



School of Computer Science & Technology
Harbin Institute of Technology

第四章 自顶向下的语法分析

重点：自顶向下分析的基本思想，预测分析器总体结构，预测分析表的构造，递归下降分析法基本思想，简单算术表达式的递归下降分析器。

难点：FIRST 和 FOLLOW 集的求法，对其的理解以及在构造 LL(1)分析表时的使用。递归子程序法中如何体现分析的结果。



第4章 自顶向下的语法分析

4.1 语法分析概述

4.2 自顶向下的语法分析面临的问题与文法的改造

4.3 预测分析法

4.4 递归下降分析法

4.5 本章小结

语法分析的功能和位置

■ **语法分析**(syntax analysis)是编译程序的核心部分,其任务是检查词法分析器输出的**单词序列是否是源语言中的句子**,亦即是否符合源语言的语法规则。

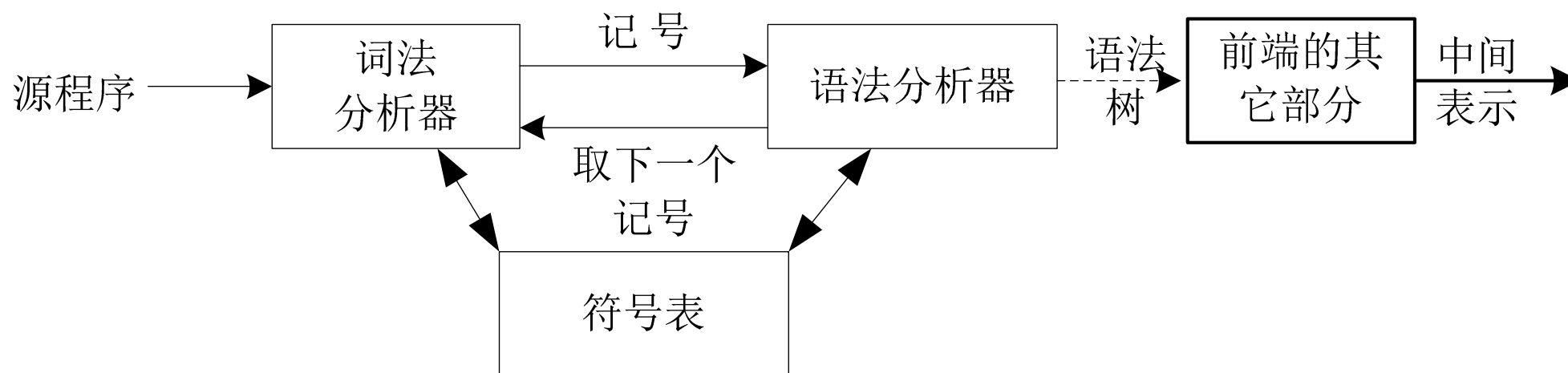


图4.1 语法分析器在编译器中的位置



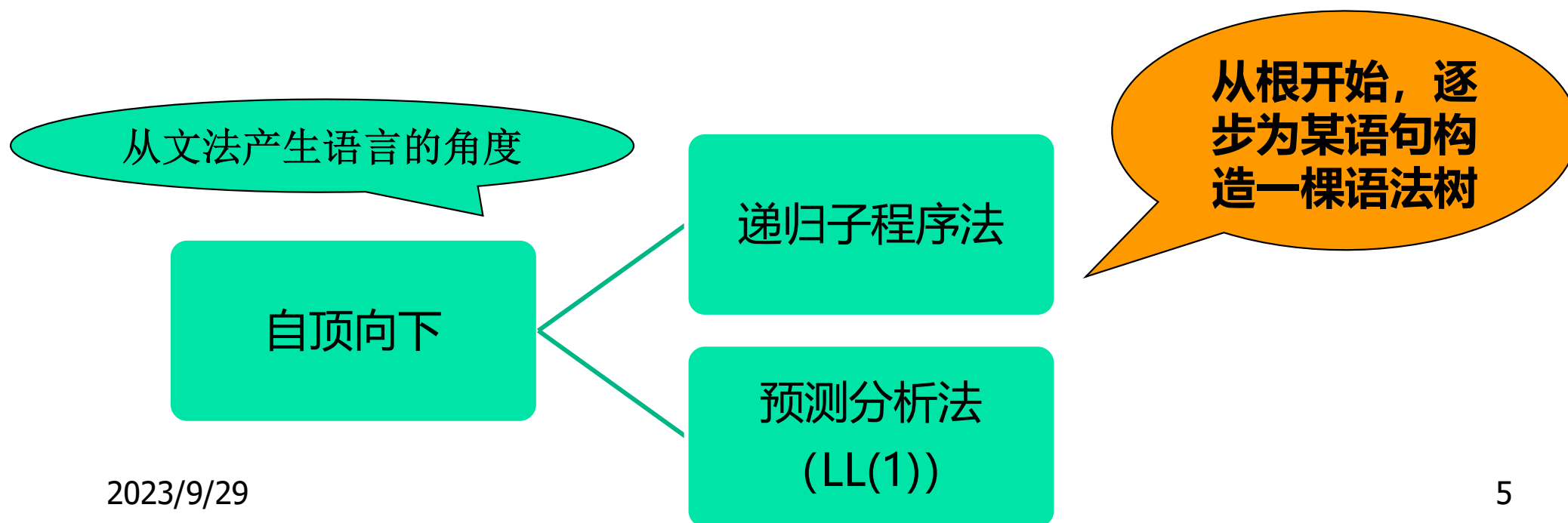
语法分析的功能和位置

- 语言是满足一定**组成规则**的句子集合，句子是满足一定**组成规则**的单词序列，单词则是满足一定**组成规则**的字符串
- 这些组成规则就是文法中的产生式
- 语法分析的主要工作是根据源语言的文法，判别**某个单词序列**是否是源语言的一个句子

4.1 语法分析概述

- 判断某个单词序列是否是源语言的句子，主要有两种方式：

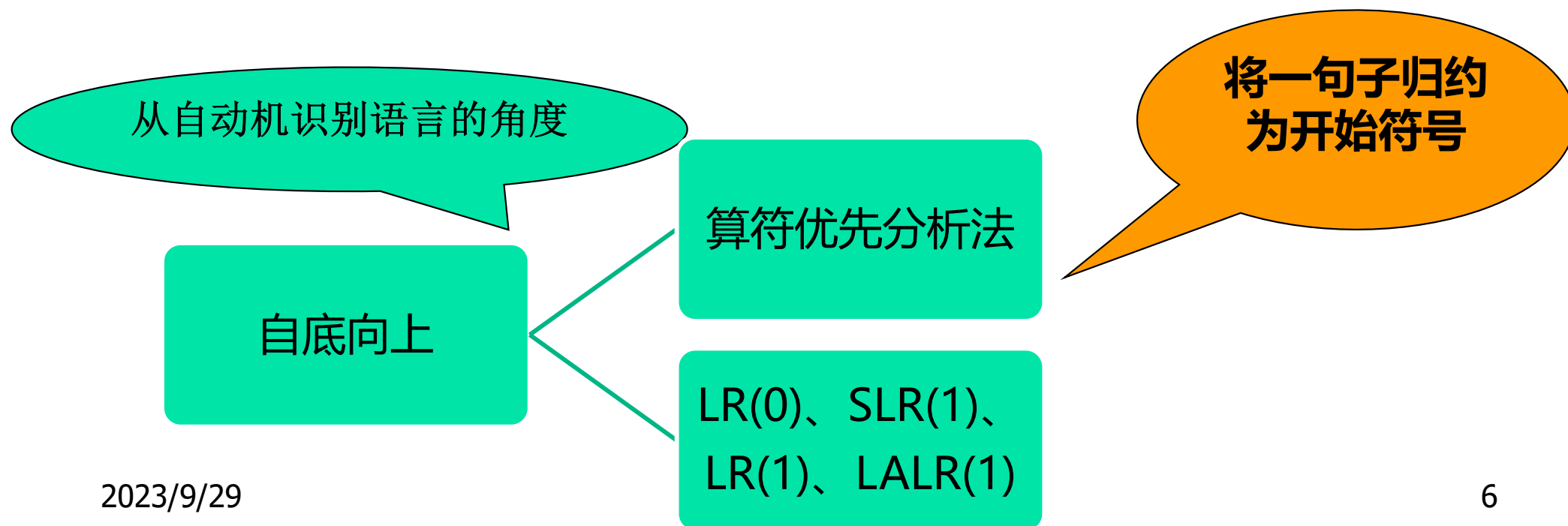
- **产生句子**的方式：从文法的开始符号开始，逐步**推导**出该单词序列，也称为**自顶向下**的语法分析



4.1 语法分析概述

- 判断某个单词序列是否是源语言的句子，主要有两种方式：

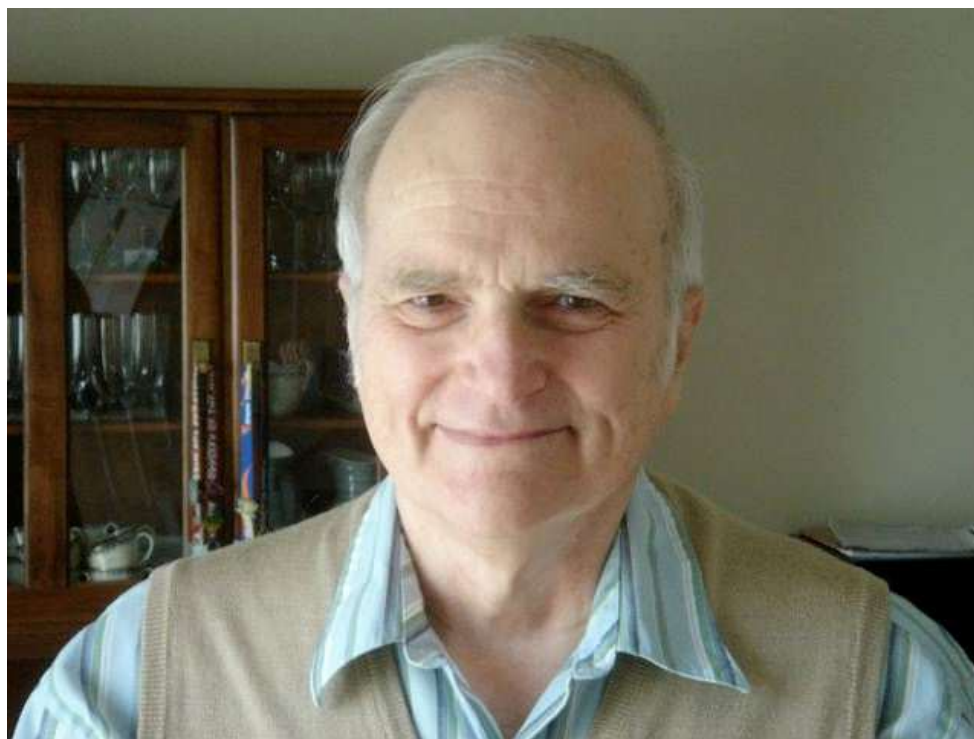
- **识别句子**的方式：逐步将构成程序的单词序列**归约**为文法的开始符号，称为**自底向上**的语法分析



