilon$ 产生式，  
将会产生什么问题？

- 预测分析法的工作过程。
  - 从文法开始符号出发，在每一步推导过程中根据当前句型的最左非终结符 $A$ 和当前输入符号 $a$ ，选择正确的 $A$ -产生式。为保证分析的确定性，选出的候选式必须是唯一的。
- S\_文法（简单的确定性文法，*Korenjak & Hopcroft*, 1966）

每个产生式的右部都以终结符开始

同一非终结符的各个候选式的首终结符都不同

S\_文法不含 $\epsilon$ 产生式

## 例

➤ 文法 <ul style="list-style-type: none"> <li>① <math>S \rightarrow aBC</math></li> <li>② <math>B \rightarrow bC</math></li> <li>③ <math>B \rightarrow dB</math></li> <li>④ <math>B \rightarrow \varepsilon</math></li> <li>⑤ <math>C \rightarrow c</math></li> <li>⑥ <math>C \rightarrow a</math></li> <li>⑦ <math>D \rightarrow e</math></li> </ul>	➤ 输入 <div style="display: inline-block; text-align: center; margin: 0 20px;"> <math>a \ d \ a</math>  <math>\uparrow \ \uparrow \ \uparrow</math> </div> <div style="display: inline-block; text-align: center;"> <math>a \ d \ e</math>  <math>\uparrow \ \uparrow \ \uparrow</math> </div>									
	➤ 推导 <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%; text-align: center;"><math>S</math></th> <th style="width: 50%; text-align: center;"><math>S</math></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><math>\Rightarrow aBC</math></td> <td style="text-align: center;"><math>\Rightarrow aBC</math></td> </tr> <tr> <td style="text-align: center;"><math>\Rightarrow adBC</math></td> <td style="text-align: center;"><math>\Rightarrow adBC</math></td> </tr> <tr> <td style="text-align: center;"><math>\Rightarrow adC</math></td> <td style="text-align: center;"><math>\Rightarrow adC</math></td> </tr> <tr> <td style="text-align: center;"><math>\Rightarrow ada</math></td> <td></td> </tr> </tbody> </table>	$S$	$S$	$\Rightarrow aBC$	$\Rightarrow aBC$	$\Rightarrow adBC$	$\Rightarrow adBC$	$\Rightarrow adC$	$\Rightarrow adC$	$\Rightarrow ada$
$S$	$S$									
$\Rightarrow aBC$	$\Rightarrow aBC$									
$\Rightarrow adBC$	$\Rightarrow adBC$									
$\Rightarrow adC$	$\Rightarrow adC$									
$\Rightarrow ada$										

可以紧跟  $B$  后面出现的终结符:  $c$ 、 $a$

### ➤ 什么时候使用 $\varepsilon$ 产生式?

- 如果当前某非终结符  $A$  与当前输入符  $a$  不匹配时, 若存在  $A \rightarrow \varepsilon$ , 可以通过检查  $a$  是否可以出现在  $A$  的后面, 来决定是否使用产生式  $A \rightarrow \varepsilon$  (若文法中无  $A \rightarrow \varepsilon$ , 则应报错)

## 非终结符的后继符号集

### ➤ 非终结符 $A$ 的后继符号集

➤ 可能在某个句型中紧跟在 $A$ 后边的终结符 $a$ 的集合，记为 $FOLLOW(A)$

$$FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, a \in V_T, \alpha, \beta \in (V_T \cup V_N)^*\}$$

