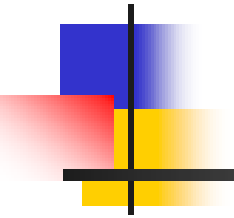


第五章 自底向上的语法分析

第5章 自底向上的语法分析

5.1 自底向上的语法分析概述

5.2 算符优先分析法

5.3 LR分析法

5.4 语法分析程序的自动生成工具Yacc

5.5 本章小结

5.1 自底向上的语法分析概述

- 思想
  - 从输入串出发，反复利用产生式进行归约，如果最后能得到文法的开始符号，则输入串是句子，否则输入串有语法错误。
- 核心
  - 寻找句型中的当前归约对象——“句柄”进行归约，用不同的方法寻找句柄，就可获得不同的分析方法

例5.1 一个简单的归约过程

- 设文法G为：

  S→aABe　A→Abc|b　B→d

句子分析：

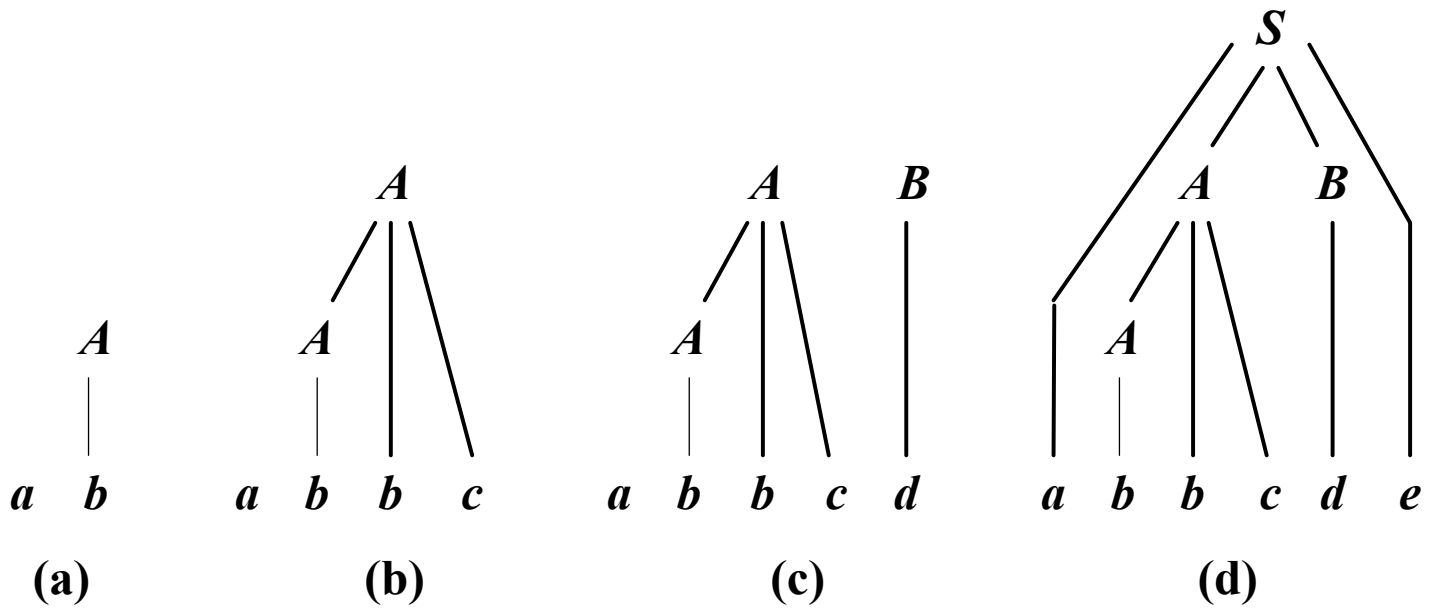
abbcde

⇐aAbcde

⇐aAde

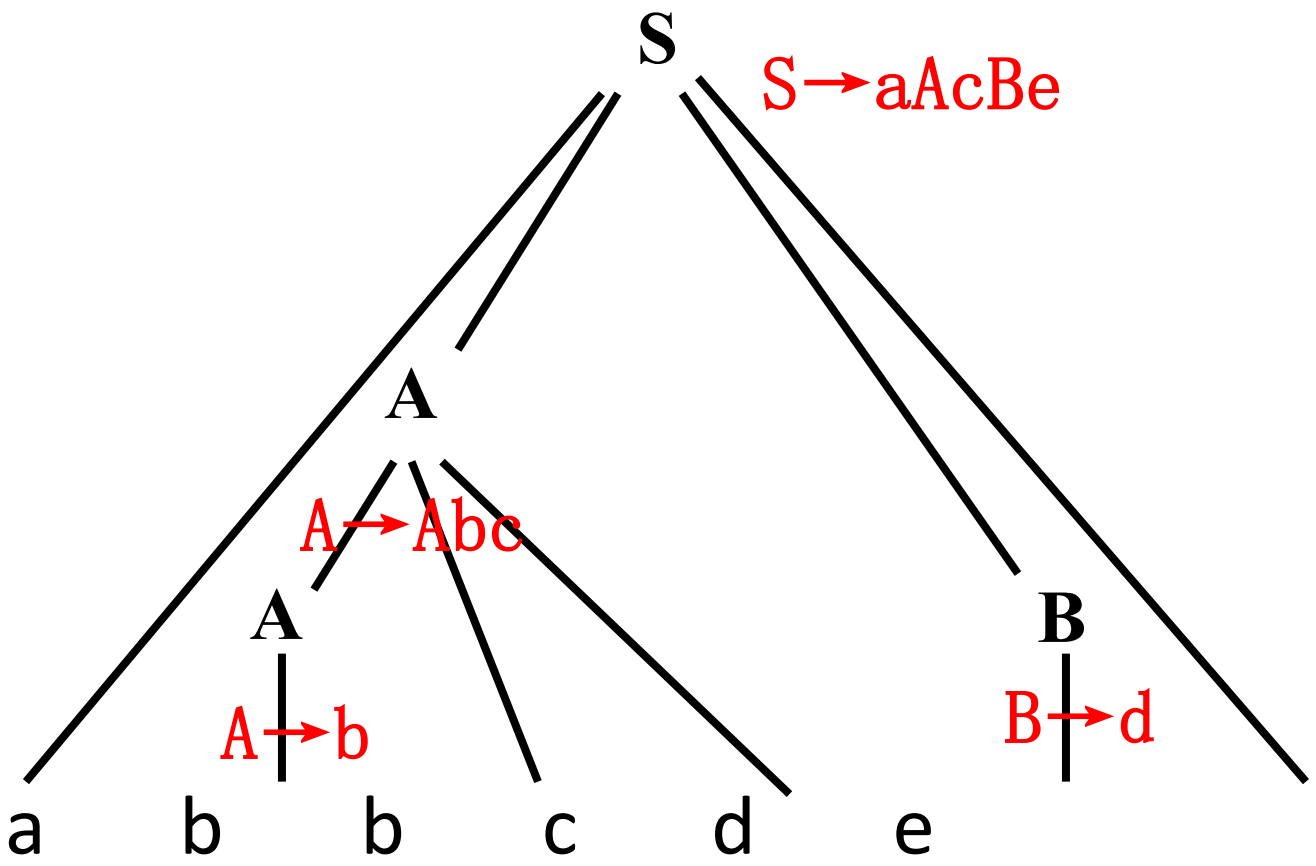
⇐aABe

⇐ S



语法树的形成过程

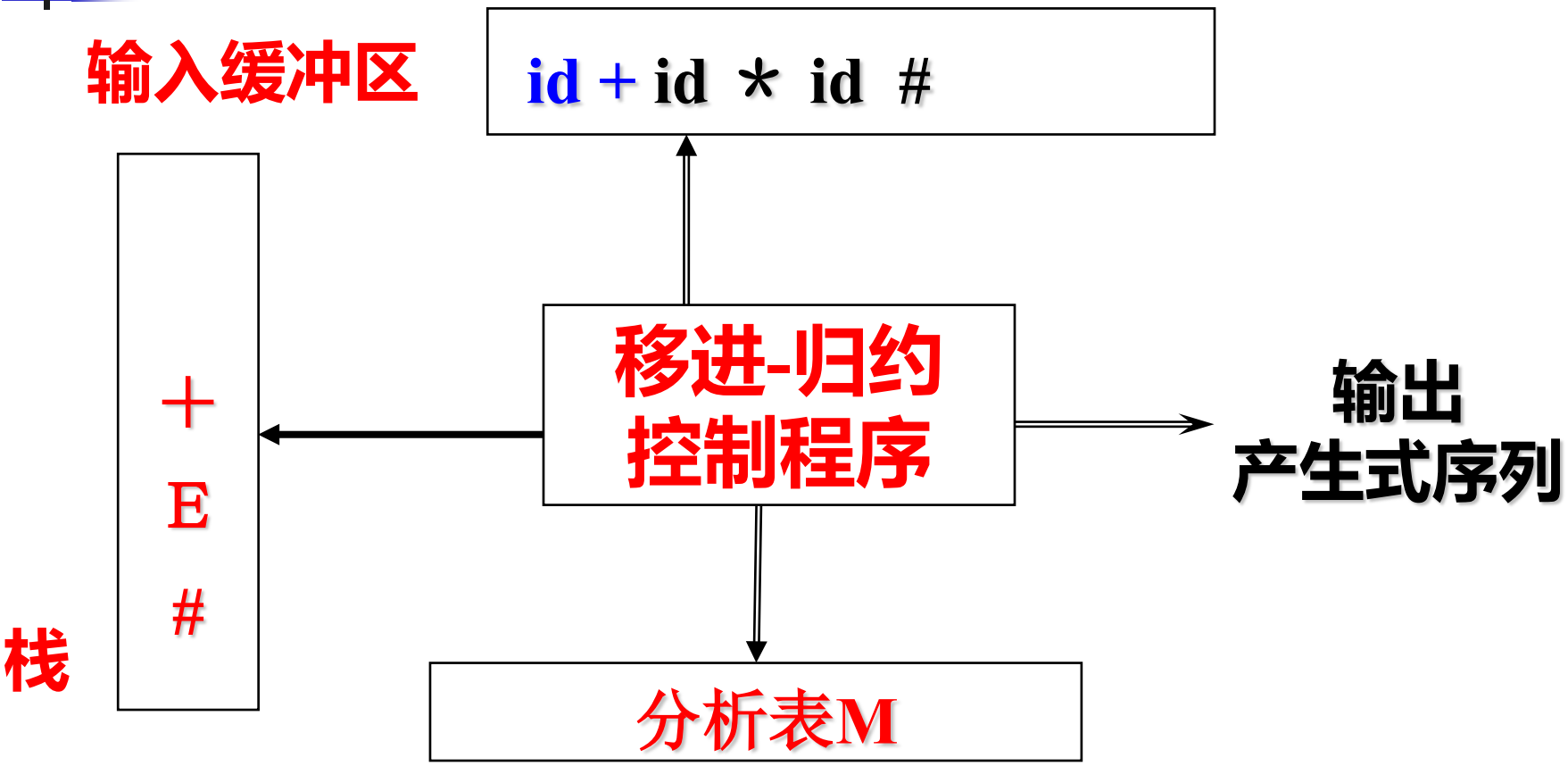
语法分析树的生成演示



5.1.1 移进-归约分析

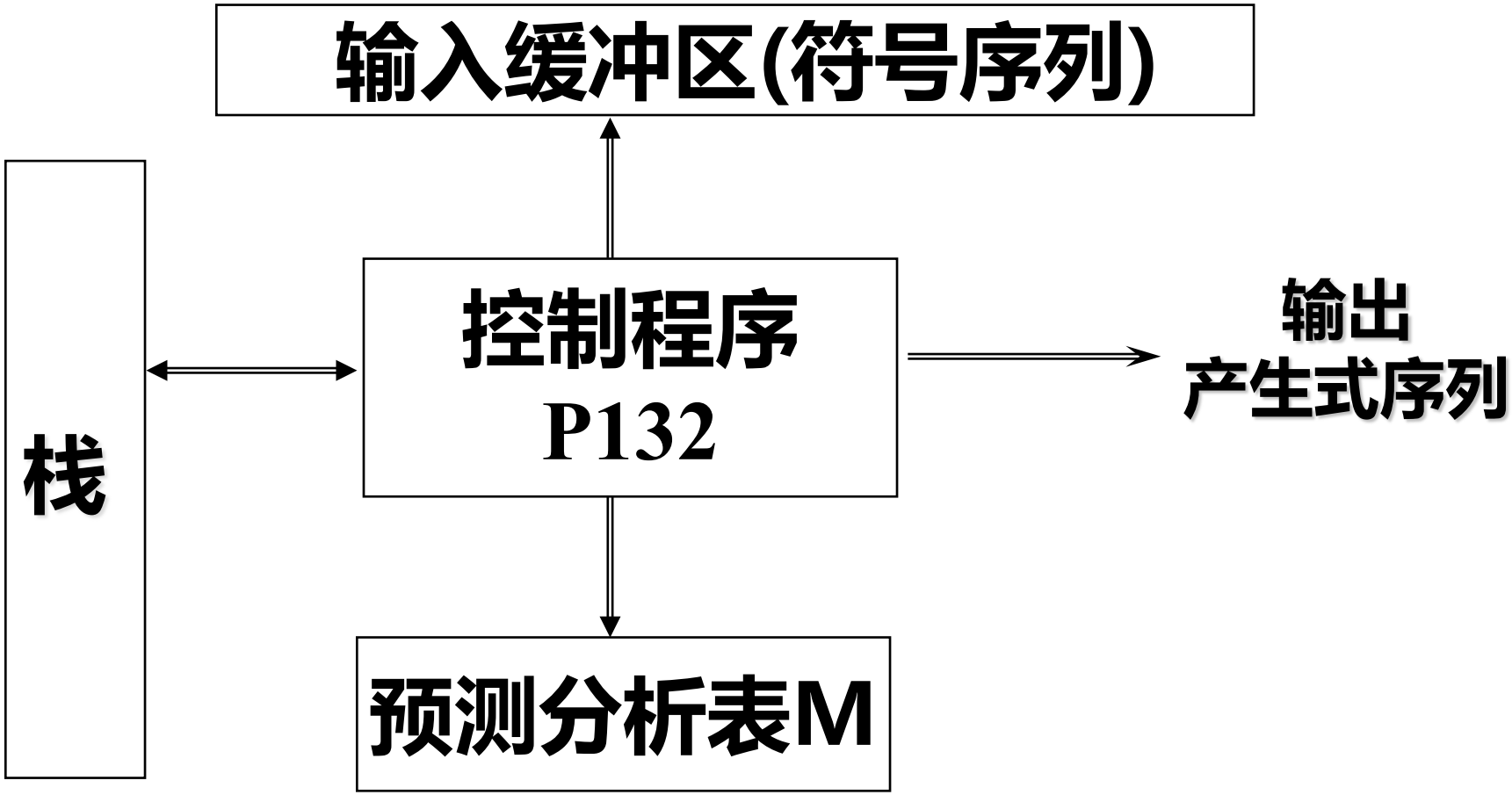
- **系统框架**
  - 采用表驱动的方式实现
  - 输入缓冲区：保存输入符号串
  - 分析栈：保存语法符号—已经得到的那部分分析结果
  - 控制程序：控制分析过程，输出分析结果——产生式序列
  - **格局：栈+输入缓冲区剩余内容="句型"**

移进-归约语法分析器的总体结构



栈内容+输入缓冲区内容 = #“当前句型” #

与LL(1)的体系结构比较



移进-归约分析的工作过程

- 系统运行
  - 开始格局
    - 栈：#；输入缓冲区：w#
  - 存放已经分析出来的结果,并将读入的符号送入栈，一旦句柄在栈顶形成，就将其弹出进行归约，并将结果压入栈
    - 问题：系统如何发现句柄在栈顶形成?
  - 正常结束：栈中为 #S，输入缓冲区只有 #

输出结果表示：用产生式序列表示语法分析树

例5.2 E→E+E|E*E|(E)|id

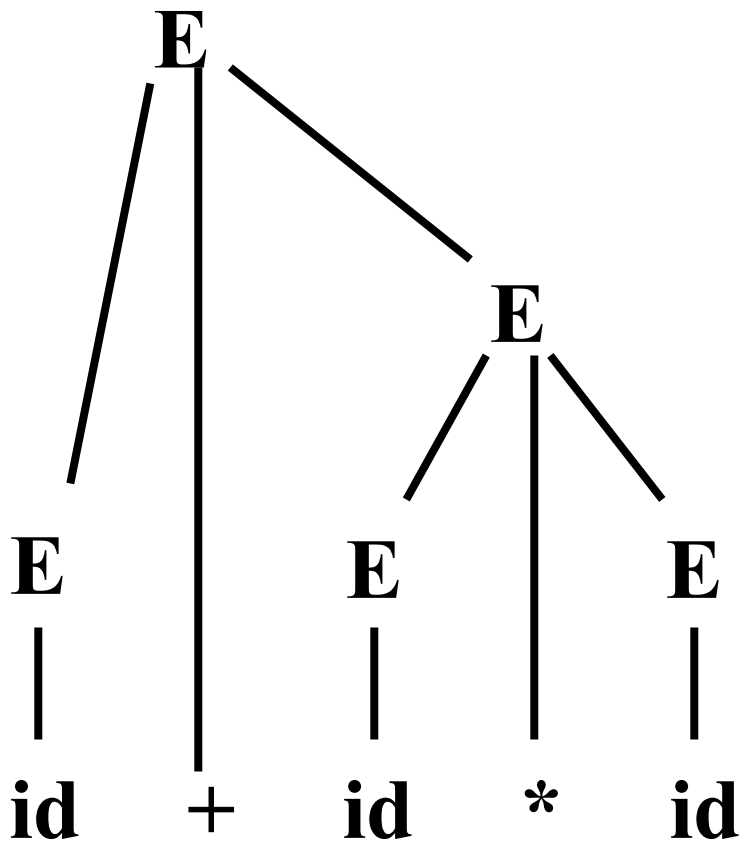
$E \rightarrow id$

$E \rightarrow id$

$E \rightarrow id$

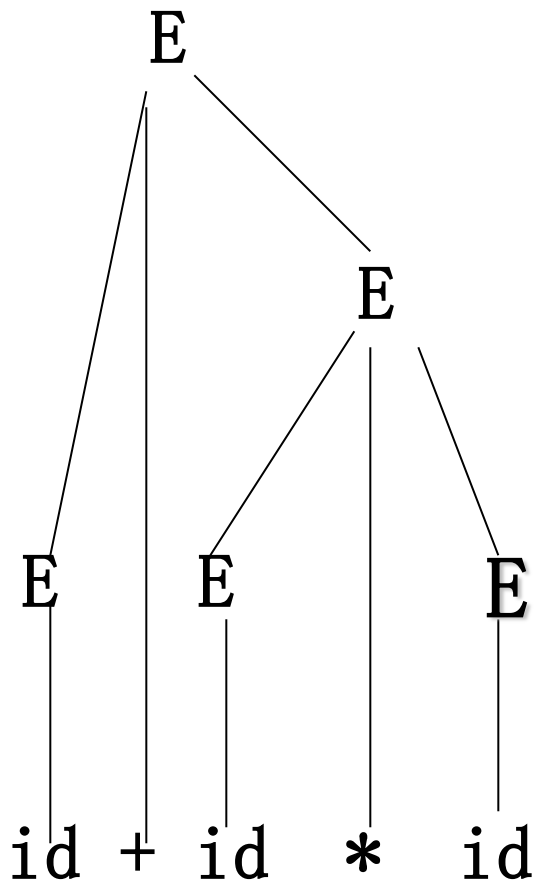
$E \rightarrow E * E$

$E \rightarrow E + E$



	动作		栈	输入缓冲区
1)			#	$id_1+id_2*id_3$#
2)	移进		#id_1	$+id_2*id_3$#
3)	归约	E→id	#E	$+id_2*id_3$#
4)	移进		#E+	id_2*id_3#
5)	移进		#E+id_2	$*id_3$#
6)	归约	E→id	#E+E	$*id_3$#
7)	移进		#E+E*	id_3#
8)	移进		#E+E*id_3	#
9)	归约	E→id	#E+E*E	#
10)	归约	E→E*E	#E+E	#
11)	归约	E→E+E	#E	#
12)	接受			

例5.2 分析过程



分析器的四种动作

1) 移进：将下一输入符号移入栈

2) 归约：用产生式左侧的非终结符替换栈顶的句柄（某产生式右部）

3) 接受：分析成功

4) 出错：出错处理

？？决定移进和归约的依据是什么

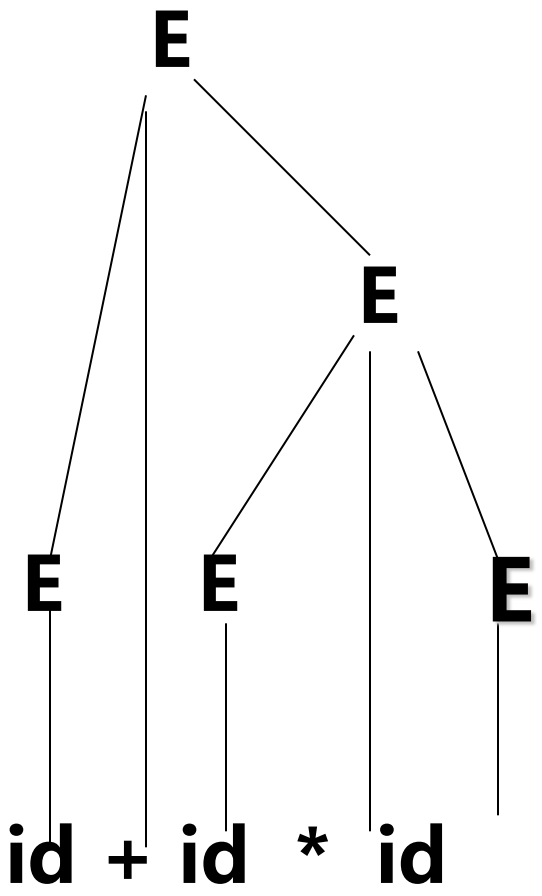
移进-归约分析中的问题

- 1) 移进归约冲突
  - 例5.2中的 6)可以移进 * 或按产生式 E→E+E归约

动作　　栈　输入缓冲区

1) # $id_1+id_2*id_3$#
2) 移进 #id_1 $+id_2*id_3$#
3) 归约 E→id #E $+id_2*id_3$#
4) 移进 #E+ id_2*id_3#
5) 移进 #E+id_2 $*id_3$#
6) 归约 E→id #E+E $*id_3$#
