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，对它们的理解以及在构造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

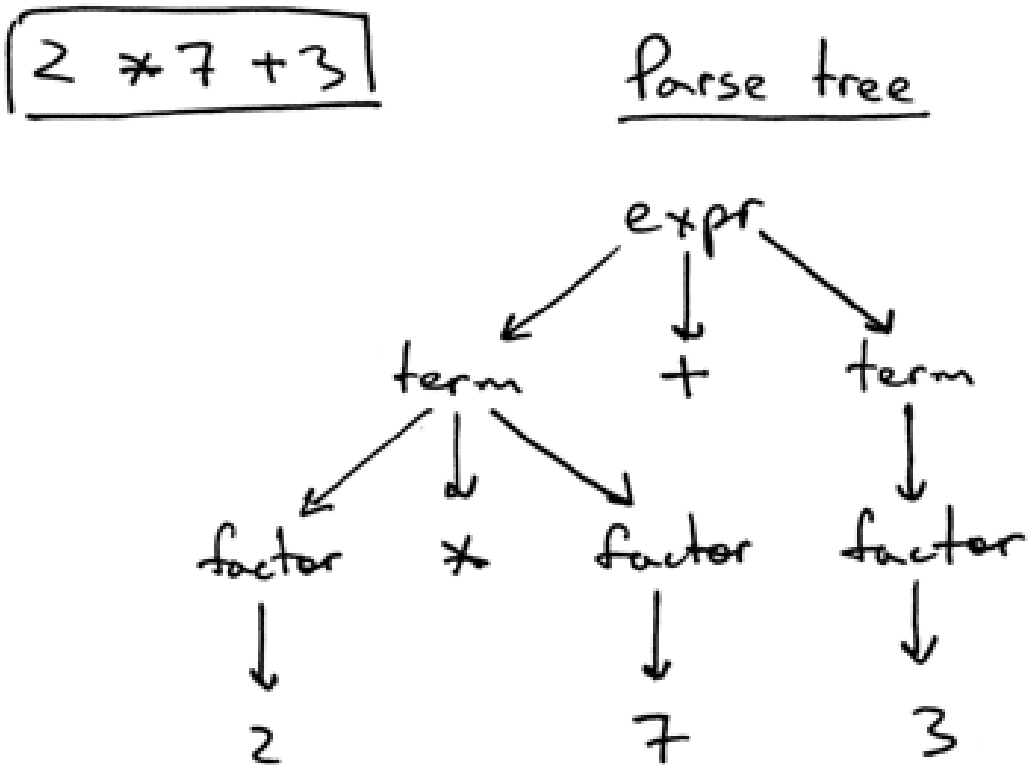
4.2 自顶向下的语法分析面临的问题与文法的改造

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。**

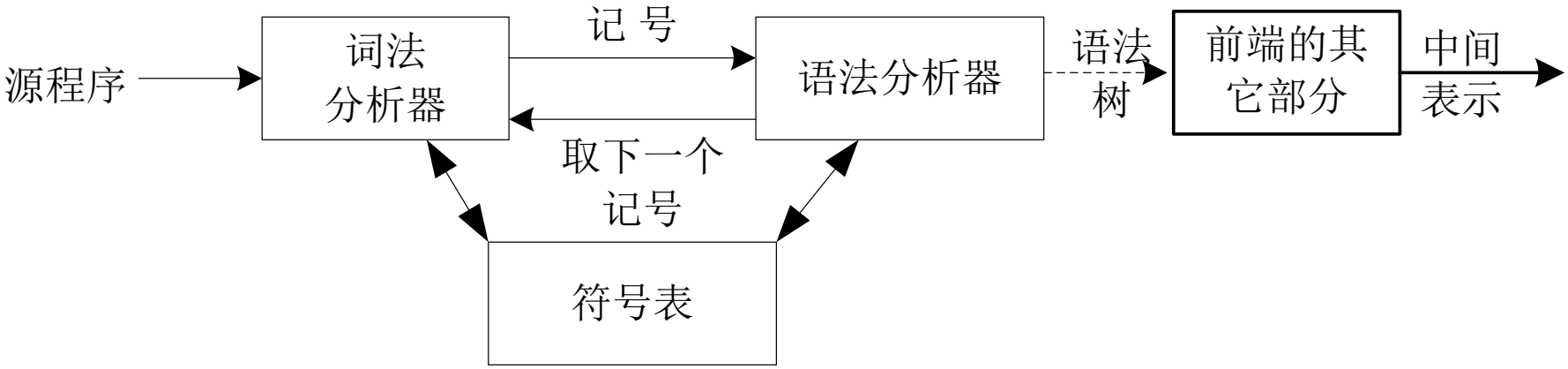


图4.1　语法分析器在编译器中的位置

4.1 语法分析概述

从文法产生语言的角度

从根开始，逐步为某语句构造一棵语法树

- 自顶向下 Top Down
  - 递归子程序法
  - 预测分析法(LL(1))
  - 算符优先分析法

从自动机识别语言的角度

相反，将一句子归约为开始符号

- 自底向上 Bottom Up
  - LR(0)、SLR(1)、LR(1)、LALR(1)