介绍的材料

自顶向下推导(**LL**文法): Philip M. Lewis and Richard Stearns

P. M. Lewis II, R. E. Stearns: Syntax-Directed Transduction,
Journal of the ACM, **15**(3): 465-488 (1968)



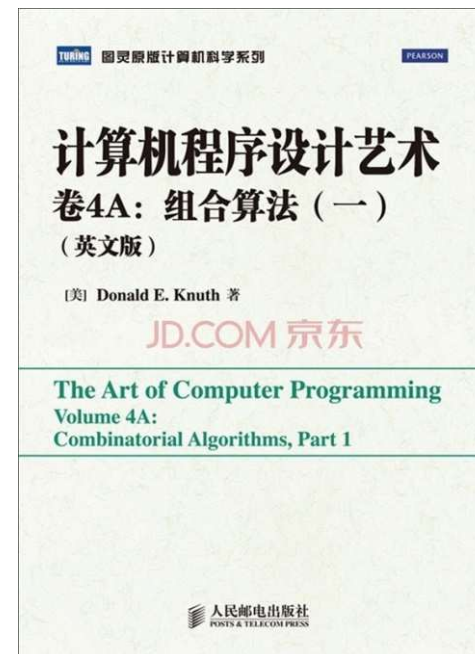
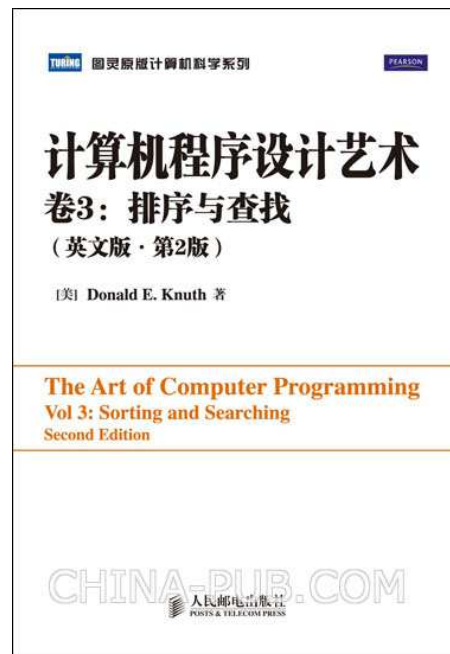
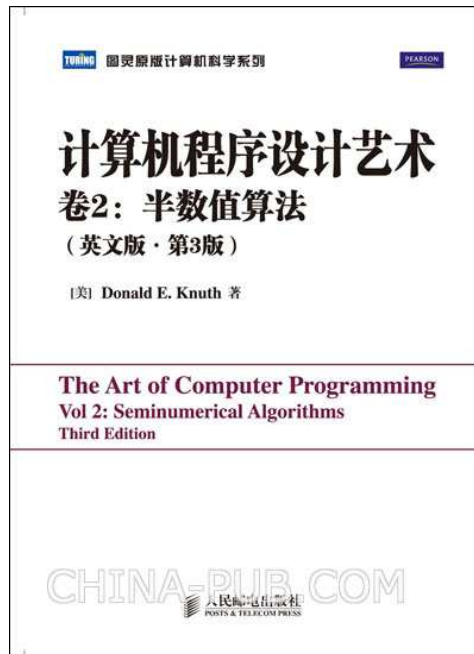
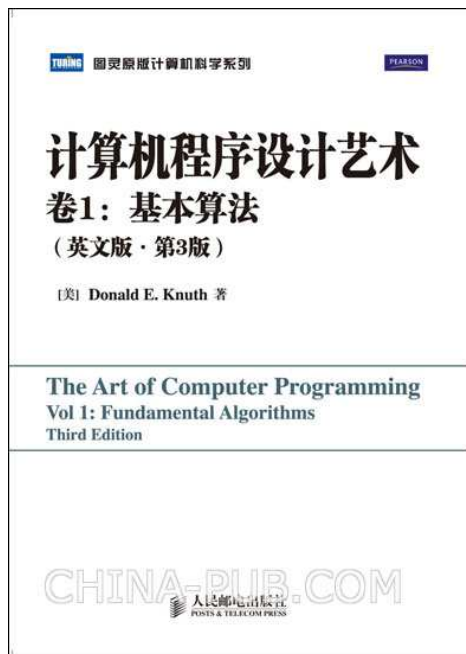
介绍的材料

自底向上归约(**LR**文法): Donald Knuth

D.E.Knuth, On the translation of languages from left to right, *Information and Control*, 8, 607-639 (1965)

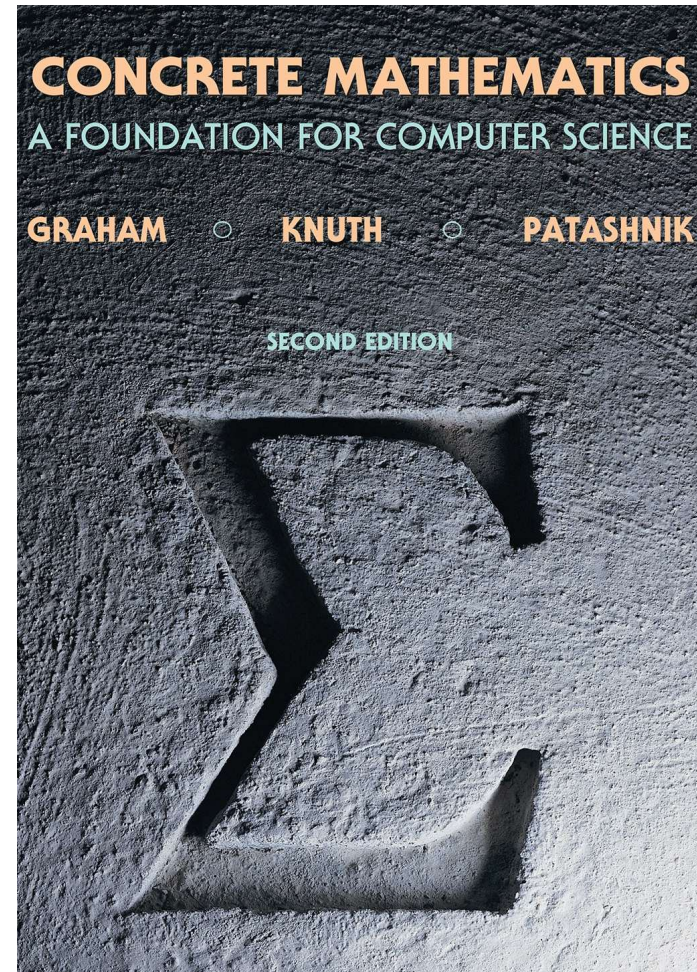
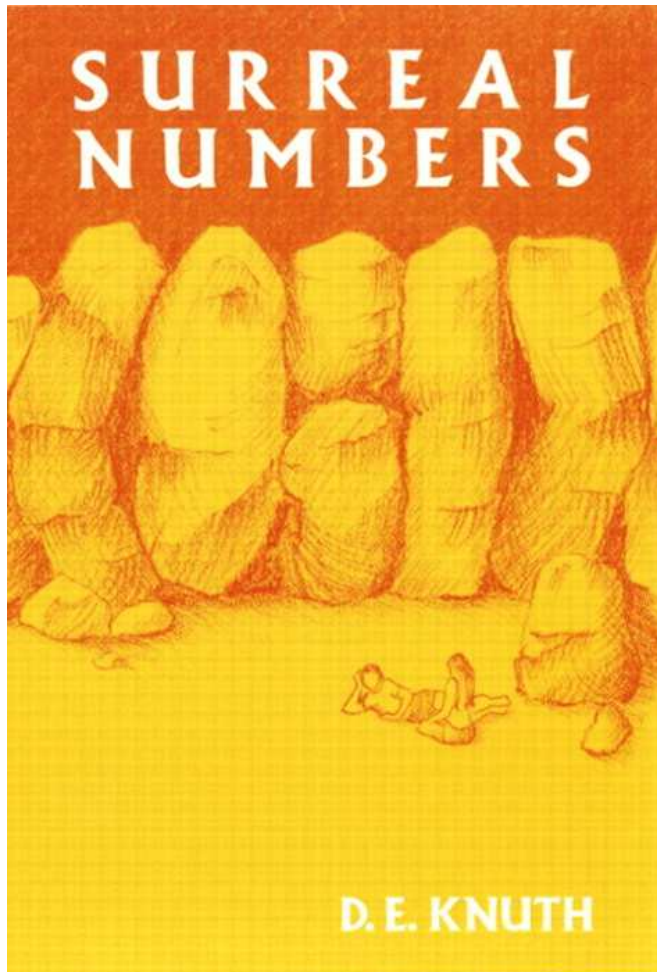


Donald Knuth(高德纳)





Donald Knuth(高德纳)





4.1 语法分析概述

- 无论是自顶向下还是自底向上，语法分析器都是自左到右地扫描输入单词序列，每次读入一个单词，针对输入单词序列建立一棵语法分析树。
- 不同的分析方法对应着不同的构建树的方式



4.1 语法分析概述

语法分析处理时的恢复策略

- 1. 紧急方式恢复策略：**丢弃输入记号，直到发现某个指定的同步记号为止。同步记号通常是定界符（分号或**end**标记），标志着一条新语句的开始。
- 2. 短语级恢复策略：**对剩余输入做局部纠正