例

(1)  $S \rightarrow aBC$

(2)  $B \rightarrow bC$  ←  $b$

(3)  $B \rightarrow dB$  ←  $d$

(4)  $B \rightarrow \varepsilon$  ←  $\{a, c\}$

(5)  $C \rightarrow c$

(6)  $C \rightarrow a$

$$FOLLOW(B) = \{a, c\}$$

输入

如果 $A$ 是某个句型的的最右符号，  
则将结束符“\$”添加到 $FOLLOW(A)$ 中



## 产生式的可选集

- 产生式 $A \rightarrow \beta$ 的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为 $SELECT(A \rightarrow \beta)$ 
  - $SELECT(A \rightarrow a\beta) = \{a\}$
  - $SELECT(A \rightarrow \varepsilon) = FOLLOW(A)$
- $q$ \_文法
  - 每个产生式的右部或为 $\varepsilon$ ，或以终结符开始
  - 具有相同左部的产生式有不相交的可选集

$q$ \_文法不含右部以非终结符打头的产生式

## 串首终结符集

### ➤ 串首终结符

➤ 串首第一个符号，并且是终结符。简称首终结符

➤ 给定一个文法符号串 $\alpha$ ， $\alpha$ 的串首终结符集 $FIRST(\alpha)$ 被定义为可以从 $\alpha$ 推导出的所有串首终结符构成的集合。如果 $\alpha \Rightarrow^* \varepsilon$ ，那么 $\varepsilon$ 也在 $FIRST(\alpha)$ 中

➤ 对于  $\forall \alpha \in (V_T \cup V_N)^+$ ,  $FIRST(\alpha) = \{ a \mid \alpha \Rightarrow^* a\beta, a \in V_T, \beta \in (V_T \cup V_N)^* \}$ ;

➤ 如果  $\alpha \Rightarrow^* \varepsilon$ ，那么  $\varepsilon \in FIRST(\alpha)$

➤ 产生式 $A \rightarrow \alpha$ 的可选集 $SELECT$

➤ 如果  $\varepsilon \notin FIRST(\alpha)$ ，那么  $SELECT(A \rightarrow \alpha) = FIRST(\alpha)$

➤ 如果  $\varepsilon \in FIRST(\alpha)$ ，那么  $SELECT(A \rightarrow \alpha) = (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A)$

## $LL(1)$ 文法

- 文法 $G$ 是 $LL(1)$ 的，当且仅当 $G$ 的任意两个具有相同左部的产生式 $A \rightarrow \alpha \mid \beta$ 满足下面的条件：
  - 如果 $\alpha$ 和 $\beta$ 均不能推导出 $\varepsilon$ ，则 $FIRST(\alpha) \cap FIRST(\beta) = \Phi$
  - $\alpha$ 和 $\beta$ 至多有一个能推导出 $\varepsilon$
  - 如果 $\beta \Rightarrow^* \varepsilon$ ，则 $FIRST(\alpha) \cap FOLLOW(A) = \Phi$ ；  
如果 $\alpha \Rightarrow^* \varepsilon$ ，则 $FIRST(\beta) \cap FOLLOW(A) = \Phi$ ；


同一非终结符的各个产生式的可选集互不相交

可以为 $LL(1)$ 文法构造预测分析器

## $LL(1)$ 文法

- 文法 $G$ 是 $LL(1)$ 的，当且仅当 $G$ 的任意两个具有相同左部的产生式 $A \rightarrow \alpha \mid \beta$ 满足下面的条件：
  - 如果 $\alpha$ 和 $\beta$ 均不能推导出 $\varepsilon$ ，则 $FIRST(\alpha) \cap FIRST(\beta) = \Phi$
  - $\alpha$ 和 $\beta$ 至多有一个能推导出 $\varepsilon$
  - 如果 $\beta \Rightarrow^* \varepsilon$ ，则 $FIRST(\alpha) \cap FOLLOW(A) = \Phi$ ；  
如果 $\alpha \Rightarrow^* \varepsilon$ ，则 $FIRST(\beta) \cap FOLLOW(A) = \Phi$ ；

- 第一个“ $L$ ”表示从左向右扫描输入
- 第二个“ $L$ ”表示产生最左推导
- “1”表示在每一步中只需要向前看一个输入符号来决定语法分析动作



第四章 语法分析

# LL(1) 文法

哈尔滨工业大学 陈鄞