问题：解决确定性问题！

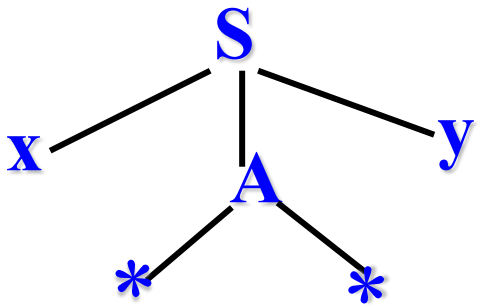
假定文法是压缩的：即删除了单位产生式和无用产生式。

4.2 自顶向下的语法分析面临的问题与文法的改造

- 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。
  - 从树根S开始，构造所给输入符号串的语法树
- 例:设有G：S→xAy A→**|*，输入串：x**y

$S \Rightarrow xAy$

$\Rightarrow x**y$



4.2.1 自顶向下分析面临的问题

什么样的文法会对自顶向下的分析造成困难?

4.2.1 自顶向下分析面临的问题

1.二义性问题

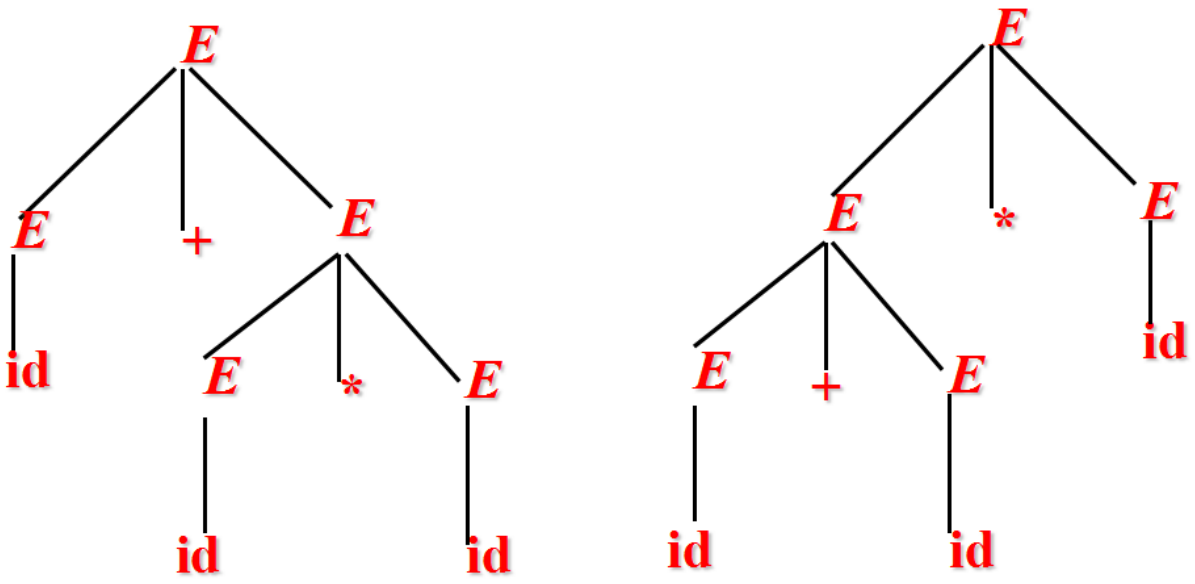
- 对于文法G，如果$L(G)$中存在一个具有两棵或两棵以上分析树的句子，则称G是二义性的。也可以等价地说：如果$L(G)$中存在一个具有两个或两个以上最左(或最右)推导的句子，则G是二义性文法。

- 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：$E \rightarrow id \mid c \mid E+E \mid E-E \mid E*E \mid E/E \mid E\uparrow E \mid (E)$



解决办法：改造文法，引入新的文法变量

4.2.1 自顶向下分析面临的问题

2.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)。
- Ge：E→T　E→E+T　E→E-T　T→F　T→T*F　T→T/F　F→(E)　F→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

2.回溯问题

- $E \Rightarrow T$ (4.1)
- $E \Rightarrow T \Rightarrow F$ (4.2)
- $E \Rightarrow T \Rightarrow F \Rightarrow (E)$ (4.3)
- $E \Rightarrow T \Rightarrow F \Rightarrow \mathrm{id}$ (4.4)
- $E \Rightarrow T \Rightarrow T^{*}F$ (4.5)
- ……

4.2.2节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。

4.2.1 自顶向下分析面临的问题

3.左递归 - 无穷推导问题

- 假设A是文法G的某个语法变量，如果存在推导$A \overset{+}{\Rightarrow} \alpha A \beta$，则称文法$G$是递归的，当$\alpha=\varepsilon$时称之为左递归；如果$A \overset{+}{\Rightarrow} \alpha A \beta$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。
- *Ger*：$E \rightarrow T$　$E \rightarrow E+T$　$E \rightarrow E-T$　$T \rightarrow \mathrm{F}$　$T \rightarrow T*F$　$T \rightarrow T/F$　$F \rightarrow (E)$　$F \rightarrow \mathrm{id}$
- 考虑为输入串id+id*id建立一个最左推导
- $E \Rightarrow E+T \Rightarrow E+T+T \Rightarrow E+T+T+T \Rightarrow \cdots\cdots$

