



第4章 语法分析（2）

Syntax Analysis

【对应教材 2.4, 4.3, 4.4】



回顾

□ 上下文无关文法

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

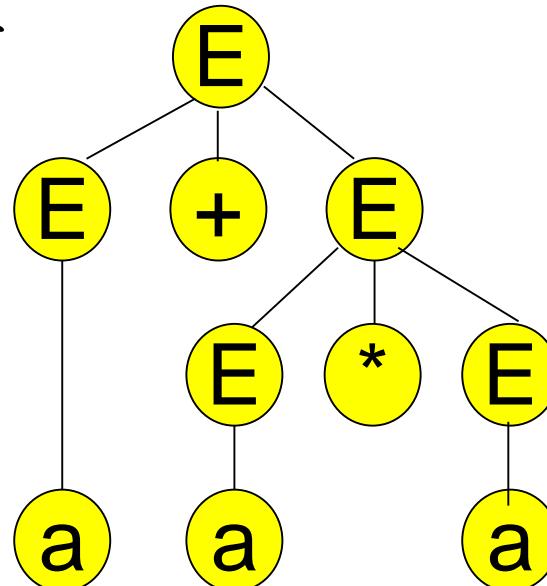
□ 推导

$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E \Rightarrow a+a*E \Rightarrow a+a*a$$

■ 最左推导、最右推导

□ 分析树

□ 二义性





练习-1

□ 考虑下面的表达式文法，符号串 $5*3+(2*7)+4$ 对应多少个不同的分析树？

- $E \rightarrow E * E \mid E + E \mid (E) \mid \text{int}$

□ 答案：

1. $\{ \{ 5 * 3 \} + (2 * 7) \} + 4$
2. $\{ 5 * 3 \} + \{ (2 * 7) + 4 \}$
3. $5 * \{ 3 + \{ (2 * 7) + 4 \} \}$
4. $\{ 5 * \{ 3 + (2 * 7) \} \} + 4$
5. $5 * \{ \{ 3 + (2 * 7) \} + 4 \}$



练习-2

□ 考虑下面的文法，它能生成多少个不同的句子？多少个不同的分析树？

- $S \rightarrow A1 \mid 1B$
- $A \rightarrow 10 \mid C \mid \epsilon$
- $B \rightarrow C1 \mid \epsilon$
- $C \rightarrow 0 \mid 1$

□ 答案：

- 5个不同的句子：1、11、111、01、101
- 对应7棵不同的分析树



内容提要

- 语法分析简介
- 上下文无关文法
- 文法的设计方法
- 自顶向下的语法分析
- 自底向上的语法分析
 - 简单LR分析: LR(0), SLR
 - 更强大的LR分析: LR(1), LALR
 - 二义性文法的使用
- 语法分析器生成工具YACC



文法的设计方法

- 文法能够描述程序设计语言的大部分语法
 - 但不是全部，比如：标识符的先声明后使用无法用上下文无关文法描述
 - 因此：语法分析器接受的语言是程序设计语言的超集。必须通过语义分析来剔除一些符合文法、但不合法的程序。
- 常见的文法变换方法
 - 消除二义性
 - 消除左递归
 - 提取左公因子

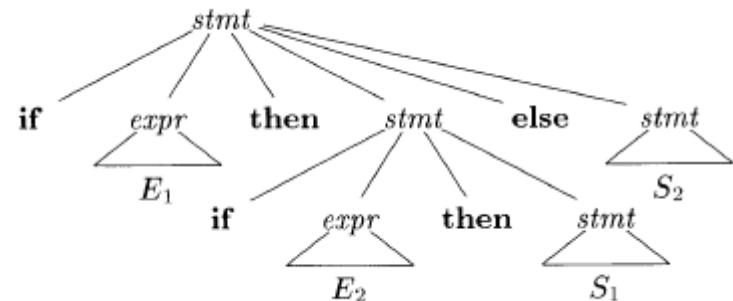
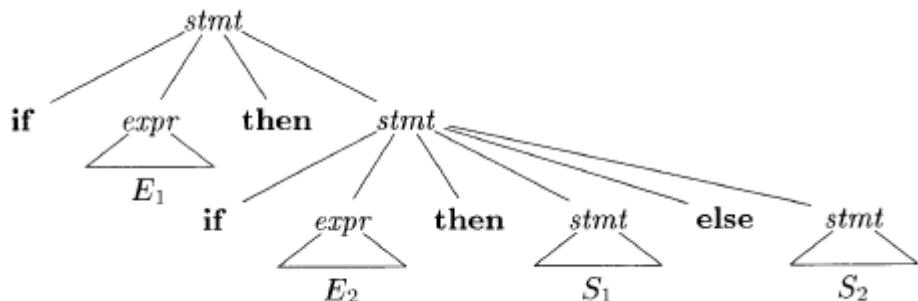


消除文法的二义性 (1)

- 一些二义性文法可以被改成等价的无二义性文法
- 例子 (dangling-else):

$stmt \rightarrow \text{if } expr \text{ then } stmt$
| $\text{if } expr \text{ then } stmt \text{ else } stmt$
| other

- $\text{if } E_1 \text{ then if } E_2 \text{ then } S_1 \text{ else } S_2$ 有两棵语法树