4.2 自顶向下的语法分析面临的问题与文法的改造

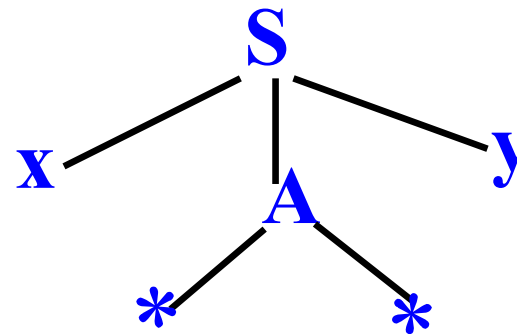
语法分析器是自左向右地扫描输入单词序列

■ 自顶向下语法分析的基本思想

- 从文法的开始符号出发，寻求所给的输入符号串的一个最左推导。
- 即从树根S开始，构造所给输入符号串的语法树

■ 例:设有G: $S \rightarrow xAy$, $A \rightarrow **|*$, 输入串: $x**y$

$$\begin{aligned} S &\Rightarrow xAy \\ &\Rightarrow x**y \end{aligned}$$





4.2.1 自顶向下分析面临的问题

- 自顶向下分析实际上是一种**试探性**的过程，可能导致分析效率极低，甚至失败
- 下面依次看自顶向下分析面临的三种问题



4.2.1 自顶向下分析面临的问题

1. 二义性问题

- 对于文法 G ，如果 $L(G)$ 中存在一个具有两棵或两棵以上分析树的句子，则称 G 是二义性的。
- 也可以等价地说：如果 $L(G)$ 中存在一个具有两个或两个以上最左推导的句子，则 G 是二义性文法。
- 如果一个文法 G 是二义性的，假设 $w \in L(G)$ 且 w 存在两个最左推导，则在对 w 进行自顶向下的语法分析时，语法分析程序将无法确定采用 w 的哪个最左推导。



4.2.1 自顶向下分析面临的问题

1. 二义性问题

- $G_{12}: E \rightarrow id \mid c \mid E + E \mid E - E \mid E * E \mid E / E \mid E ** E \mid (E)$
- 句子 $id_1 + c * id_2$
- 对应两棵不同的语法树

4.2.1 自顶向下分析面临的问题

1. 二义性问题

- $G_{12}: E \rightarrow id \mid c \mid E + E \mid E - E \mid E * E \mid E / E \mid E ** E \mid (E)$

- 解决办法1: 改造文法, 引入新的文法变量

- $G_{exp}: \begin{aligned} E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow F \uparrow P \mid P \\ P &\rightarrow c \mid id \mid (E) \end{aligned}$



4.2.1 自顶向下分析面临的问题

1. 二义性问题

- $G_{12}: E \rightarrow id \mid c \mid E + E \mid E - E \mid E * E \mid E / E \mid E ** E \mid (E)$
- 解决办法2: 根据优先级关系, 保证高优先级运算符优先的原则
- 乘法优先原则: 句子 $id_1 + c * id_2$

4.2.1 自顶向下分析面临的问题

2. 左递归引起的无穷推导问题

- 如果存在推导 $A \Rightarrow^+ \alpha A \beta$, 则称文法 G 是递归的, 当 $\alpha = \varepsilon$ 时称之为**左递归**;
- 如果 $A \Rightarrow^+ \alpha A \beta$ 至少需要两步推导, 则称文法 G 是间接递归的, 当 $\alpha = \varepsilon$ 时称之为**间接左递归**;
- 如果文法 G 中存在形如 $A \rightarrow \alpha A \beta$ 的产生式, 则称文法 G 是直接递归的, 当 $\alpha = \varepsilon$ 时称之为**直接左递归**。

4.2.1 自顶向下分析面临的问题

2. 左递归引起的无穷推导问题

- G_{er} :
 $E \rightarrow E+T$
 $E \rightarrow E-T$
 $E \rightarrow T$
 $T \rightarrow F$
 $T \rightarrow T * F$
 $T \rightarrow T / F$
 $F \rightarrow (E)$
 $F \rightarrow id$

- 考虑为输入串 $id+id*id$ 建立一个最左推导



4.2.1 自顶向下分析面临的问题

3. 回溯问题

- 文法中每个语法变量 A 的产生式右部称为 A 的**候选式**
- 如果 A 有多个候选式存在**公共前缀**，则自顶向下的语法分析程序将无法根据当前输入符号准确地选择用于推导的产生式，只能试探。
- 当试探不成功时就需要退回到上一步推导，看 A 是否还有其它的候选式，这就是**回溯**(backtracking)。

4.2.1 自顶向下分析面临的问题

3. 回溯问题

- G_e :
 - $E \rightarrow T$
 - $E \rightarrow E+T$
 - $E \rightarrow E-T$
 - $T \rightarrow F$
 - $T \rightarrow T * F$
 - $T \rightarrow T / F$
 - $F \rightarrow (E)$
 - $F \rightarrow id$

- 例如：考虑为输入串 $id+id*id$ 建立最左推导



4.2.1 自顶向下分析面临的问题

3.回溯问题

我们将采用**提取左因子**的方法来改造文法，以便减少推导过程中回溯现象的发生

当然，单纯通过提取左因子无法彻底避免回溯现象的发生。



4.2.2 对上下文无关文法的改造

1. 二义性问题

2. 左递归引起的无穷推导问题

3. 回溯问题



4.2.2 对上下文无关文法的改造

1. 消除二义性

- 改造的方法就是通过引入新的语法变量等，使文法含有更多的信息。
- 许多二义性文法是由于概念不清，即语法变量的定义不明确导致的，此时通过引入新的语法变量即可消除文法的二义性。

4.2.2 对上下文无关文法的改造

1. 消除二义性

- $G_{12}: E \rightarrow id \mid c \mid E + E \mid E - E \mid E * E \mid E / E \mid E ** E \mid (E)$

- 对 G_{exp} :
$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow F \uparrow P \mid P$$
$$P \rightarrow c \mid id \mid (E)$$

G_{exp} 比 G_{12} 提供了更多的信息

4.2.2 对上下文无关文法的改造

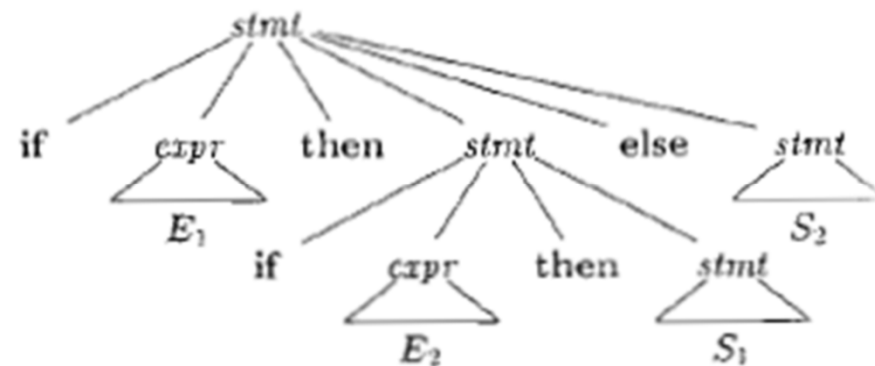
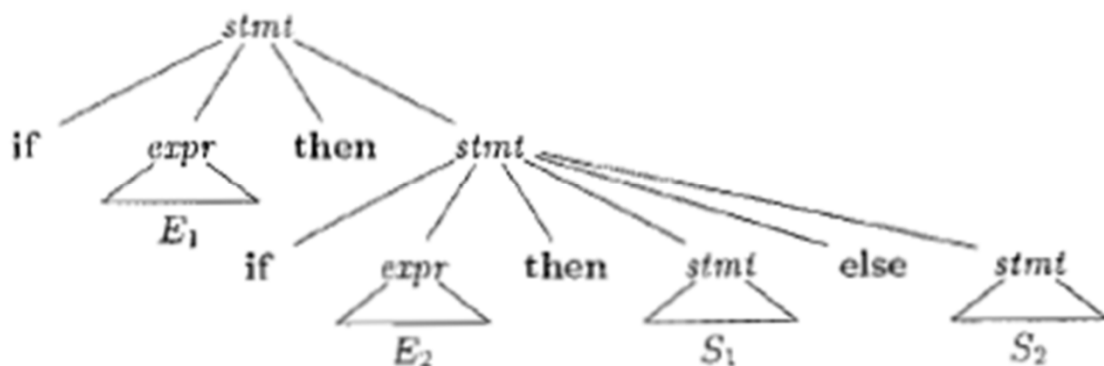
1.消除二义性

- $\langle stmt \rangle \rightarrow$ if $\langle expr \rangle$ then $\langle stmt \rangle$
| if $\langle expr \rangle$ then $\langle stmt \rangle$ else $\langle stmt \rangle$
| other
- 根据if语句中else与then配对情况将其分为配对的语句和不配对的语句两类。上述if语句的文法没有对这两个不同的概念加以区分，只是简单地将它们都定义为 $\langle stmt \rangle$ ，从而导致该文法是二义性的。
- if e_1 then if e_2 then s_1 else s_2

4.2.2 对上下文无关文法的改造

1. 消除二义性

- $\langle stmt \rangle \rightarrow$ if $\langle expr \rangle$ then $\langle stmt \rangle$
| if $\langle expr \rangle$ then $\langle stmt \rangle$ else $\langle stmt \rangle$
| other
- if e_1 then if e_2 then s_1 else s_2



4.2.2 对上下文无关文法的改造

引入语法变量 *<unmatched_stmt>* 来表示不配对语句, *<matched_stmt>* 表示配对语句

- $\langle stmt \rangle \rightarrow \langle matched_stmt \rangle \mid \langle unmatched_stmt \rangle$
 $\langle matched_stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle matched_stmt \rangle \text{ else } \langle matched_stmt \rangle \mid \text{other}$
 $\langle unmatched_stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle matched_stmt \rangle \mid \text{if } \langle expr \rangle \text{ then } \langle matched_stmt \rangle \text{ else } \langle unmatched_stmt \rangle$

4.2.2 对上下文无关文法的改造

2. 消除左递归

$A \Rightarrow^+ \alpha A \beta$, 当 $\beta = \varepsilon$ 时称之为右递归

- 直接左递归的消除(转换为右递归)
- 引入新的变量 A' , 将左递归产生式 $A \rightarrow A\alpha | \beta$ 替换为 $A \rightarrow \beta A' \quad A' \rightarrow \alpha A' | \varepsilon$

4.2.2 对上下文无关文法的改造

2.消除左递归

■ $E \rightarrow E+T \mid T \quad T \rightarrow T*F \mid F \quad F \rightarrow (E) \mid \text{id}$ 替换为:

■ $E \rightarrow TE' \quad E' \rightarrow +TE' \mid \varepsilon \quad T \rightarrow FT' \quad T' \rightarrow *FT' \mid \varepsilon$

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

4.2.2 对上下文无关文法的改造

2.消除左递归

- 一般地，假设文法 G 中的语法变量 A 的所有产生式如下： $A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$

其中， $\beta_i (i=1,2,\dots,m)$ 不以 A 打头。则用如下的产生式代替 A 的所有产生式即可消除其直接左递归：

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_m A' \quad A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon$$