4.2.2 对上下文无关文法的改造

1.消除二义性

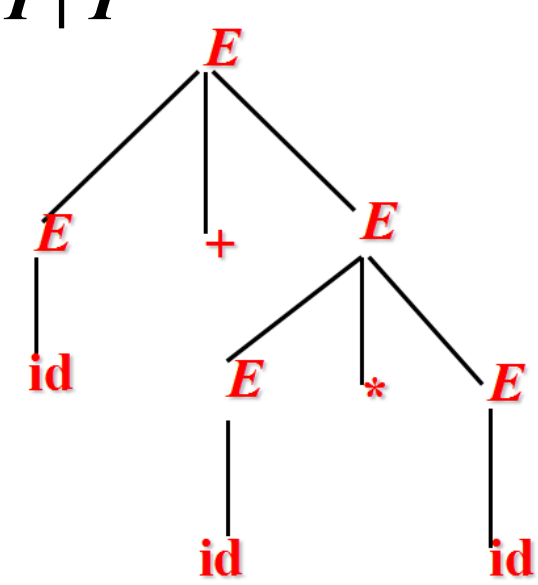
- G：$E \rightarrow id \mid c \mid E+E \mid E-E \mid E*E \mid E/E \mid E\uparrow E \mid (E)$

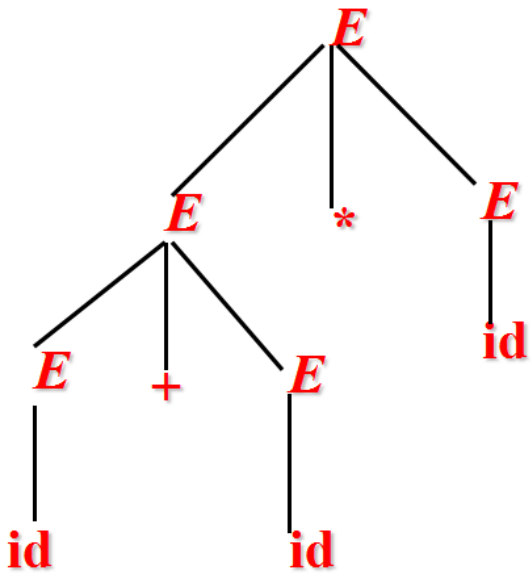
G': $E \rightarrow E+T \mid E \rightarrow E-T \mid T$

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

$F \rightarrow F\uparrow P \mid P$

$P \rightarrow (E) \mid id \mid c$





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。其实，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。
- <*stmt*>→ if <*expr*> then <*stmt*>
- | if <*expr*> then <*stmt*> else <*stmt*>
- | other (4.7)
- 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<stmt>，从而导致该文法是二义性的。

4.2.2 对上下文无关文法的改造

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

- <*stmt*>→ <*matched_stmt*>
  | <*unmathched_stmt*>
- <*matched_stmt*>→ if <*expr*> then <*matched_stmt*> else <*matched_stmt*>
  | other
- <*unmathched_stmt*>→ if <*expr*> then <*stmt*>
  | if <*expr*> then <*matched_stmt*> else <*unmathched_stmt*>

4.2.2 对上下文无关文法的改造

2.消除左递归

- **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**
  $A\rightarrow\beta A'\quad A'\rightarrow\alpha A'|\varepsilon$
- $E\rightarrow E+T|T\quad T\rightarrow T*F|F\quad F\rightarrow(E)|$**id替换为：**
- $E\rightarrow TE'\quad E'\rightarrow+TE'|\varepsilon\quad T\rightarrow FT'\quad T'\rightarrow*FT'|\varepsilon$
  $F\rightarrow(E)|$id

- **一般地，假设文法G中的语法变量A的所有产生式如下：**
  $A\rightarrow A\alpha_1|A\alpha_2|\ldots|A\alpha_n|\beta_1|\beta_2|\ldots|\beta_m$
  其中，$\beta_i(i=1,2,\ldots,m)$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：
  $A\rightarrow\beta_1A'|\beta_2A'|\ldots|\beta_mA'\quad A'\rightarrow\alpha_1A'|\alpha_2A'|\ldots|\alpha_nA'|\varepsilon$

4.2.2 对上下文无关文法的改造

2.消除左递归

- **练习：消除B→Bbc|aBc|d的直接左递归**

$B \rightarrow aBcB' | dB'$

$B' \rightarrow bcB' | \varepsilon$

4.2.2 对上下文无关文法的改造

2.消除左递归

- 如何消除所有左递归？（包括间接左递归）
- 例：消除下列文法左递归

S→Ac | c

A→Bb | b

B→Sa | a

关键：需要先将间接左递归变为直接左递归，然后再按清除直接左递归

B→Sa|a

A→Sab|ab|b

S→Sabc|abc|bc|c

S→abcS'|bcS'|cS' S'→abcS'|ε

B→Sa|a

A→Sab|ab|b

S→abcS'|bcS'|cS'

S'→abcS'|ε

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（形如$A \overset{+}{\Rightarrow} A$）和$\varepsilon$-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的所有语法变量排序(编号)，假设排序后的语法变量记为A_1，A_2，…，A_n;
2. for i←1 to n {
3. for j←1 to i-1 {
4. 用产生式$A_i \rightarrow \alpha_1\beta|\alpha_2\beta|\ldots|\alpha_k\beta$代替每个形如$A_i \rightarrow A_j\beta$的产生式，其中，$A_j \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_k$是所有的当前$A_j$产生式;
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。