消除文法的二义性（2）

- 为保证“else和最近未匹配的then匹配”，要求在then分支的语句必须是匹配好的。
- 引入`matched_stmt`表示匹配好的语句，有如下文法：

stmt

$\rightarrow matched_stmt \mid open_stmt$

matched_stmt

$\rightarrow if\ expr\ then\ matched_stmt\ else\ matched_stmt$

$\mid other$

open_stmt

$\rightarrow if\ expr\ then\ stmt$

$\mid if\ expr\ then\ matched_stmt\ else\ open_stmt$

- 二义性的消除方法没有规律可循



例：消除文法的二义性

□ 文法G的产生式如下：

$$S \rightarrow aSb \mid bSa \mid SS \mid ab \mid ba$$

■ $L(G) = ?$

□ G是二义性的，比如 **ababab** 有两个不同的最左推导。

■ $S \Rightarrow SS \Rightarrow abS \Rightarrow abSS \Rightarrow ababS \Rightarrow ababab$

■ $S \Rightarrow SS \Rightarrow SSS \Rightarrow abSS \Rightarrow ababS \Rightarrow ababab$

□ 等价的上下文无关文法：

$$S \rightarrow TS \mid T$$

$$T \rightarrow aB \mid bA$$

$$A \rightarrow a \mid bAA$$

$$B \rightarrow b \mid aBB$$



消除文法中的左递归

□ 文法左递归

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

□ 直接左递归

- 串的特点

$$\begin{aligned} A \rightarrow A\alpha &| \beta \\ &\beta\alpha\dots\alpha \end{aligned}$$

□ 消除直接左递归 (变换为右递归)

$$\begin{aligned} A &\rightarrow \beta A' \\ A' &\rightarrow \alpha A' | \varepsilon \end{aligned}$$



例：

□ 算术表达文法G:

$$E \rightarrow E + T \mid T \quad (T + T \dots + T)$$

$$T \rightarrow T * F \mid F \quad (F * F \dots * F)$$

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

□ 消除左递归后的文法G':

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

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



间接左递归的消除

□ 间接左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Sd \mid \epsilon$$

□ 先变换成直接左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Aad \mid bd \mid \epsilon$$

□ 再消除左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bd \mid A' \mid A'$$

$$A' \rightarrow ad \mid A' \mid \epsilon$$



消除所有左递归的算法

1. 把G的非终结符整理成某种顺序 A_1, A_2, \dots, A_n 。
2.

```
for i:=1 to n do {
    for j :=1 to i-1 do {
        把每个形如  $A_i \rightarrow A_j r$  的规则替换成
         $A_i \rightarrow \delta_1 r | \delta_2 r | \dots | \delta_k r$ 
        其中  $\delta_1 | \delta_2 | \dots | \delta_k$  是当前全部  $A_j$  的产生式
    }
    消除  $A_i$  规则中的直接左递归
}
```
3. 化简由2得到的文法即可。



示例：消除左递归

例：文法G[s]为

$$S \rightarrow Ac|c$$

$$A \rightarrow Bb|b$$

$$B \rightarrow Sa|a$$

该文法无直接左递归，但有间接左递归

$$S \Rightarrow Ac \Rightarrow Bbc \Rightarrow Sabc \quad \text{即: } S \Rightarrow^+ Sabc$$

非终结符顺序重新排列

$$B \rightarrow Sa|a$$

$$A \rightarrow Bb|b$$

$$S \rightarrow Ac|c$$

顺序为： B, A, S

$$\begin{aligned}B &\rightarrow Sa|a \\A &\rightarrow Bb|b \\S &\rightarrow Ac|c\end{aligned}$$

$$A \rightarrow (Sa|a) b | b$$

把B的产生式代入A中

$$A \rightarrow Sab | ab | b$$

$$S \rightarrow (Sab | ab | b) c | c$$

把A的产生式代入S中

$$S \rightarrow Sabc | abc | bc | c$$

$$S \rightarrow abcS' | bcS' | cS'$$

消除直接左递归

$$S' \rightarrow abcS' | \epsilon$$

$$A \rightarrow Sab | ab | b$$

$$B \rightarrow Sa | a$$



顺序为：S, A, B

S → Ac|c
A → Bb|b
B → Sa|a



B → (Ac|c) a | a

把S的产生式代入B中

B → Aca | ca | a

把A的产生式代入B中

B → (Bb|b) ca | ca | a

B → Bbca | bca | ca | a

B → bcaB' | caB' | aB'

消除直接左递归

B' → bcaB' | ε

S → Ac | c

A → Bb | b

B → bcaB' | caB' | aB'

最后G[s]
的产生式

B' → bcaB' | ε

由于对非终结符的排序不同，最后得到的文法在形式上可能是不一样的，但是可以证明它们是等价的。



预测分析法简介

- 试图从开始符号推导出输入符号串；
- 以开始符号作为初始的当前句型
- 每次为最左边的非终结符号选择适当的产生式；
 - 通过查看下一个输入符号来选择这个产生式。
 - 有多个可能的产生式时预测分析法无能为力
- 比如文法： $E \rightarrow +EE \mid -EE \mid id$ ； 输入为 $+ id - id id$
- 当两个产生式具有相同的前缀时无法预测
 - 文法： $stmt \rightarrow if\ expr\ then\ stmt\ else\ stmt \mid if\ expr\ then\ stmt$
 - 输入： $if\ a\ then\ ...$
 - 新文法： $stmt \rightarrow if\ expr\ then\ stmt\ elsePart$
 $elsePart \rightarrow else\ stmt \mid \epsilon$
- 需要提取公因子！