4.2.2 对上下文无关文法的改造

2.消除左递归

- **上述方法的问题：**只能消除直接左递归，无法消除间接左递归

- $$S \rightarrow Ac \mid c$$
$$A \rightarrow Bb \mid b$$
$$B \rightarrow Sa \mid a$$



4.2.2 对上下文无关文法的改造

2.消除左递归

- **上述方法的问题：**只能消除直接左递归，无法消除间接左递归
- **消除间接左递归的基本思想：**为语法变量**编号**，再采用带入法将**间接左递归变为直接左递归**，然后采用给定方法来消除直接左递归

4.2.2 对上下文无关文法的改造

算法4.1 消除左递归。

输入：不含循环推导和 ε -产生式的文法 G ;

输出：与 G 等价的无左递归文法;

步骤：

循环推导即 $A \xRightarrow{+} A$,
 ε -产生式即 $A \xRightarrow{+} \varepsilon$

1. 将 G 的所有语法变量排序(编号), 假设排序后的语法变量记为 A_1, A_2, \dots, A_n ;
2. for $i \leftarrow 1$ to n {
3. for $j \leftarrow 1$ to $i-1$ {
4. 对每个形如 $A_i \rightarrow A_j \beta$ 的产生式, 其中, $A_j \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ 是所有当前 A_j 产生式, 用产生式 $A_i \rightarrow \alpha_1 \beta | \alpha_2 \beta | \dots | \alpha_k \beta$ 替换
5. }
6. 消除 A_i 产生式中的所有直接左递归
7. }

4.2.2 对上下文无关文法的改造

算法4.1 消除左递归。

输入：不含循环推导和 ϵ -产生式的文法 G 。

输出：与 G 等价的无左递归文法。

步骤：

1. 将 G 的所有语法变量

$A_1, A_2, \dots,$

2. for $i \leftarrow 1$ to n {

3. for $j \leftarrow 1$ to $i-1$ {

4. 对每个形如 $A_i \rightarrow A_j \beta$ 的产生式，其中， $A_j \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ 是所有当前 A_j 产生式，用产生式 $A_i \rightarrow \alpha_1 \beta | \alpha_2 \beta | \dots | \alpha_k \beta$ 替换

5. }

6. 消除 A_i 产生式中的所有直接左递归

7. }

$i=1$ 时，3-5行的循环体不执行，
此时执行第6行的消除左递归操作，
消除了所有 A_1 变量的直接左递归。
此时， A_1 的所有的具有 $A_1 \rightarrow A_b \alpha$ 形式的产生式，必有 $b > 1$ 。

4.2.2

算法4.1 消除左递归

输入: **不含循环**

输出: **与G等价**

步骤:

1. 将G的所有产生式

A_1, A_2, \dots

2. for $i \leftarrow 1$ to n {

3. for $j \leftarrow 1$ to $i-1$ {

4. 对每个形如 $A_i \rightarrow A_j \beta$ 的产生式, 其中, $A_j \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ 是所有当前 A_j 产生式, 用产生式 $A_i \rightarrow \alpha_1 \beta | \alpha_2 \beta | \dots | \alpha_k \beta$ 替换

5. }

6. 消除 A_i 产生式中的所有直接左递归

7. }

再看 $i=2$, A_2 的右部最左元素是变量的产生式形式可能是 $A_2 \rightarrow A_1 \alpha \mid A_2 \beta \mid A_3 \gamma \mid \dots$

已知, 当前的 A_1 的右部最左元素是变量的产生式形式是 $A_1 \rightarrow A_2 \alpha' \mid A_3 \beta' \mid \dots$

经过第4行的替换操作, A_2 的所有的具有 $A_2 \rightarrow A_b \alpha$ 形式的产生式, 必有 $b \geq 2$

执行第6行的消除直接左递归后, A_2 的所有具有 $A_2 \rightarrow A_b \alpha$ 形式的产生式, 必有 $b > 2$

4.2.2 对上下文无关文法的改造

算法4.1 消除左递归。

输入：不含循环推理的文法

输出：与 G 等价的文法

步骤：

1. 将 G 的所有语

A_1, A_2, \dots

2. for $i \leftarrow 1$ to n

3. for $j \leftarrow 1$ to n

4. 对每个非终结符

有当前 A_j 产生式

5. }

6. 消除 A_i 产生式中的直接左递归

7. }

$i=3$, A_3 的右部最左元素是变量的产生式形式可能是 $A_3 \rightarrow A_1\alpha \mid A_2\beta \mid A_3\gamma \mid \dots$

已知, 当前的 A_1 的右部最左元素是变量的产生式形式是 $A_1 \rightarrow A_2\alpha' \mid A_3\beta' \mid \dots$,

当前的 A_2 的右部最左元素是变量的产生式形式是 $A_2 \rightarrow A_3\alpha' \mid A_4\beta' \mid \dots$

经过第4行的替换操作, A_3 的所有的具有 $A_3 \rightarrow A_b\alpha$ 形式的产生式, 必有 $b \geq 3$

执行第6行的消除直接左递归后, A_3 的所有具有 $A_3 \rightarrow A_b\alpha$ 形式的产生式, 必有 $b > 3$

4.2.2 对上下文无关文法的改造

算法4.1 消除左递归。

输入：不含循环推导和 ϵ -产生式的文法 G ；

输出：与 G 等价的无左递归文法。

步骤：

1. 将 G 的所有语法

A_1, A_2, \dots

2. for $i \leftarrow 1$ to n {

3. for $j \leftarrow 1$ to

4. 对每个形如

有当前 A_j 产生式，用产生式 $A_i \rightarrow \alpha_1 p | \alpha_2 p | \dots | \alpha_k p$ 替换

5. }

6. 消除 A_i 产生式中的所有直接左递归

7. }

以此类推， $i=n$ ，经过第4行的替换操作， A_n 的所有的具有 $A_n \rightarrow A_b \alpha$ 形式的产生式，必有 $b \geq n$

执行第6行的消除直接左递归后， A_n 的所有具有 $A_n \rightarrow A_b \alpha$ 形式的产生式，必有 $b > n$
换句话说，已经消除了左递归

4.2.2 对上下文无关文法的改造

2.消除左递归

- $B \rightarrow Sa \mid a, A \rightarrow Bb \mid b, S \rightarrow Ac \mid c$
- 语法变量排序: B、A、S
- $i = 1$ 时, $B \rightarrow Sa \mid a$ 无直接左递归
- $i = 2$ 时, $A \rightarrow Bb \mid b$, 替换变量B后得到,
 $A \rightarrow Sab \mid ab \mid b$
- $i = 3$ 时, $S \rightarrow Ac \mid c$, 替换变量A后得到
 $S \rightarrow Sabc \mid abc \mid bc \mid c$, 执行消除直接左递归操作

4.2.2 对上下文无关文法的改造

■ 3.提取左因子

- 对每个语法变量 A ，找出它的两个或更多候选式的最长公共前缀 α ， A 产生式如下：

$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_m$ ，其中 $\gamma_1, \gamma_2, \dots, \gamma_m$ 表示所有不以 α 开头的候选式。

- 如果 $\alpha \neq \varepsilon$ ，则引入新的语法变量 A' ，用下面的产生式替换所有的 A 产生式：

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

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

- 反复应用上述变换，直到任意语法变量都没有两个候选式具有公共前缀为止。

4.2.2 对上下文无关文法的改造

■ 3.提取左因子

- $\langle stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle stmt \rangle$
 $\quad \quad \quad | \text{ if } \langle expr \rangle \text{ then } \langle stmt \rangle \text{ else } \langle stmt \rangle$
- $\langle stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle stmt \rangle \langle stmt_new \rangle$
 $\langle stmt_new \rangle \rightarrow \varepsilon \mid \text{else } \langle stmt \rangle$
- 提取左因子并不能完全消除回溯

4.2.2 对上下文无关文法的改造

- 希望彻底消除回溯，实现**确定的**自顶向下分析的文法要求

- 1. 无二义性

- 2. 无左递归

- 3. 任意一个语法变量 A 的各个候选式所能推导出的第一个终结符必须各不相同

LL(1)文法就是可以彻底消除回溯实现确定自顶向下分析的文法



4.2.3 LL(1)文法

- **不确定的自顶向下分析**

- 分析需要回溯，导致分析存在不确定性
- 代价高、效率低，实际中几乎不被采用

- **确定的自顶向下分析**

- 不能处理所有文法，这里讨论什么样的文法可以进行确定的自顶向下分析

4.2.3 LL(1)文法

问题：什么样的文法对其句子才能进行确定的自顶向下分析？

- 确定的自顶向下分析首先从文法的**开始符号**出发，每一步推导都根据当前句型的最左语法变量 A 和当前输入符号 a ，选择 A 的某个**候选式** $\alpha(alpha)$ 来替换 A ，并使得从 $\alpha(alpha)$ 推导出的**第一个终结符**恰好是 a 。
- 当 A 有多个候选式时，当前选中的候选式必须是**唯一**的。
- **第一个终结符**是指符号串的第一个终结符号，可以称为**首符号**。在自顶向下的分析中，它对选取候选式具有重要的作用。为此引入首符号集(**FIRST**)的概念。

4.2.3 LL(1)文法

1. 假设 α 是文法 $G=(V, T, P, S)$ 的符号串, 即 $\alpha \in (V \cup T)^*$, 从 α 推导出的串的首符号集**FIRST(α)**记作:

$$\text{FIRST}(\alpha) = \{a \mid \alpha \xRightarrow{*} a\beta, a \in T, \beta \in (V \cup T)^*\}.$$

若不成立, 引入后续符号集的概念

2. 如果 $\alpha \Rightarrow \varepsilon$, 则 $\varepsilon \in \text{FIRST}(\alpha)$ 。

3. 如果文法 G 中的所有 A 产生式为 $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_m$,

且 $\varepsilon \notin \text{FIRST}(\alpha_1) \cup \text{FIRST}(\alpha_2) \cup \dots \cup \text{FIRST}(\alpha_m)$

且对 $\forall i, j, 1 \leq i, j \leq m; i \neq j$, 均有

$\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$ 成立, 则可以对 G 的句子进行确定的自顶向下分析



4.2.3 LL(1)文法

- 如果存在 $\alpha_i \xRightarrow{*} \varepsilon$ 形式的产生式 $A \rightarrow \alpha_i$ ，当前终结符是a
 - 如果A的所有候选式的首符号集都不含有a，那么可能选择 $A \rightarrow \varepsilon$ 来推导，因为a可能是当前句型中紧跟A后出现
 - 如果某个候选式 α_j 的首符号集包括a，并且a也可能是当前句型中紧跟A后出现，还是无法确定到底选择 α_i 还是 α_j 来进行推导