- **例：消除下列文法左递归**

S→Ac | c

A→Bb | b

B→Sa | a

排序：1.B，2.A，3.S

i=1： B→Sa | a

i=2： B→Sa | a　A→Sab | ab | b

i=3：B→Sa | a　A→Sab | ab | b　　S→Sabc|abc|bc|c

消除直接左递归：S→abcS'|bcS'|cS'　　S'→abcS'|ε

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。如果$\alpha \neq \varepsilon$，则用下面的产生式替换所有的A产生式$A \to \alpha\beta_1|\alpha\beta_2|\ldots|\alpha\beta_n|\gamma_1|\gamma_2|\ldots|\gamma_n$，其中$\gamma 1$，$\gamma 2$，…，$\gamma n$表示所有不以$\alpha$开头的候选式：

$$A \to \alpha A'|\gamma_1|\gamma_2|\ldots|\gamma_n$$

$$A' \to \beta_1|\beta_2|\ldots|\beta_n$$

  - 其中，A'是新引入的语法变量。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。请读者自行给出这个变换的算法。

4.2.2 对上下文无关文法的改造

- 3.提取左因子
  - 练习：提取左因子 S →aA|aB|aC|a|b|c

S →aS'|a|b|c

S'→A|B|C

4.2.3 LL(1)文法

什么样的文法对其句子才能进行确定的自顶向下分析?

4.2.3 LL(1)文法

什么是确定的自顶向下分析?

S→aA | aB
A→a
B→b

字符串ab的派生过程？

4.2.3 LL(1)文法

什么是确定的自顶向下分析?

S→A | B
A→aa
B→ab

字符串ab的派生过程？

启示？

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，$A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_m$，选择$A$的某个候选式$\alpha$来替换$A$，并使得从$\alpha$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惟一的。
- 第一个终结符是指符号串的第一个符号，并且是终结符号，可以称为首终结符号。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记作FIRST(α)：

FIRST(α)=$\{a|\alpha \overset{*}{\Rightarrow} a\beta, a\in T, \beta\in(V\cup T)^*\}$。

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

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

且$\varepsilon\notin$FIRST(α_1)$\cup$FIRST(α_2)$\cup\ldots\cup$FIRST(α_n)

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

FIRST(α_i)$\cap$FIRST(α_j)=$\varnothing$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

存在空产生式的情况：FIRST集不能完全解决问题

S→aAB | bB
A→ b |ε
B→ b

字符串ab的派生过程？

启示？

4.2.3 LL(1)文法

如果存在$A \to \varepsilon$这样的产生式，则需定义FOLLOW(A)

$\forall A \in V$定义A的**后续符号集为**：

1. $\text{FOLLOW}(A)=\{a \mid S \overset{*}{\Rightarrow} \alpha A a \beta,\ a \in T,\ \alpha,\beta \in (V \cup T)^*\}$
2. 如果A是某个句型的最右符号，则将结束符#添加到FOLLOW(A)中
3. 如果$\alpha_j \overset{*}{\Rightarrow} \varepsilon$，则如果对$\forall i (1 \le i \le m;\ i \ne j)$，$\text{FIRST}(\alpha_i) \cap \text{FOLLOW}(A)=\varnothing$均成立，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G的任意两个具有相同左部的产生式$A \to \alpha | \beta$满足下列条件：

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

则称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代表在分析过程中执行每步推导都要向前查看一个输入符号

$LL(1)$文法既不是二义性的，也不是左递归的

LL(1)文法的判定

算法4.2 计算FIRST(X)。

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

输出：FIRST(X);

步骤：

1．FIRST(X)= ∅;

2．if ($X\in T$) then FIRST(X):= $\{X\}$;

3．if $X\in V$ then

begin

4．if ($X\to\varepsilon\notin P$) then FIRST(X):= FIRST(X)∪$\{a|X\to a\ldots\in P\}$;

5．if ($X\to\varepsilon\in P$) then FIRST(X):= FIRST(X)∪$\{a|X\to a\ldots\in P\}\cup\{\varepsilon\}$

end

6．对$\forall X\in V$，重复如下的过程7-10，直到所有FIRST集不变为止。

7．if ($X\to Y\ldots\in P$ and $Y\in V$) then FIRST(X):= FIRST(X)∪(FIRST(Y)-$\{\varepsilon\}$);

8．if ($X\to Y_1\ldots Y_n\in P$ and $Y_1\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$) then

9．for k=2 to i do FIRST(X):= FIRST(X)∪(FIRST(Y_k)-$\{\varepsilon\}$);

10．if $Y_1\ldots Y_n \overset{*}{\Rightarrow}\varepsilon$ then FIRST(X):=FIRST(X)∪$\{\varepsilon\}$;

*

LL(1)文法的判定

算法4.3 计算FIRST(α)。

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

输出：FIRST(α);

步骤：

1．计算FIRST(X_1);

2．FIRST(α):= FIRST(X_1)-{ε};

3．k:=1;

4．while (ε∈FIRST(X_k) and $k<n$) do begin

5．　FIRST(α):= FIRST(α)∪(FIRST(X_{k+1})-{ε});