提取左公因子

- 含有左公因子的文法

$$A \rightarrow \alpha\beta_1 / \alpha\beta_2$$

- 提取左公因子

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 / \beta_2$$

提取最长公共前缀作为公因子



例：提取左公因子

□ 悬空else的文法

$$\begin{aligned}stmt \rightarrow & \text{ if } expr \text{ then } stmt \text{ else } stmt \\& | \text{ if } expr \text{ then } stmt \\& | \text{ other }\end{aligned}$$

□ 提取左公因子

$$\begin{aligned}stmt \rightarrow & \text{ if } expr \text{ then } stmt \text{ optional_else_part } \\& | \text{ other }\end{aligned}$$
$$\text{optional_else_part} \rightarrow \text{ else } stmt$$
$$| \varepsilon$$



自顶向下的语法分析

- 自顶向下分析是从文法的开始符号出发，试构造出一个最左推导，从左至右匹配输入的单词串。
- 如果在每步推导中，将要被替换的非终结符号是A、而当前从左至右读入输入串读到的单词符号是a，如果A为左部的所有产生式（假定无 ϵ -产生式）是：

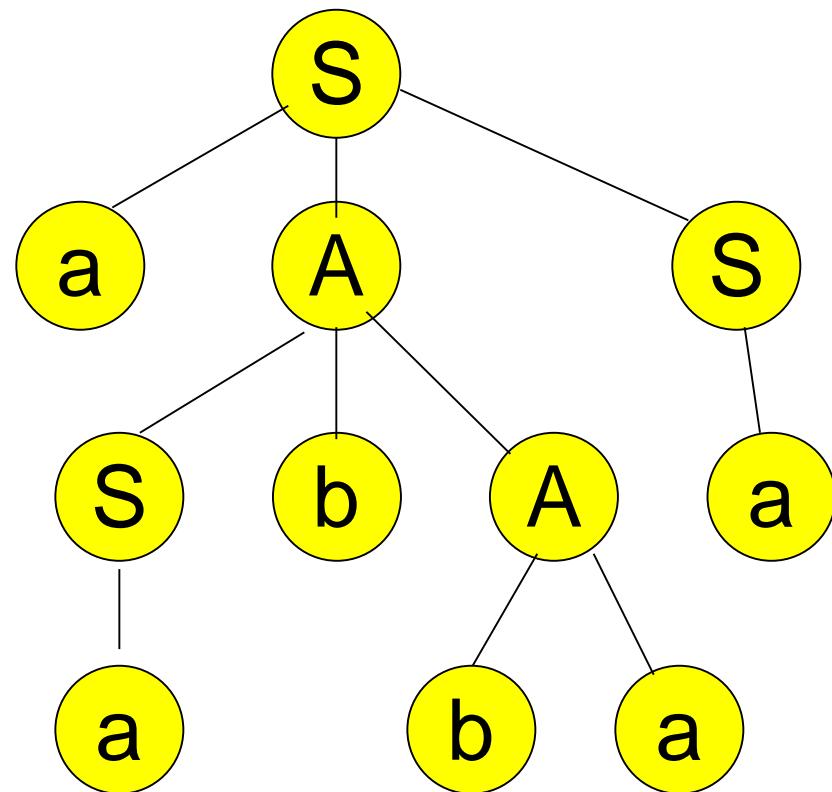
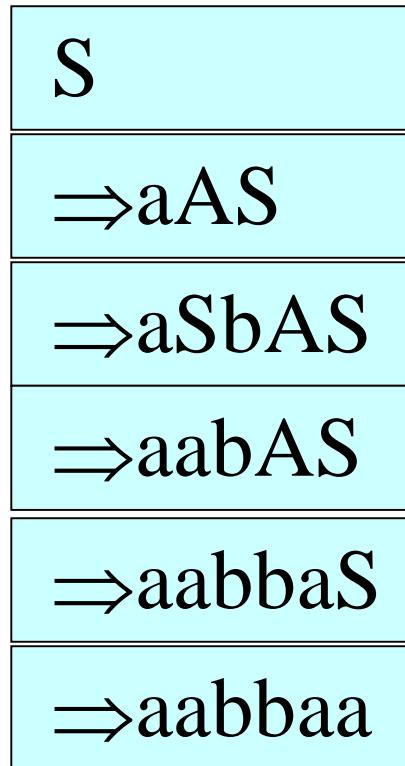
$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

并且只有 $\alpha_i (1 \leq i \leq n)$ 推导出的第一个终结符号是a，那么，就可以用产生式 $A \rightarrow \alpha_i$ 构造最左推导，即用 α_i 替换A，则可作预测分析；否则只能带回溯地进行尝试。

自顶向下分析的例子

文法: $S \rightarrow aAS \mid a \quad A \rightarrow SbA \mid SS \mid ba$

句子: aabbaa





自顶向下分析方法的特点

- 分析过程是带预测的
 - 对输入符号串要预测属于什么语法成分，然后根据该语法成分的文法建立语法树。
- 分析过程是一种试探过程
 - 尽一切办法(选用不同规则)来建立语法树
 - 由于是试探过程，难免会有失败，所以分析过程需要进行回溯，因此也称这种方法是带回溯的自顶向下分析方法。
- 最左推导可以编写程序来实现
 - 但带回溯的自顶向下分析方法在实际应用中价值不大，效率很低。