7) 移进 #E+E* id_3#
8) 移进 #E+E*id_3 #
9) 归约 E→id #E+E*E #
10) 归约 E→E*E #E+E #
11) 归约 E→E+E #E #
12) 接受

例5.2分析过程



移进-归约分析中的问题

1) 移进归约冲突
   - 例5.2中的 6)可以移进 * 或按产生式 E→E+E归约
2) 归约归约冲突
   - 存在两个可用的产生式
- 各种分析方法处理冲突的方法不同
- 如何识别句柄?
   - 如何保证找到的直接短语是最左的?利用栈
   - 如何确定句柄的开始处与结束处?

5.1.2 优先法

- 根据归约的先后次序为句型中相邻的文法符号规定优先关系
  - 句柄内相邻符号同时归约，是同优先的
  - 句柄两端符号的优先级要高于句柄外与之相邻的符号
- $a_1 \ldots a_{i-1} \lessdot a_i \equiv a_{i+1} \equiv \ldots \equiv a_{j-1} \equiv a_j \gtrdot a_{j+1} \ldots a_n$
- 定义了这种优先关系之后，语法分析程序就可以通过$a_{i-1} \lessdot a_i$和$a_j \gtrdot a_{j+1}$这两个关系来确定句柄的头和尾了

5.1.3 状态法

- 根据句柄的识别状态（句柄是逐步形成的）
  - 用状态来描述不同时刻下形成的那部分句柄
  - 因为句柄是产生式的右部，可用产生式来表示句柄的不同识别状态
- 例如： S→bBB可分解为如下识别状态
  - S→.bBB　移进b
  - S→bB.B　等待归约出B
  - S→b.BB　等待归约出B
  - S→bBB.　归约
- 采用这种方法，语法分析程序根据当前的分析状态就可以确定句柄的头和尾，并进行正确的归约。

5.2 算符优先分析法

- 算术表达式分析的启示
- 算符优先关系的直观意义
  - + ⋖ *　　+ 的优先级低于 *
  - (≡)　　(的优先级等于)
  - + ⋗ +　　+ 的优先级高于 +
- 方法
  - 将句型中的终结符号当作“算符”，借助于算符之间的优先关系确定句柄

算术表达式文法的再分析

- E→E+E
- E→E-E
- E→E*E
- E→E/E
- E→(E)
- E→id

从如何去掉二义性,看对算符优先级的利用

句型的特征：

(E+E)*(E-E)/E/E+E*E*E

- E→E+T| E-T| T
  T→T*F| T/F| F
  F→(E)|id

算符文法

- 如果文法G=（V，T，P，S）中不存在形如

$$A \rightarrow \alpha BC\beta$$

的产生式，则称之为算符文法(OG —Operator Grammar)

即：如果文法 G 中不存在具有相邻非终结符的产生式，则称为算符文法。

5.2.1 算符优先文法

- 定义5.1 假设G是一个不含ε-产生式的文法，A、B和C均是G的语法变量，G的任何一对终结符a和b之间的**优先关系定义**为：

(1) $a \equiv b$，当且仅当文法G中含有形如$A \to \ldots ab \ldots$或$A \to \ldots aBb \ldots$的产生式；

(2) $a \lessdot b$，当且仅当文法G中含有形如$A \to \ldots aB \ldots$的产生式，而且$B \overset{+}{\Rightarrow} b \ldots$或$B \overset{+}{\Rightarrow} Cb \ldots$；

(3) $a \gtrdot b$，当且仅当文法G中含有形如$A \to \ldots Bb \ldots$的产生式，而且$B \overset{+}{\Rightarrow} \ldots a$或$B \overset{+}{\Rightarrow} \ldots aC$；

(4) a与b无关系，当且仅当a与b在G的任何句型中都不相邻。

- **问题：什么是算符优先文法?**

5.2.1 算符优先文法

- E→E+E
- E→E-E
- E→E*E
- E→E/E
- E→(E)
- E→id

+ 和 * 的优先关系?