6．　k:=k+1 end

7．if ($k=n$ and ε∈FIRST(X_k)) then FIRST(α):=FIRST(α)∪{ε};

例　表达式文法的语法符号的FIRST 集

FIRST(F)={（, id}

FIRST(T)=FIRST(F)={（, id}

FIRST(E)=FIRST(T)={（, id}

FIRST(E')={+，ε}

FIRST(T')={*,ε}

FIRST(+)={+}, FIRST(*)={*}

FIRST(（)={（}

FIRST()）)={)｝

FIRST(id)={id}

E→TE'

E'→+TE’|ε

T→FT'

T'→*FT’|ε

F→(E)|id

LL(1)文法的判定

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：

1．对$\forall X\in V$，FOLLOW(X) := $\varnothing$;

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)∪FIRST(β)–{ε};

5．若$A\rightarrow\alpha B$或$A\rightarrow\alpha B\beta\in P$，且$\beta \overset{*}{\Rightarrow}\varepsilon$，$A\neq B$，则
FOLLOW(B):=FOLLOW(B)∪FOLLOW(A);

例 表达式文法的语法变量的 FOLLOW 集

E→TE'　E'→+TE'|ε

T→FT'　T'→*FT'|ε

F→(E)|id

FIRST(F)={（,id}

FIRST(T)=FIRST(F)={（,id}

FIRST(E)=FIRST(T)={（,id}

FIRST(E')={+，ε}

FIRST(T')={*,ε}

FOLLOW(E) = { #,) }

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

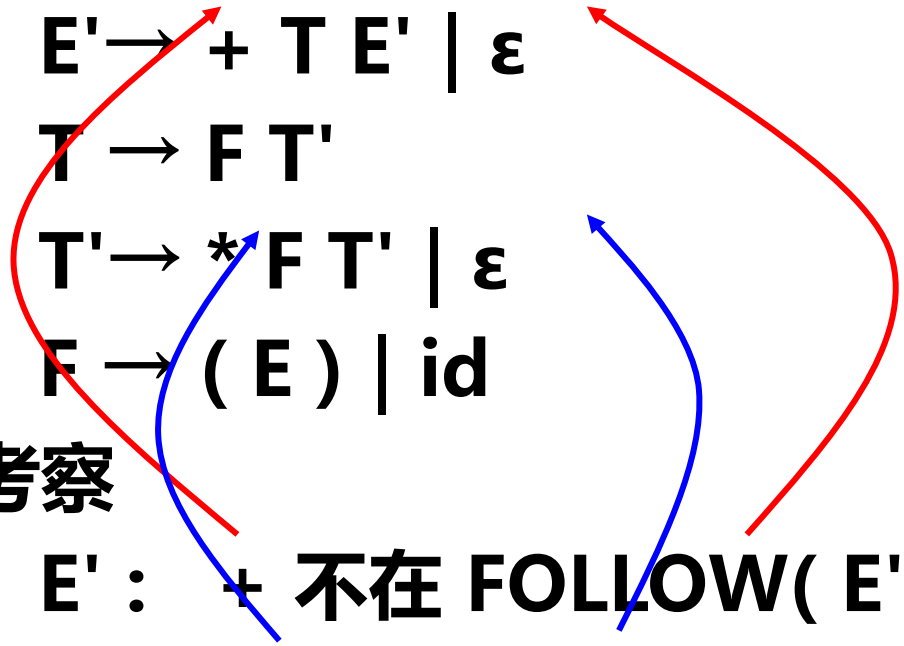
FOLLOW(T) = {FIRST(E')-{ε}}∪FOLLOW(E)∪FOLLOW(E')= {+,),#}