递归下降的语法分析

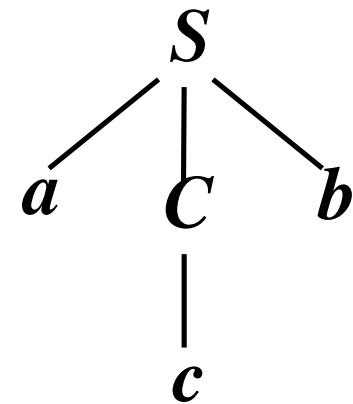
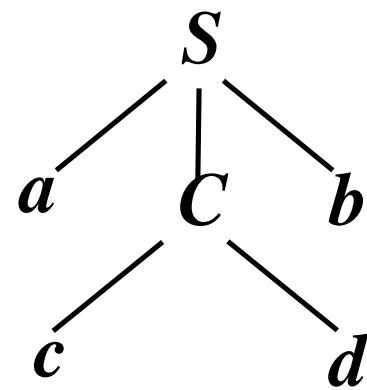
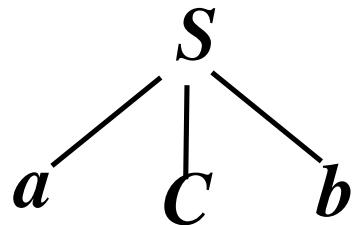
- 递归下降语法分析程序由一组过程组成
- 每个非终结符号对应于一个过程，该过程负责扫描非终结符号对应的结构
- 程序执行从开始符号对应的过程开始
 - 当扫描整个输入串时宣布分析成功完成

```
void A() {  
    1)      选择一个 A 产生式,  $A \rightarrow X_1 X_2 \cdots X_k$ ;  
    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 */;  
    }  
}
```



回溯的例子

例：文法 $S \rightarrow aCb \quad C \rightarrow cd / c$
为输入串 $w = acb$ 建立分析树：





递归下降分析技术的回溯

- 如果没有足够的信息来唯一地确定可能的产生式，那么分析过程就产生回溯。
 - 前面的算法报告错误（第7行）并不意味着输入串不是句子，而可能是表示前面选错了产生式。
 - 第1行上保存当前的扫描指针
 - 在第7行上应该改成
 - 回退到保存的指针；GOTO (1) 并选择下一个产生式；
 - 如果没有下一个产生式可选，报告错误。
- 回溯需要来回扫描，效率低
- 解决方法：设法通过一些信息确定唯一可能的产生式



如何保证没有回溯？

- 对文法加什么样的限制可以保证没有回溯？
- 在自顶向下的分析技术中，通常使用向前看几个符号来唯一地确定产生式（这里假定只看一个符号）
- 假设当前句型是 $xA\beta$ ，而输入是 $xa\dots$ 。那么选择产生式 $A \rightarrow \alpha$ 的必要条件是下列之一：
 - $\alpha \Rightarrow^* \epsilon$ ，且 β 以 a 开头；（可以用更强的条件替代：在某个句型中 a 跟在 A 之后；）
 - $\alpha \Rightarrow^* a\dots$ ；
 - 如果按照这两个条件选择时能够保证唯一性，那么我们就可以避免回溯。

First 和 Follow

□ First (α)

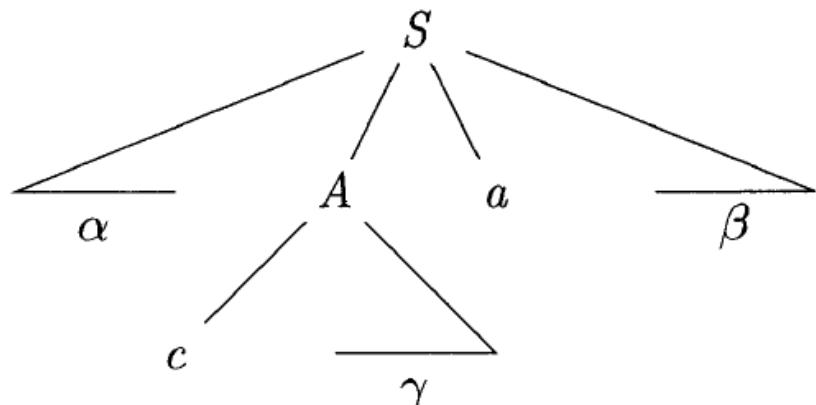
- 可以从 α 推导得到的串的首符号的集合
- $\text{First}(\alpha) = \{ a \mid \alpha \Rightarrow^* a \dots, a \in V_T \}$
 特别的, $\alpha \Rightarrow^* \epsilon$ 时, 规定 $\epsilon \in \text{First}(\alpha)$

□ Follow (A)

- 可能在某些句型中紧跟在A右边的终结符号的集合
- $\text{Follow}(A) = \{ a \mid S \Rightarrow^* \dots Aa \dots, a \in V_T \}$
 如果A是某个句型的最右符号时, 那么\$属于 $\text{Follow}(A)$

右图中:

- $c \in \text{First}(A)$,
- $a \in \text{Follow}(A)$





First集合的计算方法(1)

- 先计算单个符号X的First集合
 - 如果X是终结符号，那么 $\text{First}(X)=X$
 - 如果X是非终结符号，且 $X \rightarrow Y_1Y_2\dots Y_k$ 是产生式
 - 如果a在 $\text{First}(Y_i)$ 中，且 ϵ 在 $\text{First}(Y_1), \text{First}(Y_2), \dots, \text{First}(Y_{i-1})$ 中，那么a也在 $\text{First}(X)$ 中。
 - 如果 ϵ 在 $\text{First}(Y_1), \text{First}(Y_2), \dots, \text{First}(Y_k)$ 中，那么 ϵ 在 $\text{First}(X)$ 中。
 - 如果X是非终结符号，且 $X \rightarrow \epsilon$ 是一个产生式，那么 ϵ 在 $\text{First}(X)$ 中