由E→E+E 和 $E \overset{+}{\Rightarrow} E * E$

可得：+ ⋖ *

又由E→E*E 和 $E \overset{+}{\Rightarrow} E + E$

可得：+ ⋗ *

+ 和 * 的优先关系不唯一!

算术表达式文法的再分析

- E→E+T| E-T| T
  T→T*F| T/F| F
  F→(E)|id

+ 和 * 的优先关系?

由E→E+T 和 $E \overset{+}{\Rightarrow} T * F$

可得：+ ⋖ *

不能推出+ ⋗ *

所以+ 和 * 的优先关系唯一!

5.2.1 算符优先文法

- 设G=（V，T，P，S）为OG，如果$\forall$ a,b$\in V_T$, a$\equiv$b, a $\lessdot$b, a $\gtrdot$b 至多有一个成立，则称之为算符优先文法(OPG —Operator Precedence Grammar)

——在无ε产生式的算符文法G中，如果任意两个终结符之间至多有一种优先关系，则称为算符优先文法。

5.2.2 算符优先矩阵的构造

- 优先关系的确定
  - 根据优先关系的定义
    - $a \lessdot b \Leftrightarrow A \rightarrow \ldots aB \ldots \in P$且$(B \Rightarrow^{+} b \ldots$或者$B \Rightarrow^{+} Cb \ldots)$
    - 需要求出非终结符B派生出的第一个终结符集
    - $a \gtrdot b \Leftrightarrow A \rightarrow \ldots Bb \ldots \in P$且$(B \Rightarrow^{+} \ldots a$或者$B \Rightarrow^{+} \ldots aC)$
    - 需要求出非终结符B派生出的最后一个终结符集
- 设G=（V，T，P，S）为OG，则定义
  - $FIRSTOP(A)=\{b|A \Rightarrow^{+} b \ldots$或者$A \Rightarrow^{+} Bb \ldots, b \in T, B \in V\}$
  - $LASTOP(A)=\{b|A \Rightarrow^{+} \ldots b$或者$A \Rightarrow^{+} \ldots bB, b \in T, B \in V\}$

算符优先关系矩阵的构造

- A→…ab… ; A→…aBb…, 则a≡b
- A→…aB…,则对∀b∈FIRSTOP(B),a⋖b
- A→…Bb…,则对∀a∈LASTOP(B), a⋗b
- if A→B…∈P,then FIRSTOP(B)⊆ FIRSTOP(A)
- if A→…B∈P,then LASTOP(B)⊆ LASTOP(A)
- **问题：编程求FIRSTOP、LASTOP**

算符优先关系矩阵的构造

- $A \rightarrow X_1X_2 \ldots X_n$
  - ① 如果$X_iX_{i+1} \in TT$则：$X_i \equiv X_{i+1}$
  - ② 如果$X_iX_{i+1}X_{i+2} \in TVT$则：$X_i \equiv X_{i+2}$
  - ③ 如果$X_iX_{i+1} \in TV$则：$\forall a \in FIRSTOP(X_{i+1})$，$X_i \lessdot a$
  - ④ 如果$X_iX_{i+1} \in VT$则：$\forall a \in LASTOP(X_i)$，$a \gtrdot X_{i+1}$

算符优先关系矩阵的构造

- **练习：求解以为文法中变量的FIRSTOP和LASTOP集**

E→E+T| E-T| T　　T→T*F| T/F| F　　F→(E)|id

FIRSTOP（E）={+，-，*，/，(, id}
FIRSTOP（T）={*，/，(, id}
FIRSTOP（F）={(, id}

LASTOP(E)={+，-，*，/，), id}
LASTOP(T)={*，/，), id}
LASTOP(F)={), id}

问题：知道了FIRSTOP，LASTOP，如何求所有终结符的优先关系?

例 5.6 表达式文法的算符优先关系

	+	−	*	/	(	)	id
+	⋗	⋗	⋖	⋖	⋖	⋗	⋖
−	⋗	⋗	⋖	⋖	⋖	⋗	⋖
*	⋗	⋗	⋗	⋗	⋖	⋗	⋖
/	⋗	⋗	⋗	⋗	⋖	⋗	⋖
(	⋖	⋖	⋖	⋖	⋖	≡	⋖
)	⋗	⋗	⋗	⋗		⋗	
id	⋗	⋗	⋗	⋗		⋗	

5.2.3 算符优先分析算法

- 原理
  - 识别句柄并归约
  - 各种优先关系存放在算符优先分析表中
  - 利用$\gtrdot$识别句柄尾，利用$\lessdot$识别句柄头，分析栈存放已识别部分，比较栈顶和下一输入符号的关系，如果是句柄尾，则沿栈顶向下寻找句柄头，找到后弹出句柄，归约为非终结符。

例5.7 E→E+T|E-T|T　T→T*F|T/F|F　F→(E)|id，试利用算符优先分析法对id+id进行分析

步骤	栈	输入串	优先关系	动作
1	#	id_1+id_2#		
2	# id_1	$+id_2$#	#$\lessdot id_1$	移进id_1
3	#F	$+id_2$#	#$\lessdot id_1 \gtrdot$+	用$F\rightarrow$id归约
4	#F+	id_2#	$\lessdot$	移进+
5	#F+ id_2	#	$\lessdot$	移进id_2
6	#F+F	#	+$\lessdot id_2 \gtrdot$#	用$F\rightarrow$id归约
7	#E	#	#$\lessdot$+$\gtrdot$#	用$E\rightarrow E+T$归约

练习

- 已知文法G为：

S→SaF|F

F→FbP|P

P→c|d

- (1)求优先关系矩阵
- (2)利用上述优先关系矩阵分析符号串cadbc

问题

- 有时未归约真正的句柄（F）
- 不是严格的最左归约
- 归约的符号串有时与产生式右部不同
- 仍能正确识别句子的原因
  - OPG未定义非终结符之间的优先关系，不能识别由单非终结符组成的句柄
  - 定义算符优先分析过程识别的“句柄”为最左素短语LPP（Leftmost Prime Phase）

素短语与最左素短语

- 什么是短语？当前我们要找什么样的短语？——至少有一个算符
- $S \Rightarrow^{*} \alpha A \beta$ and $A \Rightarrow^{+} \gamma$，γ至少含一个终结符，且不含更小的含终结符的短语,则称γ是句型$\alpha\gamma\beta$的相对于变量A的素短语(Prime Phrase)
- 句型的至少含一个终结符且不含其它素短语的短语

例

- E→E+T|T T→T*F|F F→(E)|id

句型 T+T*F+i 的短语有

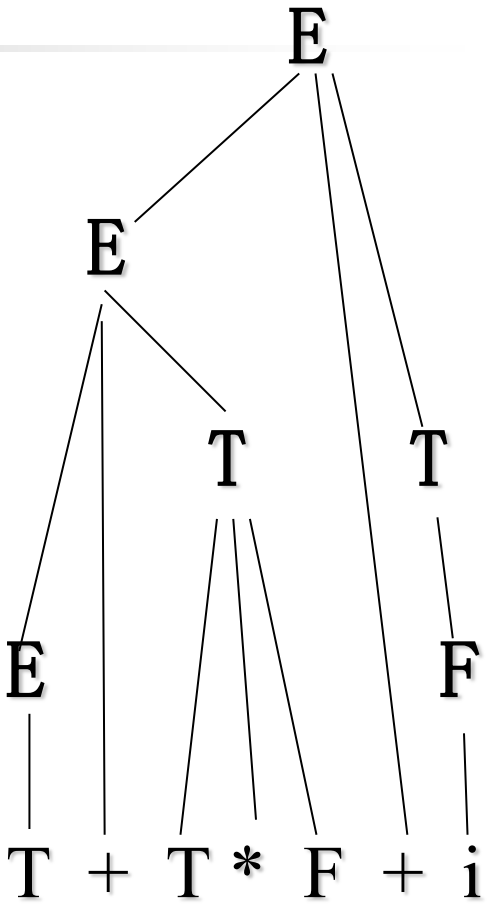
T T*F i T+T*F T+T*F+i

其中的素短语为

T*F i

T*F为最左素短语，是被归约的对象

问题：按照文法E→E+E|E*E|(E)|id，求i+E*i+i的短语和素短语



文法：E→E+E|E*E

E→(E)|id

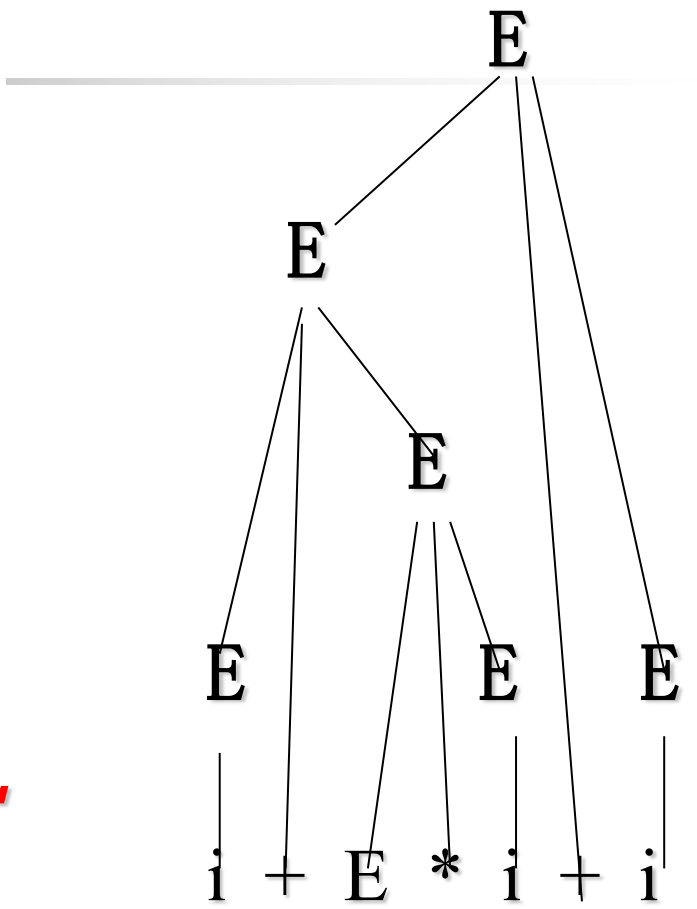
句型i+E*i+i的短语

i　i　E*i　i

i+E*i　i+E*i+i

其中的素短语为

i　i　i



问题：归约过程中如何发现“中间句型”的最左素短语?