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

FOLLOW(F) = FIRST(T')∪FOLLOW(T)∪FOLLOW(T') ={*,+,),#}

表达式文法是 LL(1) 文法

- E → T E'
- E'→ + T E' | ε
- T → F T'
- T'→ * F T' | ε
- F → (E) | id



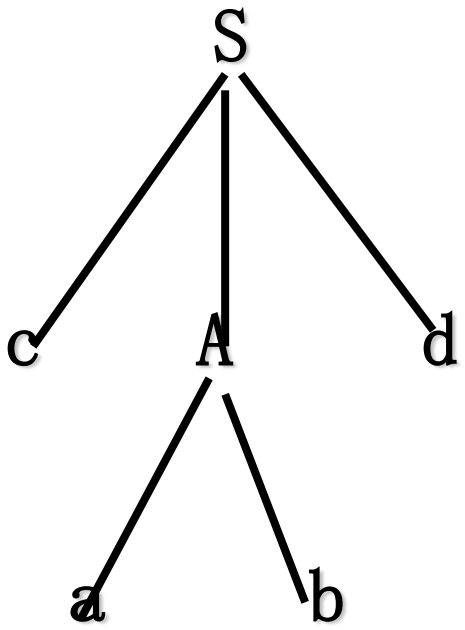
考察

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

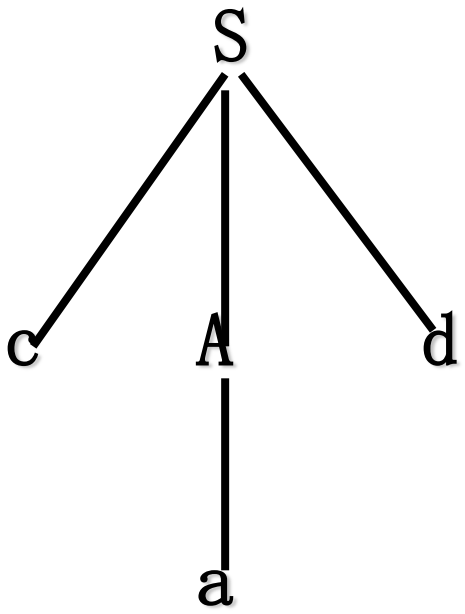
例 对文法

- S→cAd
- A→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

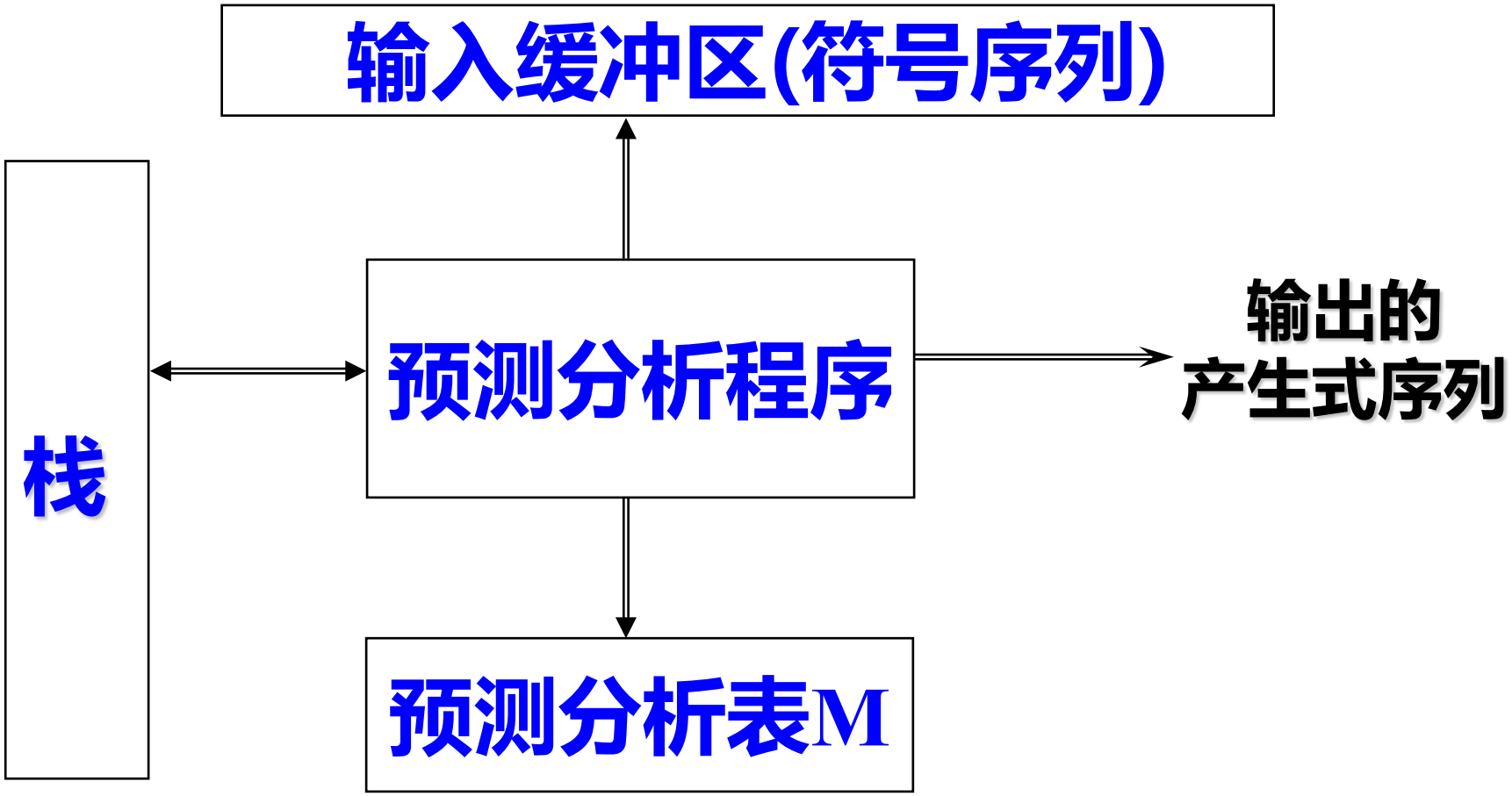
- 将非LL(1)文法改写为等价的LL(1)文法

- 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

- 系统维持一个分析表和一个分析栈，根据当前扫描到的符号，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。
- 一个通用的控制算法
- 一个分析栈，#为栈底符号
- 一个输入缓冲区，#为输入串结束符
- 一个统一形式的分析表M
  - 不同语言使用内容不同的分析表

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,以决定相应的动作。
- 优点：
  1）效率高
  2）便于维护、自动生成
- 关键——分析表M的构造