First集合的计算方法(2)

- 再计算产生式右部 $X_1X_2\dots X_n$ 的First集合
 - 向集合中加入First(X_1)中所有非 ϵ 的符号；
 - 如果 ϵ 在First(X_1)中， 再加入First(X_2)中的所有非 ϵ 的符号；
 - ...
 - 如果 ϵ 在所有的First(X_i)中， 将 ϵ 加入First($X_1X_2\dots X_n$)中。



Follow集合的计算方法

□ 算法

- 将右端结束标记\$放到Follow(S)中
- 按照下面的两个规则不断迭代，直到所有的Follow集合都不再增长为止。
 - 如果存在产生式 $A \rightarrow \alpha B \beta$ ，那么First(β)中所有非 ϵ 的符号都在Follow(B)中。
 - 如果存在一个产生式 $A \rightarrow \alpha B$ ，或者 $A \rightarrow \alpha B \beta$ 且First(β)包含 ϵ ，那么Follow(A)中的所有符号都加入到Follow(B)中。

Warning: 请注意各个步骤中将符号加入到Follow集合中的理由。



例：把上述算法应用于文法G：

$$E \rightarrow TE' \quad E' \rightarrow +TE'|\epsilon$$

$$T \rightarrow FT' \quad T' \rightarrow *FT'|\epsilon \quad F \rightarrow id \mid (E)$$

$$\text{First}(F) = \{(, \text{id}\};$$

$$\text{First}(E) = \text{First}(T) = \{(, \text{id}\}$$

$$\text{First}(E') = \{+, \epsilon\}; \quad \text{First}(T') = \{*, \epsilon\}$$

$$\text{First}(TE') = \text{First}(FT') = \{ (, \text{id}\} \quad \text{First}(\epsilon) = \{\epsilon\}$$

$$\text{First}(+TE') = \{+\} \quad \text{First}(*FT') = \{*\}$$

$$\text{First}((E)) = \{()\} \quad \text{First}(\text{id}) = \{\text{id}\}$$

$$\text{Follow}(E) = \text{Follow}(E') = \{ (), \$ \}$$

$$\text{Follow}(T) = \text{Follow}(T') = \{ +, (), \$ \}$$

$$\text{Follow}(F) = \{+, *, (), \$ \}$$



LL(1)文法

□ LL(1)文法的定义：

对于文法中任意两个不同的产生式 $A \rightarrow \alpha \mid \beta$

1. 不存在终结符号 a 使得 α 和 β 都可以推导出以 a 开头的串
2. α 和 β 最多只有一个可以推导出空串
3. 如果 β 可以推导出空串，那么 α 不能推导出以 $\text{Follow}(A)$ 中任何终结符号开头的串；

□ 等价于：

- $\text{First}(\alpha) \cap \text{First}(\beta) = \Phi$; (条件1、2)
- 如果 $\varepsilon \in \text{First}(\beta)$, 那么 $\text{First}(\alpha) \cap \text{Follow}(A) = \Phi$;
反之亦然。 (条件3)



LL(1)文法的说明

- LL(1)中的第一个“L”表示从左到右扫描输入；
- 第二个“L”表示生成一个最左推导；
- “1”表示为做出分析动作的决定，每一步只需向前看1个符号。
- 类似可以定义LL(k)文法。
- 一些非LL(1)文法通过等价变换，可变成LL(1)文法。
- 输入串以\$为结束标记。这相当于对文法作扩充，即增加产生式 $S' \rightarrow S\$$ 。所以Follow(S)一定包含\$。



预测分析表的构造方法

- 输入：文法G
- 输出：预测分析表M
- 方法：
 - 对于文法G的每个产生式 $A \rightarrow \alpha$
 - 对于First(α)中的每个终结符号a，将 $A \rightarrow \alpha$ 加入到M[A,a]中；
 - 如果 ϵ 在First(α)，那么对于Follow(A)中的每个符号b，将 $A \rightarrow \alpha$ 加入到M[A,b]中。
 - 最后在所有的空白条目中填入error。



预测分析表的例子

□ 文法:

- $E \rightarrow TE'$ $E' \rightarrow +TE' \mid \epsilon$
- $T \rightarrow FT'$ $T' \rightarrow *FT' \mid \epsilon$ $F \rightarrow (E) \mid id$

□ FIRST集:

- $F: \{(, id\};$ $E, T: \{ (, id \} ;$ $E': \{ +, \epsilon \};$ $T': \{ *, \epsilon \}$

□ FOLLOW集:

- $E: \{ \$,) \}$ $E': \{ \$,) \}$ $T, T': \{ +,), \$ \}$ $F: \{ +, *,), \$ \}$

非终结符号	输入符号					
	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		



(非回溯的) 预测分析法

- 采用预测分析法要求文法是LL(1)文法
- 递归下降子程序方法
- 表驱动的（非递归）预测分析法



递归下降子程序方法

- 递归下降语法分析程序由一组过程组成
- 每个非终结符号对应于一个过程，该过程负责扫描非终结符号对应的结构
- 可以使用当前的输入符号来唯一地选择产生式

```
void A() {  
    1)     选择一个 A 产生式,  $A \rightarrow X_1 X_2 \cdots X_k$ ;  
    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 */;  
    }  
}
```