素短语与最左素短语

设句型的一般形式为

$\#N_1a_1\,N_2a_2\ldots N_na_n\,\#\ (N_i\in V\cup\{\varepsilon\},a_i\in V_T)$

它的最左素短语是满足下列条件的最左子串

$$N_ia_i\,N_{i+1}a_{i+1}\ldots N_ja_j\,N_{j+1}$$

其中：$a_{i-1}\lessdot a_i$，

$a_i\equiv a_{i+1}\equiv\ldots\equiv a_{j-1}\equiv a_j$ ，

$a_j\gtrdot a_{j+1}$

算符优先分析的实现

- 系统组成
  - 移进归约分析器 + 优先关系表
- 分析算法
  - 参照输入串、优先关系表，完成一系列归约，生成语法分析树——输出产生式

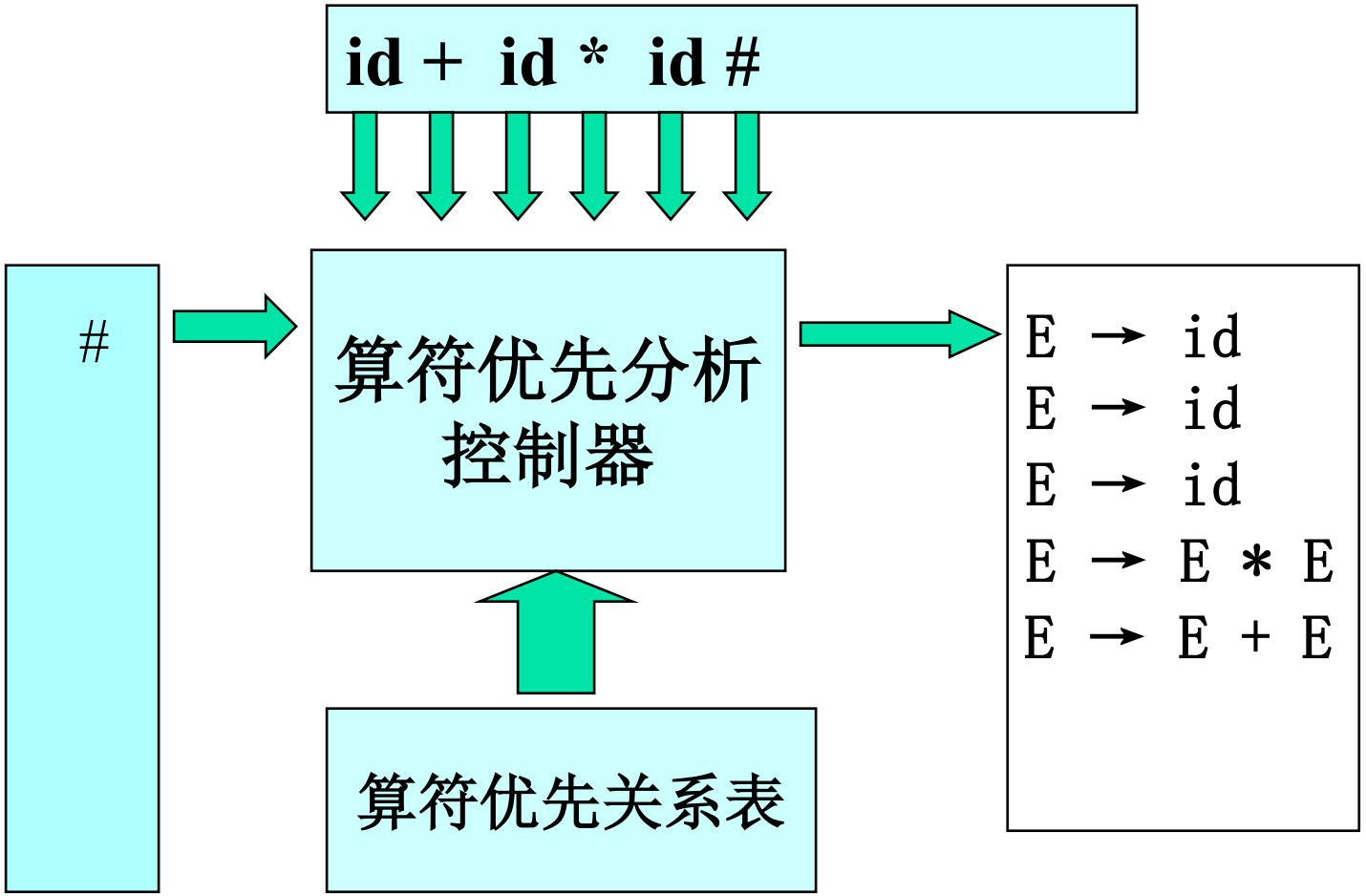
算符优先分析算法

算法5.3 算符优先分析算法。

输入： 文法$G=(V, T, P, S)$，输入字符串w和优先关系表;
输出： 如果w是一个句子则输出一个分析树架子，否则指出错误;
步骤：

```
begin  S[1]:='#';   i:=1;
       repeat 将下一输入符号读入R;
          if S[i]∈T then j:=i else j:=i-1;
          while S[j]≻R do  begin
                repeat   Q:=S[j];
                         if S[j-1]∈T then j:=j-1 else j:=j-2
                until S[j]≺Q;
                将S[j+1]…S[i]归约为N; i:=j+1;S[i]:=N end;
          if S[j]≺R or S[j]≡R then  begin i:=i+1; S[i]:=R end
          else error
       until i=2 and R='#'  end;
```

id+id*id 的分析过程



5.2.4 优先函数

- 为了节省存储空间（$n^2 \rightarrow 2n$）和便于执行比较运算，用两个优先函数f和g，它们是从终结符号到整数的映射。对于终结符号a和b选择f和g，使之满足：
  - $f(a)<g(b)$，如果$a \lessdot b$
  - $f(a)=g(b)$，如果$a \equiv b$
  - $f(a)>g(b)$，如果$a \gtrdot b$。
- 损失
  - 错误检测能力降低
    - 如：id $\gtrdot$id不存在，但f(id)>g(id)可比较

表5.2 对应的优先函数：

	+	-	*	/	(	)	id	#
f	2	2	4	4	0	4	4	0
g	1	1	3	3	5	0	5	0

1) 构造优先函数的算法不是唯一的。

2) 存在一组优先函数，那就存在无穷组优先函数。

优先函数的构造

算法5.4 优先函数的构造。

输入：算符优先矩阵;

输出：表示输入矩阵的优先函数，或指出其不存在;

步骤：

1. 对$\forall a \in T \cup \{\#\}$，建立以$f_a$和$g_a$为标记的顶点；
2. 对$\forall a, b \in T \cup \{\#\}$，若$a \succ b$或者$a \equiv b$，则从$f_a$至$g_b$画一条有向弧；若$a \prec b$或者$a \equiv b$，则从$g_b$至$f_a$画一条有向弧；
3. 如果构造的有向图中有环路，则说明不存在优先函数；如果没有环路，则对$\forall a \in T \cup \{\#\}$，将$f(a)$设为从$f_a$开始的最长路经的长度，将$g(a)$设为从$g_a$开始的最长路经的长度。

例5.10

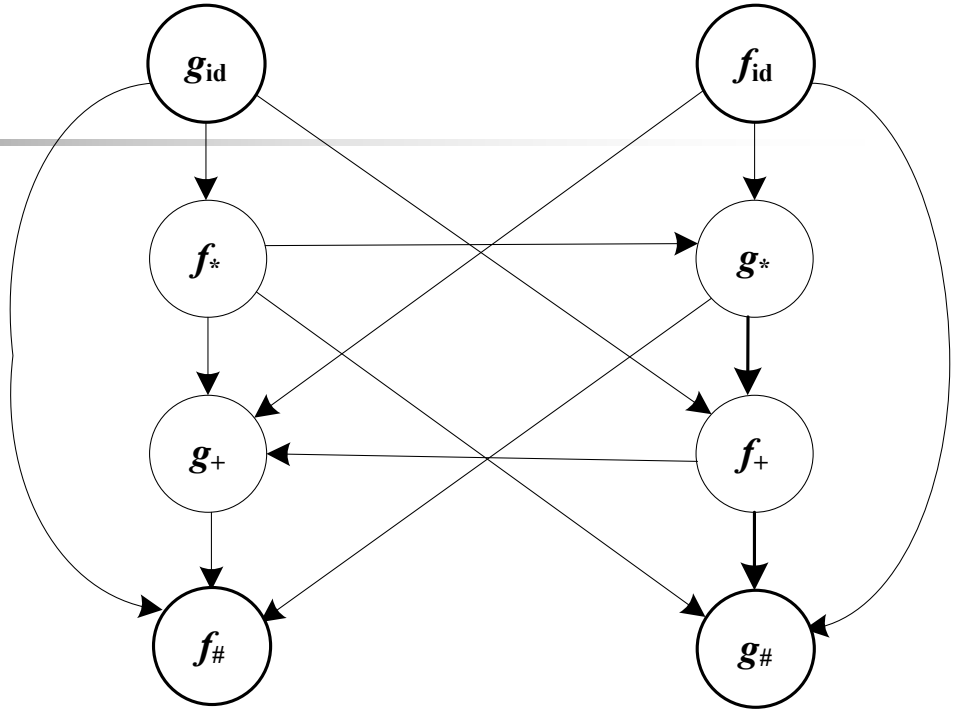
G_{es} :$E \rightarrow E+T|T$

$T \rightarrow T*F|F$

$F \rightarrow$ id

	+	*	id	#
+	⋗	⋖	⋖	⋗
*	⋗	⋗	⋖	⋗
id	⋗	⋗		⋗
#	⋖	⋖	⋖	

G_{es}的优先矩阵



	+	*	id	#
f	2	4	4	0
g	1	3	5	0

根据G_{es}的优先矩阵建立的有向图和优先函数

5.2.5 算符优先分析的出错处理

(1) 栈顶的终结符号和当前输入符号之间不存在任何优先关系；

(2) 发现被“归约对象”，但该“归约对象”不能满足要求。

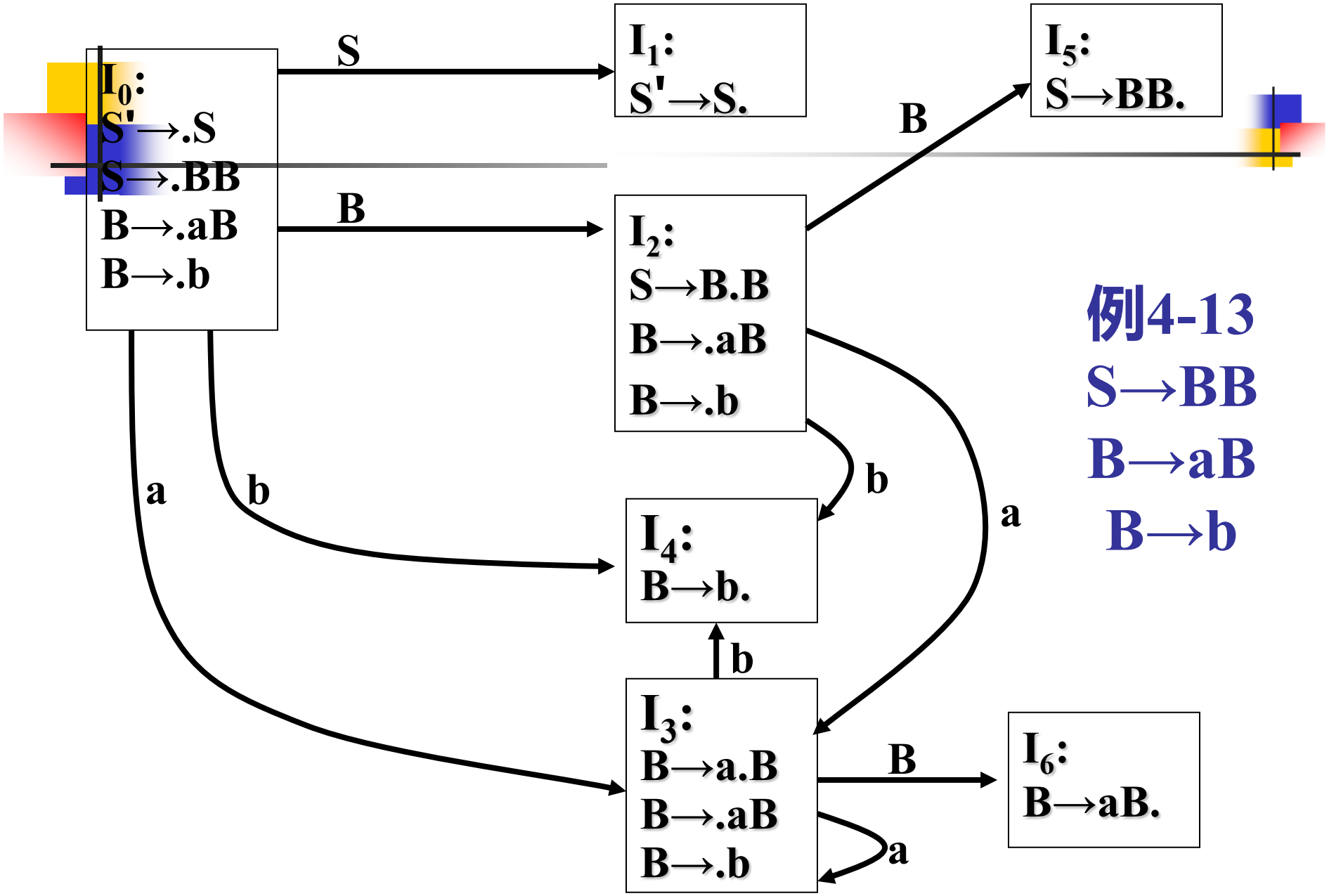
- 对于第(1)种情况，为了进行错误恢复，必须修改栈、输入或两者都修改。
- 对于优先矩阵中的每个空白项，必须指定一个出错处理程序，而且同一程序可用在多个地方。
- 对于第(2)种情况，由于找不到与“归约对象”匹配的产生式右部，分析器可以继续将这些符号弹出栈，而不执行任何语义动作。

算符优先分析法小结

- 优点
  - 简单、效率高
  - 能够处理部分二义性文法
- 缺点
  - 文法书写限制大——强调算符之间的优先关系的唯一性
  - 占用内存空间大
  - 不规范、存在查不到的语法错误
  - 算法在发现最左素短语的尾时，需要回头寻找对应的头

5.3 LR分析法　5.3.1 LR分析算法

- LR(k)分析法可分析LR(k)文法产生的语言
  - L： 从左到右扫描输入符号
  - R： 最右推导对应的最左归约
  - k： 超前读入k个符号，以便确定归约用的产生式
  - 使用语言的文法描述内涵解决句柄的识别问题，从语言的形式描述入手，为语法分析器的自动生成提供了前提和基础
- 分析器根据当前的状态，并至多向前查看k个输入符号，就可以确定是否找到了句柄，如果找到了句柄，则按相应的产生式归约，如果未找到句柄则移进输入符号，并进入相应的状态



