第4章 语法分析

LI Wensheng, SCS, BUPT

知识点：预测分析方法、LL(1)分析程序
移进-归约分析方法、LR分析程序
SLR(1)、LR(1)、LALR(1)分析表

§4 语法分析

4.1 语法分析简介

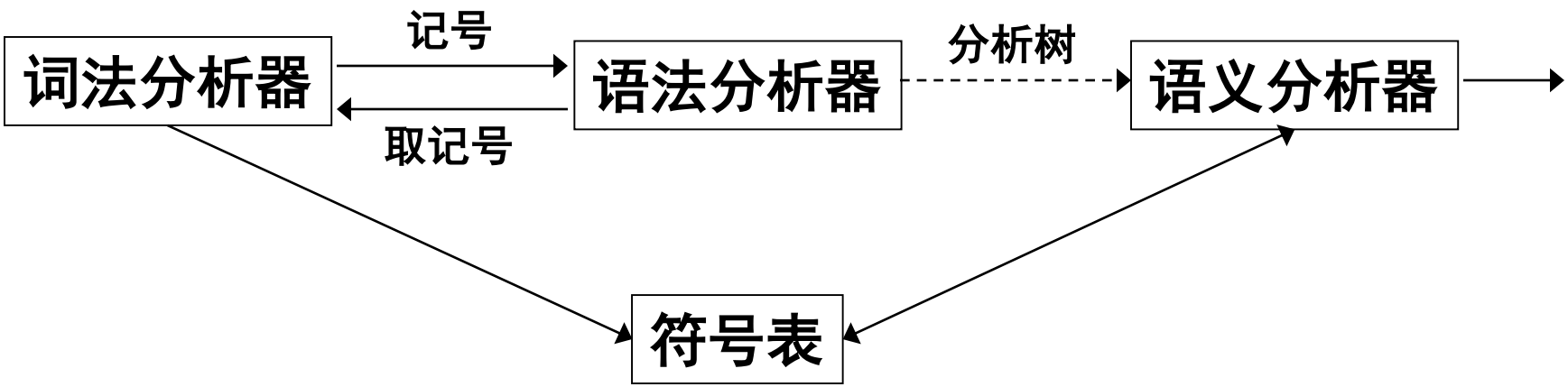
- 语法分析是编译程序的核心工作
- 由语法分析程序完成
- 工作依据：源语言的语法规则
- 文法的意义：

  语言的设计者：文法给出了语言的精确的语法规范说明

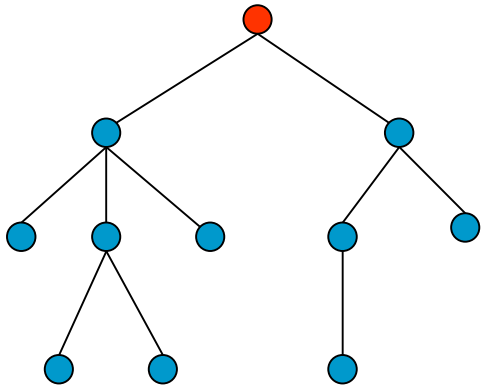
  编译程序作者：文法是进行正确编译的准则

  语言的使用者：文法是进行程序设计的依据

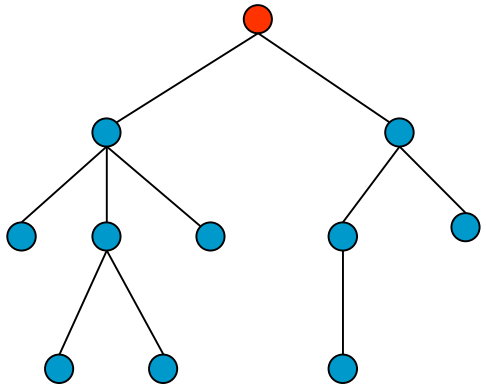
- 语法分析程序的任务
  - 从源程序记号序列中识别出各类语法成分
  - 进行语法检查
- 语法分析程序的地位：



- 语法分析程序的作用
  - 输入：记号流/记号序列
  - 工作依据：语法规则
  - 功能：将记号组合成语法成分、语法检查
  - 输出：分析树
  - 错误处理



- 常用的分析方法
  - 自顶向下的方法：
    从树根到叶子来建立分析树
  - 自底向上的方法：
    从树叶到树根来建立分析树



- 对输入符号串的扫描顺序：自左向右

语法错误的处理

- **必要性和难度：**

  程序员会出错，必须帮助他识别和定位错误

  难度：语言设计时没有考虑到错误处理

- **程序错误的种类**

  （1）词法错误：如标识符、关键字或算符的拼写错误

  （2）语法错误：如算术表达式的括号不匹配

  （3）语义错误：如算符作用与不相容的运算对象

  （4）逻辑错误：如无穷的递归调用

  语法错误占大多数

- **错误处理的基本目标**

  （1）清楚而准确地报告错误的出现

  （2）迅速地从每个错误中恢复过来，以便诊断后面的错误

  （3）错误处理不应该明显地影响对正确程序的处理效率

语法错误的处理（续）

- **怎样报告错误：**

  至少应该报告源程序的错误被检测到的位置

- **错误的恢复**

  必要性：每次尽可能多地发现错误

  困难：不妥当的恢复可能引起以假乱真的伪错误的大量涌现

- **错误恢复策略**

  有几种方法，但没有哪一种方法占明显的优势

  （1）紧急方式恢复：

  抛弃出错的语法结构，方法简单，不会陷入死循环

  （2）短语级恢复：

  局部纠错，防止发生死循环

  （3）出错产生式：

  扩充文法，增加产生错误的产生式

  （4）全局纠错

  时空代价太大，只有理论上的意义

符号使用的约定

- 终结符：
  - 次序靠前的小写字母，如：a,b,c等
  - 运算符，如：＋，－，＊，／等
  - 标点符号，如：括号，逗号等
  - 数字0，1，…，9
  - 黑体串，如：id、if、then
- 非终结符：
  - 次序靠前的大写字母，如：A, B, C等
  - 字母S，常代表文法的开始符号
  - 小写的斜体符号串，如：*expr, term, stmt*等
- 文法符号（终结符或非终结符）：
  - 字母表后面的大写字母，如：X, Y, Z等
- 终结符串：字母表后面的小写字母，如：u, v, …, z等
- 文法符号串：小写的希腊字母，如：α，β，γ等
- 通常用产生式集合代替四元组描述文法，第一个产生式左边符号是文法开始符号

4.2 自顶向下分析方法

4.2.1 递归下降分析

4.2.2 递归调用预测分析

4.2.3 非递归预测分析

4.2.1 递归下降分析

- 从文法的开始符号出发，进行推导，试图推出要分析的输入串的过程。
- 对给定的输入串，从对应于文法开始符号的根结点出发，自顶向下地为输入串建立一棵分析树。
- 试探过程，是反复使用不同产生式谋求匹配输入串的过程。
- 例：试分析输入串ω=abbcde是否为如下文法的一个句子

S→aAcBe

A→b | Ab

B→d （文法4.1）

示例：abbcde的递归下降分析

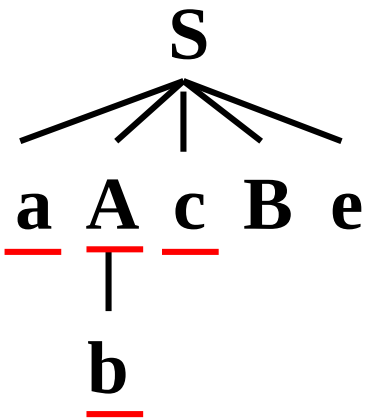
a b b c d e $

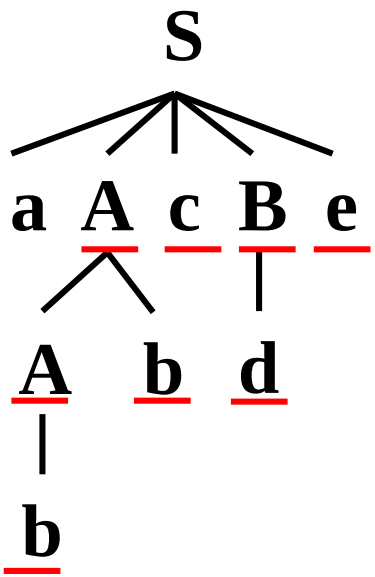
a b b c d e $

S→aAcBe

A→b | Ab

B→d





S⇒aAcBe⇒aAbcBe⇒abbcBe⇒abbcde

试图为输入符号串建立一个最左推导序列的过程。

- 为什么采用最左推导？
  - 因为对输入串的扫描是自左至右进行的，只有使用最左推导，才能保证按扫描的顺序匹配输入串。
- 递归下降分析方法的实现
  - 文法的每一个非终结符号对应一个递归过程，即可实现这种带回溯的递归下降分析方法。
  - 每个过程作为一个布尔过程，一旦发现它的某个产生式与输入串匹配，则用该产生式展开分析树，并返回true，否则分析树不变，返回false。
- 实践中存在的困难和缺点
  - 左递归的文法，可能导致分析过程陷入死循环。
  - 回溯
  - 工作的重复
  - 效率低、代价高：穷尽一切可能的试探法。

4.2.2 递归调用预测分析

- 一种确定的、不带回溯的递归下降分析方法

一、如何克服回溯？
二、对文法的要求
三、预测分析程序的构造
四、LL(1)文法

一、如何克服回溯？

- 能够根据所面临的输入符号准确地指派一个候选式去执行任务。
- 该选择的工作结果是确信无疑的。
  - 匹配成功则不是虚假的
  - 匹配不成功则任何其他候选式肯定也无法成功
- 例：

  $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_i | \ldots | \alpha_n$

  当前输入符号：a

  指派α_i去匹配输入符号串
- 怎样实现
  - 让每一个候选式的开始符号各不相同

二、对文法的要求

FIRST(α)={ }

- **定义4.1：文法符号串α的首符集**

$$FIRST(\alpha)=\{\ a \mid \alpha \overset{*}{\Rightarrow} a\beta, a\in V_T,\ \alpha、\beta\in(V_T\cup V_N)^*\ \}$$

如果 $\alpha \overset{*}{\Rightarrow} \varepsilon$，则规定$\varepsilon\in FIRST(\alpha)$。

1. 文法不含左递归　　$A \overset{*}{\Rightarrow} A\alpha$
2. 对于文法的每一个非终结符A，有产生式：

$A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$

则：$FIRST(\alpha_i)\cap FIRST(\alpha_j)=\phi$　　$(i\neq j)$

即：非终结符号A的所有候选式的首符集两两互不相交

示例:

- 有如下产生PASCAL类型子集的文法：
  type→*simple* | ↑id | array[*simple*] of *type*
  simple→integer | char | num dotdot num

  分析输入串：array [num dotdot num] of char

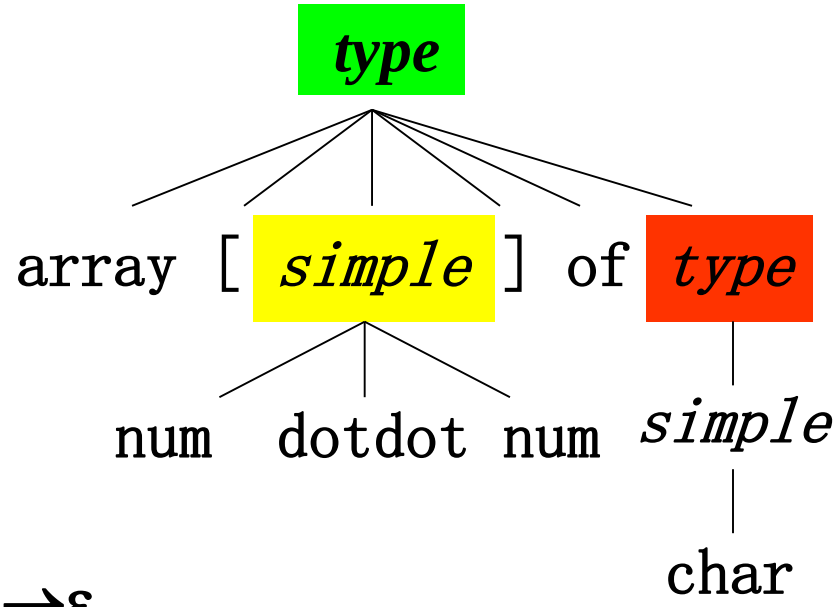
- *type*的三个候选式，有：
  FIRST(*simple*)={ integer, char, num }
  FIRST(↑id)={ ↑ }
  FIRST(array[simple] of type)={ array }

- *simple*的三个候选式，有：
  FIRST（integer）={ integer }
  FIRST(char)={ char }
  FIRST(num dotdot num)={ num }

示例（续）

输入：array [num dotdot num] of char

树：



产生式：A→ε

缺席匹配

当递归下降分析程序没有适当候选式时，可以用一个 ε-候选式，表示其它候选式可以缺席。

三、预测分析程序的构造

- 预测分析程序的转换图
- 转换图的工作过程
- 转换图的化简
- 预测分析程序的实现

预测分析程序的转换图

- 为预测分析程序建立转换图作为其实现蓝图
- 每一个非终结符号有一张有向图
- 边的标记可以是终结符号，也可以是非终结符号。
- 在一个非终结符号A上的转移意味着对相应A的过程的调用。
- 在一个终结符号a上的转移，意味着下一个输入符号若为a，则应做此转移。

从文法构造转换图

- 改写文法
  - 重写文法
  - 消除左递归
  - 提取左公因子
- 对每一个非终结符号A，做如下工作：
  - 创建一个初始状态和一个终结状态。
  - 对每一个产生式$A \rightarrow X_1X_2...X_n$创建一条从初态到终态的路径，有向边的标记依次为$X_1, X_2, ..., X_n$。

示例：

- 为如下文法构造预测分析程序转换图

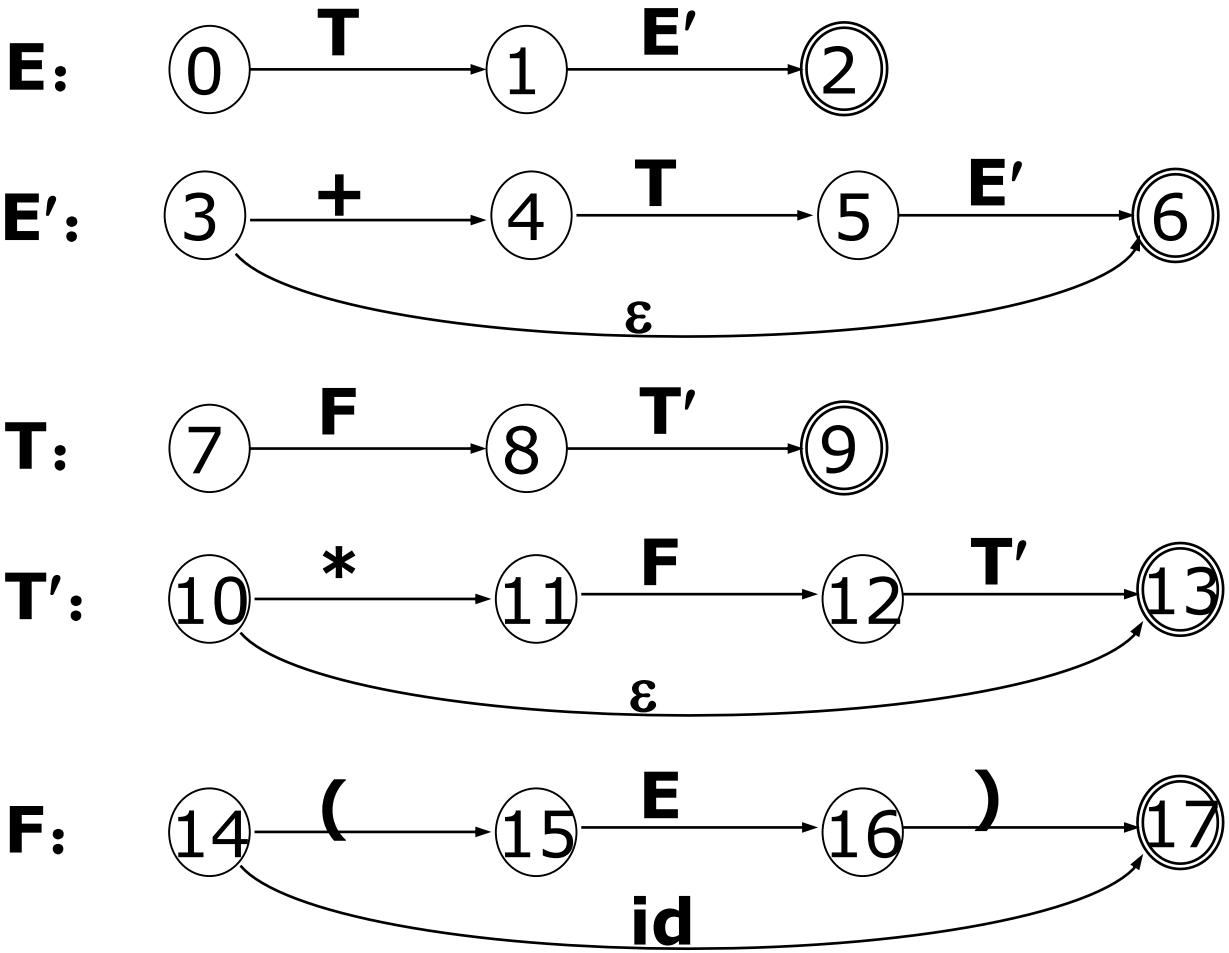
  E→E+T|T
  T→T*F|F
  F→(E)|id　　(文法4.3)

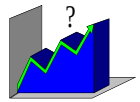
- 消除文法中存在的左递归，得到

  E→TE′
  E′→+TE′|ε
  T→FT′
  T′→*FT′|ε
  F→(E)|id　　(文法4.4)