如果当前输入
符号为a，那么
选择M[A,a]中
的产生式



分析表达式的递归下降子程序(1)

```
void f_e ( );
{
    f_t( ); f_e'( );
}
```

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

```
void f_e' ( );
{
    if (lookahead == '+' )
        { match('+'); f_t(); f_e'(); }
    else {
        if ( lookahead != ')' && lookahead != '$' )
            { error (); }
    }
};
```



分析表达式的递归下降子程序(2)

```
void f_t()
{
    f_f(); f_t'();
}
```

$T \rightarrow FT'$
 $T' \rightarrow *FT'^* | \epsilon$

```
void f_t'()
{
    if (lookahead == '*' )
        { match('*'); f_f(); f_t'(); }
    else {
        if (lookahead != '+' && lookahead != ')'
            && lookahead != '$' )
            { error(); }
    }
}
```



分析表达式的递归下降子程序(3)

```
void f_f ()  
{  
    if ( lookahead== '(' )  
    {  
        match('('); f_e(); match(')');  
    }  
    else { if ( lookahead == id)  
            { match(id); }  
        else { error(); }  
    }  
}
```

$F \rightarrow id \mid (E)$



表驱动的(非递归)预测分析

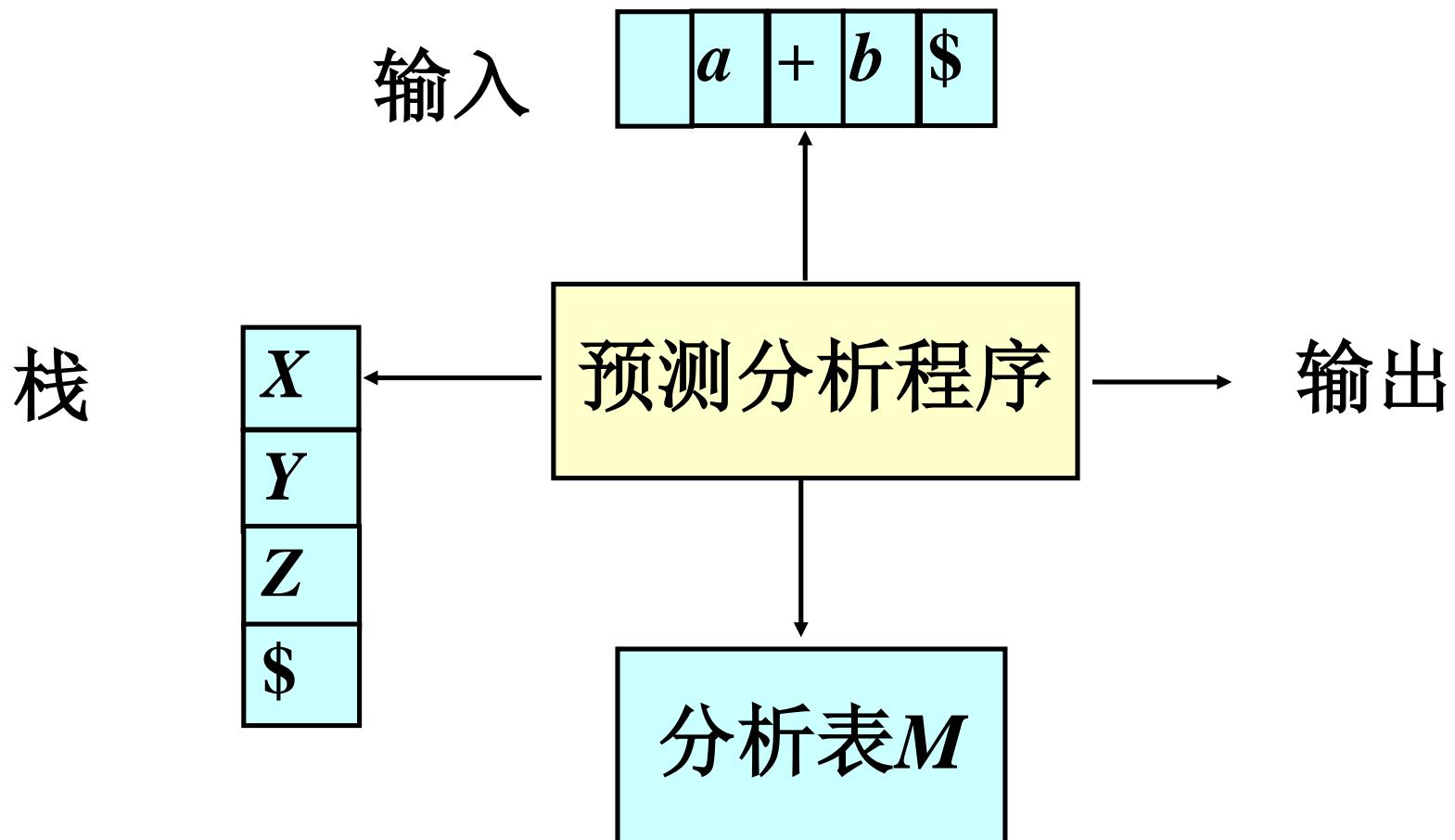
□ 需要

- 一个预测分析驱动程序
- 一个预测分析栈
- 一个预测分析表

- 前两项与具体文法无关，对所有文法都是相同的；
- 预测分析表依赖于具体的LL(1)文法。

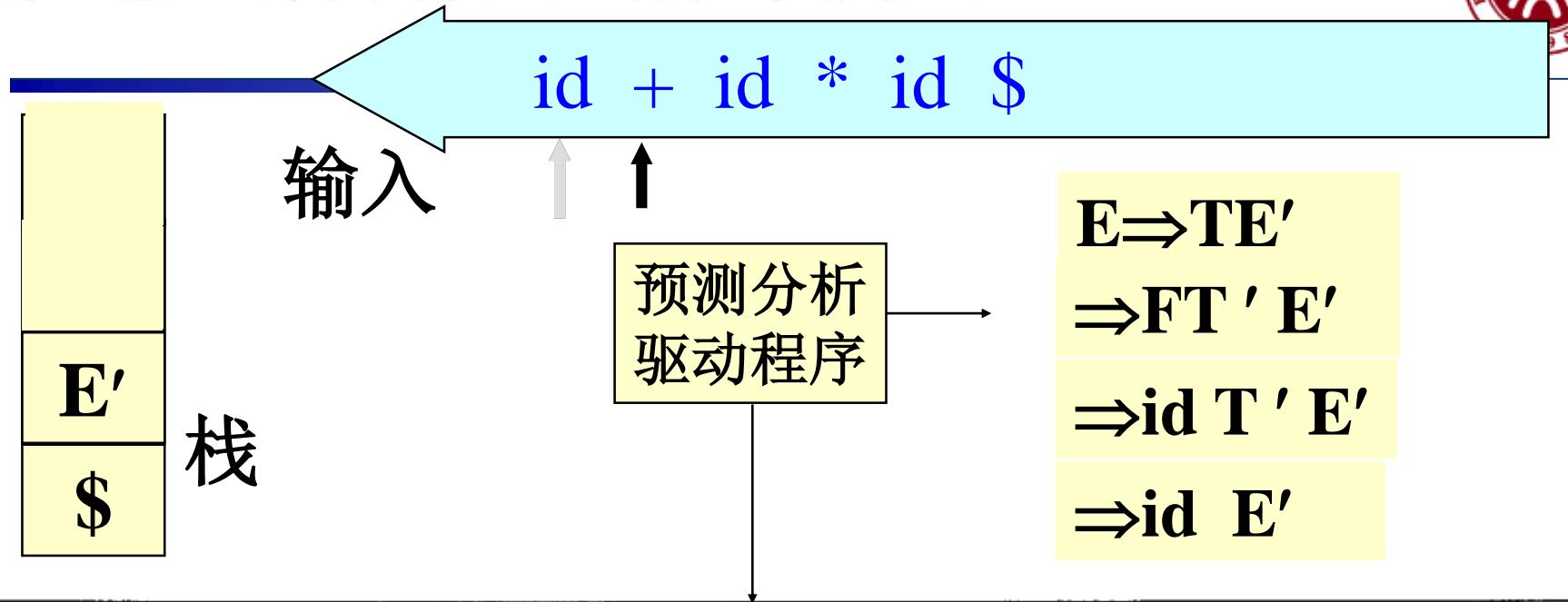


非递归的预测分析





非递归预测分析器的模型



非终结符号	输入符号					
	id	+	*	()	\$
E	$E \Rightarrow TE'$			$E \Rightarrow TE'$		
E'		$E' \Rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		