例4-13

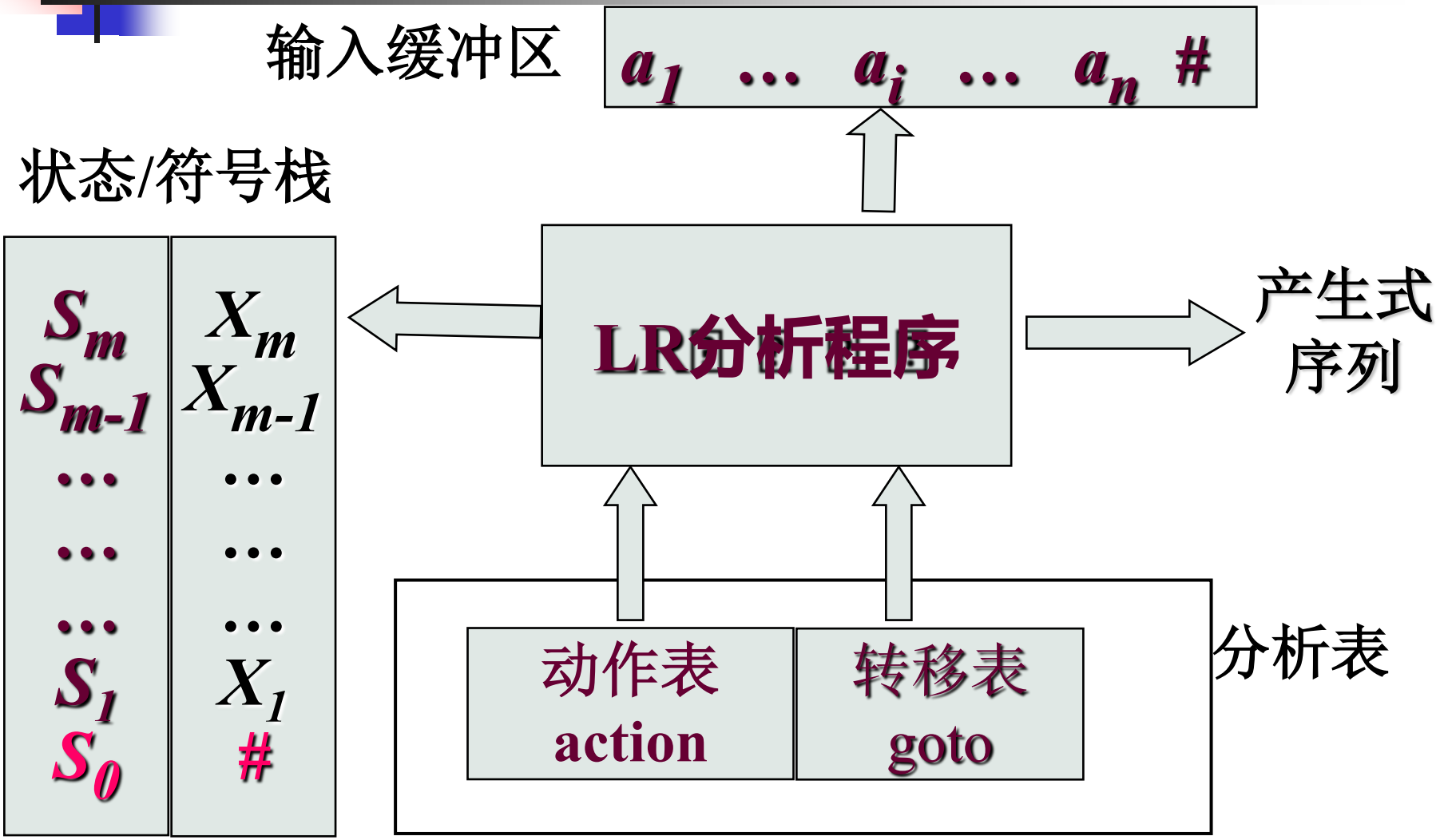
S→BB

B→aB

B→b

2023/8/17

LR语法分析器的总体结构



LR 分析表： action[s,a]； goto[s,X]

约定：
sn:将符号a、状态n压入栈
rn:用第n个产生式进行归约

LR(0)、SLR(1)、LR(1)、LALR(1)将以不同的原则构造这张分析表

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

LR分析器的工作过程

- 书上的下式（格局）

  $(s_0s_1\cdots s_m,\ X_1X_2\cdots X_m,\ a_ia_{i+1}\cdots a_n\#)$

- 在这里表示为

$s_0s_1\cdots s_m$
$\#X_1\cdots X_m$　　$a_ia_{i+1}\cdots a_n\#$

LR分析器的工作过程

- 1. 初始化

s_0
#　　$a_1a_2\ldots a_n$#　　对应“句型”　$a_1a_2\ldots a_n$

- 2. 在一般情况下，假设分析器的格局如下：

$s_0s_1\ldots s_m$
#$X_1\ldots X_m$　$a_ia_{i+1}\ldots a_n$#　　对应“句型”　$X_1\ldots X_ma_ia_{i+1}\ldots a_n$

- ①If　action[s_m,a_i]= si(shift i) then 格局变为

$s_0s_1\ldots s_m$　i
#$X_1\ldots X_ma_i$　$a_{i+1}\ldots a_n$#

$s_0 s_1 \cdots s_m$
$\#X_1 \cdots X_m \quad a_i a_{i+1} \cdots a_n \#$

②If action[s_m,a_i]= ri(Reduce i) then 表示用第i个产生式$A \rightarrow X_{m-(k-1)} \cdots X_m$进行归约，格局变为

$s_0 s_1 \cdots s_{m-k}$
$\#X_1 \cdots X_{m-k} A \quad a_i a_{i+1} \cdots a_n \#$

查goto表，如果goto[s_{m-k},A]=i then 格局变为

$s_0 s_1 \cdots s_{m-k}\ i$
$\#X_1 \cdots X_{m-k} A \quad a_i a_{i+1} \cdots a_n \#$

③If action[s_m,a_i]=acc then 分析成功

④If action[s_m,a_i]=err then 出现语法错

LR分析算法

算法5.5 *LR*分析算法。

输入：文法*G*的*LR*分析表和输入串*w*;
输出：如果*w*∈*L*(*G*)，则输出*w*的自底向上分析，否则报错;
步骤:

1. **将#和初始状态*S*0压入栈，将*w*#放入输入缓冲区;**
2. **令输入指针ip指向*w*#的第一个符号;**
3. **令*S*是栈顶状态，*a*是ip所指向的符号;**
4. **repeat**
5. **if *action*[*S*,*a*]=*Si* then　　/* *Si*表示移进*a*并转入状态*i**/**
6. **　　begin**
7. **　　把符号*a*和状态*i*先后压入栈;**
8. **　　令ip指向下一输入符号**
9. **　　end**

```
10.  elseif action[S,a]=rkthen  /* ri表示按第k个产生式A→β归约 */
11.      begin
12.          从栈顶弹出2*|β|个符号;
13.          令S'是现在的栈顶状态;
14.          把A和goto[S',A]先后压入栈中;
15.          输出产生式 A→β
16.      end
17.  elseif action[S,a]= acc then
18.      return
19.  else
20.      error();
```

例5.12

文法

1) S→BB
2) B→aB
3) B→b

分析表

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

栈 输入 动作说明 bab 的分析过程:

1) S→BB
2) B→aB
3) B→b

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

0
bab# action(0,b)=s4
04
#b ab# action(4,a)=r3
0
#B ab# goto(0,B)=2
02
#B ab# action(2,a)=s3
023
#Ba b# action(3,b)=s4
0234
#Bab # action(4,#)=r3
023
#BaB # goto(3,B)=6
0236
#BaB # action(6,#)=r2
02
#BB # goto(2,B)=5
025
#BB # action(5,#)=r1

0
#S # goto(0,S)=1
01
#S # action(1,#)=acc

规范句型活前缀

- 分析栈中内容+剩余输入符号=规范句型
  - 分析栈中内容为某一句型的前缀
- 来自分析栈的活前缀(Active Prefix)
  - 不含句柄右侧任意符号的规范句型的前缀
- 例：id + id * id 的分析中
  - 句型 E + id . * id 和 E + E * . id

$$S \Rightarrow^*_{rm} \alpha Aw \Rightarrow_{rm} \alpha\beta_1\beta_2 w$$

规范句型活前缀

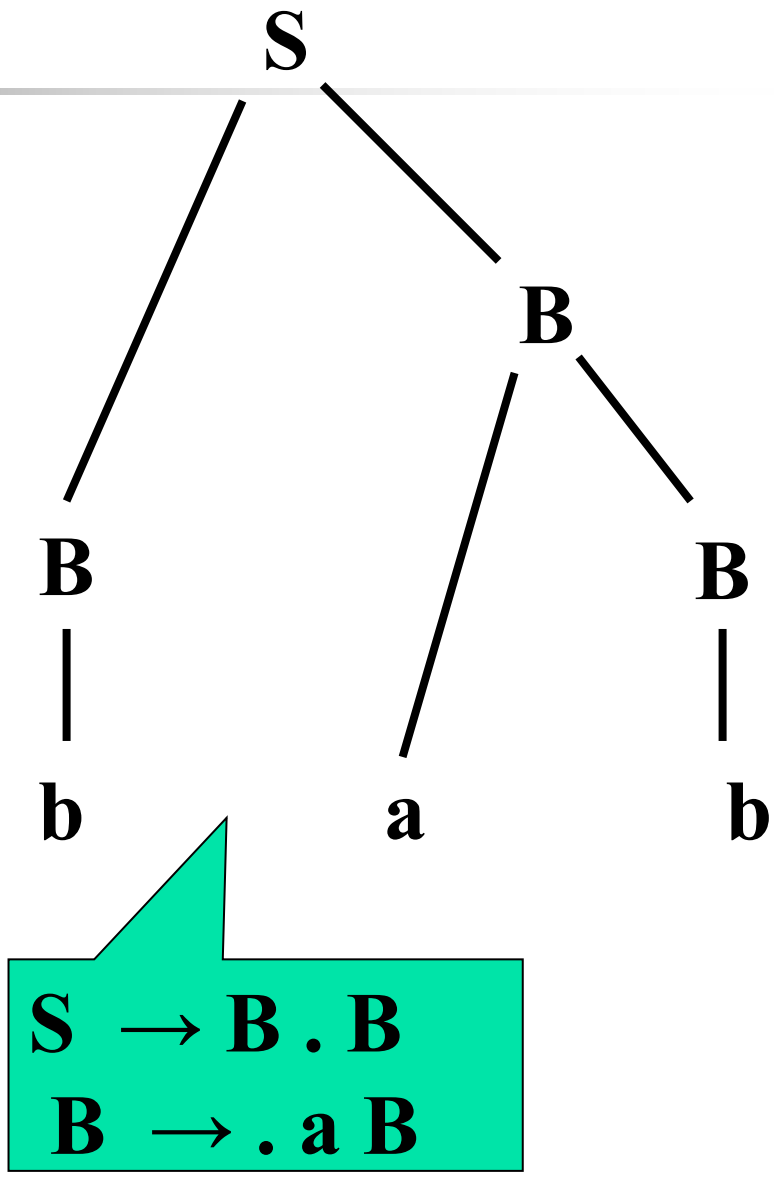
- 规范归约所得到的规范句型(Canonical Sentential Form)的活前缀是出现在分析栈中的符号串，所以，不会出现句柄之后的任何字符，而且相应的后缀正是输入串中还未处理的终结符号串。
- 活前缀与句柄的关系
  - 包含句柄 $A \rightarrow \beta.$
  - 包含句柄的部分符号$A \rightarrow \beta_1.\beta_2$
  - 不含句柄的任何符号$A \rightarrow .\beta$

5.3.2 LR(0)分析表的构造

- LR(0)项目——从产生式寻找归约方法
  - 右部某个位置标有园点的产生式称为相应文法的LR(0)项目（Item）
  - 例 S→.bBB　S→bB.B　S→b.BB S→bBB.
  - 归约（Reduce）项目: S→aBB.
  - 移进（Shift）项目：S→.bBB
  - 待约项目：S→b.BB　S→bB.B

项目的意义

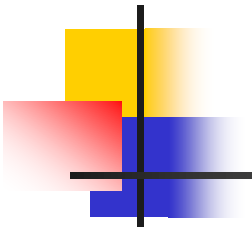
- 用项目表示分析的进程(句柄的识别状态)
- 方法：在产生式右部加一园点以分割已获取的内容和待获取的内容：构成句柄



拓广(Augmented)文法

- 需要一个对“归约成S” 的表示（只有一个接受状态）
- 文法 G= (V, T, P, S)的拓广文法 G'：
  - G'=（V ∪ {S'}，T，P ∪{S'→S}，S'）
  - S'∉V
  - 对应S'→.S（分析开始）和S'→S.（分析成功）
- 例5.13

  0) S'→S
  1) S→BB
  2) B→aB
  3) B→b



- **问题：如何设计能够指导分析器运行，并且能够根据当前状态（栈顶）确定句柄——归约对象的头——的装置**

构造识别G的所有规范句型活前缀的DFA

项目集闭包的计算

项目集 I 的闭包（Closure）

$$CLOSURE(I)=I \cup \{B\rightarrow .\gamma \mid A\rightarrow \alpha .B\beta \in I,\ B\rightarrow \gamma \in P\}$$

算法

J:=I;

repeat

$$J=J\cup \{B\rightarrow .\eta \mid A\rightarrow \alpha .B\beta \in J,\ B\rightarrow \eta \in P\}$$

until J不再扩大

闭包之间的转移

- 后继项目（Successive Item）
  - A→α.Xβ的后继项目是A→αX.β
- 闭包之间的转移
  - go(I,X)=CLOSURE({A→αX.β| A→α.Xβ∈I}

状态转移的计算

- 确定在某状态遇到一个文法符号后的状态转移目标

```
function GO(I, X);
begin
    J:=∅;
for I中每个形如A→α.Xβ的项目do
    begin  J:=J∪{A→αX.β}  end;
    return CLOSURE(J)
end;
```