- 为每个非终结符号构造转换图：

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



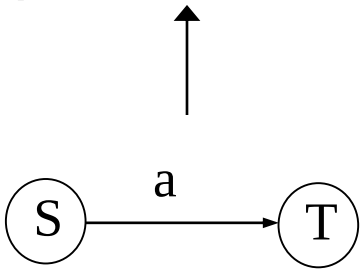


转换图的工作过程

- 从文法开始符号所对应的转换图的开始状态开始分析
- 经过若干动作之后，处于状态S

箭弧标志是终结符

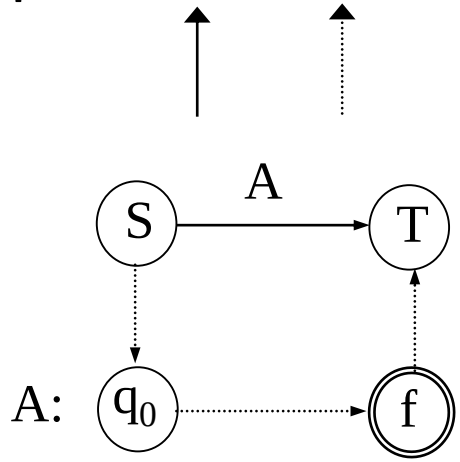
输入串：a …



把输入指针向右移一个位置并转到状态T

箭弧标志是非终结符

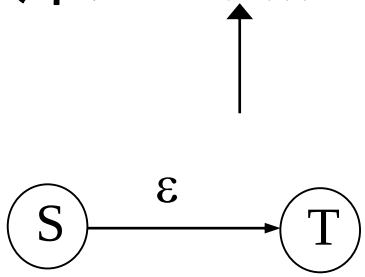
输入串：a ……b …



控制转移到A的开始状态，不移动输入指针

箭弧的标志符号是ε

输入串：a …



从状态S转移到状态T，不移动输入指针

转换图的确定性

- 转换图的确定性：

  对每个状态结点的所有射出弧，

  (1)不存在同名弧；

  (2)不存在ε-弧；

  (3)非终结符号为标志的弧是唯一射出弧

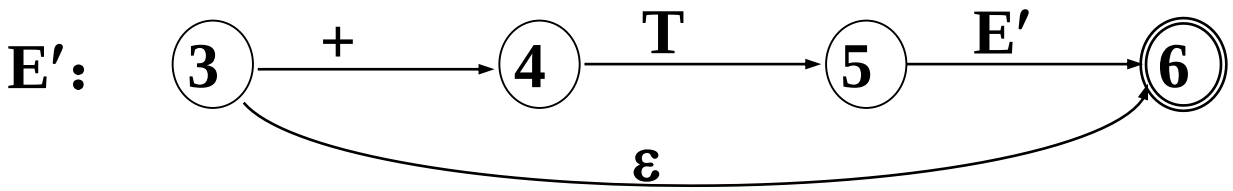
- 非确定性的解决：

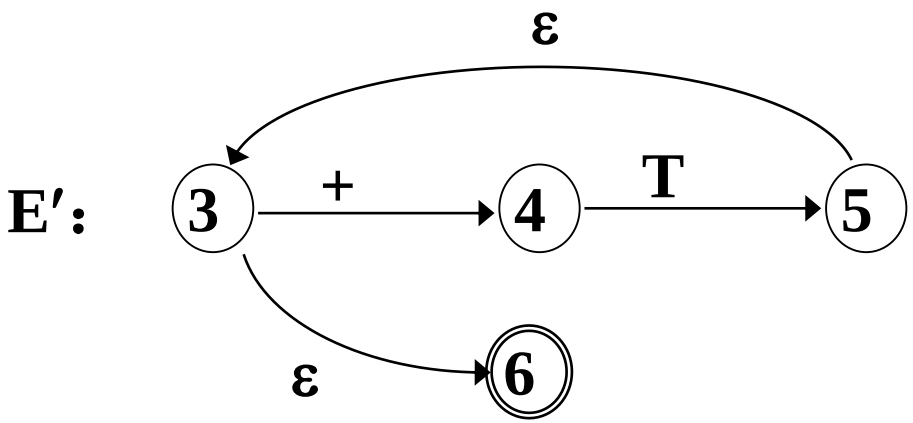
  (1)用特殊方法；

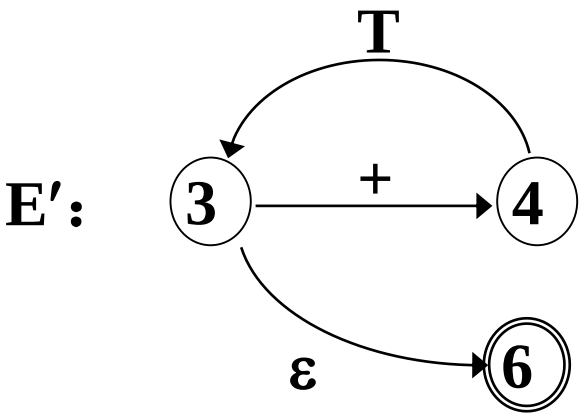
  (2)建立带回溯的系统

转换图的化简

- 用代入的方法进行化简

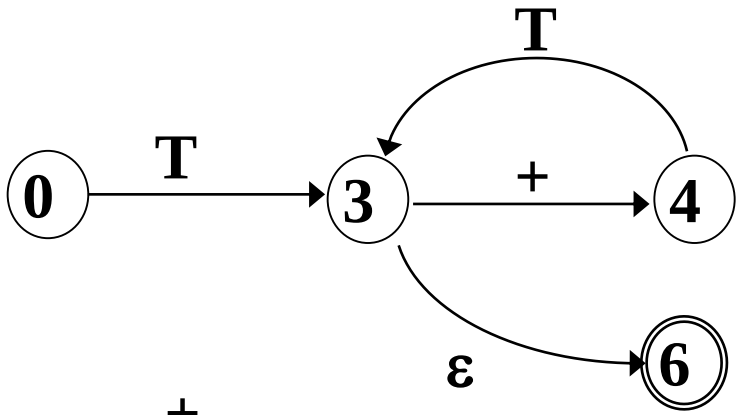




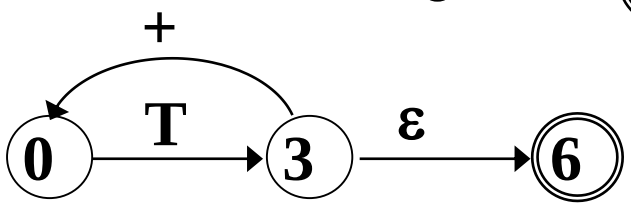


- 把E′的转换图代入E的转换图：

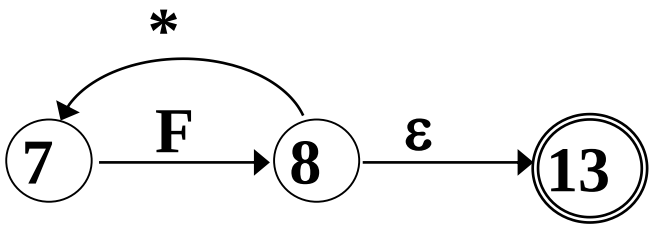
E：



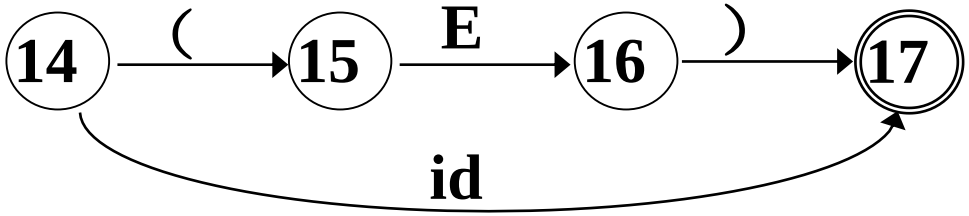
E：



T：



F：



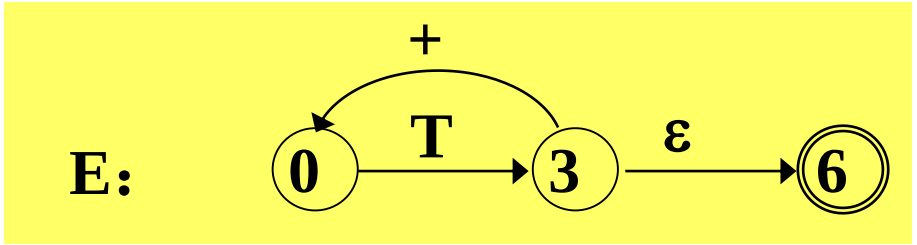
预测分析程序的实现

- 要求用来描述预测分析程序的语言允许递归调用
- 为每一个非终结符编写一个递归子程序
- 方法：从转换图的起始状态出发，根据状态的射出边
  - 如果是终结符，则程序为匹配该终结符，输入指针移向下一个记号
  - 如果是非终结符，则程序为调用该非终结符所对应的子程序，输入指针不移动
  - 如果有两个以上的射出边，则使用分枝语句在程序中产生相应的分支部分

预测分析程序的实现(续1)

- E的过程：

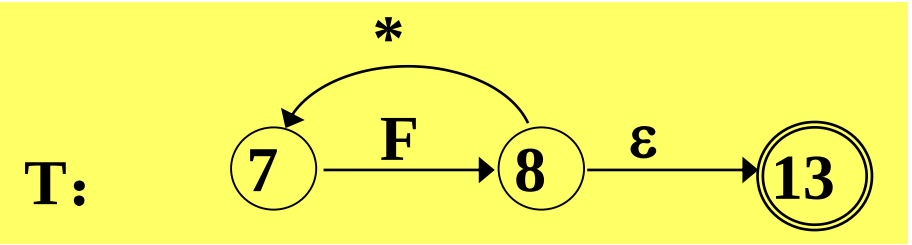
```
void procE(void)
{
  procT();
  if (char=='+') {
    forward pointer;
    procE();
  }
}
```



预测分析程序的实现(续2)

- T的过程：

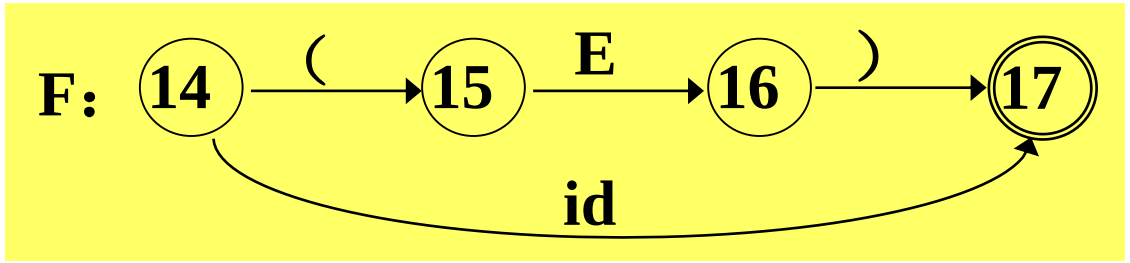
```
void procT(void)
{
  procF();
  if (char=='*') {
     forward pointer;
     procT();
  }
}
```



预测分析程序的实现（续3）

- F的过程：

```
void procF(void)
{
  if (char==′(′) {
     forward pointer;
     procE();
     if (char==′)′) {
        forward pointer;
     };
     else  error();
  };
  else  if (char==′id′) {
             forward pointer;
          else  error();
}
```



四、LL(1)文法

■ FIRST集合及其构造

定义：对任何文法符号串$\alpha \in (V_T \cup V_N)^*$，
FIRST(α)是α可以推导出的开头终结符号集合

描述为：

$$FIRST(\alpha)=\{\ a \mid \alpha \overset{*}{\Rightarrow} a\cdots,\ a\in V_T\ \}$$

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

构造每个文法符号$X \in V_T \cup V_N$的FIRST(X)

- 若$X \in V_T$，则FIRST(X)={X}；
- 若$X \in V_N$，且有产生式$X \to a\ldots$，其中 $a \in V_T$，则把a加入到FIRST(X)中；
- 若$X \to \varepsilon$也是产生式，则 ε 也加入到FIRST(X)中。
- 若$X \to Y\ldots$是产生式，且$Y \in V_N$，则把FIRST(Y)中的所有非ε元素加入到FIRST(X)中；

若$X \to Y_1Y_2\ldots Y_k$是产生式，如果对某个i，
$FIRST(Y_1)$、$FIRST(Y_2)$、…、$FIRST(Y_{i-1})$都含有ε，
即$Y_1Y_2\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$，
则把$FIRST(Y_i)$中的所有非ε元素加入到FIRST(X)中；

若所有$FIRST(Y_i)$均含有ε，其中i=1、2、…、k，则把 ε 加入到FIRST(X)中。

构造文法G的任何符号串$\alpha=X_1X_2...X_n$的FIRST(α)

把FIRST(X_1)的所有非ε-元素加入到FIRST(α)中；

若$\varepsilon\in$FIRST(X_1)，
　把FIRST(X_2)的所有非ε-元素加入到FIRST(α)中；

若$\varepsilon\in$FIRST(X_1)，且$\varepsilon\in$FIRST(X_2)，
　把FIRST(X_3)的所有非ε-元素加入到FIRST(α)中；
以次类推。

最后，若对于所有的i，FIRST(X_i)都含有ε，
　把ε加入到FIRST(α)中

FOLLOW集合及其构造

- **FOLLOW集合**

定义：假定S是文法G的开始符号，对于G的任何非终结符号A，集合FOLLOW(A)是在所有句型中，紧跟A之后出现的终结符号或$组成的集合。

描述为：

$$FOLLOW(A)=\{\ a \mid S \overset{*}{\Rightarrow} \cdots Aa \cdots,\ a\in V_T\ \}$$

特别地，若$S \overset{*}{\Rightarrow} \cdots A$，则规定$\$\in FOLLOW(A)$。

因此：$\$\in FOLLOW(S)$

其中：$为输入符号串的右尾标志

构造每个非终结符号A的集合FOLLOW(A)

- 对文法开始符号S，置$于FOLLOW(S)中，$为输入符号串的右尾标志。
- 若A→αBβ是产生式，则把FIRST(β)中的所有非ε元素加入到FOLLOW(B)中。
- 若A→αB是产生式，或A→αBβ是产生式并且$\beta \overset{*}{\Rightarrow} \varepsilon$，则把FOLLOW(A)中的所有元素加入到FOLLOW(B)中。
- 重复此过程，直到所有集合不再变化为止。

示例：

- 构造文法4.4中每个非终结符号的FIRST集合和FOLLOW集合

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

	FIRST	FOLLOW
E	(, id	$,)
E’	+, ε	$,)
T	(, id	$,), +
T’	*, ε	$,), +
F	(, id	$,), +, *

LL(1)文法定义

一个文法是LL(1)文法，当且仅当它的每一个产生式 $A\to\alpha|\beta$，满足：

- $FIRST(\alpha)\cap FIRST(\beta)=\phi$
- 若β推导出ε，则$FIRST(\alpha)\cap FOLLOW(A)=\phi$

- **判断文法4.4是否是LL(1)文法**

$FIRST(+TE')\cap FOLLOW(E')$

$=\{+\}\cap\{),\$\}=\phi$

$FRIST(*FT')\cap FOLLOW(T')$

$=\{*\}\cap\{+,),\$\}=\phi$

$FIRST((E))\cap FIRST(id)=\{(\}\cap\{id\}=\phi$

因此该文法是LL(1)文法

$E\to TE'$
$E'\to +TE'|\varepsilon$
$T\to FT'$
$T'\to *FT'|\varepsilon$
$F\to (E)|id$

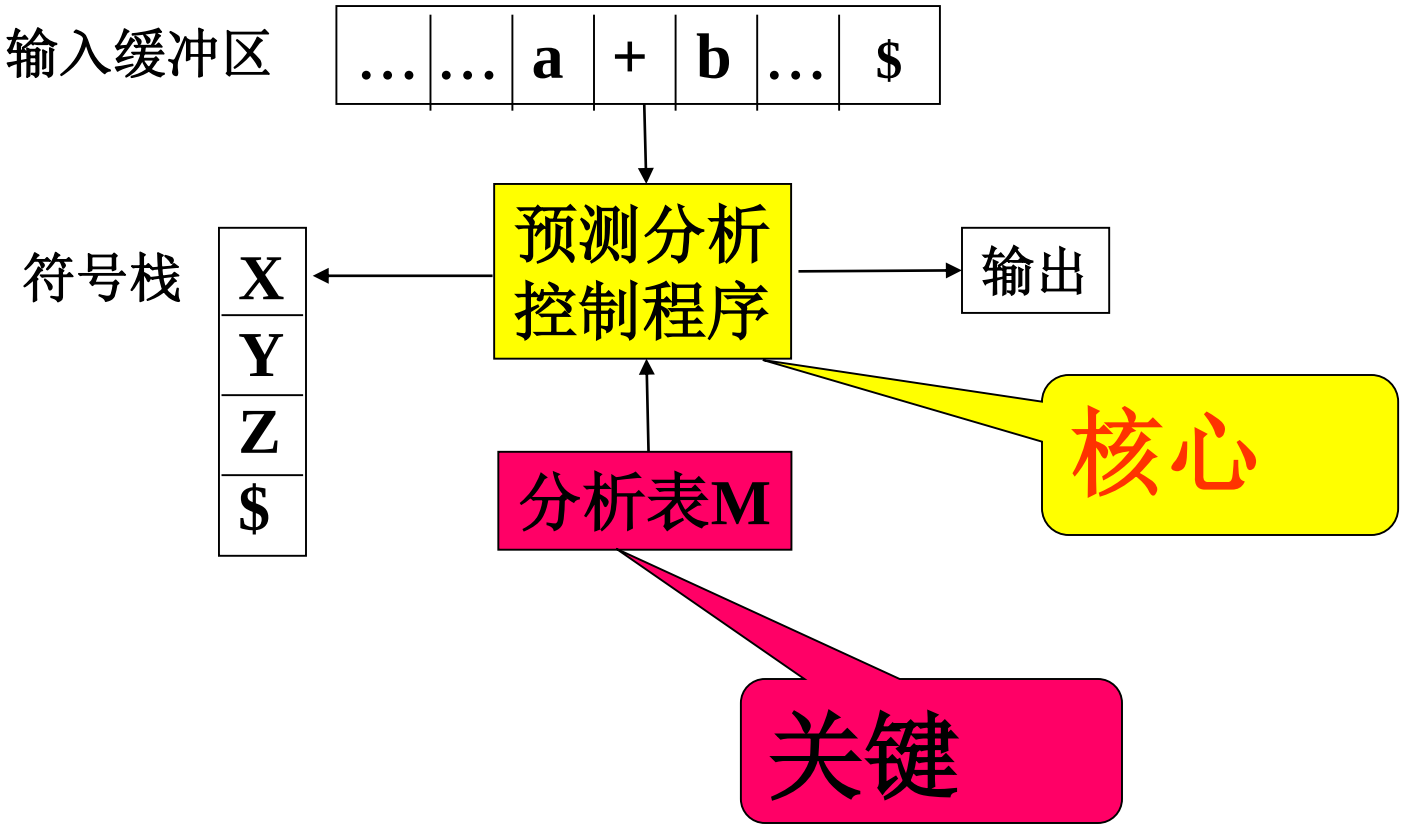
4.2.3 非递归预测分析

- 使用一张分析表和一个栈联合控制，实现对输入符号串的自顶向下分析。

- 预测分析程序的模型及工作过程
- 预测分析表的构造
- LL(1)文法
- 预测分析方法中的错误处理示例

预测分析程序的模型及工作过程

- 关键：决定哪个产生式运用于非终结符
- 模型：



每部分的作用

- **输入缓冲区：**

  存放被分析的输入符号串，串后随右尾标志符$。

… … $

- **符号栈：**

  存放一系列文法符号，$存于栈底。分析开始时，先将$入栈，以标识栈底，然后再将文法的开始符号入栈。

$S

- **分析表：**

  二维数组M[A,a]，$A \in V_N$，$a \in V_T \cup \{\$\}$。

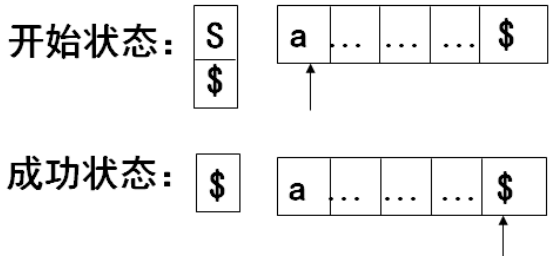
  根据给定的A和a，在分析表M中找到将被调用的产生式。

- **输出流：**

  分析过程中不断产生的产生式序列。

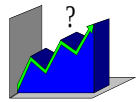
	… a …
A	A→a…

预测分析控制程序



- 根据栈顶符号X和当前输入符号a，决定分析动作，有4种可能：
  - (1) X=a=$，宣告分析成功，停止分析；
  - (2) X=a≠$，从栈顶弹出X，输入指针前移一个位置；
  - (3) $X \in V_T$，但X≠a，报告发现错误，调用错误处理程序，以报告错误及进行错误恢复；
  - (4) 若$X \in V_N$，访问分析表M[X, a]
    - $M[X, a]=X \rightarrow Y_1Y_2...Y_n$
      先将X从栈顶弹出，然后把产生式的右部符号串按反序推入栈中（即按Y_n、…、Y_{n-1}、Y_2、Y_1的顺序）；
    - $M[X, a]=X \rightarrow \varepsilon$　　从栈顶弹出X；
    - M[X, a]=error　　调用出错处理程序

算法4.1　非递归预测分析方法



输入：输入符号串ω，文法G的一张预测分析表M。

输出：若ω在L(G)中，则输出ω的最左推导，否则报告错误。

方法：分析开始时，$在栈底，文法开始符号S在栈顶，ω$在输入缓冲区中

```
    置ip指向 ω$ 的第一个符号;
    do {
        令X是栈顶符号，a是ip所指向的符号;
        if (X是终结符号或$) {
            if (X==a) {
                从栈顶弹出X; ip前移一个位置;
            };
            else error();
        else /* X是非终结符号 */
        if (M[X,a]=X→Y1Y2...Yk ) {
            从栈顶弹出X;
            把Yk、Yk-1、…、Y2、Y1压入栈，Y1在栈顶;
            输出产生式X→Y1Y2...Yk;
        };
        else error();
    }while(X!=$) /* 栈不空，继续 */
```

示例：文法4.4的预测分析表M，试分析输入串 id+id*id。

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E’		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T’		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