4.2.3 LL(1)文法

如果存在 $A \rightarrow \varepsilon$ 这样的产生式，则需定义 $\text{FOLLOW}(A)$

$\forall A \in V$ ，定义 A 的**后续符号集**为：

1. $\text{FOLLOW}(A) = \{a \mid S \xRightarrow{*} \alpha A a \beta, a \in T, \alpha, \beta \in (V \cup T)^*\}$
2. 如果 A 是某个句型的最右符号，则将结束符 $\#$ 添加到 $\text{FOLLOW}(A)$ 中
3. 给定 $\alpha_j \xRightarrow{*} \varepsilon$ 和当前终结符 $a \in \text{FOLLOW}(A)$ ，则如果对 $\forall i (1 \leq i \leq m; i \neq j)$ ， $\text{FIRST}(\alpha_i) \cap \text{FOLLOW}(A) = \emptyset$ 均成立，即 A 的所有候选式的首符号集都不含有 a ，可以确定当前终结符 a 是由 α_j 后面的变量生成的

4.2.3 LL(1)文法

综上所述，对文法 G 的句子进行确定的自顶向下语法分析的充分必要条件是：如果 G 的任意两个具有相同左部的产生式 $A \rightarrow \alpha | \beta$ 满足下列条件：

1. 如果 α 、 β 均不能推导出 ε ，则 $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$ ；
2. α 和 β 至多有一个能推导出 ε ；
3. 如果 $\beta \Rightarrow^* \varepsilon$ ，则 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

称 G 为LL(1)文法。

第一个 L 代表从左向右扫描输入符号串，第二个 L 代表产生最左推导，1代表在分析过程中每步推导都要向前查看一个输入符号

4.2.3 LL(1)文法

LL(1)文法要求:

$\varepsilon \in \text{FIRST}(\alpha_1), \varepsilon \in \text{FIRST}(\alpha_2), \dots, \varepsilon \in \text{FIRST}(\alpha_n)$

中至多有一个成立。

如果有多个
成立呢？

算法4.2 计算FIRST(X)。

输入：文法 $G=(V, T, P, S)$, $X \in (V \cup T)$;

1. $\text{FIRST}(X) = \emptyset$;
2. if $(X \in T)$ then $\text{FIRST}(X) := \{X\}$;
3. if $X \in V$ then
begin
4. if $(X \rightarrow \varepsilon \notin P)$ then
 $\text{FIRST}(X) := \text{FIRST}(X) \cup \{a | X \rightarrow a... \in P\}$;
5. if $(X \rightarrow \varepsilon \in P)$ then
 $\text{FIRST}(X) := \text{FIRST}(X) \cup \{a | X \rightarrow a... \in P\} \cup \{\varepsilon\}$
- end
6. 对 $\forall X \in V$, 重复过程7-10, 直到所有FIRST集不变为止。
7. if $(X \rightarrow Y... \in P \text{ and } Y \in V)$ then
 $\text{FIRST}(X) := \text{FIRST}(X) \cup (\text{FIRST}(Y) - \{\varepsilon\})$;
8. if $(X \rightarrow Y_1...Y_n \in P \text{ and } Y_1...Y_{i-1} \Rightarrow \varepsilon)$ then
9. for $k=2$ to i do $\text{FIRST}(X) := \text{FIRST}(X) \cup (\text{FIRST}(Y_k) - \{\varepsilon\})$;
10. if $Y_1...Y_n \Rightarrow \varepsilon$ then $\text{FIRST}(X) := \text{FIRST}(X) \cup \{\varepsilon\}$;

算法4.3 计算FIRST(α)。

输入：文法 $G=(V, T, P, S)$, $\alpha \in (V \cup T)^*$, $\alpha = X_1 \dots X_n$;

输出：FIRST(α);

步骤：

1. 计算FIRST(X_1);
2. FIRST(α) := FIRST(X_1) - $\{\epsilon\}$;
3. $k := 1$;
4. while ($\epsilon \in \text{FIRST}(X_k)$ and $k < n$) do begin
5. FIRST(α) := FIRST(α) \cup (FIRST(X_{k+1}) - $\{\epsilon\}$);
6. $k := k + 1$ end
7. if ($k = n$ and $\epsilon \in \text{FIRST}(X_k)$) then
 FIRST(α) := FIRST(α) \cup $\{\epsilon\}$;

例 表达式文法的语法符号的FIRST 集

$\text{FIRST}(F) = \{ (, \text{id} \}$

$\text{FIRST}(T) = \text{FIRST}(F) = \{ (, \text{id} \}$

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

$\text{FIRST}(E') = \{ +, \epsilon \}$

$\text{FIRST}(T') = \{ *, \epsilon \}$

$\text{FIRST}(+) = \{ + \}$

$\text{FIRST}(*) = \{ * \}$

$\text{FIRST}(() = \{ (\}$

$\text{FIRST}()) = \{) \}$

$\text{FIRST}(\text{id}) = \{ \text{id} \}$

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

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

计算某变量FIRST集时，关注该变量出现在左部的产生式

算法4.4 计算FOLLOW集。

输入：文法 $G=(V, T, P, S)$, $A \in V$;

输出：FOLLOW(A);

步骤：

第2行表示，如果A是某个句型的最右符号，将结束符'#'添加到FOLLOW(A)

1. 对 $\forall X \in V$, FOLLOW(X) := \emptyset ;
2. FOLLOW(S) := {#}, #为句子的结束符;
3. 对 $\forall X \in V$, 重复下面的第4步到第5步, 直到所有FOLLOW集不变为止。
4. 若 $A \rightarrow \alpha B \beta \in P$, 则
FOLLOW(B) := FOLLOW(B) \cup (FIRST(β) - { ϵ });
5. 若 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B \beta \in P$ 且 $\beta \xRightarrow{*} \epsilon$, $A \neq B$, 则
FOLLOW(B) := FOLLOW(B) \cup FOLLOW(A);

第5行表示，任何一个终结符，如果可以紧跟着A出现，那么该终结符也可以紧跟着B出现，反之不然

例 表达式文法的语法变量的 FOLLOW 集

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

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

$F \rightarrow (E) | id$

计算某变量 FOLLOW 集时关注其出现在右部的产生式

$FIRST(F) = \{ (, id \}$

$FIRST(T) = FIRST(F) = \{ (, id \}$

$FIRST(E) = FIRST(T) = \{ (, id \}$

$FIRST(E') = \{ +, \epsilon \}$

$FIRST(T') = \{ *, \epsilon \}$

$FOLLOW(E) = \{ \#,) \}$

$FOLLOW(E') = FOLLOW(E) = \{ \#,) \}$

$FOLLOW(T) = \{ FIRST(E') - \{ \epsilon \} \} \cup FOLLOW(E) \cup FOLLOW(E') = \{ +,), \# \}$

$FOLLOW(T') = FOLLOW(T) = \{ +,), \# \}$

$FOLLOW(F) = \{ FIRST(T') - \{ \epsilon \} \} \cup FOLLOW(T) \cup FOLLOW(T')$

2023/9/29 $= \{ *, +,), \# \}$

表达式文法是 LL(1) 文法

- $E \rightarrow T E'$
- $E' \rightarrow + T E' \mid \varepsilon$
- $T \rightarrow F T'$
- $T' \rightarrow * F T' \mid \varepsilon$
- $F \rightarrow (E) \mid \text{id}$

考察

- E' : $+$ 不在 $\text{FOLLOW}(E') = \{), \# \}$
- T' : $*$ 不在 $\text{FOLLOW}(T') = \{ +,), \# \}$
- F : $($ 和 id 不同

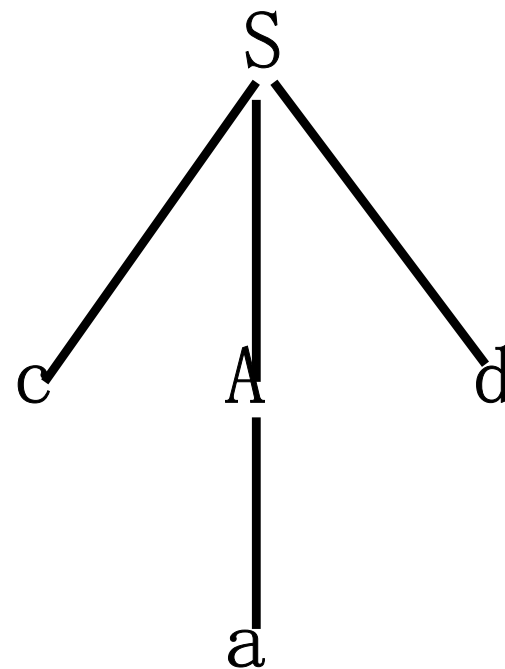
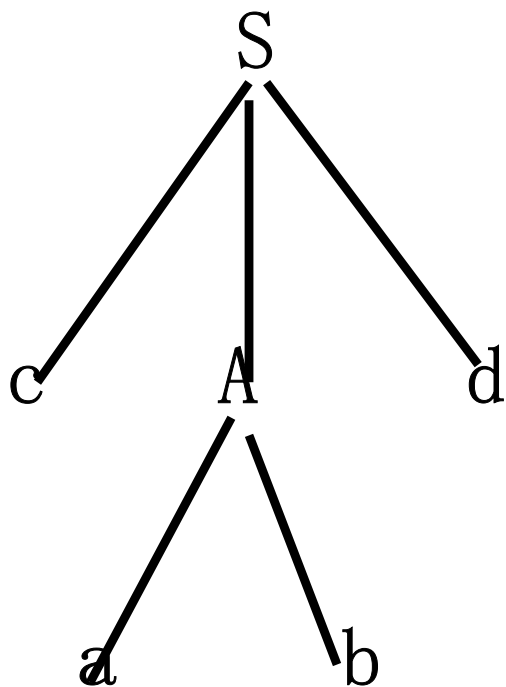