识别拓广文法所有规范句型活前缀的DFA

- 识别文法G=（V，T，P，S）的拓广文法G'的所有规范句型活前缀的DFA：

$$M=(C, V\cup T, go, I_0, C)$$

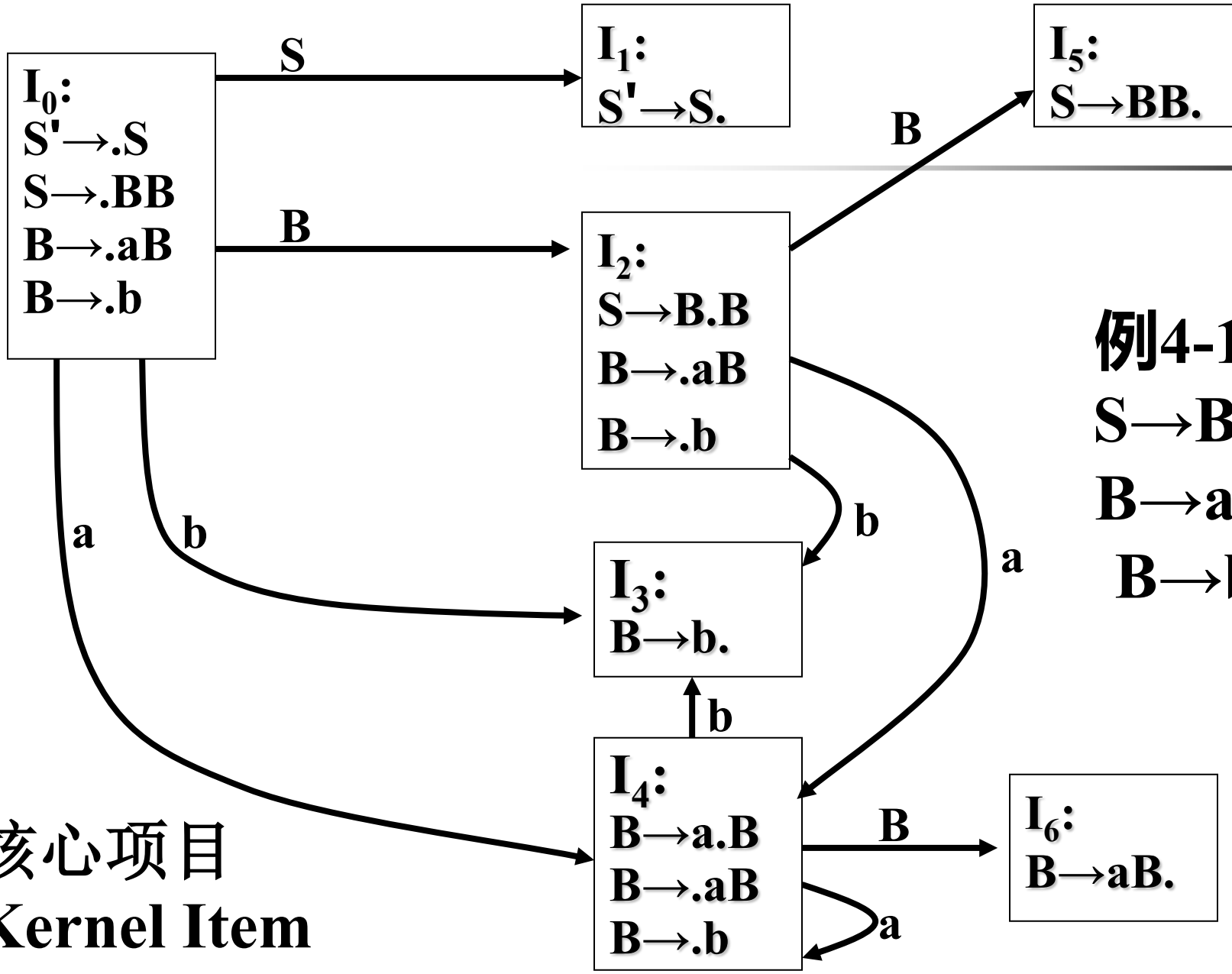
  - $I_0=CLOSURE(\{S' \rightarrow .S\})$
  - $C=\{I_0\}\cup\{I|\exists J\in C, X\in V\cup T, I=go(J,X)\}$

称为G'的**LR(0)项目集规范族**(Canonical Collection)

计算LR(0)项目集规范族C

即：分析器状态集合

```
begin
C := {closure({ S'→.S})};
  repeat
     for ∀I∈C，∀X∈V∪T
        if go(I,X)≠Φ & go(I,X)∉C then
                        C=C∪{go(I,X)}
  until C不变化
end.
```



例4-13
S→BB
B→aB
B→b

核心项目
Kernel Item

LR(0)分析表的构造算法

算法5.6 *LR*(0)分析表的构造。

输入： 文法$G=(V, T, P, S)$的拓广文法G';

输出： G'的*LR*(0)分析表，即*action*表和*goto*表;

步骤：

1.令I_0= CLOSURE($\{S' \rightarrow .S\}$)，构造G'的*LR*(0)项目集规范族$C=\{I_0,I_1,\ldots,I_n\}$

2．让I_i对应状态i，I_0对应状态0，0为初始状态。

3．for k=0 to n do begin

(1) if $A\rightarrow\alpha.a\beta\in I_k$ & $a\in T$ & GO(I_k, a)=I_j then *action*[k,a]:=Sj;

(2) if $A\rightarrow\alpha.B\beta\in I_k$ & $B\in V$ & GO(I_k, B)=I_j then *goto*[k,B]:=j;

(3) if $A\rightarrow\alpha.\in I_k$ & $A\rightarrow\alpha$为G的第j个产生式then

for $\forall a\in T\cup\{\#\}$ do *action*[k,a]:=rj;

(4) if $S'\rightarrow S.\in I_k$ then *action*[k,#]:=acc end;

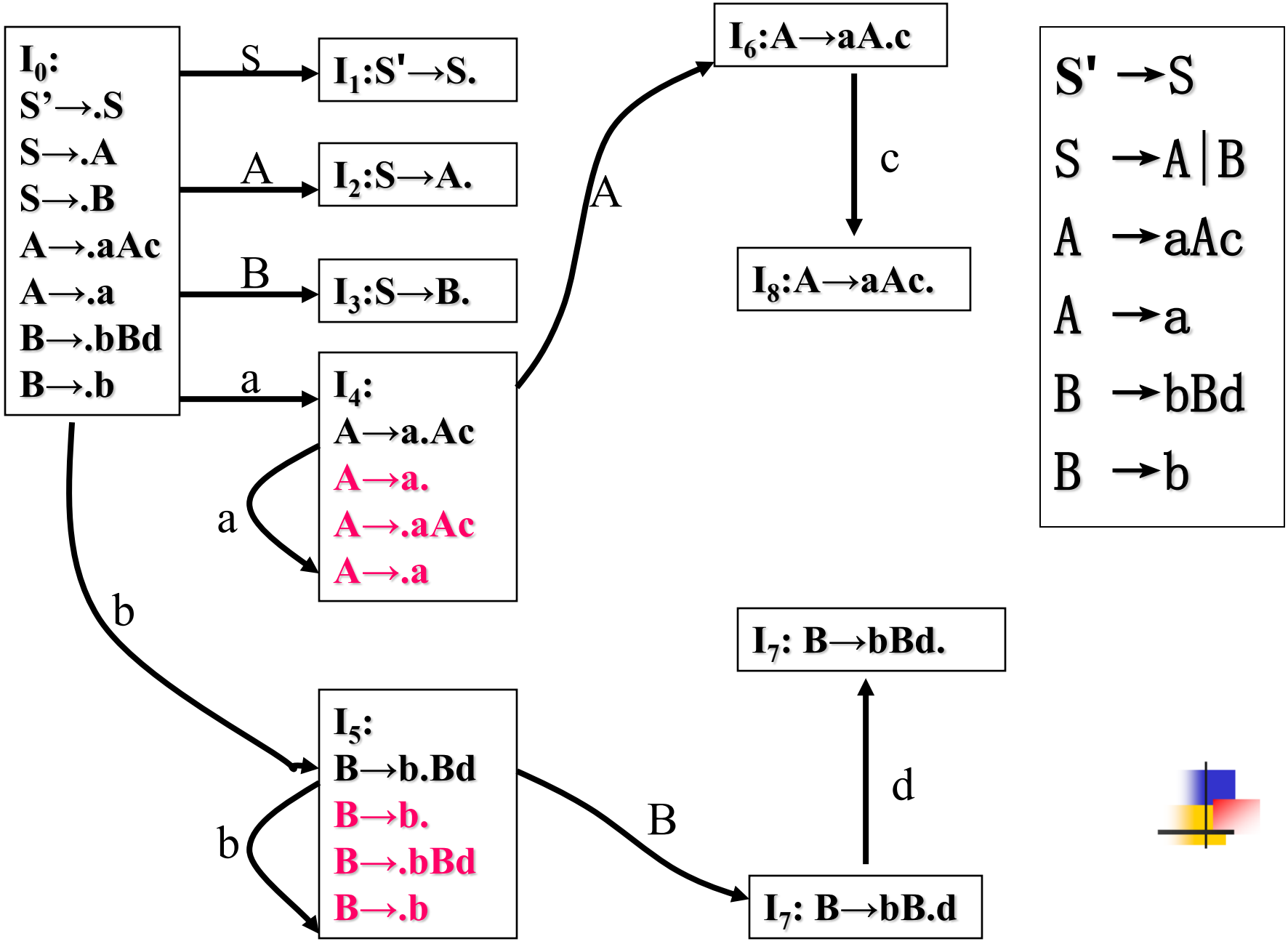
4．上述(1)到(4)步未填入信息的表项均置为error。

LR(0)不是总有效的

(S' →S)

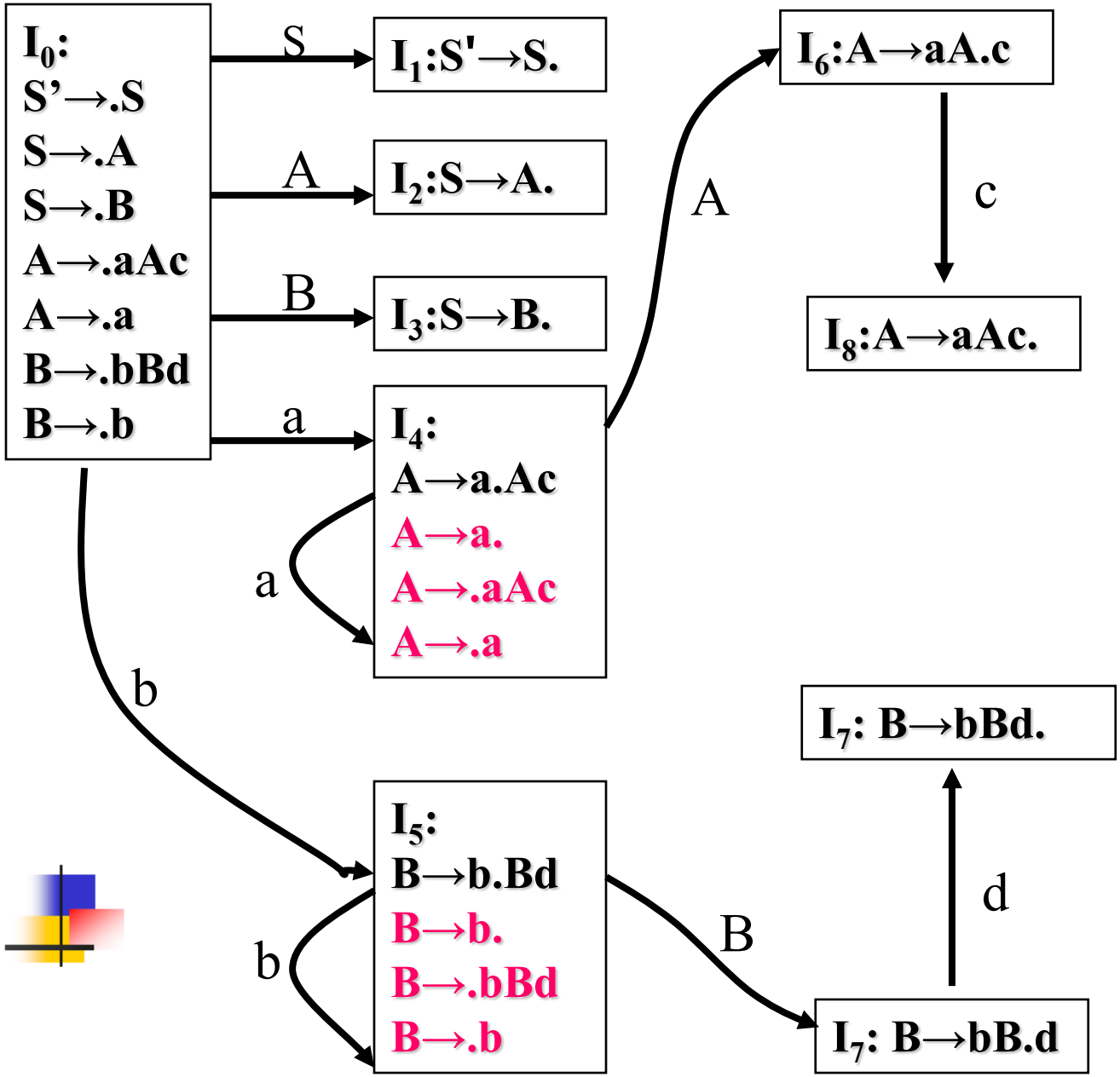
1) S →A|B
2) A →aAc
3) A →a
4) B →bBd
5) B →b

上下文无关文法不是都能用LR(0)方法进行分析的，也就是说，CFG不总是LR(0)文法.