栈	输入	输出	左句型
$E	id+id*id$		E
$E′T	id+id*id$	E→TE′	TE′
$E′T′F	id+id*id$	T→FT′	FT′E′
$E′T′id	id+id*id$	F→id	idT′E′
$E′T′	+id*id$		idT′E′
$E′	+id*id$	T′→ε	idE′

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E’		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T’		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

$E′T+	+id*id$	E′→+TE′	id+TE′
$E′T	id*id$		id+TE′
$E′T′F	id*id$	T→FT′	id+FT′E′
$E′T′id	id*id$	F→id	id+idT′E′
$E′T′	*id$		id+idT′E′
$E′T′F*	*id$	T′→*FT′	id+id*FT′E′
$E′T′F	id$		id+id*FT′E′
$E′T′id	id$	F→id	id+id*idT′E′
$E′T′	$		id+id*idT′E′
$E′	$	T′→ε	id+id*idE′
$	$	E′→ε	id+id*id

预测分析表的构造

- **主要思想：**

  如果A→α是产生式，当A呈现于栈顶，并且当前输入符号a∈FIRST(α)时，α应被选作A的唯一代表，即用α展开A，所以M[A,a]中应放入产生式A→α。

  当α=ε或$\alpha \overset{*}{\Rightarrow} \varepsilon$时，如果当前输入符号b(包括$)∈FOLLOW(A)，则认为A自动得到匹配，因此，应把产生式A→α放入M[A,b]中。

算法4.2 预测分析表的构造方法

输入：文法G

输出：文法G的预测分析表M

方法：

```
for (文法G的每个产生式A→α) {
    for (每个终结符号a∈FIRST(α))
        把A→α放入M[A,a]中;
    if (ε∈FIRST(α))
        for (任何b∈FOLLOW(A))
            把A→α放入M[A,b]中;
 };
for (所有无定义的M[A,a]) 标上错误标志。
```

示例：为文法4.4构造预测分析表

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | id

	FIRST	FOLLOW
E	(, id	$,)
E’	+, ε	$,)
T	(, id	$,), +
T’	*, ε	$,), +
F	(, id	$,), +, *

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

LL(1)文法

例：考虑如下映射程序设计语言中if语句的文法

S→iEtSS′ | a

S′→eS | ε

E→b　　（文法4.5）

构造各非终结符号的FIRST和FOLLOW集合：

	S	S′	E
FIRST	i, a	e, ε	b
FOLLOW	$, e	$, e	t

应用算法4.2，构造该文法的分析表：

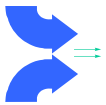
	a	b	e	i	t	$
S	S→a			S→iEtSS′		
S′			S′→eS S′→ε			S′→ε
E		E→b				

LL(1)文法（续）

- 如果一个文法的预测分析表M不含多重定义的表项，则称该文法为LL(1)文法。
- LL(1)的含义：
  - 第一个L表示从左至右扫描输入符号串
  - 第二个L表示生成输入串的一个最左推导
  - 1表示在决定分析程序的每步动作时，向前看一个符号

LL(1)文法的判断

- 根据文法产生式判断：
  一个文法是LL(1)文法，当且仅当它的每一个产生式$A\rightarrow\alpha|\beta$，满足：
  - $FIRST(\alpha)\cap FIRST(\beta)=\phi$　并且
  - 若β推导出ε，则$FIRST(\alpha)\cap FOLLOW(A)=\phi$
- 根据分析表判断：
  如果利用算法4.2构造出的分析表中不含多重定义的表项，则文法是LL(1)文法。
- 文法4.5不是LL(1)文法
  - 因为FIRST(es)∩FOLLOW(S′)={e}
  - M[S′,e]中有两个产生式



预测分析方法中的错误处理示例

- 分析过程中，有两种情况可以发现源程序中的语法错误：

  (1) $X \in V_T$，但$X \neq a$；

  (2) $X \in V_N$，但M[X,a]为空。

- 错误处理方法：
  - 第(1)种情况，弹出栈顶的终结符号；
  - 第(2)种情况，跳过剩余输入符号串中的若干个符号，直到可以继续进行分析为止。

带有同步化信息的分析表

- 带有同步化信息的分析表的构造
  - 对于$A \in V_N$，$b \in FOLLOW(A)$，若M[A,b]为空，则加入“synch”
- 带有同步化信息的分析表的使用
  - 若M[A,b]为空，则跳过a；
  - 若M[A,b]为synch，则弹出A。

示例：

- 构造文法4.4的带有同步化信息的分析表

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | id

	FIRST	FOLLOW
E	(, id	$,)
E′	+, ε	$,)
T	(, id	$,), +
T′	*, ε	$,), +
F	(, id	$,), +, *

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

栈	输入	输出
$E	*id*+id$	出错，M[E,*]=空白，跳过*
$E	id*+id$	E→TE′
$E′T	id*+id$	T→FT′
$E′T′F	id*+id$	F→id
$E′T′id	id*+id$	
$E′T′	*+id$	T′→*FT′
$E′T′F*	*+id$	
$E′T′F	+id$	出错，M[F,+]=synch，弹出F
$E′T′	+id$	T′→ε
$E′	+id$	E′→+TE′
$E′T+	+id$	
$E′T	id$	T→FT′
$E′T′F	id$	F→id
$E′T′id	id$	
$E′T′	$	T′→ε
$E′	$	E′→ε
$	$	

	id	+	*	(	)	$
E	E→TE′			E→TE′	synch	synch
E’		E′→+TE′			E′→ε	E′→ε
T	T→FT′	synch		T→FT′	synch	synch
T’		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id	synch	synch	F→(E)	synch	synch

- 带有同步化信息的分析表的使用
  - 若M[A,b]为空，则跳过a；
  - 若M[A,b]为synch，则弹出A。

– $X \in V_T$，但$X \neq a$ 弹出栈顶的终结符号；

4.3 自底向上分析方法

- 对输入串的扫描：自左向右
- 分析树的构造：自底向上
- 分析过程：
  - 从输入符号串开始分析
  - 查找当前句型的“可归约串”
  - 使用规则，把它归约成相应的非终结符号
  - 重复
- 关键：找出“可归约串”
- 常用方法：
  - 优先分析方法
  - LR分析方法

优先分析法

- 分为：简单优先分析法和算符优先分析法。
- 简单优先分析法
  - 规范归约。
  - 按照文法符号（包括终结符号和非终结符号）之间的优先关系确定当前句型的“可归约串”。
  - 分析效率低，且只适用于简单优先文法。
  - 简单优先文法，满足以下两个条件：

    （1）任何两个文法符号之间最多存在一种优先关系。

    （2）不存在具有相同右部的产生式。

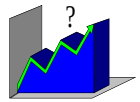
优先分析法（续）

- 算符优先分析法
  - 只考虑终结符号之间的优先关系。
  - 分析速度快，不是规范归约，只适用于算符优先文法。
  - 算符文法：没有形如$A\rightarrow\cdots BC\cdots$的产生式，其中$B、C\in V_N$
  - 算符优先文法：
    - 算符文法、且不含有ε-产生式；
    - 任何两个构成序对的终结符号之间最多有 $>$、$=$ 和 $<$ 三种优先关系中的一种成立。
  - “可归约串”是句型的“最左素短语”。
    - 素短语：句型的一个短语，至少含有一个终结符号，并且除它自身之外不再含有其他更小的素短语。
    - 最左素短语：处于句型最左边的那个素短语。

“移进-归约”分析方法

- 符号栈：存放文法符号
- 分析过程：
  （1）把输入符号一个个地移进栈中。
  （2）当栈顶的符号串形成某个产生式的一个候选式时，在一定条件下，把该符号串替换(即归约)为该产生式的左部符号。
  （3）重复(2)，直到栈顶符号串不再是“可归约串”为止。
  （4）重复(1)~(3)，直到最终归约出文法开始符号S。

规范归约



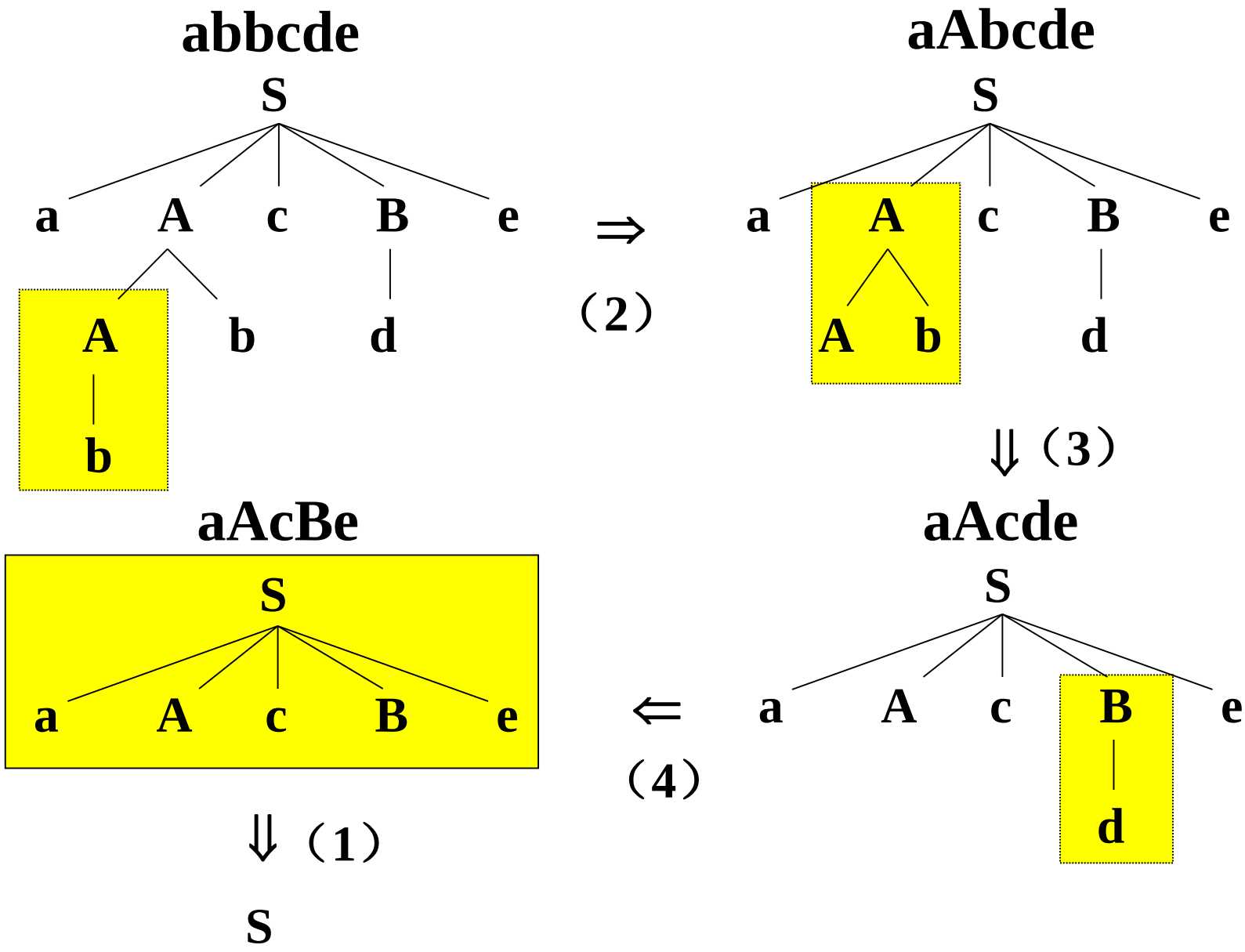
例：分析符号串abbcde是否为如下文法的句子。

1) S→aAcBe
2) A→b
3) A→Ab
4) B→d　　（文法4.1）

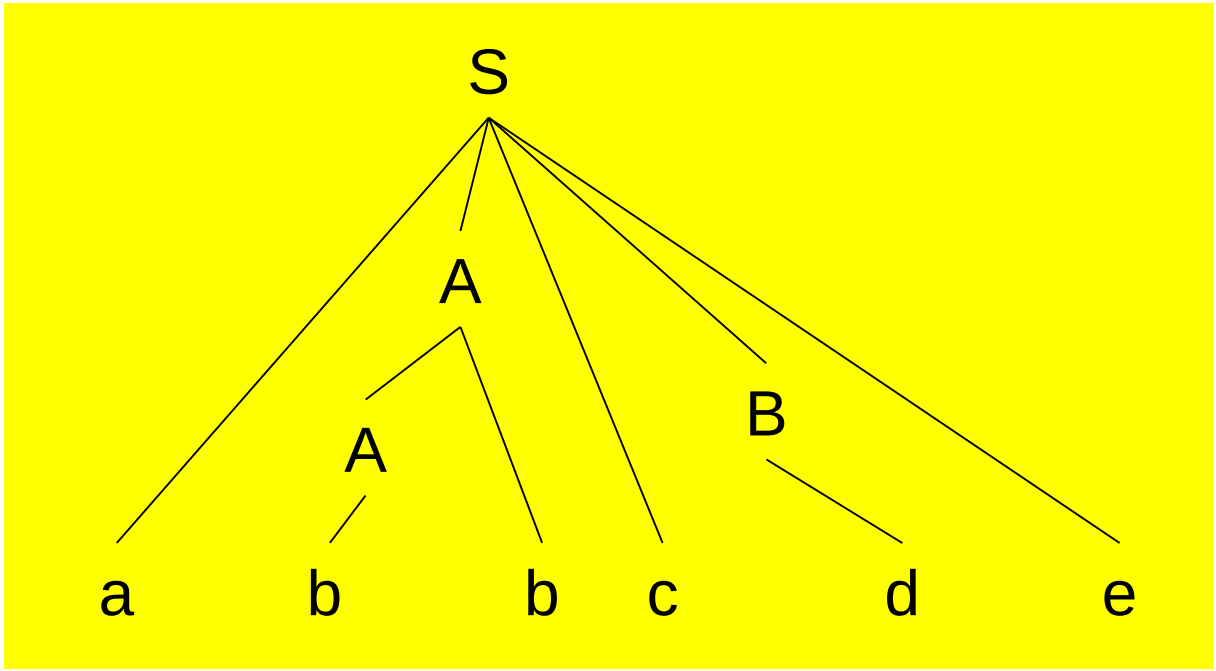
- 最右推导

$$S \underset{rm}{\overset{1)}{\Rightarrow}} aAcBe \underset{rm}{\overset{4)}{\Rightarrow}} aAcde \underset{rm}{\overset{3)}{\Rightarrow}} aAbcde \underset{rm}{\overset{2)}{\Rightarrow}} abbcde$$

最右推导的逆过程(剪句柄)



最右推导的逆过程



a b b c d e

a A b c d e

a A c d e

a A c B e

S

A→b

A→Ab

B→d

S→aAcBe

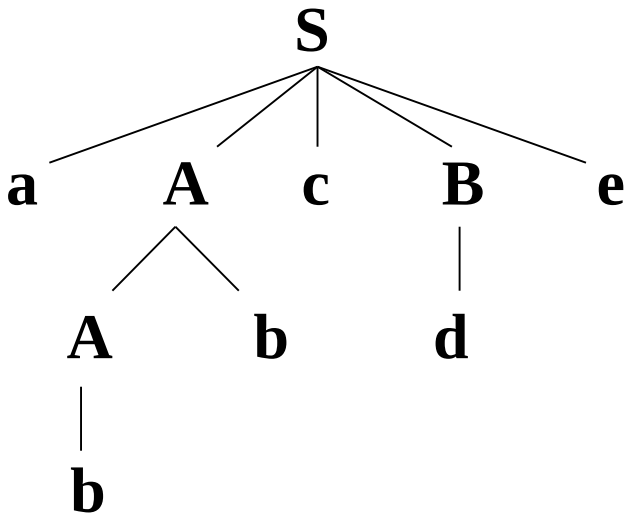
最右推导：

S⇒aAcBe

⇒aAcde

⇒aAbcde

⇒abbcde



规范归约的定义

定义：假定α是文法G的一个句子，我们称右句型序列 α_n，α_{n-1}，…，α_1，α_0 是α的一个规范归约，如果序列满足：

(1) $\alpha_n=\alpha$，$\alpha_0=S$

(2) 对任何i($0<i\leq n$)，α_{i-1}是经过把α_i的句柄替换为相应产生式的左部符号而得到的。

- 规范归约是关于α的一个最右推导的逆过程，因此规范归约也称为最左归约。
- abbcde的一个规范归约是如下的右句型序列：abbcde，aAbcde，aAcde，aAcBe，S。
- 规范规约的中心问题：如何寻找或确定一个右句型的句柄

句柄的最左性

- 规范句型：最右推导得到的句型
- 规范句型的特点：句柄之后没有非终结符号
- 利用句柄的最左性：与符号栈的栈顶相关
- 不同的最右推导，其逆过程也是不同

例：考虑文法 E→E+E|E*E|(E)|id的句子id+id*id

E⇒E+E⇒E+E*E⇒E+E*id⇒E+id*id⇒id+id*id

E⇒E*E⇒E*id⇒E+E*id⇒E+id*id⇒id+id*id

句柄：E+E
产生式：E→E+E

句柄：id
产生式：E→id

“移进-归约”方法的实现

- 必须解决两个问题：
  1. 确定右句型中将要归约的子串
  2. 如果这个子串是多个产生式的右部，如何确定选择哪个产生式
- 使用一个寄存文法符号的栈和一个存放输入符号串的缓冲区
  - 分析开始时，先将符号$入栈，以示栈底；
  - 将$置入输入符号串之后，以示符号串的结束。

栈	输入	栈	输入
$	ω$	$S	$
开始格局		结束格局	

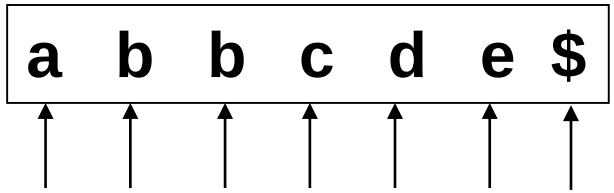
- 为什么可以使用栈：

  由于句柄的最左性决定了任何可归约串的出现必须在栈顶而不是在栈的里面

例：对文法4.6的句子abbcde的规范归约过程

	栈	输入	分析动作
1)	$	abbcde$	shift
2)	$a	bbcde$	shift
3)	$ab	bcde$	reduce by A→b
4)	$aA	bcde$	shift
5)	$aAb	cde$	reduce by A→Ab
6)	$aA	cde$	shift
7)	$aAc	de$	shift
8)	$aAcd	e$	reduce by B→d
9)	$aAcB	e$	shift
10)	$aAcBe	$	reduce by S→aAcBe
11)	$S	$	accept