例 对文法

■ $S \rightarrow cAd$

■ $A \rightarrow ab|a$

输入 cad 的分析



非 LL(1)文法 的不确定性



不确定性的解决方法

1) 采用回溯算法

- 过于复杂，效率低下

2) 改写文法

- 将非LL(1)文法改写为等价的LL(1)文法
- 无法改写时：
 - 增加其它的判别因素
 - 文法过于复杂，无法用自顶向下方法处理



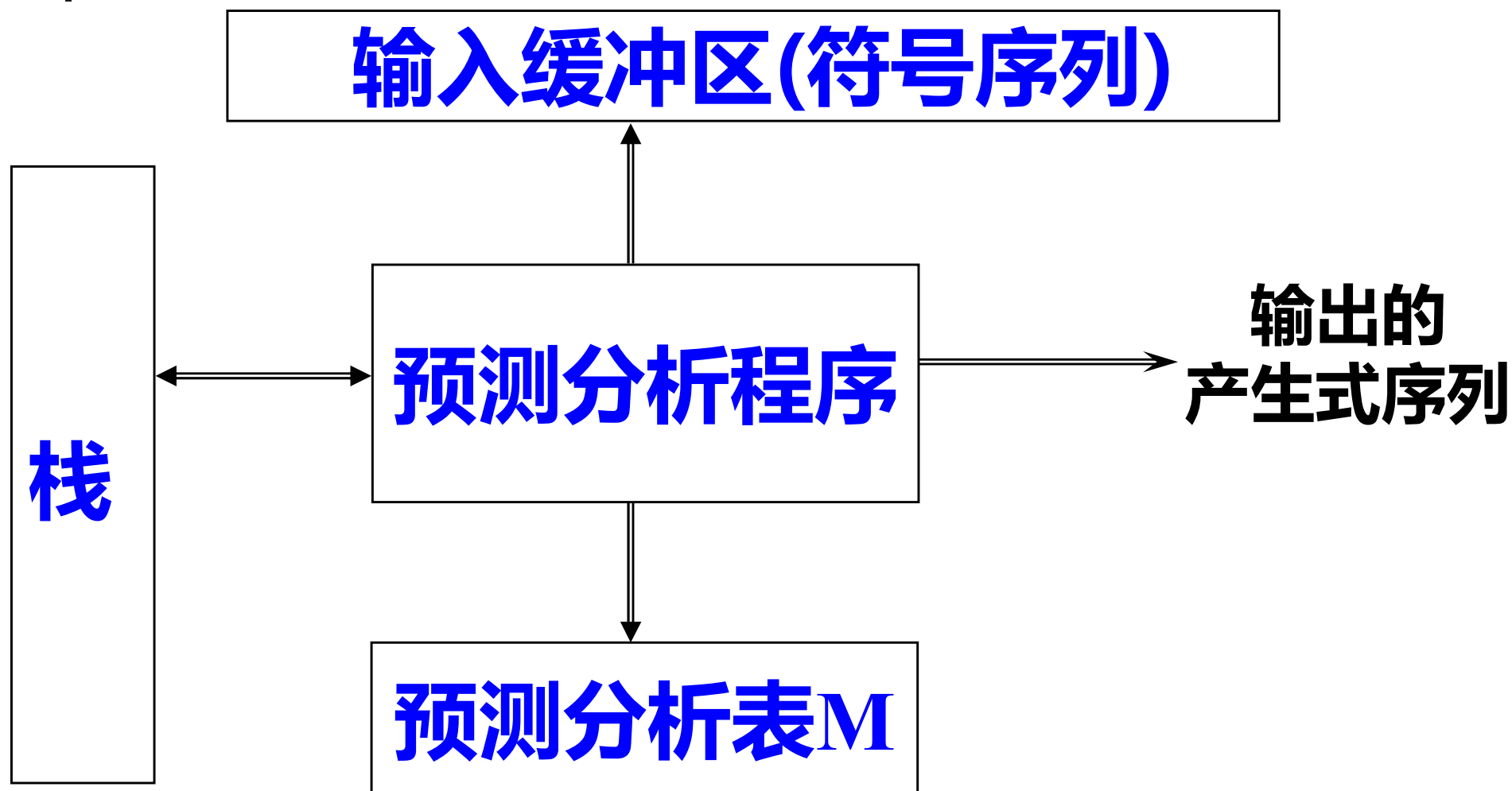
4.3 预测分析法

- **一种高效的自顶向下分析法**
- **能够对LL(1)文法实现确定的自顶向下分析**

4.3 预测分析法

- 通用控制算法，采用表驱动方式实现
 - 分析表 $M[A,a]$ ，即**LL(1)分析表**，存储执行LL(1)分析的信息，其中A是语法变量，a是输入符号。
 - 分析栈，存放文法符号序列，#为栈底符号，**初始时栈顶是开始符号**
 - 输入缓冲区，包括待分析的串和结束符
- 系统维持一个**分析表**和一个**分析栈**，根据当前**输入缓冲区**中扫描到的符号，选择当前语法变量（**处于栈顶**）的候选式进行推导——希望找到相应输入符号串的最左推导。

4.3.1 预测分析器的构成





系统的执行与特点

- 在系统启动时，输入指针指向输入串的第一个字符，分析栈中存放着栈底符号#和文法的开始符号。
- 根据栈顶符号A和读入的符号a，查看分析表M,决定相应的动作。
 - 如果 $A=a=\#$ ，分析成功并停机
 - 如果 $A=a\neq\#$ ，弹出栈顶符号A，并将输入指针指向下一个符号
 - 如果A是语法变量，程序访问分析表M的 $M[A,a]$ 表项，该表项或者是一个A产生式，或者是出错信息。
 - 如果 $M[A,a]=\{A\rightarrow UVW\}$ ，则用WVU(栈顶)替换原栈顶符号A，输出该产生式



系统的执行与特点

- **优点：**
 - 1) **效率高**
 - 2) **便于维护、自动生成**
- **关键——分析表M的构造**



预测分析表的构造算法

算法4.6 预测分析表($LL(1)$ 分析表)的构造算法。

输入：文法 G ;

输出：分析表 M ;

步骤：

- 1. 对 G 中的任意一个产生式 $A \rightarrow \alpha(\text{alpha})$, 执行第2和第3步;**
- 2. for $\forall a \in \text{FIRST}(\alpha)$, 将 $A \rightarrow \alpha(\text{alpha})$ 填入 $M[A, a]$;**
- 3. if $\varepsilon \in \text{FIRST}(\alpha)$ then $\forall a \in \text{FOLLOW}(A)$, 将 $A \rightarrow \alpha(\text{alpha})$ 填入 $M[A, a]$;**
- 4. 将所有无定义的 $M[A, b]$ 标上出错标志。**

例4.10 考虑简单算术表达式文法的实现

$\text{FOLLOW}(E') = \{), \# \}$

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

$\text{FIRST}(TE') = \{ (, \text{id} \}$

$\text{FIRST}(+TE') = \{ + \}$

$\text{FIRST}(FT') = \{ (, \text{id} \}$

$\text{FIRST}(*FT') = \{ * \}$

$\text{FIRST}((E)) = \{ (\}$

$\text{FIRST}(\text{id}) = \{ \text{id} \}$

$E \rightarrow TE' \quad E' \rightarrow +TE' | \varepsilon \quad T \rightarrow FT'$

$T' \rightarrow *FT' | \varepsilon \quad F \rightarrow (E) | \text{id}$


$$E \rightarrow TE'$$
$$E' \rightarrow +TE'$$
$$E' \rightarrow \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'$$
$$T' \rightarrow \varepsilon$$
$$F \rightarrow (E)$$
$$F \rightarrow \text{id}$$

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

$$\text{FOLLOW}(E') = \{), \# \}$$
$$\text{FOLLOW}(T') = \{ +,), \# \}$$
$$\text{FIRST}(TE') = \{ (, \text{id} \}$$
$$\text{FIRST}(+TE') = \{ + \}$$
$$\text{FIRST}(FT') = \{ (, \text{id} \}$$
$$\text{FIRST}(*FT') = \{ * \}$$
$$\text{FIRST}((E)) = \{ (\}$$
$$\text{FIRST}(\text{id}) = \{ \text{id} \}$$

2023/9/29

简单算术表达式文法的预测分析表

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



预测分析法的实现步骤

1. 构造文法
2. 改造文法：消除二义性、消除左递归、提取左因子
3. 求每个候选式的FIRST集和变量的FOLLOW集
4. 检查是不是 LL(1) 文法
若不是 LL(1), 说明文法的复杂性超过自顶向下方法的分析能力, 需要附加新的“信息”
5. 构造预测分析表
6. 实现预测分析器

算法4.5 预测分析程序的总控程序。

输入：输入串 w 和文法 $G=(V, T, P, S)$ 的分析表 M ;

输出：如果 w 属于 $L(G)$ ，则输出 w 的最左推导，否则报告错误;

步骤：

1. 将栈底符号 $\#$ 和文法开始符号 S 压入栈中;

2. repeat

3. $X:=$ 当前栈顶符号;

4. $a:=$ 当前输入符号;

5. if $X \in T \cup \{\#\}$ then

6. if $X=a$ then

7. {if $X \neq \#$ then begin

8. 将 X 弹出栈;

9. 前移输入指针

10. end}

预测分析程序的总控程序

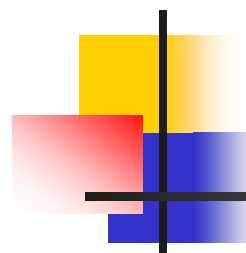
```
11.      else error// 即当前栈顶终结符和当前
           // 输入符号不一致
12.      else //也就是X属于一个语法变量
13.          if  $M[X, a]=Y_1Y_2...Y_k$  then begin
14.              将X弹出栈;
15.              依次将 $Y_k, ..., Y_2, Y_1$ 压入栈;
16.              输出产生式 $X \rightarrow Y_1Y_2...Y_k$ 
17.          end
18.      else error
19.  until  $X=\#$ 
```

简单算术表达式文法的预测分析表

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

对输入串id+id*id进行分析的过程

栈	输入缓冲区	输出
#E	id+id*id#	
#E'T	id+id*id#	$E \rightarrow TE'$
#E'T'F	id+id*id#	$T \rightarrow FT'$
#E'T'id	id+id*id#	$F \rightarrow id$
#E'T'	+id*id#	
#E'	+id*id#	$T' \rightarrow \varepsilon$
#E'T+	+id*id#	$E' \rightarrow +TE'$
#E'T	id*id#	



#E'T

id*id#

(接上表)

#E'T'F

id*id#

$T \rightarrow FT'$

#E'T'id

id*id#

$F \rightarrow id$

#E'T'

***id#**

#E'T'F*

***id#**

$T' \rightarrow *FT'$

#E'T'F

id#

#E'T'id

id#

$F \rightarrow id$

#E'T'

#

#E'