I0:
S'→.S
S→.A
S→.B
A→.aAc
A→.a
B→.bBd
B→.b
S
I1:S'→S.
A
I2:S→A.
B
I3:S→B.
a
I4:
A→a.Ac
A→a.
A→.aAc
A→.a
a
A
I6:A→aA.c
c
I8:A→aAc.
b
I5:
B→b.Bd
B→b.
B→.bBd
B→.b
b
B
I7: B→bB.d
d
I7: B→bBd.
S' →S
S →A|B
A →aAc
A →a
B →bBd
B →b

项目集 I 的相容

- 如果 I 中至少含两个归约项目，则称 I 有归约—归约冲突（Reduce/Reduce Conflict）
- 如果 I 中既含归约项目，又含移进项目，则称 I 有移进—归约冲突（Shift/Reduce Conflict）
- 如果 I 既没有归约—归约冲突，又没有移进—归约冲突，则称 I 是相容的(Consistent)，否则称 I 是不相容的
- 对文法G，如果∀ I∈C,都是相容的，则称G为LR(0)文法



S' →S

S →A|B

A →aAc

A →a

B →bBd

B →b

问题:

如何构造其分析表?

5.3.3 SLR(1)分析表的构造算法

算法5.6 *LR*(0)分析表的构造。

输入：文法$G=(V, T, P, S)$的拓广文法G';

输出：G'的*LR*(0)分析表，即*action*表和*goto*表;

步骤：

1.令I_0= CLOSURE($\{S'\rightarrow .S\}$)，构造G'的*LR*(0)项目集规范族$C=\{I_0, I_1, \ldots, I_n\}$

2．让I_i对应状态i，I_0对应状态0，0为初始状态。

3．for k=0 to n do begin

(1) if $A\rightarrow\alpha.a\beta\in I_k$ & $a\in T$ & GO(I_k, a)=I_j then *action*[k,a]:=Sj;

(2) if $A\rightarrow\alpha.B\beta\in I_k$&$B\in V$&GO($I_k$, B)=I_j then *goto*[k,B]:=j;

(3) if $A\rightarrow\alpha.\in I_k$ & $A\rightarrow\alpha$为G的第j个产生式then

for $\forall a\in$FOLLOW(A) do *action*[k,a]:=rj;

(4) if $S'\rightarrow S.\in I_k$ then *action*[k,#]:=acc end;

4．上述(1)到(4)步未填入信息的表项均置为error。

识别表达式文法的所有活前缀的DFA

拓广文法

0) $E' \rightarrow E$

1) $E \rightarrow E + T$

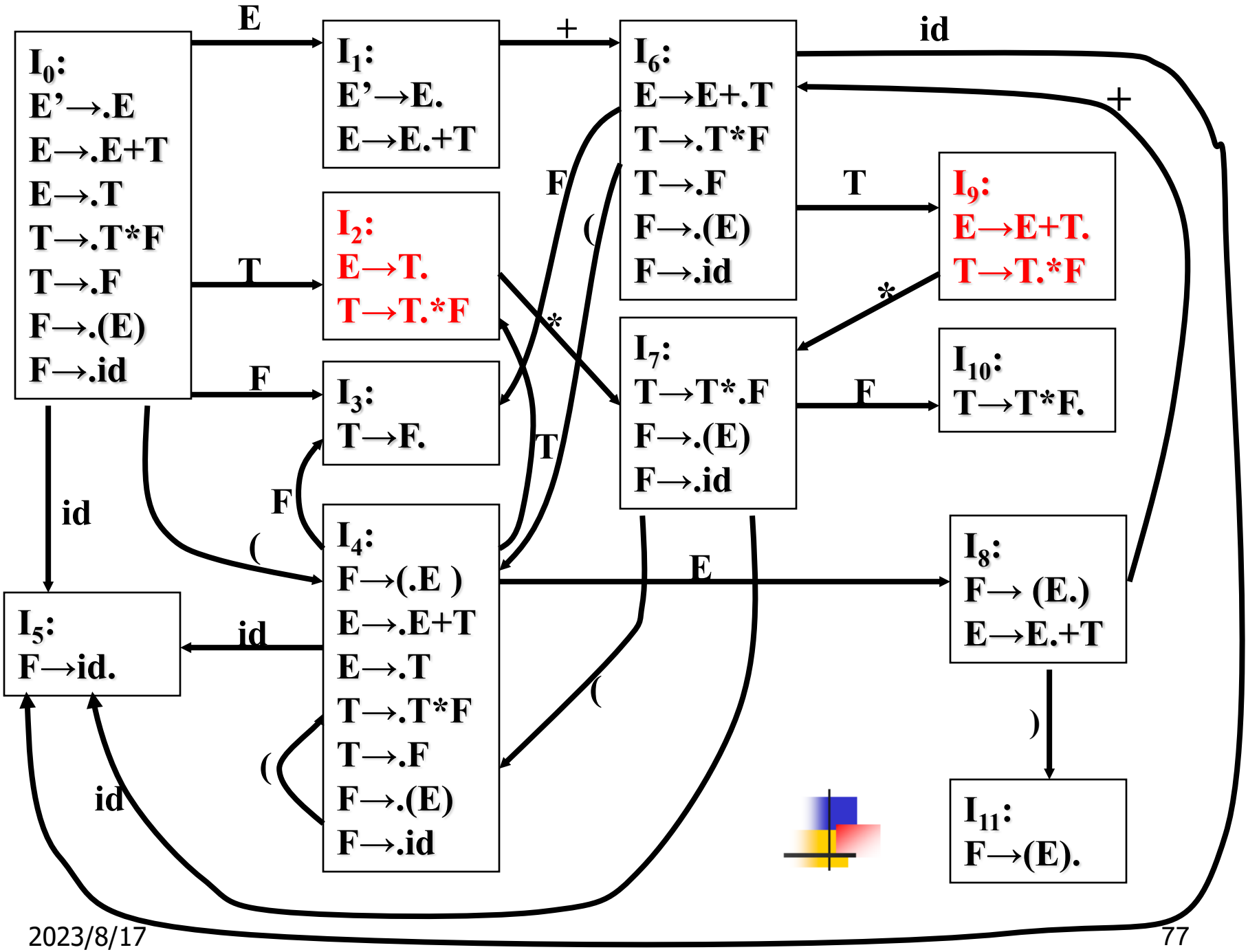
2) $E \rightarrow T$

3) $\mathbf{T \rightarrow T * F}$

4) $\mathbf{T \rightarrow F}$

5) $\mathbf{F \rightarrow (E)}$

6) $\mathbf{F \rightarrow id}$


I0:
E'→.E
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id
I1:
E'→E.
E→E.+T
I2:
E→T.
T→T.*F
I3:
T→F.
I4:
F→(.E)
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id
I5:
F→id.
I6:
E→E+.T
T→.T*F
T→.F
F→.(E)
F→.id
I7:
T→T*.F
F→.(E)
F→.id
I8:
F→ (E.)
E→E.+T
I9:
E→E+T.
T→T.*F
I10:
T→T*F.
I11:
F→(E).
E
T
F
id
(
)
+
*

表达式文法的 LR(0)分析表含有冲突

状态	ACTION					
	id	+	*	(	)	#
2	r2	r2	r2/s7	r2	r2	r2
3	r4	r4	r4	r4	r4	r4
	...					
5	r6	r6	r6	r6	r6	r6
	...					
9	r1	r1	r1/s7	r1	r1	r1
10	r3	r3	r3	r3	r3	r3
11	r5	r5	r5	r5	r5	r5

- 在状态 2、9 采用归约，出现移进归约冲突

表达式文法的SLR(1)分析表

- 求非终结符的 FISRT 集和 FOLLOW 集
  - FIRST(F) = { id, (}
  - FIRST(T) = { id, (}
  - FIRST(E) = { id, (}
  - FOLLOW(E) = {), +, # }
  - FOLLOW(T) = {), +, #, * }
  - FOLLOW(F) = {), +, #, * }

1) E → E + T

2) E → T

3) T → T * F

4) T → F

5) F → (E)

6) F → id

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			

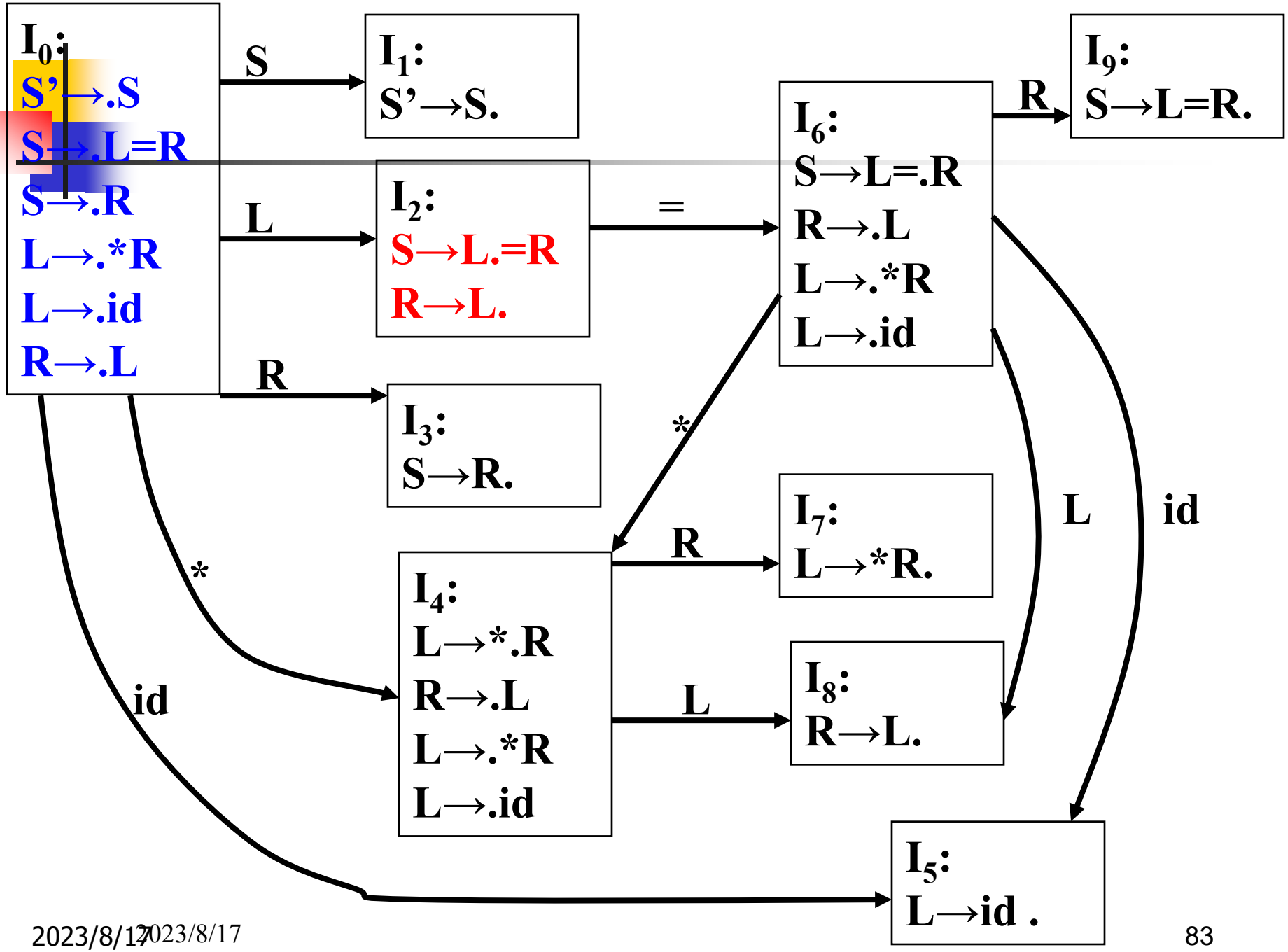
si 表示移进到状态i, ri 表示用i号产生式归约

SLR(1) 分析的特点

- 描述能力强于 LL(1)
  - SLR(1)还考虑Follow集中的符号
  - LL(1) 仅考虑产生式的首符号
- SLR(1) 文法：SLR(1)分析表无冲突的CFG

SLR(1)分析的局限性

- 如果 SLR(1) 分析表仍有多重入口（移进归约冲突或归约归约冲突），则说明该文法不是 SLR(1) 文法；
- 说明仅使用 LR(0) 项目集和 FOLLOW 集还不足以分析这种文法


I0:
S'→.S
S→.L=R
S→.R
L→.*R
L→.id
R→.L
S
I1:
S'→S.
L
I2:
S→L.=R
R→L.
R
I3:
S→R.
=
I6:
S→L=.R
R→.L
L→.*R
L→.id
R
I9:
S→L=R.
*
I4:
L→*.R
R→.L
L→.*R
L→.id
R
I7:
L→*R.
L
I8:
R→L.
L
id
*
id
I5:
L→id .