句子abbcde的规范归约过程



栈 | $ S |

右句型

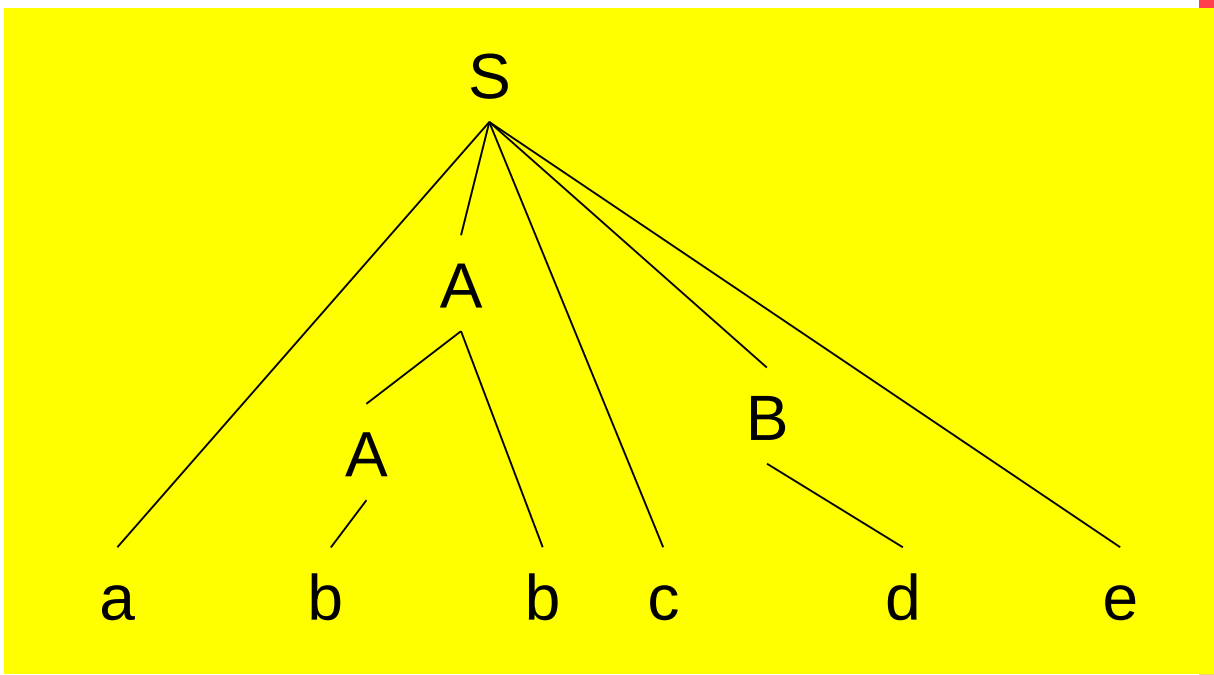
a b b c d e

a A b c d e

a A c d e

a A c B e

S



“移进-归约”分析方法的分析动作

移进：把下一个输入符号移进到栈顶。

归约：用适当的归约符号去替换这个串。

接受：宣布分析成功，停止分析。

错误处理：调用错误处理程序进行诊断和恢复。

分析过程中的动作冲突：

“移进-归约”冲突

“归约-归约”冲突

4.4 LR分析方法

LR分析技术概述

4.4.1 LR分析程序的模型及工作过程

4.4.2 SLR(1)分析表的构造

4.4.3 LR(1)分析表的构造

4.4.4 LALR(1)分析表的构造

4.4.5 LR分析方法对二义文法的应用

4.4.6 LR分析的错误处理与恢复

LR分析技术概述

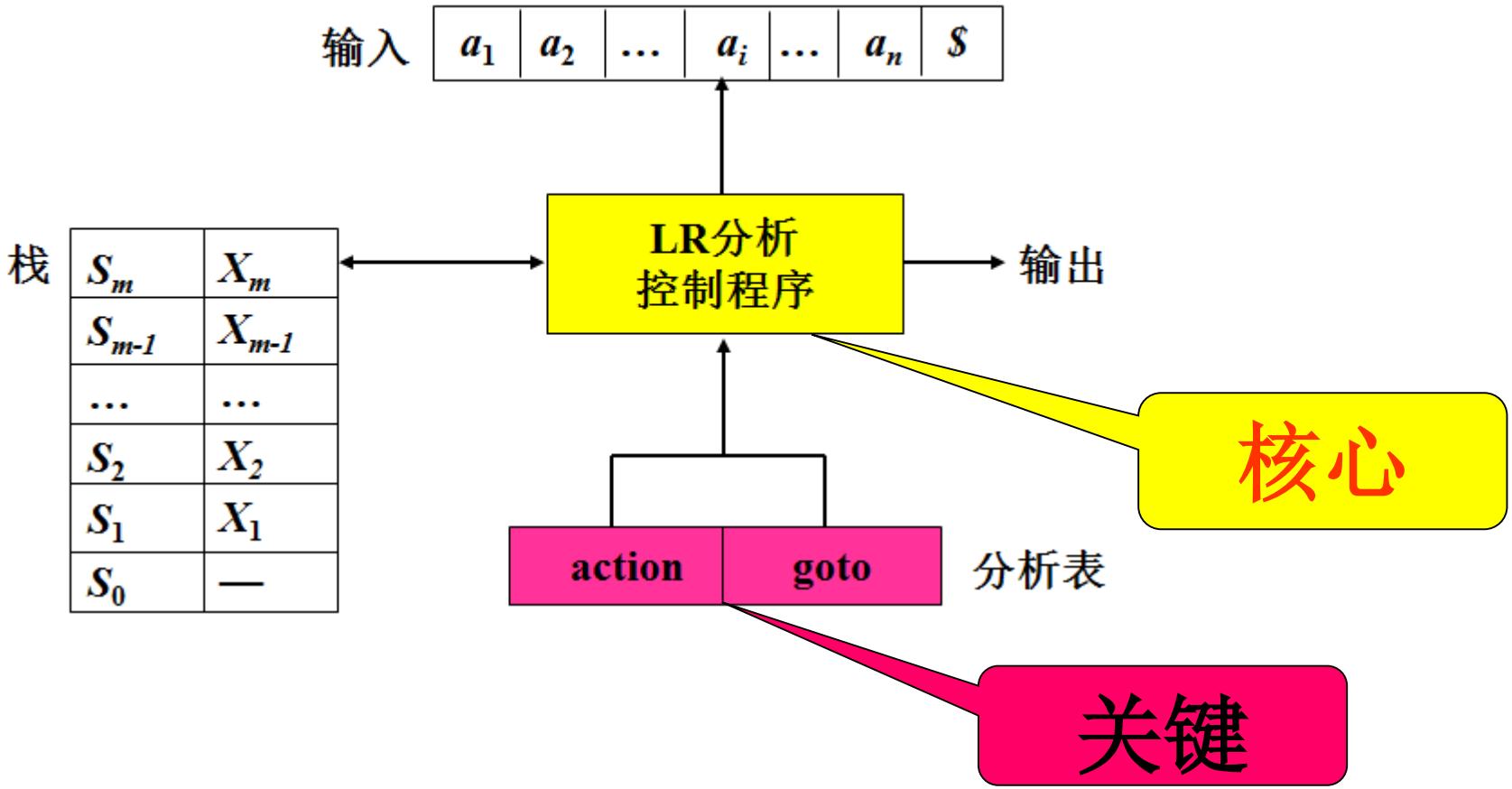
- LR(k)的含义：
  - L 表示自左至右扫描输入符号串
  - R 表示为输入符号串构造一个最右推导的逆过程
  - k 表示为作出分析决定而向前看的输入符号的个数。
- LR分析方法的基本思想
  - “历史信息”：记住已经移进和归约出的整个符号串；
  - “预测信息”：根据所用的产生式推测未来可能遇到的输入符号；
  - 根据“历史信息”和“预测信息”，以及“现实”的输入符号，确定栈顶的符号串是否构成相对于某一产生式的句柄。

LR分析技术概述(续)

- LR分析技术是一种比较完备的技术
  - 可以分析所有能用上下文无关文法书写的程序设计语言的结构；
  - 最一般的无回溯的“移进-归约“方法；
  - 能分析的文法类是预测分析方法能分析的文法类的真超集；
  - 分析过程中，能及时发现错误，快到自左至右扫描输入的最大可能。
- LR分析方法的不足之处
  - 手工编写LR分析程序的工作量太大
  - 需要专门的工具，即LR分析程序生成器（如YACC）

4.4.1、LR分析程序的模型及工作过程

- LR分析程序的模型



- 分析程序：LR分析器的核心，对所有的LR分析器都一样，不同语言的LR分析器仅分析表不同

- 栈：存放“状态”和移进、归约的文法符号

  $S_0X_1S_1\ldots X_mS_m$

  其中，S_i表示状态，X_i表示文法符号

分析表

- LR分析器工作的依据，由两部分组成
  - 状态转换表goto[S_m，X]：栈顶状态S_m，对于文法符号X(X可能是移进或归约时的文法符号）时，下一个状态是什么
  - 动作表action[S_m，a_i]：栈顶状态S_m面临输入符号a_i时，分析程序应采取的动作，可能有四个值

    移进：把当前输入符号a_i及由S_m和a_i所决定的下一个状态S=goto[S_m，a_i]推进栈，向前扫描指针前移。

    归约：用某产生式A→β进行归约，若β的长度为r，归约的动作从栈顶起向下弹出r项，使S_{m-r}成为栈顶状态，然后把文法符号A及状态S=goto[S_{m-r}，A]推进栈。

    接受：宣布分析成功，停止分析。

    出错：调用出错处理程序，进行错误恢复。
- 构造LR分析表的三种技术
  - 简单的LR方法（SLR)：易于实现，功能较弱
  - 规范的LR方法（LR(1))：功能最强，代价最大
  - 向前看的LR方法（LALR)：功能和代价介于两者之间，能分析大多数程序设计语言的结构，并且能比较有效地实现

LR分析控制程序

- 可以看作栈中符号序列和剩余输入符号串所构成的二元式的变化过程。
- LR分析控制程序的工作过程：
  - 分析开始时，初始的二元式为：$(S_0, a_1a_2...a_n\$)$
  - 分析过程中每步的结果，均可表示为如下的二元式：$(S_0X_1S_1...X_mS_m, a_ia_{i+1}...a_n\$)$，这里$X_1X_2...X_ma_ia_{i+1}...a_n$是一个右句型，$X_1X_2...X_m$是它的一个活前缀
  - 若action[S_m，a_i]=shift S，且S=goto[S_m，a_i]，
    则二元式变为：$(S_0X_1S_1X_2...X_mS_ma_iS, a_{i+1}...a_n\$)$
  - 若action[S_m，a_i]=reduce by $A\rightarrow\beta$，
    则二元式变为：$(S_0X_1S_1X_2...X_{m-r}S_{m-r}AS, a_ia_{i+1}...a_n\$)$
    其中：$S=goto[S_{m-r}, A]$，r为β的长度，且$\beta= X_{m-r+1}X_{m-r+2}...X_m$
  - 若action[S_m，a_i]=accept（接受）
    分析成功，二元式变化过程终止。
  - 若action[S_m，a_i]=error（出错）
    发现错误，调用错误处理程序。
- 所有LR分析器都按这种形式动作，唯一的区别是分析表的内容不一样（表驱动程序）

“活前缀”的概念

定义：一个规范句型的一个前缀，如果不含句柄之后的任何符号，则称它为该句型的一个活前缀。

例：右句型aAbcde的句柄是Ab

该句型的活前缀有ε、a、aA、aAb；

句型aAcde的句柄是d

它的活前缀有：ε、a、aA、aAc、aAcd。

之所以称它们为活前缀，是因为在其右边增加某些终结符号之后，就可以使之成为一个规范句型。

分析过程中 ($S_0X_1S_1X_2\ldots X_mS_m$，$a_ia_{i+1}\ldots a_n\$$)

$X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$是一个右句型，

$X_1X_2\ldots X_m$是它的一个活前缀。

算法4.3 LR分析控制程序

输入：文法G的一张分析表和一个输入符号串ω

输出：若ω∈L(G)，得到ω的自底向上的分析，否则报错

方法：开始时，初始状态S_0在栈顶，ω$在输入缓冲区中。

```
置ip指向ω$的第一个符号；
do {
    令S是栈顶状态，a是ip所指向的符号
    if (action[S，a]==shift S′) {
        把a和S′分别压入符号栈和状态栈;
        推进ip，使它指向下一个输入符号;
    };
    else if (action[S，a]==reduce by A→β) {
            从栈顶弹出|β|个符号；（令S′是现在的栈顶状态）
            把A和goto[S′，A]分别压入符号栈和状态栈;
            输出产生式A→β;
        };
        else if (action[S，a]==accept) return;
            else  error();
} while(1).
```

例：对算术表达式文法G

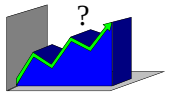
(1) E →E+T　　(2) E → T　　(3) T → T*F

(4) T → F　　(5) F →(E)　　(6) F → id

分析表如下：

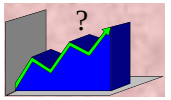
状态	action						goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

示例：对输入符号串id+id*id的分析过程



栈	输入	分析动作	
0	id+id*id$	shift 5	
--			
0 5	+id*id$	reduce by F→id	goto[0,F]=3
--id			
0 3	+id*id$	reduce by T→F	goto[0,T]=2
-- F			
0 2	+id*id$	reduce by E→T	goto[0,E]=1
--T			
0 1	+id*id$	shift 6	
-- E			
0 1 6	id*id$	shift 5	
-- E +			
0 1 6 5	*id$	reduce by F→id	goto[6,F]=3
-- E + id			
0 1 6 3	*id$	reduce by T→F	goto[6,T]=9
-- E + F			

示例：对输入符号串id+id*id的分析过程（续）



栈	输入	分析动作	
0 1 6 9	*id$	shift 7	
-- E + T			
0 1 6 9 7	id$	shift 5	
-- E + T *			
0 1 6 9 7 5	$	reduce by F→id	goto[7,F]=10
-- E + T * id			
0 1 6 9 7 10	$	reduce by T→T*F	goto[6,T]=9
-- E + T * F			
0 1 6 9	$	reduce by E→E+T	goto[0,E]=1
-- E + T			
0 1	$	acc	
-- E			

4.4.2 SLR(1)分析表的构造

- 中心思想：
  - 为给定的文法构造一个识别它所有活前缀的DFA
  - 根据该DFA构造文法的分析表

- 构造识别给定文法的所有活前缀的DFA
- SLR(1)分析表的构造

构造识别给定文法所有活前缀的DFA

- 活前缀与句柄之间的关系
  - 活前缀不含有句柄的任何符号
  - 活前缀只含有句柄的部分符号
  - 活前缀已经含有句柄的全部符号
- 分析过程中，分析栈中出现的活前缀
  - 第一种情况，期望从剩余输入串中能够看到由某产生式$A\rightarrow\alpha$的右部α所推导出的终结符号串；
  - 第二种情况，某产生式$A\rightarrow\alpha_1\alpha_2$的右部子串$\alpha_1$已经出现在栈顶，期待从剩余的输入串中能够看到$\alpha_2$推导出的符号串；
  - 第三种情况，某一产生式$A\rightarrow\alpha$的右部符号串α已经出现在栈顶，用该产生式进行归约。

LR(0)项目

- 右部某个位置上标有圆点的产生式称为文法G的一个LR(0)项目
- 产生式A→XYZ对应有4个LR(0)项目

A→·XYZ
A→X·YZ
A→XY·Z
} 移进-待约项目

A→XYZ· 归约项目

- 归约项目：圆点在产生式最右端的LR(0)项目
- 接受项目：对文法开始符号的归约项目
- 待约项目：圆点后第一个符号为非终结符号的LR(0)项目
- 移进项目：圆点后第一个符号为终结符号的LR(0)项目
- 注意：产生式A→ε只有一个LR(0)归约项目 A→·。

拓广文法

- 任何文法$G=(V_T,V_N,S,\varphi)$，都有等价的文法$G'=(V_T,V_N\cup\{S'\},S',\varphi\cup\{S'\rightarrow S\})$，称$G'$为$G$的拓广文法。
- 拓广文法G'的接受项目是唯一的(即$S'\rightarrow S\cdot$)

定义4.10：LR(0)有效项目

项目$A\rightarrow\beta_1\ \beta_2$对活前缀$\gamma=\alpha\beta_1$是有效的，如果存在一个规范推导：

$$S \overset{*}{\Rightarrow} \alpha A\omega \Rightarrow \alpha\beta_1\beta_2\omega。$$

- 推广：若项目$A\rightarrow\alpha\ B\beta$对活前缀$\gamma=\delta\alpha$是有效的，并且$B\rightarrow\eta$是一个产生式，则项目$B\rightarrow\ \eta$对活前缀$\gamma=\delta\alpha$也是有效的。

- LR(0)项目$S'\rightarrow\ S$是活前缀ε的有效项目
  （这里取$\gamma=\varepsilon$，$\beta_1=\varepsilon$，$\beta_2=S$，$A=S'$）

LR(0)有效项目集和LR(0)项目集规范族

文法G的某个活前缀γ的所有LR(0)有效项目组成的集合称为γ的LR(0)有效项目集。

文法G的所有LR(0)有效项目集组成的集合称为G的LR(0)项目集规范族。

定义4.11：闭包(closure)

设I是文法G的一个LR(0)项目集合，closure(I)是从I 出发，用下面的方法构造的项目集：

(1) I中的每一个项目都属于closure(I)；

(2) 若项目$A\rightarrow\alpha\cdot B\beta$属于closure(I)，
且$B\rightarrow\eta$是G的一个产生式，
若$B\rightarrow\cdot\eta$不属于closure(I)，
则将$B\rightarrow\cdot\eta$加入closure(I)；

(3) 重复规则(2)，直到closure(I)不再增大为止。

算法4. 4　closure(I)的构造过程

输入：项目集合*I*。

输出：集合J=closure(I)。

方法：

```
J=I;
do {
    J_new=J;
    for ( J_new中的每一个项目A→α·Bβ
          和文法G的每个产生式B→η)
        if (B→ η∉J)   把B→ η加入J;
} while (J_new!=J).
```

定义4.12：转移函数go

若I是文法G的一个LR(0)项目集，X是一个文法符号，定义

$$go(I,X)=closure(J)$$

其中：$J=\{ A\rightarrow\alpha X\cdot\beta \mid$ 当 $A\rightarrow\alpha\cdot X\beta$ 属于 I 时 $\}$

- go(I,X)称为转移函数
- 项目$A\rightarrow\alpha X\cdot\beta$称为$A\rightarrow\alpha\cdot X\beta$的后继

直观含义： 若I中的项目$A\rightarrow\alpha\cdot X\beta$是某个活前缀$\gamma=\delta\alpha$的有效项目，J中的项目$A\rightarrow\alpha X\cdot\beta$是活前缀$\delta\alpha X$（即$\gamma X$）的有效项目。

算法4.5　构造文法G的LR(0)项目集规范族

输入：文法G

输出：G的LR(0)项目集规范族C

方法：

```
C={closure({S′→·S})};
do
   for (对C中的每一个项目集I和每一个文法符号X)
      if  (go(I,X)不为空，且不在C中)
          把go(I,X)加入C中;
while (没有新项目集加入C中);
```

这里closure({S′→·S})是活前缀 ε 的有效项目集

示例：

- 构造如下文法G的LR(0)项目集规范族：
  S→aA|bB　A→cA|d　B→cB|d　（文法4.6）
- 拓广文法G′：
  S′→S　S→aA|bB　A→cA|d　B→cB|d
- 活前缀ε的有效项目集
  I_0=closure({S′→·S})={ S′→·S，S→·aA，S→·bB }
- 从I_0出发的转移有

活前缀

I_1=go(I_0,S)=closure({S′→S·})={S′→S·}　——S

I_2 =go(I_0,a)=closure({S→a·A})
=={S→a·A，A→·cA，A→·d}　——a

I_3 =go(I_0,b)=closure({S→b·B})
=={S→b·B，B→·cB，B→·d}　——b

示例（续1）：

- 从I_2出发的转移有

活前缀

$I_4=go(I_2,A)=closure(\{S\rightarrow aA\cdot\})=\{ S\rightarrow aA\cdot\}$ — aA

$I_5=go(I_2,c)=closure(\{A\rightarrow c\cdot A\})$

$=\{ A\rightarrow c\cdot A,\ A\rightarrow\cdot cA,\ A\rightarrow\cdot d\}$ — ac

$I_6=go(I_2,d)=closure(\{A\rightarrow d\cdot\})=\{A\rightarrow d\cdot\}$ — ad

- 从I_3出发的转移有

活前缀

$I_7=go(I_3,B)=closure(\{S\rightarrow bB\cdot\})=\{ S\rightarrow bB\cdot\}$ — bB

$I_8=go(I_3,c)=closure(\{B\rightarrow c\cdot B\})$

$=\{ B\rightarrow c\cdot B,\ B\rightarrow\cdot cB,\ B\rightarrow\cdot d\}$ — bc

$I_9=go(I_3,d)=closure(\{B\rightarrow d\cdot\})=\{B\rightarrow d\cdot\}$ — bd

示例（续2）：

- 从I_5出发的转移有

$I_{10}=go(I_5,A)=closure(\{A\rightarrow cA\cdot\})=\{A\rightarrow cA\cdot\}$ 活前缀—acA

$go(I_5,c)=closure(\{A\rightarrow c\cdot A\})=I_5$ 活前缀—acc

$go(I_5,d)=closure(\{A\rightarrow d\cdot\})=I_6$ 活前缀—acd

- 从I_8出发的转移有

$I_{11}=go(I_8,B)=closure(\{B\rightarrow cB\cdot\})=\{B\rightarrow cB\cdot\}$ 活前缀—bcB