#

$T' \rightarrow \epsilon$

#

#

$E' \rightarrow \epsilon$

输出的产生式序列形成了最左推导对应的分析树



4.3.3 预测分析中错误的处理

■ 错误处理

- 1. 栈顶终结符号和下一个输入符号不匹配
- 2. 栈顶是语法变量 A , a 是下一个输入符号,
 $M[A, a]$ 是空白表项



4.3.3 预测分析中错误的处理

- **紧急方式错误恢复策略**
- **发现错误时跳过一些输入符号，直到下一个语法成分包含的第一个符号为止（同步记号）**



4.3.3 预测分析中错误的处理

- **同步记号的一般选择策略：对语法变量 A ，如果 $M[A,a]$ 无定义，并且 a 属于 $FOLLOW(A)$ ，则增加 $M[A,a]$ 为“同步点” (synch)。当程序到达这个同步点时，放弃对 A 的识别，而转入分析 A 后面的符号。**

4.3.3 预测分析中错误的处理

- 同步记号的启发式选择方法如下：
 - 把FOLLOW(A)的所有符号放入语法变量 A 的同步记号集合。

跳过当前输入符号直到
FOLLOW(A)的某个元素出现，
那么分析栈弹出 A ，继续执行
下面的分析

4.3.3 预测分析中错误的处理

- 同步记号的启发式选择方法如下：
 - 把FOLLOW(A)的所有符号放入语法变量 A 的同步记号集合。
 - 把高层结构的开始符号加到低层结构的同步记号集合中。

语言通常具有层次结构，例如，表达式包括在语句中，而语句包括在块中，等等。如果表达式的识别出现问题，可以直接从下一个语句继续识别，或者如果当前语句出现问题，可以直接从下一个块继续识别。

4.3.3 预测分析中错误的处理

同步记号的启发式选择方法如下：

- 把FOLLOW(A)加入A的同步记号集
- 把高层结集中。
- 把FIRST(A)的符号加入A的同步记号集合。

很可能，对于变量A的识别错误是因为误输入某几个字符，因此如果当前输入字符串出现FIRST(A)的字符，那么可能对A重新开始识别

4.3.3 预测分析中错误的处理

同步记号的启发式选择方法如下：

- 把FOLLOW(A)的所有符号放入语法变量 A 的同步记号集合
- 把高层结构集合中。
- 把FIRST(A)的所有符号放入同步记号集合。
- 如果语法变量可以产生空串，若出错时栈顶是这样的语法变量，则可以使用产生空串的产生式。
- 如果符号在栈顶而不能匹配，则弹出此符号。

一个可用的错误处理选项，可以减少在错误处理阶段需要考虑的变量的数量

4.3.3 预测分析中错误的处理

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$E \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

$FOLLOW(E) = \{ \#,) \}$

$FOLLOW(E') = \{ \#,) \}$

$FOLLOW(T) = \{ +,), \# \}$

$FOLLOW(T') = \{ +,), \# \}$

$FOLLOW(F) = \{ *, +,), \# \}$

4.3.3 预测分析中错误的处理

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$E \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

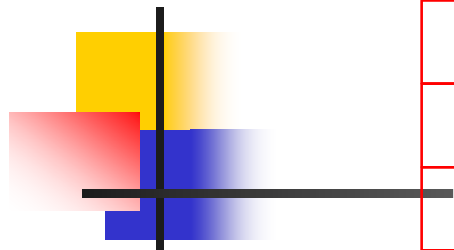
使用方法:

1. 如果表项 $M[A, a]$ 为空, 则跳过输入符号 a ;
2. 如果表项是**synch**, 则弹出栈顶的语法变量并试图恢复分析;
3. 如果栈顶的记号与输入符号不匹配, 则从栈顶弹出该记号。

4.3.3 预测分析中错误的处理

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$E \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

识别 **+id * + id** 的动作



识别 **+id ***
+ id 的动作

步骤	栈	输入	备注
1	#E	+id*+id#	跳过+
2	#E	id*+id#	正常识别
3	#E'T	id*+id#	
4	#E'T'F	id*+id#	
5	#E'T'id	id*+id#	
6	#E'T'	*+id#	
7	#E'T'F*	*+id#	
8	#E'T'F	+id#	Synch

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$E \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch



识别
+ id

语法变量	输入符号					
	id	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$E \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

9	#E'T'	+id#	
10	#E'	+id#	
11	#E'T+	+id#	
12	#E'T	id#	
13	#E'T'F	id#	
14	#E'T'id	id#	
15	#E'T'	#	
16	#E'	#	
17	#	#	



4.4 递归下降分析法

所谓递归下降分析法，是指根据各个候选式的结构，为文法的**每个语法变量编写一个处理程序**，用来识别该语法变量所代表的语法成分



4.4 递归下降分析法

1. 对应每个变量设置一个处理子程序，例如对于语法变量A，有产生式：

$$A \rightarrow X_1 X_2 \dots X_k \dots X_n$$

(1) 当遇到 X_k 是终结符号时直接进行匹配；

(2) 当遇到 X_k 是语法变量时就调用 X_k 对应的处理子程序。

2. 要求处理子程序是可以**递归调用**的



4.4.1 递归下降分析法的基本思想

例4.14 对于产生式 $E' \rightarrow +TE'$ ，与 E' 对应的子程序可以按如下方式来编写：

```
procedure  $E'$ 
begin
    match('+');
     $T$ ;           /*调用识别 $T$ 的过程*/
     $E'$            /*调用识别 $E'$ 的过程*/
end;
```




4.4.1 递归下降分析法的基本思想

服务子程序 $match$ 用来匹配当前的输入记号，其代码为：

```
procedure match( $t:token$ );  
begin  
    if  $lookhead=t$  then // lookhead用于保存当前输入记号  
         $lookhead:=nexttoken$ ; // nexttoken读取下一个符号  
    else error           /*调用出错处理程序*/  
end;
```



4.4.2 语法图和递归子程序法

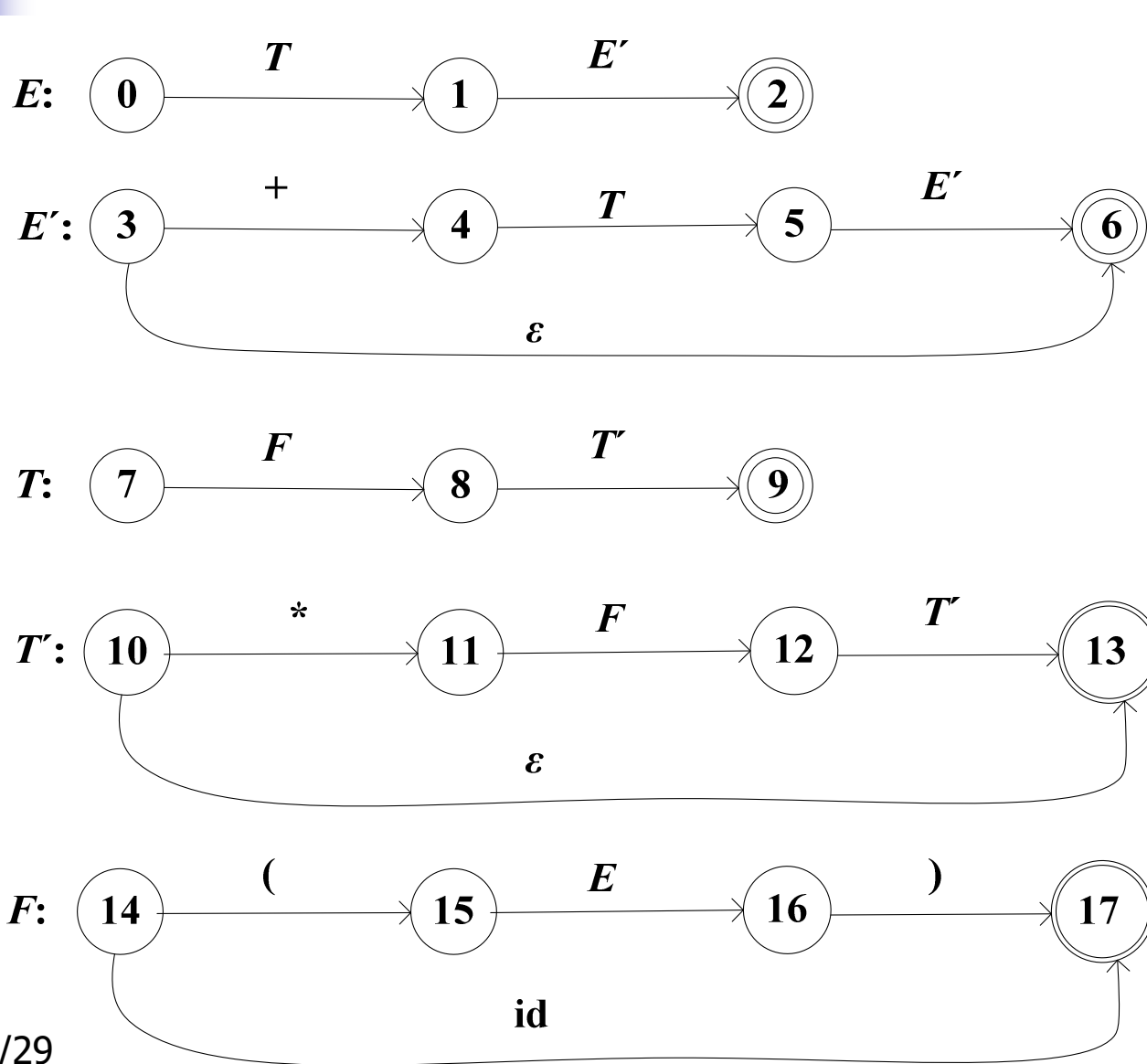
- **状态转换图（语法图）是非常有用的设计工具，既可用于词法分析，也可以用来帮助建立语法分析程序**



4.4.2 语法图和递归子程序法

- 从文法构造状态转换图，对每个非终结符A执行如下操作
 - 创建一个开始状态和一个终止状态（返回状态）
 - 对每个产生式 $A \rightarrow X_1 X_2 \dots X_n$ ，创建一条从开始状态到终止状态的路径，边上的标记分别为 X_1, X_2, \dots, X_n
 - 如上的状态转换图表达的是语法成分的构成，简称**语法图**

例4.15 简单表达式文法的语法图



$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) | id$

4.4.3 基于语法图的语法分析器工作方式

- 初始时，分析器进入状态图的开始状态，输入指针指向输入符号串的第一个符号。
- 如果经过一些动作后，它进入状态 s ，且从状态 s 到状态 t 的边上标记了终结符 a ，此时下一个输入符又正好是 a ，则分析器将输入指针向右移动一位，并进入状态 t 。

4.4.3 基于语法图的语法分析器工作方式

- 另一方面，如果边上标记的是非终结符 A ，则分析器进入 A 的初始状态，但不移动输入指针。一旦到达 A 的终态，则立刻进入状态 t ，事实上，分析器从状态 s 转移到状态 t 时，它已经从输入符号串读了 A （调用 A 对应的过程）。
- 最后，如果从 s 到 t 有一条标记为 ϵ 的边，那么分析器从状态 s 直接进入状态 t 而不移动输入指针。