预测分析程序的总控程序

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∈T∪{#} then
6．                    if X=a then
7．                    {if X≠# 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
12.           else
13.                 if M[X, a]=Y1Y2…Yk then begin
14.                 将X弹出栈;
15.                 依次将Yk，…，Y2，Y1压入栈;
16.                 输出产生式X→Y1Y2…Yk
17.                 end
18.                 else error
19.   until X=#
```

例4.10 考虑简单算术表达式文法的实现

FOLLOW(E')={), # }

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

FIRST(TE')={(,id}　　FIRST(+TE')={+}

FIRST(FT')={(,id}　　FIRST(*FT')={*}

FIRST((E))={(}　　FIRST(id)={id}

E→TE'　E'→+TE'|ε　T→FT'

T'→*FT'|ε　F→(E)|id

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	→TE′			→TE′		
E′		→+TE			→ε	→ε
T	→FT′			→FT′		
T′		→ε	→*FT′		→ε	→ε
F	→id			→(E)		

对输入串id+id*id进行分析的过程

（在黑板上同时画出语法树）

栈	输入缓冲区	输出
#E	id+id*id#	
#E'T	id+id*id#	E→TE'
#E'T'F	id+id*id#	T→FT'
#E'T'id	id+id*id#	F→id
#E'T'	+id*id#	
#E'	+id*id#	T'→ε
#E'T+	+id*id#	E'→+TE'
#E'T	id*id#	

#E'T	id*id#	
#E'T'F	id*id#	T→FT'
#E'T'id	id*id#	F→id
#E'T'	*id#	
#E'T'F*	*id#	T'→*FT'
#E'T'F	id#	
#E'T'id	id#	F→id
#E'T'	#	
#E'	#	T'→ε
#	#	E'→ε

输出的产生式序列形成了最左推导对应的分析树

4.3.2 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$A\to\alpha$, 执行第2步和第3步;

2． for $\forall a\in$FIRST(α), 将$A\to\alpha$填入$M[A, a]$;

3． if $\varepsilon\in$FIRST(α) then $\forall a\in$FOLLOW(A),将$A\to\alpha$填入$M[A, a]$;

if $\varepsilon\in$FIRST(α)&#$\in$FOLLOW(A) then将$A\to\alpha$填入$M[A, \#]$;

4．将所有无定义的$M[A, b]$标上出错标志。

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

4.3.3 预测分析中错误的处理

- 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), 同步记号选择方法如下：
  - 把FOLLOW(*A*)的所有符号放入语法变量*A*的同步记号集合中。
  - 把高层结构的开始符号加到低层结构的同步记号集合中。
  - 把FIRST(*A*)的符号加入*A*的同步记号集合。
  - 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。

4.4 递归下降分析法—一个设想

E→TE'

E'→+TE'|ε

T→FT'

T'→*FT'|ε

F→(E)|id

1. 对应每个变量设置一个处理子程序：

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

(1) 当遇到X_k是终极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

其中，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:=nexttoken;
     else error                  /*调用出错处理程序*/
  end;
```