$go(I_8,c)=closure(\{B\rightarrow c\cdot B\})=I_8$ 活前缀—bcc

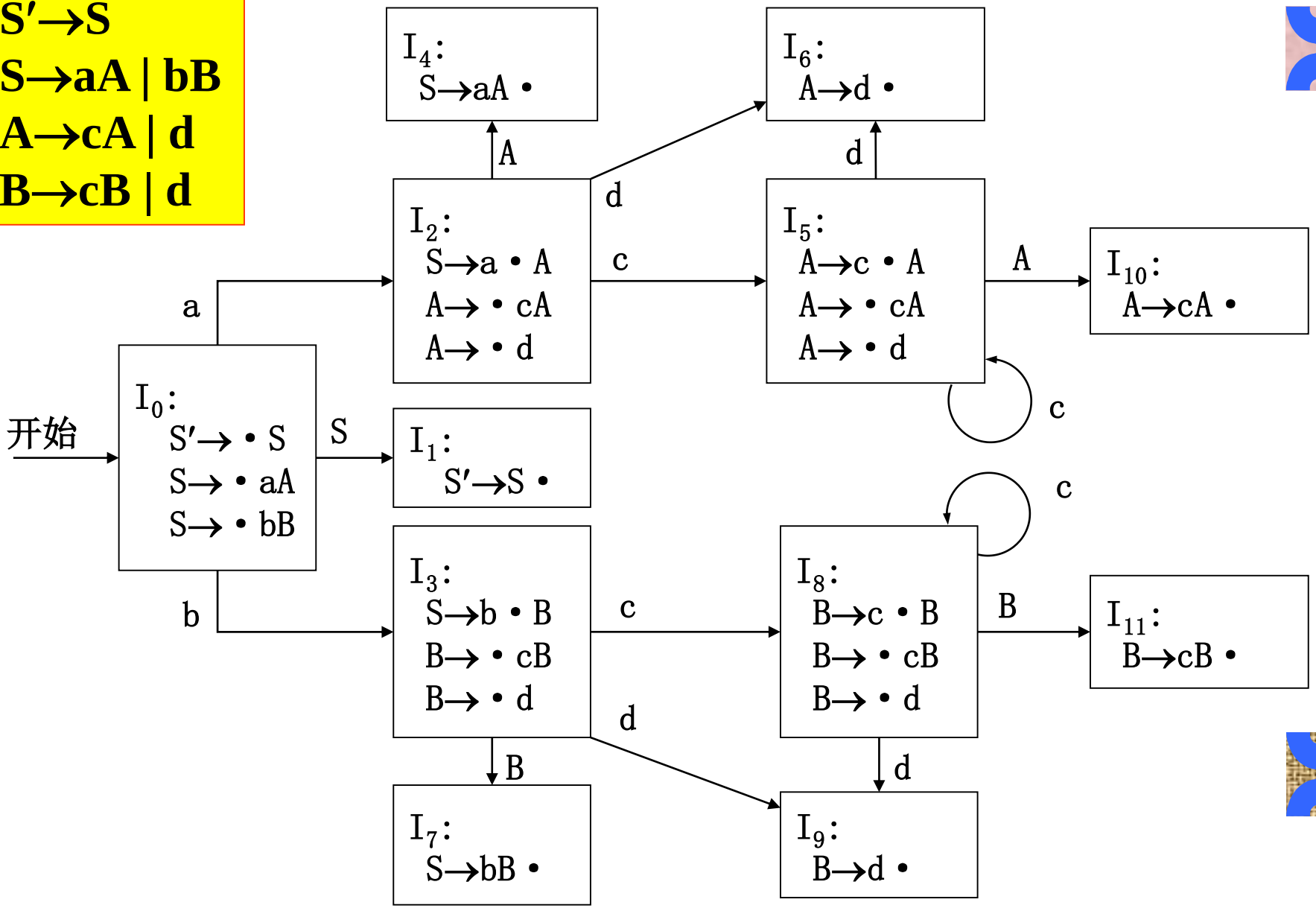
$go(I_8,d)=closure(\{B\rightarrow d\cdot\})=I_9$ 活前缀—bcd

- 文法G′的LR(0)项目集规范族

$C=\{I_0、I_1、\cdots、I_{11}\}$

识别文法G′的所有活前缀的DFA

S′→S
S→aA | bB
A→cA | d
B→cB | d



几点讨论

- 可以构造一个识别活前缀的NFA
  - 状态是项目本身，从$A\rightarrow\alpha\cdot X\beta$到$A\rightarrow\alpha X\cdot\beta$有转换标记X，从$A\rightarrow\alpha\cdot B\beta$到$B\rightarrow\cdot\eta$有转换标记ε
  - 此NFA与上述DFA等价
- 如果项目$A\rightarrow\beta_1\cdot\beta_2$对活前缀$\gamma=\alpha\beta_1$是有效的，当$\alpha\beta_1$在分析栈的栈顶时
  - 如果$\beta_2\neq\varepsilon$，表示句柄没有完全入栈，分析器应该移进
  - 如果$\beta_2=\varepsilon$，　β_1是句柄，应该用产生式$A\rightarrow\beta_1$归约
- 一个活前缀γ的有效项目集就是从上述DFA的初态出发，沿着标记为γ的路径到达的那个项目集（状态）
  - 这是LR分析理论的一条基本定理
  - 当活前缀γ在栈顶时，其有效项目集（即栈顶状态）概括了所有可以从栈中收集到的有用信息
- 冲突通过向前看几个输入符号或许能够解决

SLR(1)分析表的构造

- SLR分析方法的一个特征
  - 如果文法的有效项目集中有冲突动作，多数冲突可通过考察有关非终结符号的FOLLOW集合而得到解决
- 如项目集：$I=\{X\rightarrow\alpha\cdot b\beta,\ A\rightarrow\alpha\cdot,\ B\rightarrow\alpha\cdot\}$
  - 存在移进-归约冲突
  - 存在归约-归约冲突
- 冲突的解决方法，查看FOLLOW(A)和FOLLOW(B)
  - 假设：
    - $FOLLOW(A)\cap FOLLOW(B)=\Phi$
    - $b\notin FOLLOW(A)$ 并且 $b\notin FOLLOW(B)$
  - 决策：
    - 当a=b时，把b移进栈里；
    - 当$a\in FOLLOW(A)$时，用产生式$A\rightarrow\alpha$进行归约；
    - 当$a\in FOLLOW(B)$时，用产生式$B\rightarrow\alpha$进行归约。

算法4.6　构造SLR(1)分析表

输入：拓广文法G′　输出：G′的SLR分析表　方法如下：

1.构造G′的LR(0)项目集规范族C={I_0,I_1,...I_n}。

2.对于状态i（对应于项目集I_i的状态）的分析动作如下

a) 若$A\to\alpha\cdot a\beta\in I_i$，且$go(I_i,a)=I_j$，则置
action[i,a]=sj

b) 若$A\to\alpha\cdot\in I_i$，则对所有$a\in FOLLOW(A)$，置
action[i,a]=r $A\to\alpha$

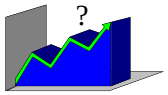
c) 若$S'\to S\cdot\in I_i$，则置action[i,$]=acc，表示分析成功

3.若$go(I_i,A)=I_j$，A为非终结符号，则置goto[i,A]=j

4.分析表中凡不能用规则(2)、(3)填入信息的空白表项，均置为出错标志error。

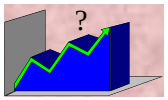
5.分析程序的初态是包含项目$S'\to\cdot S$的有效项目集所对应的状态。

示例：构造文法4.6的SLR(1)分析表



拓广文法：(0) S′→S　(1) S→aA　(2) S→bB
(3) A→cA　(4) A→d　(5) B→cB　(6) B→d

- 构造出该文法的LR(0)项目集规范族、及识别文法活前缀的DFA
- 考察I_0={ S′→·S，S→·aA，S→·bB }
  - 对项目S′→·S，有go(I_0,S)=I_1，所以 goto[0,S]=1
  - 对项目S→·aA，有go(I_0,a)=I_2，所以action[0,a]=s2
  - 对项目S→·bB，有go(I_0,b)=I_3，所以action[0,b]=s3
- 考察I_1={S′→S·}
  - 项目S′→S·是接受项目，所以action[1,$]=acc
- 考察I_2={S→a·A，A→·cA，A→·d}
  - 对项目S→a·A，有go(I_2,A)=I_4，所以 goto[2,A]=4
  - 对项目A→·cA，有go(I_2,c)=I_5，所以action[2,c]=s5
  - 对项目A→·d，　有go(I_2,d)=I_6，所以action[2,d]=s6



- 考察I_3={S→b·B，B→·cB，B→·d}
  - 对项目S→b·B，有go(I_3,B)=I_7，所以 goto[3,B]=7
  - 对项目B→·cB，有go(I_3,c)=I_8，所以action[3,c]=s8
  - 对项目B→·d，有go(I_3,d)=I_9，所以action[3,d]=s9
- 考察I_4={ S→aA·}
  - 项目S→aA·是归约项目，因为FOLLOW(S)={$}，所以action[4,$]=r1
- 考察I_5={ A→c·A，A→·cA，A→·d}
  - 对项目A→c·A，有go(I_5,A)=I_{10}，所以goto[5,A]=10
  - 对项目A→·cA，有go(I_5,c)=I_5，所以action[5,c]=s5
  - 对项目A→·d，有go(I_5,d)=I_6，所以action[5,d]=s6
- 考察I_6={A→d·}
  - 项目A→d·是归约项目，因为FOLLOW(A)={$}，所以action[6,$]=r4

文法4.6的SLR(1)分析表

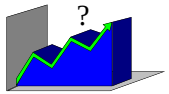
状态	action					goto		
	a	b	c	d	$	S	A	B
0	s2	S3				1		
1					acc			
2			s5	s6			4	
3			s8	s9				7
4					r1			
5			s5	s6			10	
6					r4			
7					r2			
8			s8	s9				11
9					r6			
10					r3			
11					r5			

SLR(1)文法

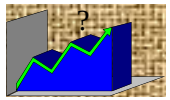
- 若用算法4.5构造出来的分析表不含有冲突，则该分析表称为该文法的SLR(1)分析表
- 具有SLR(1)分析表的文法称为SLR(1)文法
- 使用SLR分析表的分析器，称为SLR分析器

LR(0)分析表和LR(0)文法

- 如果在执行上述算法的过程中，始终没有向前看任何输入符号，则构造的SLR分析表称为LR(0)分析表



- 具有LR(0)分析表的文法称为LR(0)文法。
- 一个文法是LR(0)文法，当且仅当该文法的每个活前缀的有效项目集中：
  - 要么所有元素都是移进-待约项目
  - 要么只含有唯一的归约项目



- 具有如下产生式的文法是一个LR(0)文法。

$$A \rightarrow (A) \mid a$$

示例：
判断文法4.3是LR(0)文法，还是SLR(1)文法。

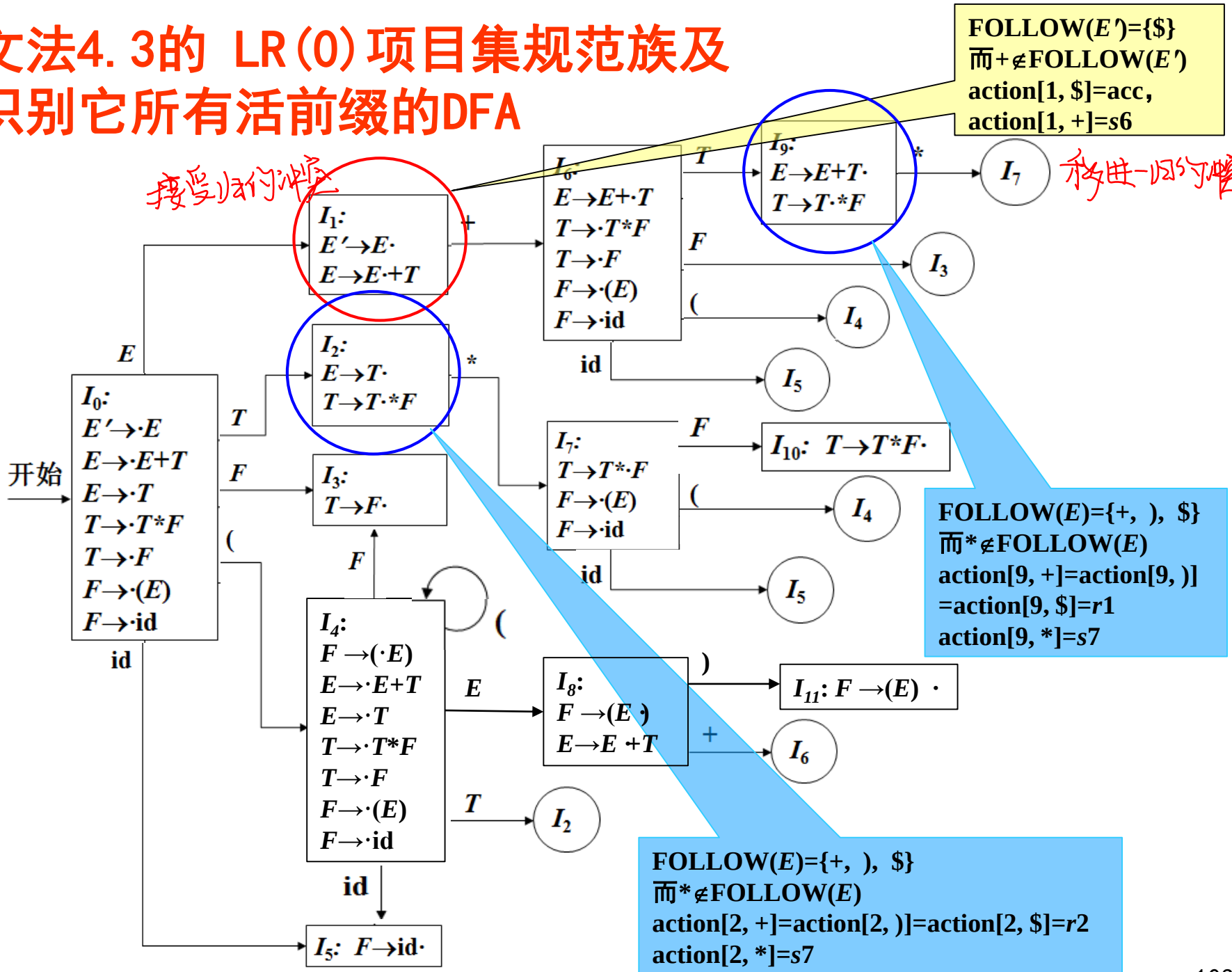
- 文法4.3的拓广文法G′：

(0) E′→E　(1) E→E+T　(2) E→T　(3) T→T*F

(4) T→F　(5) F→(E)　(6) F→id

- 构造G′的LR(0)项目集规范族及识别它所有活前缀的DFA

文法4.3的 LR(0)项目集规范族及识别它所有活前缀的DFA



文法4.3的SLR(1)分析表

状态	action						goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

每一个SLR(1)文法都是无二义的文法
但并非无二义的文法都是SLR(1)文法。

- 例：有文法G具有如下产生式：

  每一个SLR(1)文法都是无二义的文法

  S→L=R

  S→R

  L→*R

  L→id

  R→L　　（文法4.7）

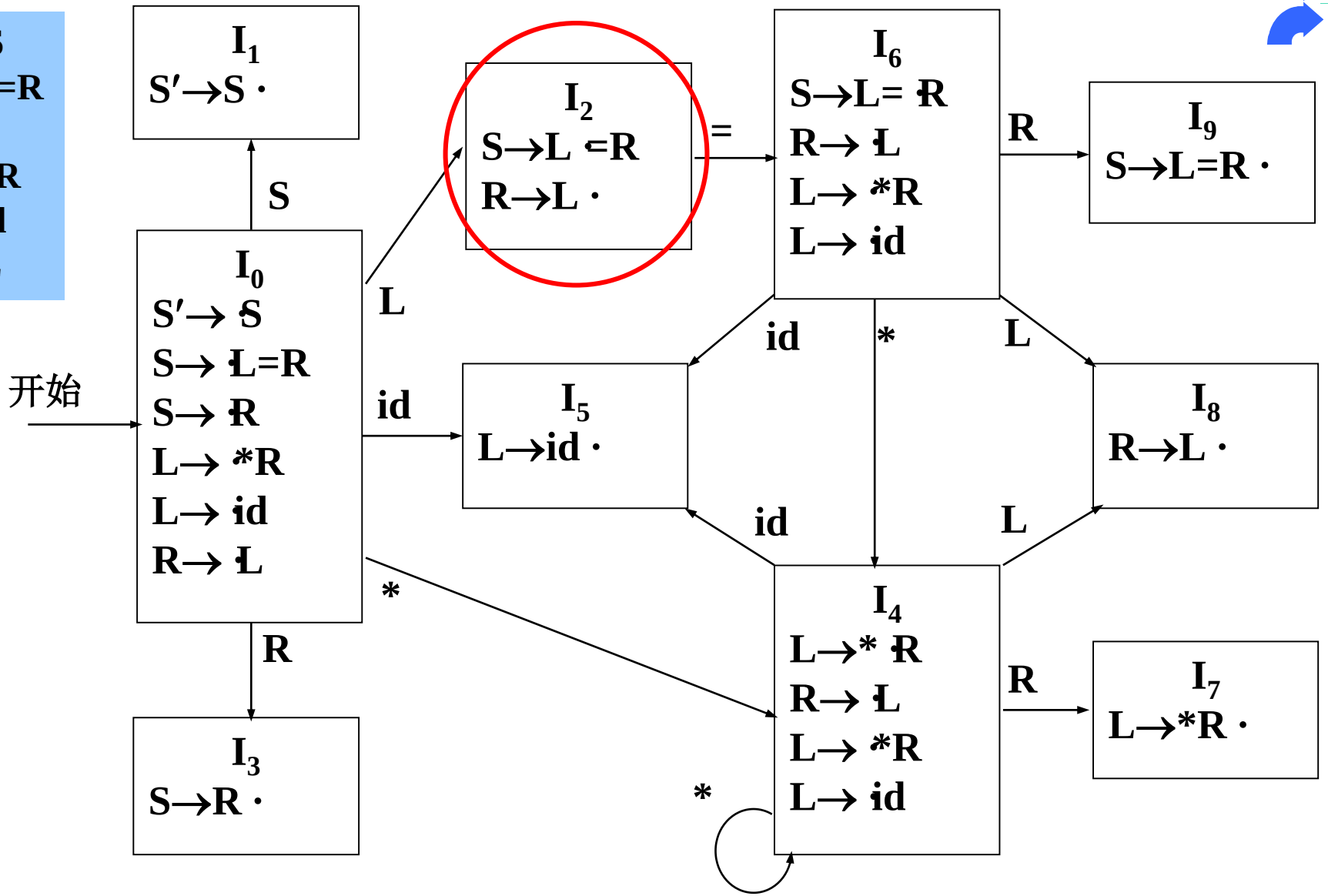
- 该文法无二义性，但不是SLR(1)文法。
- 拓广文法G′的产生式如下：

  (0) S′→S　　(1) S→L=R　　(2) S→R

  (3) L→*R　　(4) L→id　　(5) R→L

构造文法G′的LR(0)项目集规范族及识别活前缀的DFA

S'→S
S→L=R
S→R
L→*R
L→id
R→L



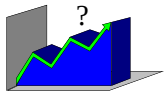
应用算法4.5，为之构造SLR分析表

- 考察I_2:
  - 项目S→L·=R与R→L·存在移进-归约冲突，
  - 根据第一个项目，有：action[2,=]=s6
  - 根据第二个项目，由于FOLLOW(R)={=,$}
    所以有： action[2,$]=action[2,=]=r5
- 存在“移进-归约”冲突。
- 原因是在SLR分析表中未包含足够多的预测信息。

4.4.3 LR(1)分析表的构造

- SLR分析法的问题：FOLLOW函数的信息太粗略

  构造SLR(1)分析表时，如果项目集I_k包含项目$A\rightarrow\alpha$；且$a\in FOLLOW(A)$，则在状态k用产生式$A\rightarrow\alpha$归约。但是当状态k出现在栈顶时，栈中的活前缀$\beta\alpha$有时却使得右句型中的βA后面不能跟随a，在这种情况下，当输入符号为a时，用产生式$A\rightarrow\alpha$归约将是不合法的