4.4.4 语法图的化简与实现

(1) 左因子提取

将形如 $A \rightarrow YX \mid YZ$ 的产生式替换为 $A \rightarrow Y(X \mid Z)$;

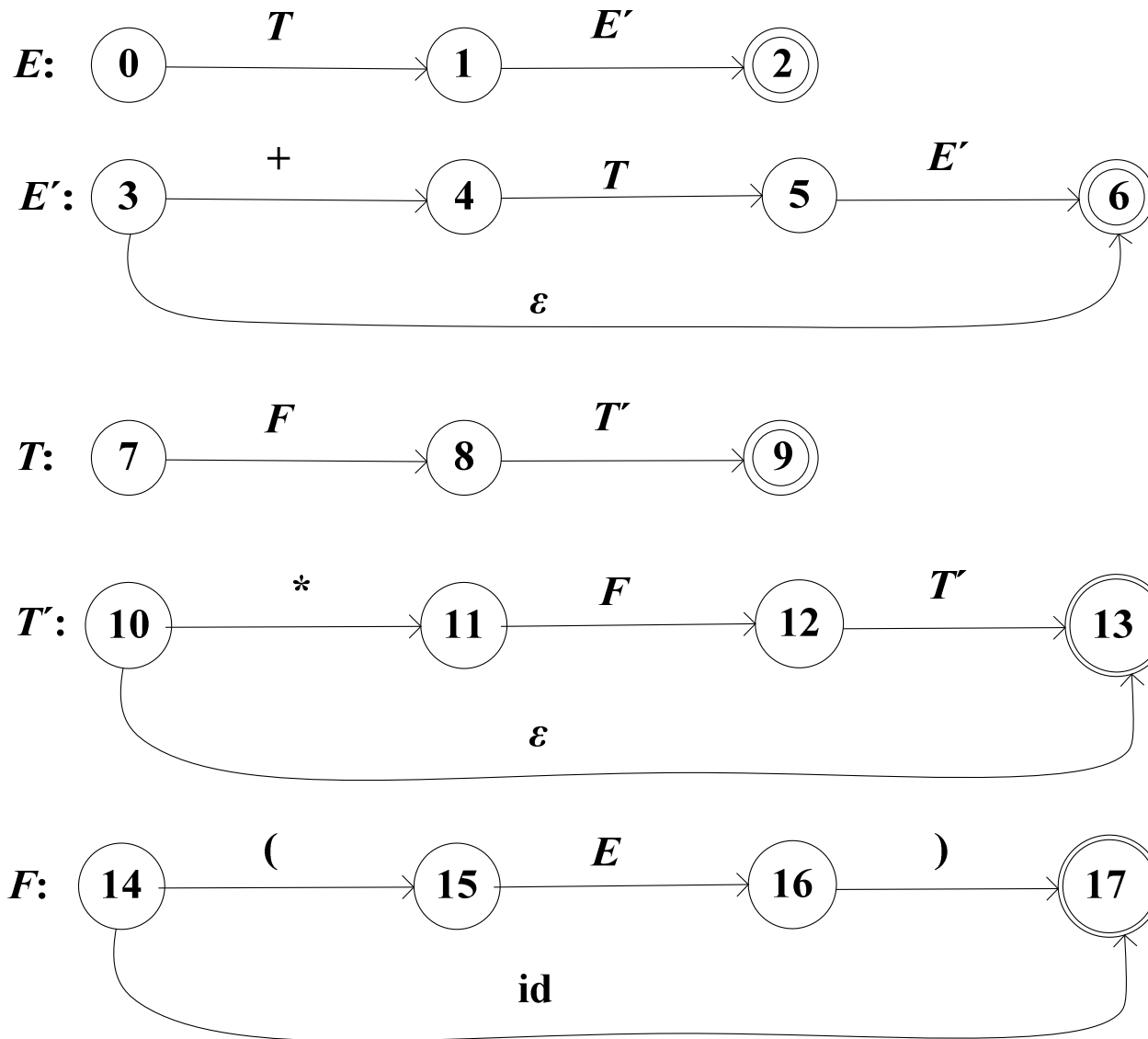
(2) 右因子提取

将形如 $A \rightarrow YX \mid ZX$ 的产生式替换为 $A \rightarrow (Y \mid Z)X$;

(3) 尾递归消除

将形如 $X \rightarrow YX \mid Z$ 的产生式替换为 $X \rightarrow Y^*Z$ 。

4.4.4 语法图的化简与实现



$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T' \rightarrow *FT' | \varepsilon$

$F \rightarrow (E) | id$

4.4.4 语法图的化简与实现

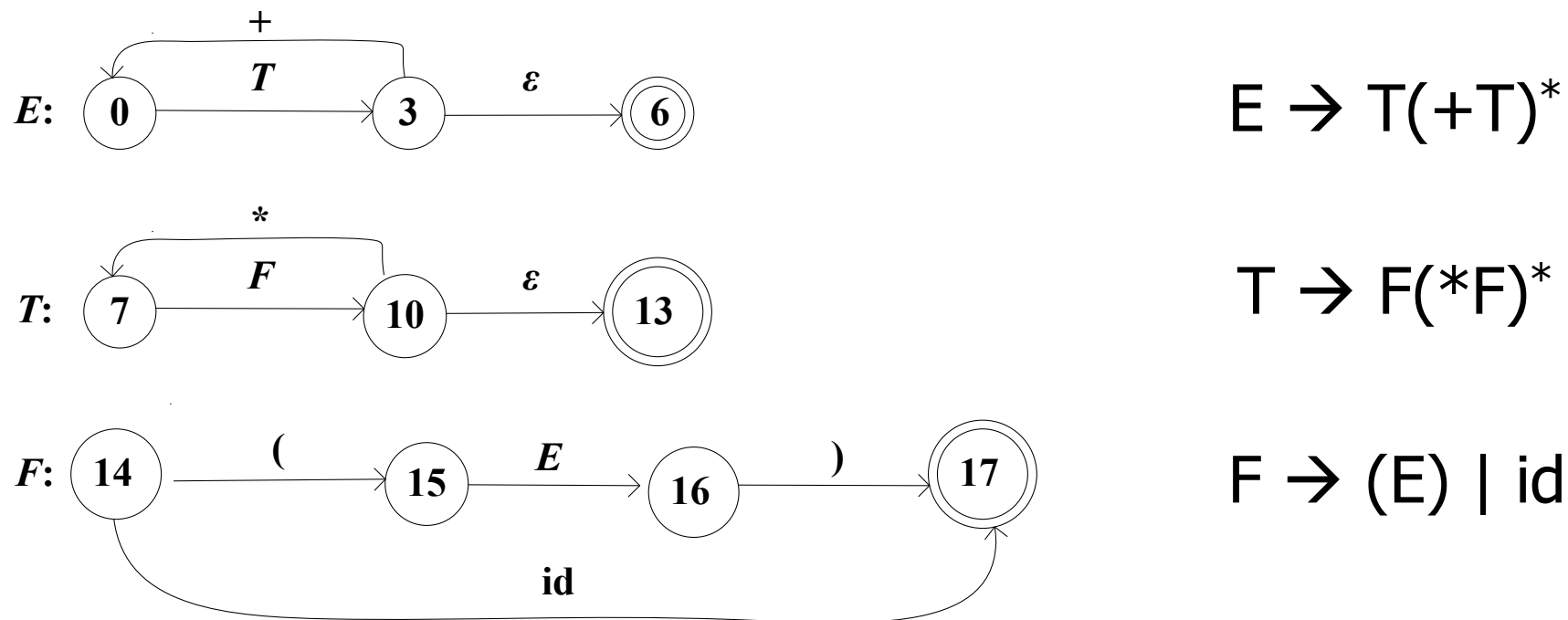


图4.6算术表达式的简化语法图



例4.16 简单算术表达式的语法分析器

- **E 的子程序**($E \rightarrow T(+T)^*$)

procedure E;

begin

T; **T的过程调用**

while lookahead='+' do

begin **当前符号等于+时**

match('+'); **处理终结符+**

T **T的过程调用**

end

end; **lookhead: 当前符号**



T 的子程序 ($T \rightarrow F (*F) *$)

procedure T;

begin

F;

F的过程调用

while lookahead='*' then

begin

当前符号等于*时

match('*');

处理终结符*

F

F的递归调用

end

end;



F 的子程序($F \rightarrow (E)|id$)

procedure F;

begin

if lookahead='(' then

begin

match('(');

E;

match(')');

end

else if lookahead=id then

match(id)

else error

end

当前符号等于 (

处理终结符 (

E 的递归调用

处理终结符)

处理终结符id

出错处理



主程序

begin

lookhead:=nexttoken;

E

end

**调用词法分析程序
调用识别 E 的过程**

procedure match(t:token); 服务子程序

begin

if lookhead=t then

lookhead:=nexttoken

else error 出错处理程序

end;



4.4.5 递归子程序法的实现步骤

- 1) 构造文法;
- 2) 改造文法: 消除二义性、消除左递归、提取左因子;
- 3) 求每个候选式的FIRST集和语法变量的FOLLOW集;
- 4) 检查 G 是不是 $LL(1)$ 文法, 若 G 不是 $LL(1)$ 文法, 说明文法 G 的复杂性超过了自顶向下方法的分析能力, 需要附加新的“信息”;
- 5) 按照 $LL(1)$ 文法画语法图;
- 6) 化简语法图;
- 7) 按照语法图为每个语法变量设置一个子程序。



递归子程序法的优缺点分析

- **优点:**

- 1) **直观、简单、可读性好**
- 2) **便于扩充**

- **缺点:**

- 1) **递归算法的实现效率低**
- 2) **处理能力相对有限**
- 3) **通用性差，难以自动生成**



本章小结

1. 自顶向下分析法和自底向上分析法分别寻找输入串的最左推导和最左归约
2. 自顶向下分析会遇到二义性问题、回溯问题、左递归引起的无穷推导问题，需对文法进行改造：消除二义性、消除左递归、提取公共左因子
3. $LL(1)$ 文法是一类可以进行确定分析的文法，利用FIRST集和FOLLOW集可以判定某个上下文无关文法是否为 $LL(1)$ 文法



本章小结

4. $LL(1)$ 文法可以用 $LL(1)$ 分析法进行分析。
5. 递归下降分析法根据各个候选式的结构为每个非终结符编写一个子程序。
6. 使用语法图可以方便地进行递归子程序的设计。