非递归预测分析器的驱动程序

把 ip 指向 w\$ 的第一个符号；令 x 为栈顶符号，
a 是 ip 所指向的符号。

```
push(st,$); push(st,S);
REPEAT
    IF x ∈ VT ∪ {$}
        THEN IF x=a THEN { pop(st); ip:=ip+1}
              ELSE error( )
        ELSE IF M[x,a] = x → y1y2...yk
              THEN { pop(st); push(st,yk); ...; push(st,y1);
                      write('x → y1y2...yk') }
              ELSE error( )
        UNTIL x=$ /*栈为空*/
```



非LL(1)文法

- 二义性文法肯定不是LL(1)文法，例如：

$G[S]: S \rightarrow iEtSS' \mid a \quad S' \rightarrow eS \mid \epsilon \quad E \rightarrow b$

- $\text{Follow}(S') = \{e, \$\}$
- 其相应的分析表如下表

	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow eS$ $S' \rightarrow \epsilon$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				



非LL(1)文法

- 某些非二义性文法也不是LL(1)文法，如表达式文法（左递归文法）：
 - 表达式文法： $E \rightarrow E+T \mid T$
 - $\text{First}(E+T) = \{\text{id},()\}$
 - $\text{First}(T) = \{\text{id},()\}$
- 有些语言不存在相应的LL(1)文法。如G的产生式为：

$$S \rightarrow aSb \mid ab \mid a$$

对任意k，语言L(G) 没有相应的LL(k)文法。



左递归文法

- 左递归文法不是LL(1)文法，也不适于自顶向下分析。

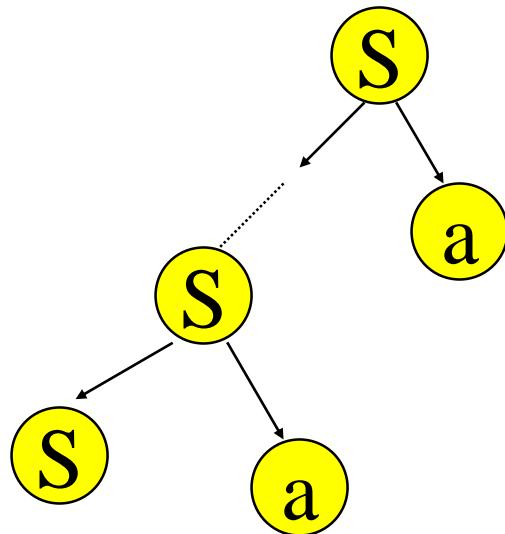
例如，若直接左递归的文法G为： $S \rightarrow Sa|b$

- (1) $\text{First}(Sa) \cap \text{First}(b) \neq \Phi$ (都包含b)。
- (2) $L(G)=\{ba^n|n\geq0\}$ 。例如，对于输入baaa\$，从左到右扫描输入串，最开始向前看符号是b，此时，选用S的哪条候选式？