- 例子：
  - 对于文法4.8，当状态2处于栈顶，面临输入符号“=”时，由于这个文法不存在以“R=”为前缀的规范句型，因此不能用$R\rightarrow L$对栈顶的L进行归约。
  - “=”属于FOLLOW(R)是因为存在以“*R=”为前缀的规范句型，但不存在以“R=”为前缀的规范句型，FOLLOW函数不能区分这两种情况

- 解决方法：

  对LR(0)项目进行分裂，使得LR分析器的每个状态能确定地指出，当α后紧跟那些终结符时，才允许把α归约为A

LR(k)项目

- LR(k)项目：$[A\rightarrow\alpha\cdot\beta, a_1a_2...a_k]$
  - $A\rightarrow\alpha\cdot\beta$是一个LR(0)项目
  - $a_i(i=1,2,...,k)$是终结符号
  - $a_1a_2...a_k$称为该项目的向前看符号串
- 向前看符号串仅对归约项目$[A\rightarrow\alpha\cdot, a_1a_2...a_k]$起作用，对任何移进或待约项目$[A\rightarrow\alpha\cdot\beta, a_1a_2...a_k]$，$(\beta\neq\varepsilon)$是没有意义的。
- 归约项目$[A\rightarrow\alpha\cdot, a_1a_2...a_k]$意味着，当它所属项目集对应的状态在栈顶，且后续的k个输入符号为$a_1a_2...a_k$时，才允许把栈顶的文法符号串α归约为A。
- 绝大多数程序设计语言是LR(1)

定义4.13：LR(1)有效项目

称一个LR(1)项目[A→α β，a]对活前缀γ=δα是有效的，如果存在一个规范推导：

$$S \overset{*}{\Rightarrow} \delta A\omega \Rightarrow \delta\alpha\beta\omega。$$

其中ω的第一个符号为**a**，或者ω=ε，且**a=$**。

- **推广**：若项目[A→α Bβ，a]对活前缀γ=δα是有效的，并且B→η是一个产生式，则对任何b∈FIRST(βa)，项目[B→ η，b]对活前缀γ=δα 也是有效的。
- b或是从β推出的开头终结符号，或者$\beta \overset{*}{\Rightarrow} \varepsilon$，而b=a。
- LR(1)项目[S′→ S, $]对活前缀 ε 是有效的。

向前看符号串

LR(1)有效项目集和 LR(1)项目集规范族

- 文法G的某个活前缀γ的所有LR(1)有效项目组成的集合称为γ的LR(1)有效项目集。
- 文法G的所有LR(1)有效项目集组成的集合称为G的LR(1)项目集规范族。

定义4.14：闭包（closure）

设I是文法G的一个LR(1)项目集，closure(I)是从I出发，用下面的方法构造的项目集

(1) I中的每一个项目都属于closure(I)；

(2) 若项目[A→α·Bβ，a]属于closure(I)，且B→η是G的一个产生式，则对任何终结符号b∈FIRST(βa)，若项目[B→·η，b]不属于集合closure(I)，则将它加入closure(I)；

(3) 重复规则(2)，直到closure(I)不再增大为止。

算法4.7 closure(I)的构造过程

输入：项目集合*I*。

输出：集合*J*=closure(I)。

方法：

```
J=I;
do {
J_new=J;
for ( J_new中的每一个项目[A→α·Bβ，a] 和
      文法G的每个产生式B→η )
   for (FIRST(βa)中的每一个终结符号b)
      if ([B→ η，b]∉J) 把[B→ η，b]加入J；
} while (J_new!=J);
```

定义4.15：转移函数go

若I是文法G的一个LR(1)项目集，X是一个文法符号，定义：

$$go(I,X)=closure(J)$$

其中：J={ [A→αX·β，a] | 当[A→α·Xβ，a]属于I时 }

项目[A→αX·β，a]称为[A→α·Xβ，a]的后继。

直观含义：

若I是某个活前缀 γ 的有效项目集，

则 go(I,X) 便是对活前缀 γX 的有效项目集。

算法4.8 构造文法G的LR(1)项目集规范族

输入：拓广文法G′

输出：G′的LR(1)项目集规范族

方法：

```
C={closure({[S′→·S，$]})};
do
  for (C中的每一个项目集I和每一个文法符号X)
    if (go(I,X)不为空，且不在C中)
      把go(I,X)加入C中;
while (没有新项目集加入C中).
```

示例：构造如下文法的LR(1)项目集规范族：

(1) S→CC　(2) C→cC　(3) C→d　（文法4.8）

- 拓广文法：

  (0) S′→S　(1) S→CC　(2) C→cC　(3) C→d

- 根据算法4.6构造其LR(1)项目集规范族。

I_0=closure({[S′→·S, $]})

={[S′→·S, $] [S→·CC, $] [C→·cC, c/d] [C→·d, c/d]}

I_1=go(I_0, S)=closure({[S′→S·, $]})={[S′→S·, $]}

I_2=go(I_0, C)=closure({[S→C·C, $]})

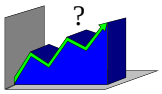
={[S→C·C, $] [C→·cC, $] [C→·d, $]}

I_3=go(I_0, c)=closure({[C→c·C, c/d]})

={[C→c·C, c/d] [C→·cC, c/d] [C→·d, c/d]}

I_4=go(I_0, d)=closure({[C→d·, c/d]})={[C→d·, c/d]}

示例（续）



$I_5=go(I_2, C)=closure(\{[S\rightarrow CC\cdot, \$]\})=\{[S\rightarrow CC\cdot, \$]\}$

$I_6=go(I_2, c)=closure(\{[C\rightarrow c\cdot C, \$]\})$

$=\{[C\rightarrow c\cdot C, \$]\ [C\rightarrow\cdot cC, \$]\ [C\rightarrow\cdot d, \$]\}$

$I_7=go(I_2, d)=closure(\{[C\rightarrow d\cdot, \$]\})=\{[C\rightarrow d\cdot, \$]\}$

$I_8=go(I_3, C)=closure(\{[C\rightarrow cC\ ;\ c/d]\})=\{[C\rightarrow cC\ ;\ c/d]\}$

$go(I_3, c)=closure(\{[C\rightarrow c\ C, c/d]\})=I_3$

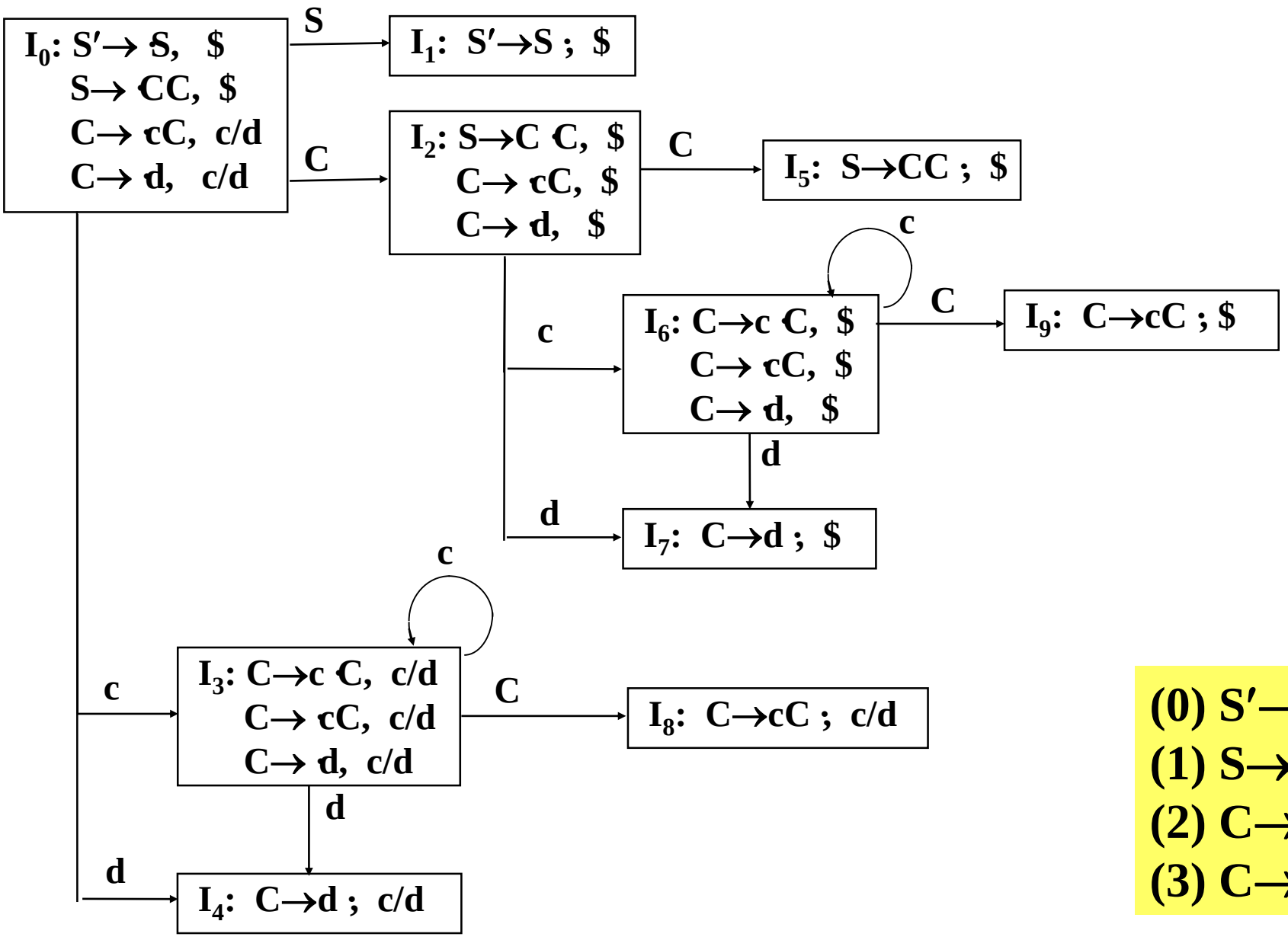
$go(I_3, d)=closure(\{[C\rightarrow d\ ;\ c/d]\})=I_4$

$I_9=go(I_6, C)=closure(\{[C\rightarrow cC\ ;\ \$]\})=\{[C\rightarrow cC\ ;\ \$]\}$

$go(I_6, c)=closure(\{[C\rightarrow c\ C, \$]\})=I_6$

$go(I_6, d)=closure(\{[C\rightarrow d\ ;\ \$]\})=I_7$

识别文法4.8所有活前缀的DFA



(0) S′→S
(1) S→CC
(2) C→cC
(3) C→d

算法4.9　构造LR(1)分析表

输入：拓广文法G′　输出：文法G′的分析表

方法如下：

1.构造文法G′的LR(1)项目集规范族C={$I_0,I_1,\ldots,I_n$}

2.对于状态i（代表项目集I_i），分析动作如下：

a) 若[A→α·aβ,b]∈I_i，且go(I_i,a)=I_j，则置action[i,a]=sj

b) 若[A→α·,a]∈I_i，且A≠S′，则置action[i,a]=rj

c) 若[S′→S·,$]∈$I_i$，则置action[i,$]=acc

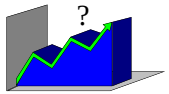
3.若对非终结符号A，有go(I_i,A)=I_j，则置goto[i,A]=j

4.凡是不能用上述规则填入信息的空白表项，均置上出错标志error。

5.分析程序的初态是包括[S′→·S,$]的有效项目集所对应的状态。

- 与SLR分析表构造方法的不同点：
  - 对归约项目，LR(1)用向前看符号，SLR用FOLLOW集
- 规范的LR(1)分析表
  - 用以上算法构造的分析表，如果不含多重定义的表项，即没有动作冲突，则称它为文法G的LR(1)分析表
- 规范的LR(1)分析器
  - 使用LR(1)分析表的分析器
- LR(1)文法
  - 具有LR(1)分析表的文法

例：构造文法4.8的LR(1)分析表



- 考察I_0：
  - $[S' \rightarrow \cdot S, \$]$ 且 $go(I_0, S)=I_1$， 故：$goto[0, S]=1$
  - $[S \rightarrow \cdot CC, \$]$ $go(I_0, C)=I_2$ $goto[0, C]=2$
  - $[C \rightarrow \cdot cC, c/d]$ $go(I_0, c)=I_3$ $action[0, c]=s3$
  - $[C \rightarrow \cdot d, c/d]$ $go(I_0, d)=I_4$ $action[0, d]=s4$
- 考察I_1：由于$[S' \rightarrow S \cdot, \$]$ 故$action[1, \$]=acc$
- 考察I_2：
  - $[S \rightarrow C \cdot C, \$]$ 且 $go(I_2, C)=I_5$ 故：$goto[2, C]=5$
  - $[C \rightarrow \cdot cC, \$]$ $go(I_2, c)=I_6$ $action[2, c]=s6$
  - $[C \rightarrow \cdot d, \$]$ $go(I_2, d)=I_7$ $action[2, d]=s7$
- 考察I_4：
  - $[C \rightarrow d \cdot, c/d]$ 故 $action[4, c]=action[4, d]=r3$

文法4.8的LR(1)分析表

状态	action			goto	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

示例：

- 试构造文法4.7的LR(1)分析表

  (1) S→L=R　(2) S→R　(3) L→*R

  (4) L→id　　(5) R→L

- 拓广文法G′

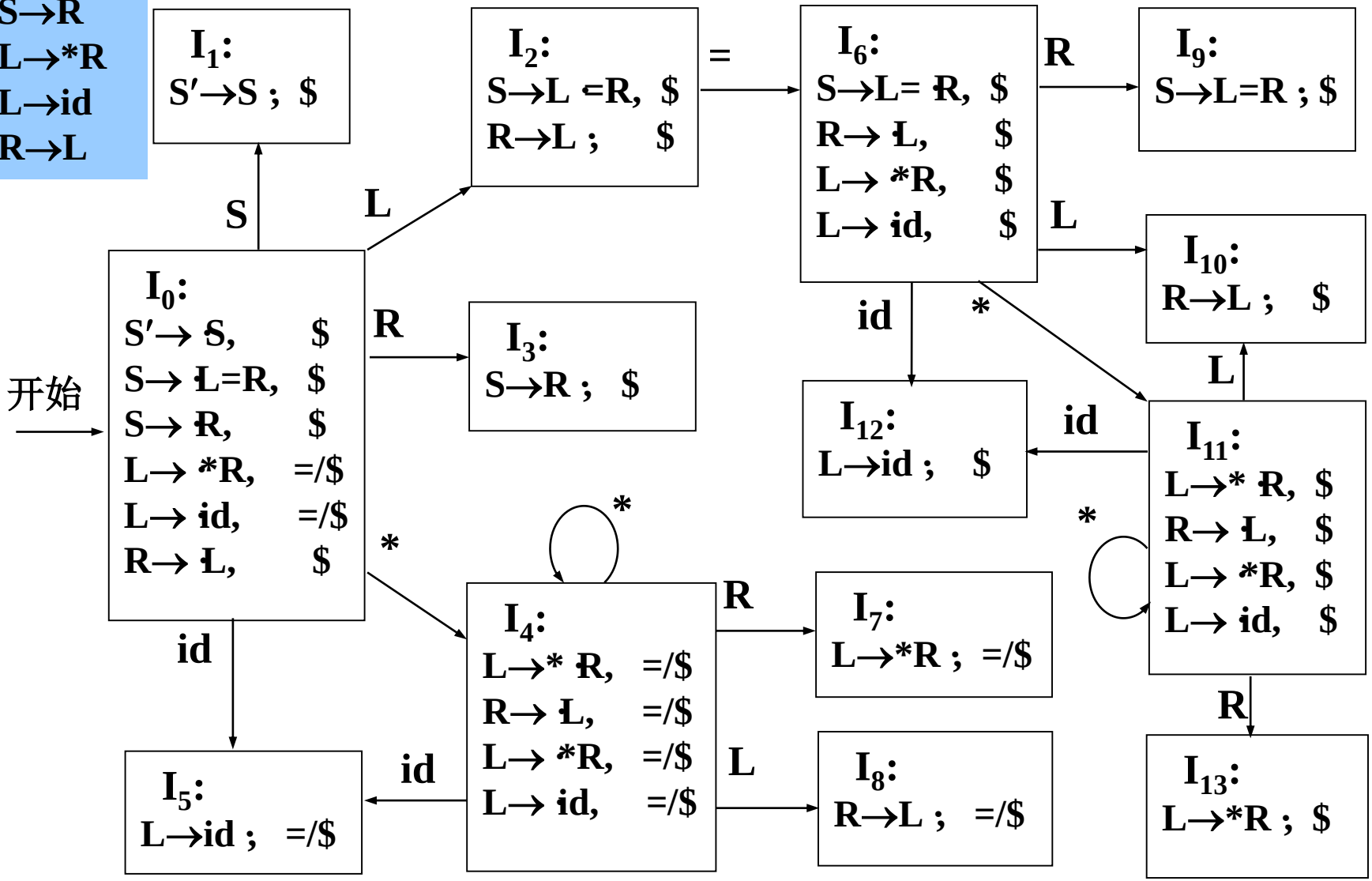
  (0) S′→S　(1) S→L=R　(2) S→R

  (3) L→*R　(4) L→id　　(5) R→L

- 构造文法G′的LR(1)项目集规范族及识别所有活前缀的DFA

0) S'→S
1) S→L=R
2) S→R
3) L→*R
4) L→id
5) R→L

示例（续）：



文法4.7的LR(1)分析表

状态	action				goto		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				acc			
2	s6			r5			
3				r2			
4		s4	s5			8	
5	r4			r4			
6		s11	s12			10	9
7	r3			r3			
8	r5			r5			
9				r1			
10				r5			
11		s11	s12			10	13
12				r4			
13				r3			

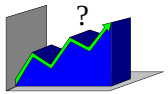
分析方法的比较

- 规范的LR分析法和预测分析法的比较：
  - LL(1)文法要求每个非终结符的产生式的首符集均不相同，用首字符决定所用的产生式；
  - 规范的LR(1)分析法只有在看到整个产生式的右部时（句柄出现在栈顶），才进行归约
  - 可以证明所有LL(1)文法均是LR(1)文法，LR方法更一般化
- 规范的LR分析法与SLR分析法的比较：
  - 规范的LR分析法比SLR分析法功能更强，但代价更高（由于状态的细化）
  - 折衷方法：LALR分析方法，状态数与SLR分析法相同，功能界与两者之间

4.4.4 LALR(1)分析表的构造

- 描述LR(1)项目集特征的两个定义

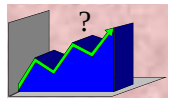
定义4.16 如果两个LR(1)项目集去掉向前看符号之后是相同的，则称这两个项目集具有相同的心（core），即这两个项目集是同心集。



定义4.17 除去初态项目集外，一个项目集的核（kernel）是由该项目集中那些圆点不在最左边的项目组成。LR(1)初态项目集的核中有且只有项目[S′→·S,$]。

构造LALR(1)分析表的基本思想

- 合并LR(1)项目集规范族中的同心集，以减少分析表的状态数。



- 用核代替项目集，以减少项目集所需的存储空间

合并同心集可能的问题

1. **报错问题：**不会遗漏错误，但报错会推迟（多做些归约动作），不会移进更多的符号
2. **转移函数：**go(I,X)仅仅依赖于I的心，因此LR(1)项目集合并后的转移函数可以通过go(I,X)自身的合并得到，不用考虑修改
3. **动作冲突：**不会产生新的移进-归约冲突，可能产生新的归约-归约的冲突