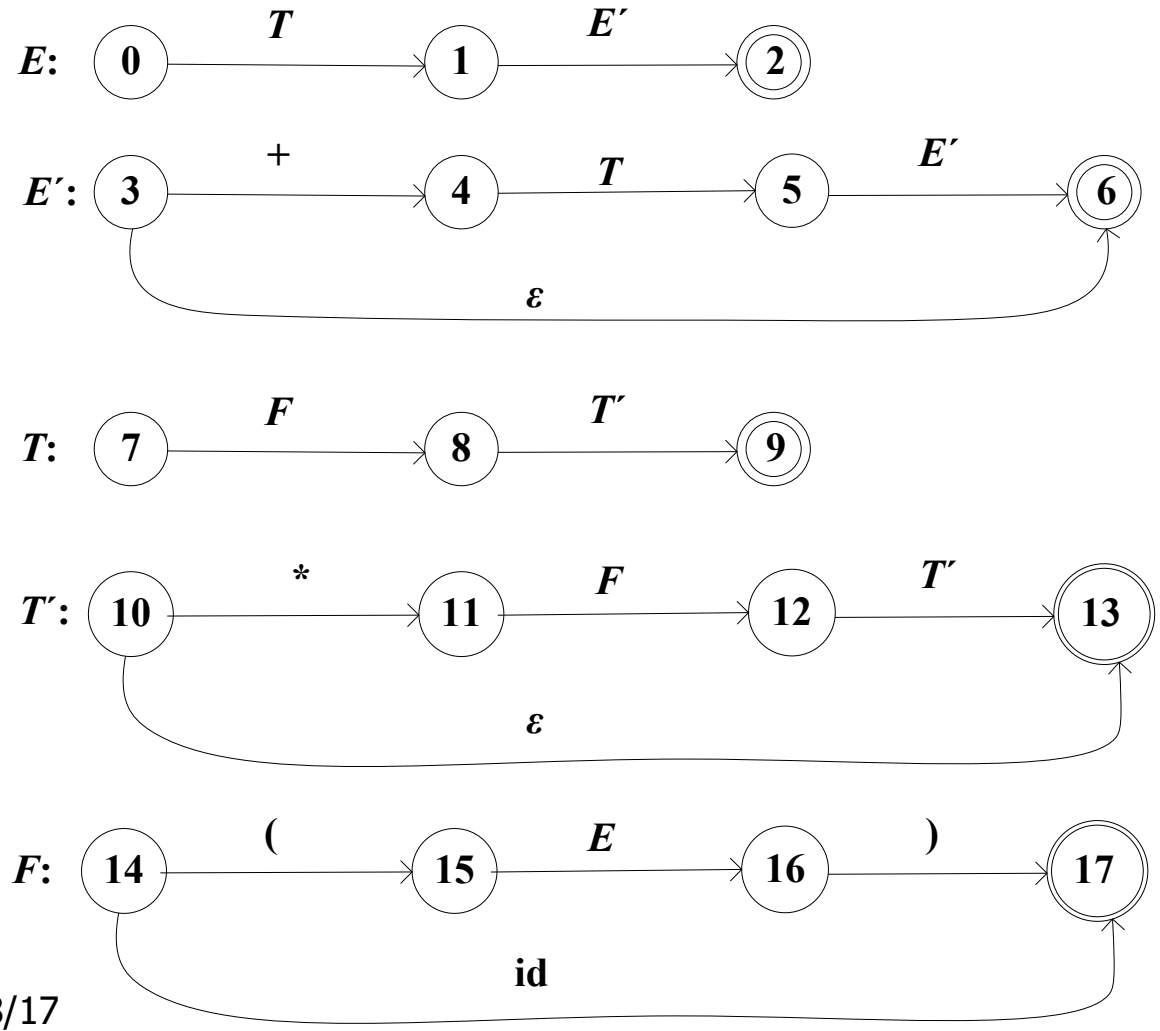
4.4.2 语法图和递归子程序法

- 状态转换图（语法图）是非常有用的设计工具
- 语法分析器和词法分析器的状态转换图不同
  - 每个非终结符对应一个状态转换图，边上的标记是记号和非终结符
  - 记号上的转换意味着如果该记号是下一个输入符号，就应进行转换
  - 非终结符A上的转换是对与A对应的过程的调用

4.4.2 语法图和递归子程序法

- 从文法构造语法图，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$A \rightarrow X_1 X_2 \ \ldots X_n$，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，…，X_n

例4.15 简单表达式文法的语法图



E→TE‘

E'→+TE'|ε

T→FT'

T'→*FT'|ε

F→(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$A \rightarrow YX|YZ$的产生式替换为$A \rightarrow Y(X|Z)$;

(2) 右因子提取

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

(3) 尾递归消除

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

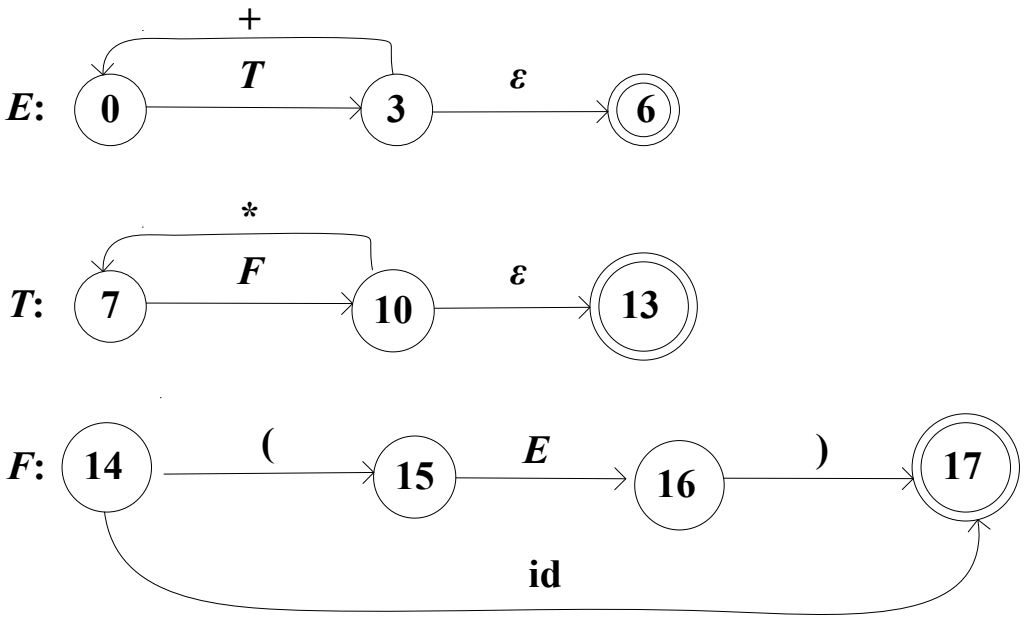


图4.6算术表达式的简化语法图

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 lookhead='+' 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→F(*F)*)

```
procedure T;
  begin
     F;                          F的过程调用
     while lookhead='*' 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→(E)|id)

```
procedure F;
  begin
     if lookhead='(' then
        begin                  当前符号等于（
           match('(');         处理终结符（
           E;                          E 的递归调用
           match(')');         处理终结符)
        end
     else if lookhead=id then
                    match(id)          处理终结符id
     else error                出错处理
  end
```

主程序

```
begin
    lookhead:=nexttoken;        调词法分析程序
    E                           E 的过程调用
end
```

服务子程序

```
procedure match(t:token);
 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
- 从递归子程序法及FIRST与FOLLOW集看如何进一步用好当前的输入符号?

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。