- 若选用 $S \rightarrow b$ ，则肯定构造不出 $S \Rightarrow^+ ba^{n+1}$ ；
- 若选用 $S \rightarrow Sa$ ，则面对向前看符号b，下次仍选用 $S \rightarrow Sa$ ，不知何时选用 $S \rightarrow b$ 。
- 构造分析树的过程如下图所示。



含直接左递归文法的分析树结构



用左递归规则构造递归预测分析的过程：一进入这种过程不匹配任何输入符号，又直接执行递归调用，形成自己调用自己的永久循环。



语法错误的处理

□ 程序中可能存在不同层次的错误

- 词法错误：例如标识符、关键字或运算符拼写错误
- 语法错误：例如分号位置错误、{}多余或缺失等
- 语义错误：例如运算符和运算分量类型不匹配
- 逻辑错误：例如程序的错误推理引起的任何错误。（在C程序中==和=的无用）

□ 语法分析错误处理的目标

- 清晰准确报告错误
- 错误恢复的能力：继续检测下面的错误



预测分析中的错误恢复

□ 错误恢复

- 当预测分析器报错时，表示输入的串不是句子。
- 用户希望预测分析器能够进行恢复，并继续语法分析过程，在一次分析中找到更多的语法错误。
 - 有可能恢复得并不成功，之后找到的语法错误有可能是假的。
- 进行错误恢复时可用的信息：栈里面的符号，待分析的符号

□ 两类错误恢复方法：

- 恐慌模式
- 短语层次的恢复



恐慌模式 (panic mode)

□ 基本思想

- 语法分析器忽略输入中的一些符号，直到出现由设计者选定的某个**同步词法单元**

□ 解释：

- 语法分析过程总是试图在输入的前缀中找到和某个非终结符号对应的串；出现错误表明现在已经不可能找到这个非终结符号（程序结构）的串。
- 如果编程错误仅仅局限于这个程序结构，那么我们可以考虑跳过这个程序结构，假装已经找到了这个结构；然后继续进行语法分析处理。
- **同步词法单元**就是这个程序结构结束的标志。



同步词法单元的确定

- 确定非终结符号A的**同步集合**的启发式规则
 - Follow(A)的所有符号放到非终结符号A的同步集合中
 - 这些符号的出现可能表示之前的那些符号是错误的A的串
 - 将高层次的非终结符号对应串的开始符号加入到较低层次的非终结符号的同步集合
 - 比如语句的开始符号，if, while等可以作为表达式的同步集合；
 - First(A)中的符号加入到非终结符号A的同步集合。
 - 碰到这些符号时，可能意味着前面的符号是多余的符号
 - 如果A可以推导出空串，那么把此产生式当作默认值
 - 在匹配时出现错误，可以直接弹出相应的符号；
- 根据特定的应用决定哪些符号需要确定同步集合



恐慌模式的例子（1）

- 对文法4.28的表达式进行语法分析时，可以直接使用First、Follow作为同步集合。
- 为E、T和F设定同步集合；
- 空条目表示忽略输入符号；synch条目表示忽略到同步集合，然后弹出非终结符号；
- 终结符号不匹配时，弹出栈中终结符号；

非终结符号	输入符号					
	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E \rightarrow +TE'$			$E \rightarrow \epsilon$	$E \rightarrow \epsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch



恐慌模式的例子（2）

□ 错误输入：

+ id * + id

栈	输入	说明
$E \$$	$+ id * + id \$$	错误，略过 $+$
$E \$$	$id * + id \$$	id 在 FIRST(E) 中
$TE' \$$	$id * + id \$$	
$FT'E' \$$	$id * + id \$$	
$id T'E' \$$	$id * + id \$$	
$T'E' \$$	$* + id \$$	
$* FT'E' \$$	$* + id \$$	
$FT'E' \$$	$+ id \$$	错误， $M[F, +] = \text{synch}$
$T'E' \$$	$+ id \$$	F 已经被弹出栈
$E' \$$	$+ id \$$	
$+ TE' \$$	$+ id \$$	
$TE' \$$	$id \$$	
$FT'E' \$$	$id \$$	
$id T'E' \$$	$id \$$	
$T'E' \$$	$\$$	
$E' \$$	$\$$	
$\$$	$\$$	



短语层次的恢复

- 在预测分析表的空白条目中插入错误处理例程的函数指针
 - 例程可以改变、插入或删除输入中的符号
 - 发出适当的错误消息
 - 也可以选择弹栈
 - 确保不会出现无限循环！



作业

- 10月18日交
- 文法变换
 - Ex. 4.3.1, 4.3.3
- 自顶向下分析
 - Ex. 4.4.1 (1, 3, 5小题)
 - (同时计算First和Follow集合)