同心集的合并不会引进新的移进–归约冲突

如果合并后的项目集中存在移进-归约冲突，则意味着：

项目[A→α·,a]和[B→β·aγ,b] 处于合并后的同一项目集中

合并前必存在某个c，使得[A→α·,a]和[B→β·aγ,c]同处于某个项目集中。

说明，原来的LR(1)项目集中已经存在移进-归约冲突。

同心集的合并可能导致归约-归约冲突

例：有文法G：

S→aAd|bBd|aBe|bAe

A→c

B→c　　（文法4.9）

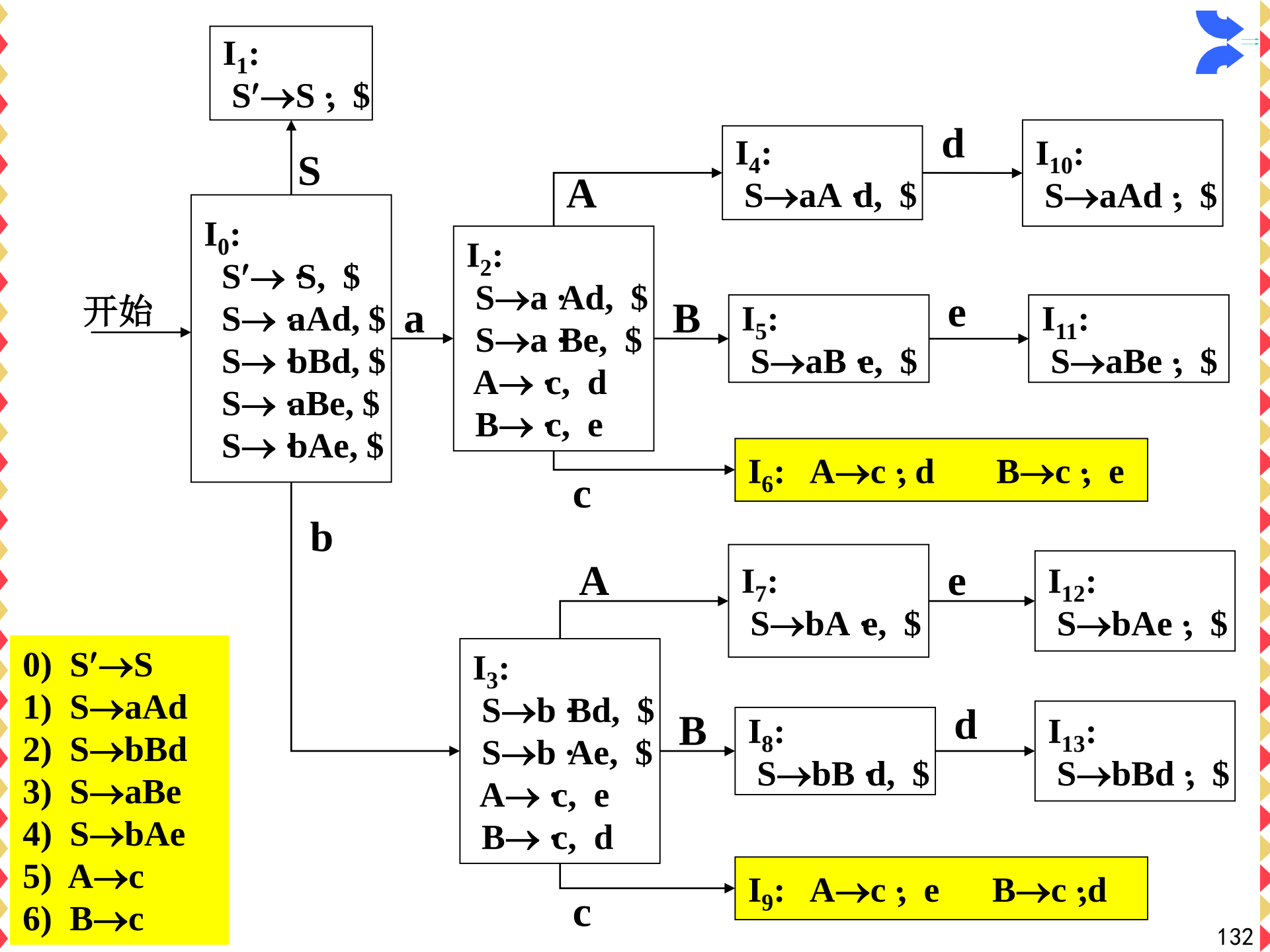
- 拓广文法G′

  (0) S′→S

  (1) S→aAd　(2) S→bBd　(3) S→aBe　(4) S→bAe

  (5) A→c　(6) B→c

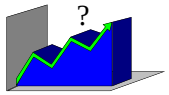
- 构造识别文法G′的所有活前缀的DFA


I1:
S′→S ; $
S
I0:
S′→ S, $
S→ aAd, $
S→ bBd, $
S→ aBe, $
S→ bAe, $
开始
a
b
A
I2:
S→a Ad, $
S→a Be, $
A→ c, d
B→ c, e
I4:
S→aA d, $
d
I10:
S→aAd ; $
B
I5:
S→aB e, $
e
I11:
S→aBe ; $
c
I6: A→c ; d B→c ; e
A
I3:
S→b Bd, $
S→b Ae, $
A→ c, e
B→ c, d
I7:
S→bA e, $
e
I12:
S→bAe ; $
B
I8:
S→bB d, $
d
I13:
S→bBd ; $
c
I9: A→c ; e B→c ;d
0) S′→S
1) S→aAd
2) S→bBd
3) S→aBe
4) S→bAe
5) A→c
6) B→c

文法4.9的LR(1)分析表

状态	action						goto		
	a	b	c	d	e	$	S	A	B
0	s2	s3					1		
1						acc			
2			s6					4	5
3			s9					7	8
4				s10					
5					s11				
6				r5	r6				
7					s12				
8				s13					
9				r6	r5				
10						r1			
11						r3			
12						r4			
13						r2			

合并同心集



- LR(1)项目集规范族中
  - 活前缀ac的有效项目集是I_6
  - 活前缀bc的有效项目集是I_9
  - 这两个项目集都不含冲突项目，且是同心集。

- 它们合并后得到的集合为：
  {[A→c·, d/e] [B→c·, d/e]}

- 含有归约-归约冲突。

LALR(1)分析表的构造

- 基本思想是：
  - 首先构造LR(1)项目集规范族；
  - 如果它不存在冲突，就把同心集合并在一起；

    如果LR(1)项目集规范族中含有冲突，
    则该文法不是LR(1)文法，
    不能为它构造LALR分析表。

  - 若合并后的项目集规范族不存在归约-归约冲突，就按这个项目集规范族构造分析表。

    如果合并后得到的项目集规范族中含有冲突，
    则该文法是LR(1)文法，但不是LALR文法，
    因而，也不能为它构造LALR分析表。

算法4.10　构造LALR(1)分析表

输入：一个拓广文法G′

输出：文法G′的LALR(1)分析表

方法：

1.构造文法G′的LR(1)项目集规范族
$C=\{I_0, I_1, \ldots, I_n\}$。

2. 合并C中的同心集，得到一个新的项目集规范族$C'=\{J_0,J_1,\ldots,J_m\}$，其中含有项目$[S'\rightarrow\cdot S,\$]$的$J_k$为分析表的初态。

3.从C′出发，构造action子表

a) 若$[A\rightarrow\alpha\cdot a\beta,b]\in J_i$，且$go(J_i,a)=J_j$，则置action[i,a]=sj

b) 若$[A\rightarrow\alpha\cdot,a]\in J_i$，则置action[i,a]=r $A\rightarrow\alpha$

c) 若$[S'\rightarrow S\cdot,\$]\in J_i$，则置action[i,$]=acc

算法4.10　构造LALR(1)分析表（续）

4.构造goto子表

设$J_k=\{I_{i1}，I_{i2}，…，I_{it}\}$

由于这些I_i是同心集，因此

$go(I_{i1},X)$、$go(I_{i2},X)$、…、$go(I_{it},X)$也是同心集

把所有这些项目集合并后得到的集合记作J_i，则有：

$$go(J_k,X)=J_i$$

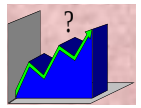
于是，若$go(J_k,A)=J_i$，则置goto[k,A]=I

5.分析表中凡不能用上述规则填入信息的空表项，均置上出错标志。

LALR(1)文法

- LALR(1)项目集规范族：
  - 算法第二步产生的项目集规范族
- LALR(1)分析表：
  - 构造的分析表不存在冲突
- LALR(1)文法：
  - 存在LALR(1)分析表的文法

示例：构造文法4.8的LALR(1)分析表。



- 有三对同心集可以合并，即
  - I_3和I_6合并，得到项目集：

    I_{36}={[C→c C, c/d/$] [C→ cC, c/d/$] [C→ d, c/d/$]}
  - I_4和I_7合并，得到项目集：

    I_{47}={[C→d •， c/d/$]}
  - I_8和I_9合并，得到项目集：

    I_{89}={[C→cC •， c/d/$]}
- 同心集合并后得到的新的项目集规范族为：

  C′={I_0, I_1, I_2, I_{36}, I_{47}, I_5, I_{89}}
- 利用算法4.8，可以为该文法构造LALR(1)分析表。

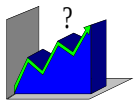
文法4.8的LALR(1)分析表

状态	action			goto	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

- 此文法是LALR(1)文法
- 转移函数go的计算只依赖于项目集的心

LALR(1)分析程序和LR(1)分析程序的比较

- LALR(1)分析程序对符号串cdcd的分析过程

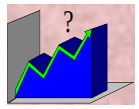


步骤	栈	输入	分析动作
0	0	cdcd$	shift
1	0c36	dcd$	shift
2	0c36d47	cd$	reduce by C→d
3	0c36C89	cd$	reduce by C→cC
4	0C2	cd$	shift
5	0C2c36	d$	shift
6	0C2c36d47	$	reduce by C→d
7	0C2c36C89	$	reduce by C→cC
8	0C2C5	$	reduce by S→CC
9	0S1	$	accept

- LR(1)分析程序对符号串cdcd的分析过程

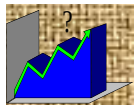
步骤	栈	输入	分析动作
0	0	cdcd$	shift
1	0c3	dcd$	shift
2	0c3d4	cd$	reduce by C→d
3	0c3C8	cd$	reduce by C→cC
4	0C2	cd$	shift
5	0C2c6	d$	shift
6	0C2c6d7	$	reduce by C→d
7	0C2c6C9	$	reduce by C→cC
8	0C2C5	$	reduce by S→CC
9	0S1	$	accept

- LR(1)分析程序对符号串ccd的分析过程



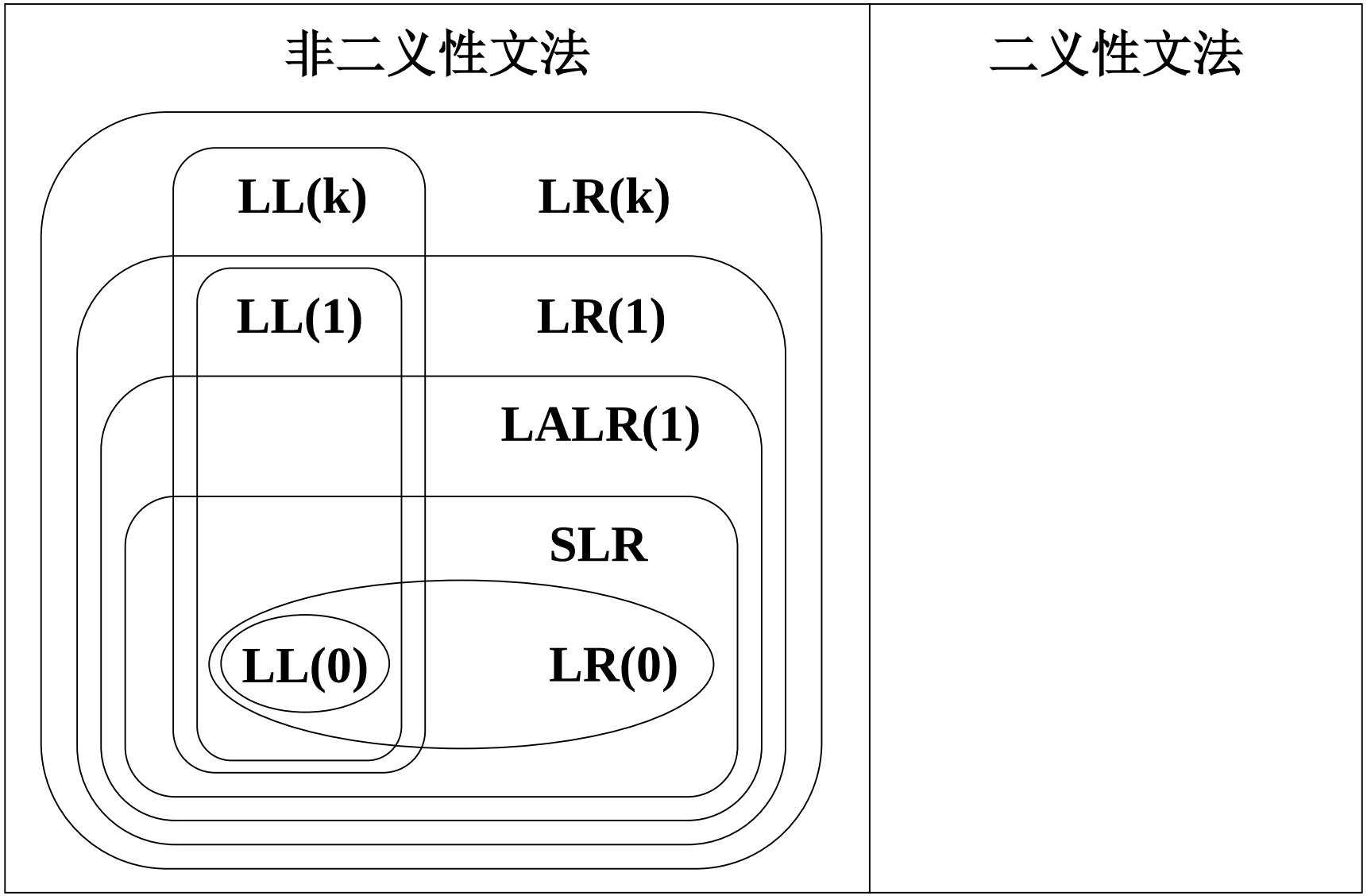
步骤	栈	输入	分析动作
0	0	ccd$	shift
1	0c3	cd$	shift
2	0c3c3	d$	shift
3	0c3c3d4	$	error

- LALR(1)分析程序对符号串ccd的分析过程



步骤	栈	输入	分析动作
0	0	ccd$	shift
1	0c36	cd$	shift
2	0c36c36	d$	shift
3	0c36c36d47	$	reduce by C→d
4	0c36c36C89	$	reduce by C→cC
5	0c36C89	$	reduce by C→cC
6	0C2	$	error

上下文无关文法分类



4.4.5 LR分析方法对二义文法的应用

- 定理：任何二义性文法决不是LR文法，因而也不是SLR或LALR文法。
- 程序设计语言的某些结构用二义性文法描述比较直观，使用方便。例如关于算术表达式的文法和if语句的文法。
- 在所有情况下，都要说明了消除二义性的一些规则（即这类结构的使用限制），以保证对每个句子只有一颗分析树。

利用优先级和结合规则解决表达式冲突

- 描述算术表达式集合的二义性文法：

  $E \rightarrow E+E|E*E|(E)|id$　　（文法4.10）

- 无二义性的文法：

  $E \rightarrow E+T|T$

  $T \rightarrow T*F|F$

  $F \rightarrow (E)|id$

- 前者具有两个明显的优点：
  - 改变运算符号的优先级或结合规则时，文法本身无需改变，只需改变限制条件。
  - LR分析表所包含的状态数比后者少得多，后者含有单非产生式$E \rightarrow T$和$T \rightarrow F$，作用在于定义运算符的优先级和结合规则

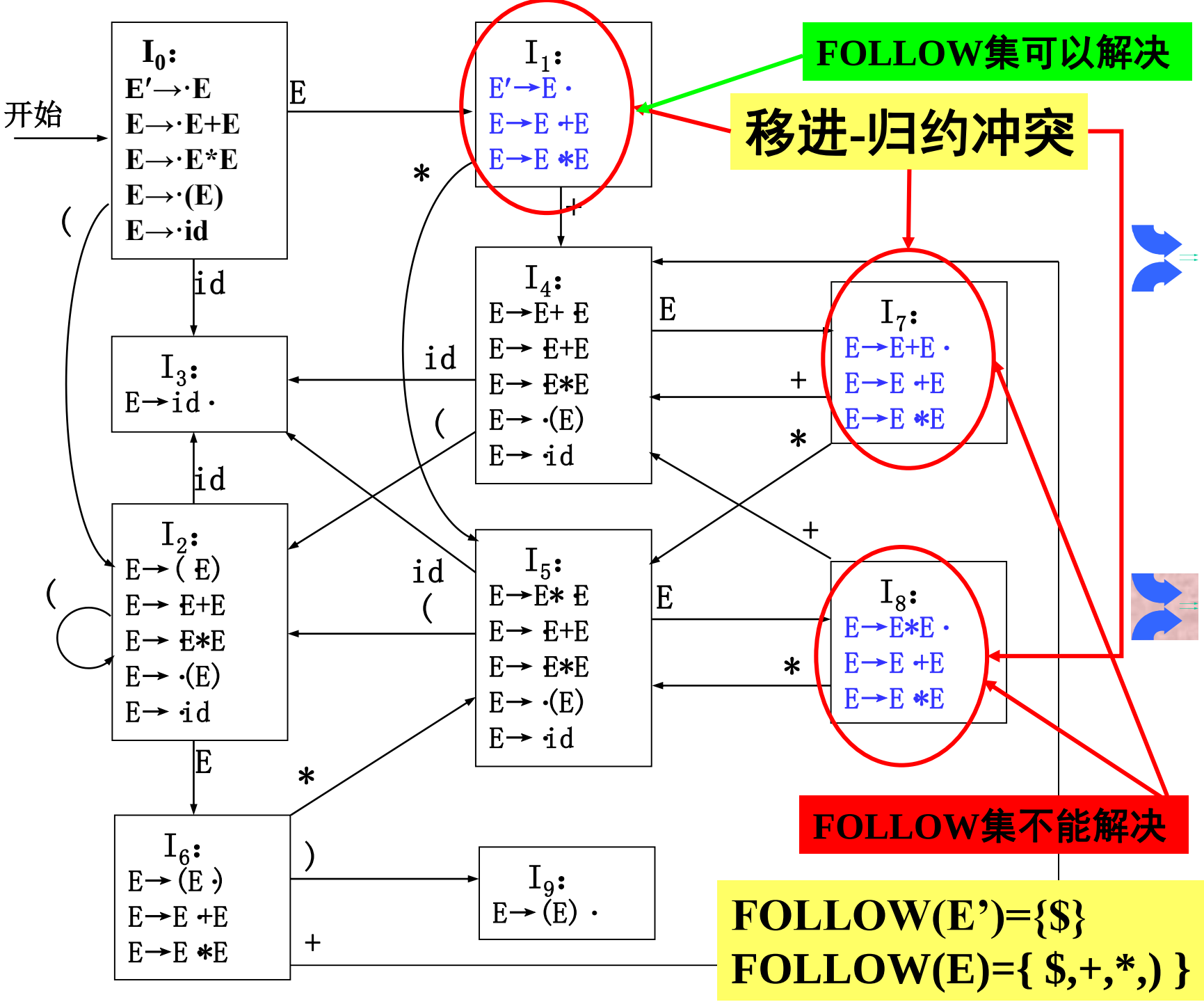
例：构造文法4.10的LR分析表。

- 其拓广文法G′具有产生式：

  (0) E′→E　　(1) E→E+E　(2) E→E*E

  (3) E→(E)　(4) E→id

- 构造文法G′的LR(0)项目集规范族及识别所有活前缀的DFA

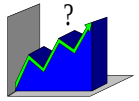

I0:
E′→·E
E→·E+E
E→·E*E
E→·(E)
E→·id
开始
E
I1:
E′→E·
E→E·+E
E→E·*E
(
id
I3:
E→id·
I2:
E→(·E)
E→·E+E
E→·E*E
E→·(E)
E→·id
I4:
E→E+·E
E→·E+E
E→·E*E
E→·(E)
E→·id
I5:
E→E*·E
E→·E+E
E→·E*E
E→·(E)
E→·id
I7:
E→E+E·
E→E·+E
E→E·*E
I8:
E→E*E·
E→E·+E
E→E·*E
I6:
E→(E·)
E→E·+E
E→E·*E
I9:
E→(E)·
)
+
*
FOLLOW集可以解决
移进-归约冲突
FOLLOW集不能解决
FOLLOW(E')={$}
FOLLOW(E)={ $,+,*,) }

‘+’和 ‘*’ 的优先级及结合规则共有4种情况

(1) * 优先于 + ，遵从左结合规则
(2) * 优先于 + ，遵从右结合规则
(3) + 优先于 * ，遵从左结合规则
(4) + 优先于 * ，遵从右结合规则

I_7:
E→E+E·
E→E·+E
E→E·*E

I_8:
E→E*E·
E→E·+E
E→E·*E



条件	状态	action					goto
		id	+	*	(	)	$	E
(1)	7		r1	s5		r1	r1	
	8		r2	r2		r2	r2	
(2)	7		s4	s5		r1	r1	
	8		r2	s5		r2	r2	
(3)	7		r1	r1		r1	r1	
	8		s4	r2		r2	r2	
(4)	7		s4	r1		r1	r1	
	8		s4	s5		r2	r2	

文法4.10的LR分析表

状态	action						goto
	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

利用最近最后匹配原则解决if语句冲突

- 映射程序设计语言中if-then-else结构的文法：

  S→ if E then S else S
  | if E then S
  | others （文法4.11）

- 对该文法进行抽象
  - 用i表示“if E then”
  - 用e表示“else”
  - 用a表示“others”

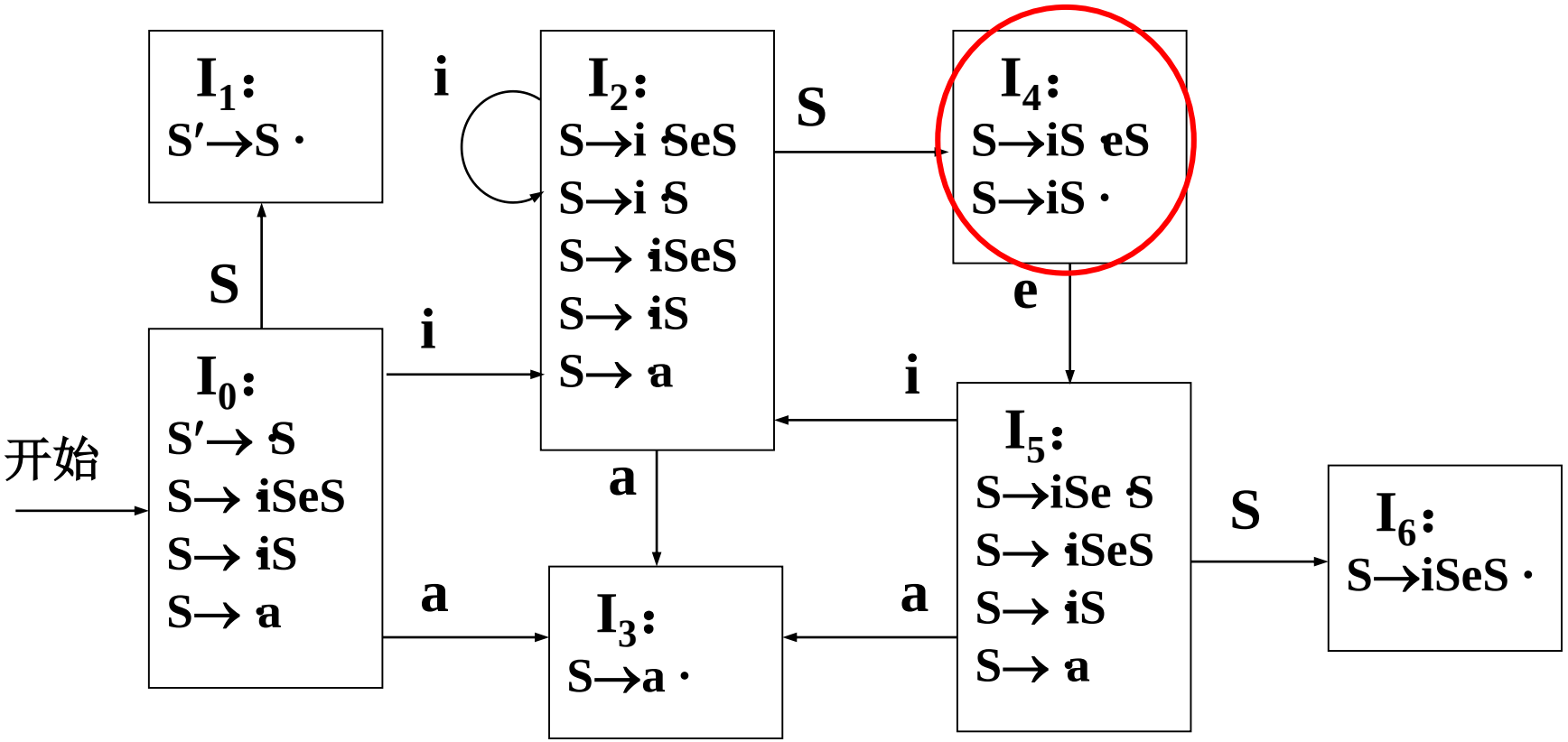
- 得到文法：

  S→iS | iSeS | a （文法4.12）

文法4.12 的LR(0)项目集规范族及识别其所有活前缀的DFA

其拓广文法G′为：

(0) S′→S　(1) S→iSeS　(2) S→iS　(3) S→a



FOLLOW(S)={ $, e }

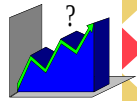
最近最后匹配原则：

else与离它最近的一个未匹配的then相匹配

文法4.14的LR分析表

状态	action				goto
	i	e	a	$	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r2	
5	s2		s3		6
6		r1		r1	

例：分析输入符号串iiaea



步骤	栈	输入	分析动作
1	0 -	iiaea$	shift
2	0 2 - i	iaea$	shift
3	0 2 2 - i i	aea$	shift
4	0 2 2 3 - i i a	ea$	reduce by S→a
5	0 2 2 4 - i i S	ea$	shift
6	0 2 2 4 5 - i i S e	a$	shift
7	0 2 2 4 5 3 - i i S e a	$	reduce by S→a
8	0 2 2 4 5 6 0 i i S e S	$	reduce by S→iSeS
9	0 2 4 - i S	$	reduce by S→iS
10	0 1 - S	$	accept

非上下文无关的语言结构

- 在很多程序设计语言中，仅用上下文无关文法难以完成其中的一些语法结构的说明
- 例：$L=\{\omega c \omega \mid \omega \in (a|b)^* \}$
  - 是检查程序中标识符的声明应先于引用问题的抽象，第一个ω表示声明，第二个ω表示它的引用。Pascal和C不是上下文无关的语言
  - 解决办法：由语义分析检查
- 例：$L=\{a^n b^m c^n d^m \mid m, n \geqslant 0\}$
  - 是检查函数声明的形参个数和函数引用的实参个数一致问题的抽象，a^n和b^m代表两个函数声明的形参表中分别有n和m个参数，c^n和d^m分别代表这两个函数调用的实参表
  - 解决办法：由语义分析检查

4.4.6 LR分析的错误处理与恢复

- LR分析器发现错误：
  - 在访问动作表时遇到出错条目，但在访问转移表时决不会遇到出错条目（只有在归约时访问转移表）
- 只要已扫描的输入出现一个不正确的后继，LR分析器便立即报错，决不会把不正确的后继移进栈
  - 规范的LR分析器在报告错误之前决不做任何无效归约
  - SLR和LALR分析器在报告错误前有可能执行几步归约
- 动作表的出错条目存放相应的错误处理程序的入口

LR分析的错误恢复策略

- **紧急方式恢复：**
  1. 从栈顶开始退栈，可能弹出0个或若干个状态，直到出现状态S为止，根据S在goto子表中可以找到一个非终结符号A，即它有关于A的转移；
  2. 抛弃0个或若干个输入符号，直到找到符号a为止，a∈FOLLOW(A)，即a可以合法地跟在A的后面；
  3. 分析器把状态goto[S,A]压入栈，继续进行语法分析。
  - 在弹栈过程中出现的A可能不止一个，通常选择表示较大程序结构的非终结符号，如表达式、语句或程序块。例如，若A是语句的非终结符，那么a可以是分号或*end*。
  - 实际上是跳过包含错误的一部分终结符号串。
- **短语级恢复：**
  - 对剩余输入作局部纠正，用可以使分析器继续分析的串来代替剩余输入的前缀
  - 尽量避免从分析栈中弹出与非终结符有关的状态，因为归约出的非终结符都是分析成功的

例（短语

于+，运

期待输入符号为运算符号或右括号，而遇到的却是运算对象（id或左括号）。

诊断信息：“缺少运算符号”

恢复策略：把运算符号‘+’压入栈，转移到状态4。

状态						goto
	id			(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1		
8							
9							

期待输入符号为运算符号或右括号，而遇到的却是输入串结束标志‘$’。

诊断信息：“缺少右括号”

恢复策略：把右括号压入栈，转移到状态9。

e1：在状态0、2、4、5，期待输入符号为运算对象的首字符，即id或(，而输入中出现的却是运算符号‘+’或‘*’，或是输入串结束标志‘$’。

- 策略：把一个假想的id压入栈，并将状态3推入栈顶。
- 诊断信息：“缺少运算对象”

e2：在状态0、1、2、4、5，期待输入符号为运算对象的首字符或运算符号，但却遇到右括号。

- 策略：删掉输入的右括号
- 诊断信息：“括号不匹配”

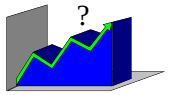
e3：在状态1、6，期待输入符号为运算符号或右括号，而遇到的却是运算对象（id或左括号）。

- 策略：把运算符号‘+’压入栈，转移到状态4。
- 诊断信息：“缺少运算符号”

e4：在状态6，期待输入符号为运算符号或右括号，而遇到的却是输入串结束标志‘$’。

- 策略：把右括号压入栈，转移到状态9。
- 诊断信息：“缺少右括号”

示例：分析符号串id+)



步骤	栈	输入	分析动作
(1)	0	id+)$	shift
	-		
(2)	0 3	+)$	reduce by E→id
	- id		
(3)	0 1	+)$	shift
	- E		
(4)	0 1 4	)$	CALL e2 “括号不匹配”，删掉 ‘)’
	- E +		
(5)	0 1 4	$	CALL e1 “缺少运算对象”，id压入栈
	- E +		
(6)	0 1 4 3	$	reduce by E→id
	- E + id		
(7)	0 1 4 7	$	reduce by E→E+E
	- E + E		
(8)	0 1	$	accept
	- E		

4.5 软件工具YACC

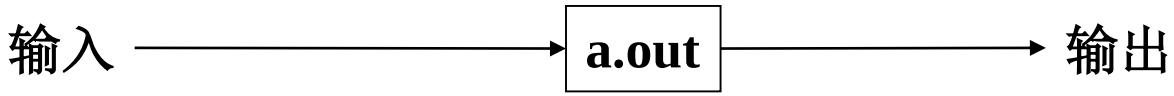
- 语法分析程序自动生成工具
  YACC：Yet Another Compiler-Compiler

一、YACC使用方式
二、YACC源程序结构
三、YAC处理二义文法的处理
四、用LEX建立YACC的词法分析程序
五、YACC内部名称

一、YACC 使用方式


```
%yacc   translate.y
```


```
%cc  y.tab.c  -ly  -o   a.out
```



二、YACC源程序结构

- 由说明部分、翻译规则和辅助过程三部分组成，各部分之间用双百分号分隔。

说明部分：有任选的两节

- 第一节是处于%{和%}之间的部分，在这里是一些普通的C语言的声明。
- 第二节是文法记号的声明，一般
  - 以 %start S 的形式说明文法的开始符号。
  - 用 %token IF、DO、ID、... 的形式说明记号。
  - 记号被YACC赋予了不会与任何字符值冲突的数字值。

翻译规则部分

每条规则由一个产生式和有关的语义动作组成

- 产生式$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$，在YACC说明文件中写成

```
A : α1 { 语义动作1 }
  | α2 { 语义动作2 }
  ...
  | αn { 语义动作n }
  ;
```

- 用单引号括起来的单个字符，如‘c’，是由终结符号c组成的记号
- 没有用引号括起来、也没有被说明成token类型的字母数字串是非终结符号
- 语义动作是用C语言描述的语句序列
  - ‘$$’表示和产生式左部非终结符号相关的属性值，‘$i’表示和产生式右部第i个文法符号相关的属性值。

辅助过程部分

用C语言书写一些语义动作中用到的辅助程序

- 名字为yylex()的词法分析程序必须提供
- 函数main

```
main()
{
  return yyparse();
}
```

- YACC生成的yyparse过程调用一个词法分析程序yylex
- yyparse 每次调用 yylex() 时，得到一个二元式记号：<记号，属性值>。
  - 返回的记号必须事先在YACC说明文件的第一部分中用%token说明
  - 属性值必须通过YACC定义的变量yylval传给分析程序

```
%{
#include  <ctype.h>
%}
%token  DIGIT
%%
line     :   expr  '\n'          {printf("%d\n",  $1); }
         ;
expr     :   expr '+' term       {$$=$1 + $3 ; }
         |   term                {$$=$1; }
         ;
term     :   term '*' factor     {$$=$1 * $3 ; }
         |   factor              {$$=$1; }
         ;
factor   :   '(' expr  ')'        {$$=$2 ; }
         |   DIGIT               {$$=$1; }
         ;
%%
yylex( )  {
      int  c;
      c = getchar( );
      if  (isdigit(c))  {
          yylval = c-'0' ;
          return  DIGIT;
       }
       return  c;
}
```

三、YACC对二义文法的处理

- 处理冲突的两条缺省规则
  - “归约-归约”冲突，选择排在前面的产生式进行归约
  - “移进-归约”冲突，选择执行移进动作
- 处理移进-归约冲突的机制
  - 利用 %left '+' '-' 说明'+'和'-'具有同样的优先级，并且遵从左结合规则。
  - 利用 %right '↑' 声名算符 '↑' 遵从右结合规则。
  - 利用 %nonassoc '<' '>'说明某些二元运算符不具有结合性。
  - 先出现的记号的优先级低
  - 同一声明中的记号具有相同的优先级

-产生式的优先级

- 和它最右边的终结符号的优先级一致。
- 最右终结符号不能给产生式以适当的优先级
  - 通过给产生式附加标记 %prec <ternimal> 来限制它的优先级
  - 它的优先级和结合性质同这个指定的终结符号的一样
  - 这个终结符号可以是一个占位符，不是由词法分析程序返回的记号，仅用来决定一个产生式的优先级。
- YACC不报告用这种优先级和结合性质能够解决的移进-归约冲突。

四、用LEX建立YACC的词法分析程序

- LEX编译器将提供词法分析程序yylex()
- 如果用LEX产生词法分析程序，则YACC说明文件中第三部分的函数yylex()应由语句

  #include "lex.yy.c" 代替。

- 使用这条语句，程序yylex()可以访问YACC中记号的名字，因为LEX的输出是YACC输出文件的一部分，所以每个LEX动作都返回YACC知道的终结符号。

五、YACC内部名称

YACC内部名称	说明
y.tab.c	YACC输出文件名
y.tab.h	YACC生成的头文件，包含有记号定义
yyparse	YACC分析程序
yylval	栈中当前记号的值
yyerror	由YACC使用的用户定义的错误信息打印程序
error	YACC错误伪记号
yyerrok	在错误处理之后，使分析程序回到正常操作方式的程序
yychar	变量，记录导致错误的先行记号
YYSTYPE	定义分析栈值类型的预处理器符号
yydebug	变量，当由用户设置为1时，生成有关分析动作的运行信息

小 结

一、自顶向下的分析方法

- 递归下降分析方法
  - 试探性、回溯
  - 要求：文法不含左递归
- 递归调用预测分析方法
  - 不带回溯的递归分析方法
  - 要求：文法不含左递归

    对任何产生式：$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$

    $FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$
  - 构造步骤：描述结构的上下文无关文法

    根据文法构造预测分析程序的状态转换图

    状态转换图化简

    根据状态转换图构造递归过程

非递归预测分析方法

- 不带回溯、不含递归
- 模型：

输入缓冲区：存放输入符号串$a_1a_2...a_n\$$

符号栈：分析过程中存放文法符号

分析表：二维表，每个A有一行，每个a包括$有一列

表项内容是产生式（关键）

控制程序：根据栈顶X和当前输入a决定分析动作（永恒的核心）

$X=a=\$$　分析成功

$X=a\neq\$$　弹出X，扫描指针前移

X是非终结符号，查分析表：M[X,a]

$M[X,a]=X\rightarrow Y_1Y_2...Y_K$，弹出X，$Y_K$、...、$Y_2$、$Y_1$入栈

$M[X,a]= X\rightarrow\varepsilon$，弹出X

M[X,a]=空白，出错处理

输出：对输入符号串进行最左推导所用的产生式序列

- 预测分析表的构造

  构造每个文法符号的FIRST集合

  构造每个非终结符号的FOLLOW集合

  检查每个产生式$A\rightarrow\alpha$

  对任何$a\in FIRST(\alpha)$，$M[A,a]= A\rightarrow\alpha$

  若$\alpha\Rightarrow\varepsilon$，对所有$b\in FOLLOW(A)$，$M[A,b]= A\rightarrow\alpha$

- LL(1)文法
  - LL(1)的含义
  - 判断一个文法是否为LL(1) 文法

    构造分析表，或者

    检查每个产生式：$A\rightarrow\alpha|\beta$

    $FIRST(\alpha)\cap FIRST(\beta)=\phi$

    若$\beta\Rightarrow\varepsilon$，则$FIRST(\alpha)\cap FOLLOW(A)=\phi$

二、自底向上分析方法

- 移进-归约分析方法
  - 分析栈、输入缓冲区
  - 可归约串
  - 规范归约：最右推导的逆过程
- LR分析方法
  - 模型：

    输入缓冲器：　　输出：分析动作序列

    分析栈：$S_0X_1S_1...XnSn$

    分析表：包括action和goto两部分（关键）

    控制程序：根据栈顶状态S_n和当前输入符号a查分析表

    action[S_n,a]，决定分析动作（永恒的核心）

    action[S_n,a]=s i，a入栈，i入栈 i=goto(S_n,a)

    action[S_n,a]=r A→β，弹出2*|β|个符号，

    A入栈，goto(S_n-r,A)入栈

    action[S_n,a]=acc，分析成功

    action[S_n,a]=空白，出错处理

SLR(1)分析表的构造

- LR(0)项目集规范族
- 识别文法所有活前缀的DFA
- 构造分析表：检查每个状态集

若$A\to\alpha\cdot a\beta \in I_i$，且$go(I_i,a)=I_j$，则置action[i,a]=sj，

若$A\to\alpha\cdot \in I_i$，则对所有$a\in FOLLOW(A)$，置action[i,a]=r $A\to\alpha$

若$S'\to S\cdot \in I_i$，则置action[i,$]=acc，表示分析成功。

若$go(I_i,A)=I_j$，A为非终结符号，则置goto[i,A]=j

LR(1)分析表的构造

- LR(1)项目集规范族
- 构造分析表：检查每个状态集

若$[A\to\alpha\cdot a\beta\ ,\ b]\in I_i$，且$go(I_i,a)=I_j$，则置action[i,a]=sj，

若$[A\to\alpha\cdot,\ a] \in I_i$，则置action[i,a]=r $A\to\alpha$，

若$[S'\to S\cdot,\ \$] \in I_i$，则置action[i,$]=acc，表示分析成功。

若$go(I_i,A)=I_j$，A为非终结符号，则置goto[i,A]=j

LALR(1)分析表的构造

- LR(1)项目集规范族，若没有冲突，继续
- 合并同心集，若没有冲突，则为LALR(1)项目集规范族
- 构造分析表，方法同LR(1)分析表的构造方法

利用LR技术处理二义文法

- 利用运算符号的优先级和结合性质，处理算术表达式文法
- 利用最近最后匹配原则，处理IF语句文法。

YACC的使用

- YACC源程序结构
- 冲突的缺省处理原则
- 二义性处理机制

算法清单

给定文法，构造 候选式/非终结符号 的 FIRST 集合
构造 非终结符号 的 FOLLOW 集合