SLR分析中的冲突——需要更强的分析方法

$I_2 = \{S \rightarrow L.=R,\ R \rightarrow L.\ \}$

- 输入符号为 = 时，出现了移进归约冲突：

$S \rightarrow L .=R \in I_2$ and $go(I_2,=)=I_6$

$\Rightarrow$ action[2,=] = Shift 6

$R \rightarrow L . \in I_2$ and $= \in FOLLOW(R)=\{=,\#\ \}$

$\Rightarrow$ action[2,=] = Reduce $R \rightarrow L$

- 说明该文法不是SLR(1)文法，分析这种文法需要更多的信息。

SLR分析中存在冲突的原因

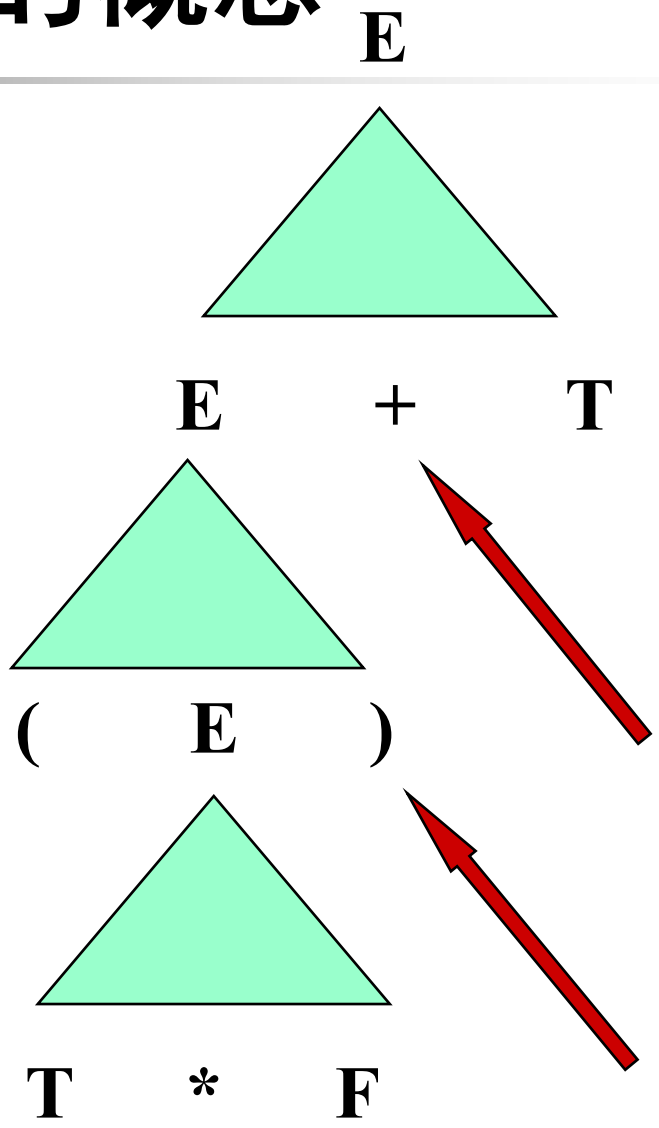
- SLR（1）只孤立地考察输入符号是否属于归约项目A→α.相关联的集合FOLLOW（A），而没有考察符号串α所在规范句型的“上下文”。
- 所以试图用某一产生式A→α归约栈顶符号串α时，不仅要向前扫描一个输入符号，还要查看栈中的符号串δα，只有当δAa的确构成文法某一规范句型的活前缀时才能用A→α归约。亦即要考虑归约的有效性：
- 问题：怎样确定δAa是否是文法某一规范句型的活前缀

5.3.4 LR(1)分析表的构造

- LR(0)不考虑后继符(搜索符)，SLR(1)仅在归约时考虑后继符(搜索符)，因此，对后继符(搜索符)所含信息量的利用有限，未考虑栈中内容。
- 希望在构造状态时就考虑后继符(搜索符)的作用：考虑对于产生式 A→α的归约，不同使用位置的 A 会要求不同的后继符号

后继符(搜索符)的概念

- 不同的归约中有不同的后继符。
- 特定位置的后继符是FOLLOW集的子集



LR(k) 项目

- 定义5.11
  - $[A\rightarrow\alpha.\beta,a_1a_2\ldots a_k]$为LR（k）项目，根据圆点所处位置的不同又分为三类：
  - 归约项目：$[A\rightarrow\alpha.,a_1a_2\ldots a_k]$
  - 移进项目：$[A\rightarrow\alpha.a\beta,a_1a_2\ldots a_k]$
  - 待约项目：$[A\rightarrow\alpha.B\beta,a_1a_2\ldots a_k]$
- 利用LR(k)项目进行(构造)LR(k)分析(器)，当k=1时，为LR(1)项目，相应的分析叫LR(1)分析(器)

LR(1) 项目的有效性

- 形式上
  - 称LR(1)项目[A→α.β,a]对活前缀γ=δα是有效的，如果存在规范推导
  - $S \Rightarrow^{*} \delta Aw \Rightarrow \delta\alpha\beta w$
  - 其中a为w的首字符，如果w=ε，则a=#
- 与LR(0)文法类似，识别文法全部活前缀的DFA的每一状态也是用一个LR(1)项目集来表示，为保证分析时，每一步都在栈中得到规范句型的活前缀，应使每一个LR(1)项目集仅由若干个对相应活前缀有效的项目组成

识别文法全部活前缀的DFA

- LR(1) 项目集族的求法
  - CLOSURE（I）：求I的闭包，目的是为了合并某些状态，节省空间
  - GO（I，X）：转移函数

闭包的计算

- CLOSURE(I)的计算
  - (核心位置：A→α.Bβ,a 扩展成闭包)
- 同时考虑可能出现的后继符
  - b ∈FIRST(βa)

闭包的计算

- 如果[A→α.Bβ,a]对γ=δα有效

/*即存在$S\Rightarrow^{*}\delta Aax\Rightarrow\delta\alpha B\beta ax$*/

- 假定$\beta ax\Rightarrow^{*}by$，则对任意的B→η有：
  - [B→.η，b]对γ=δα也是有效的，其中
  - b∈FIRST（βa）

闭包的计算

J:=I;

repeat

J=J∪{[B→.η,b]|[A→α.Bβ,a]∈J,

b∈FIRST(βa)}

until J 不再扩大

- **当β⇒$^{+}$ε时，此时b=a叫继承的后继符，否则叫自生的后继符**

状态 I 和文法符号 X 的转移函数

go(I,X) =

closure([A→αX.β,a]|[A→α.Xβ,a]∈I)

计算LR(1)项目集规范族C

即：分析器状态集合

$C=\{I_0\}\cup\{I|\exists J\in C，X\in V\cup T，I=go(J,X)\}$称为G'的LR(1)项目集规范族（算法：P185)

begin

C:= {closure({ S'→.S,#})};

repeat

for ∀I∈C，∀ X ∈ V∪T

if go(I,X)≠Φ & go(I,X)∉C then

C=C∪go(I,X)

until C不变化

end.

识别活前缀的关于LR(1) 的DFA

- 识别文法G=（V，T，P，S）的拓广文法G’的所有活前缀的DFA M=(C, V∪T, go, I_0, C)
  - I_0=CLOSURE({S’ →.S, #}
- 如果CFG G的LR(1)分析表无冲突则称G为LR(1)文法

LR(1) 分析表的构造

1．令I_0= CLOSURE({S'→.S})，构造C={ I_0, I_1, …, I_n}，即G'的LR(1)项目集规范族。

2．从Ii构造状态i，0为初始状态。

for k=0 to n do

begin

(1) if [A→α.aβ, b]∈I_k & a∈T & GO(I_k, a)=I_j then action[k,a]:=Sj;

(2) if GO(I_k, B)=I_j & B∈V then goto[k,B]:=j;

(3) if [A→α., a]∈I_k & A→α为G'的第j个产生式then action[k,a]:=rj;

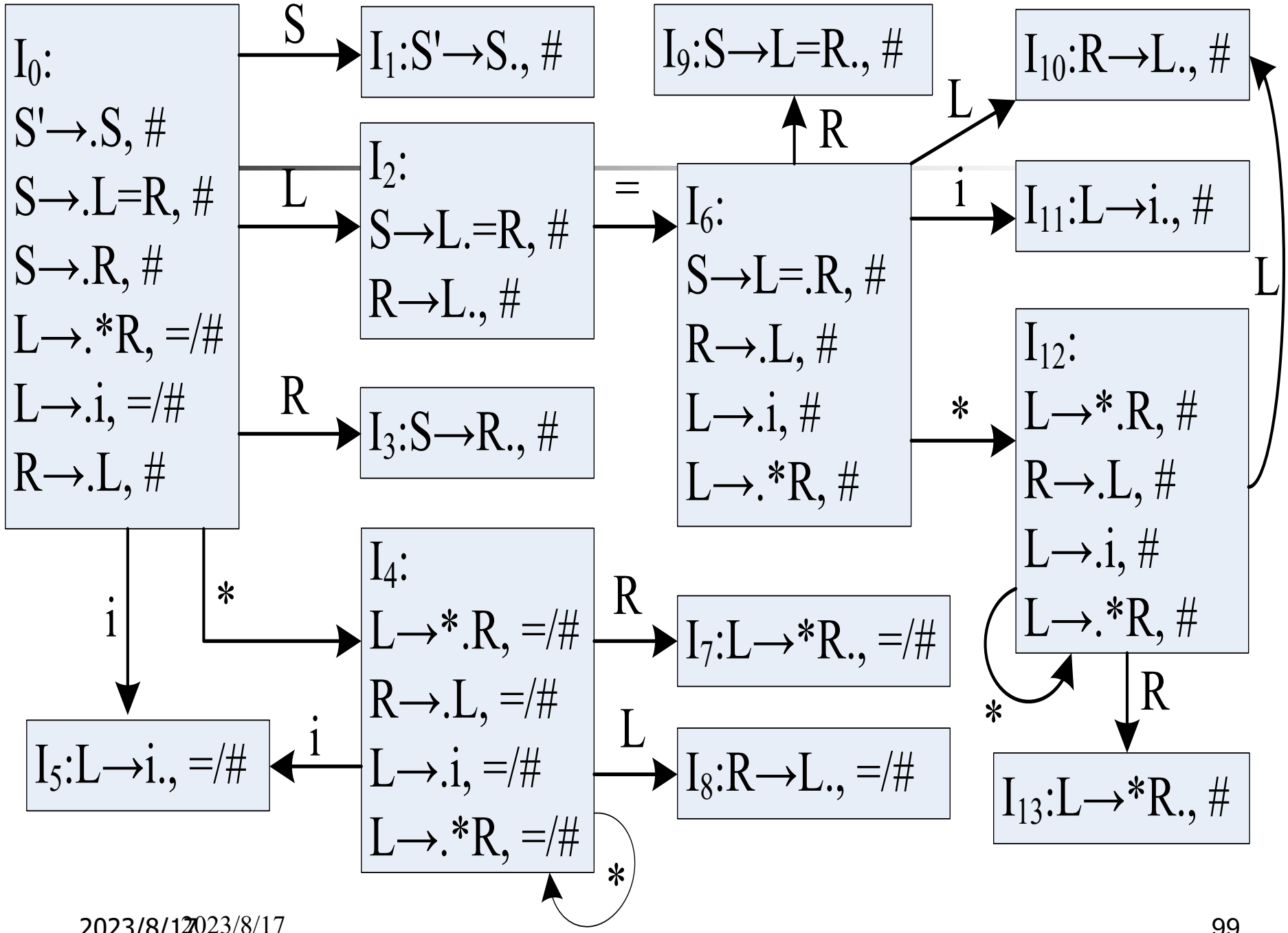
(4) if [S'→S., #]∈I_k then action[k,#]:=acc;

end

- 上述(1)到(4)步未填入信息的表项均置为error。

LR(1) 分析表的构造

- 与LR(0)的不同点主要在归约动作的选择：
  - LR(0) 分析考虑所有终结符
  - SLR(1) 分析参考 FOLLOW 集
  - LR(1) 分析仅考虑 LR(1)项目中的后继符


I0:
S'→.S, #
S→.L=R, #
S→.R, #
L→.*R, =/#
L→.i, =/#
R→.L, #
S
I1:S'→S., #
L
I2:
S→L.=R, #
R→L., #
R
I3:S→R., #
i
I5:L→i., =/#
*
I4:
L→*.R, =/#
R→.L, =/#
L→.i, =/#
L→.*R, =/#
i
*
R
I7:L→*R., =/#
L
I8:R→L., =/#
=
I6:
S→L=.R, #
R→.L, #
L→.i, #
L→.*R, #
R
I9:S→L=R., #
L
I10:R→L., #
i
I11:L→i., #
*
I12:
L→*.R, #
R→.L, #
L→.i, #
L→.*R, #
L
*
R
I13:L→*R., #

5.3.5 LALR(1)分析表的构造

- LR(1)对应的C太大
- 问题：是否可以将某些闭包/状态合并?
  - 不同的LR(1)项目闭包可能有相同的LR(0)项目，但后继符可能不同——同心
  - 合并后可能带来归约归约冲突
  - 合并那些不会带来冲突的同心的LR(1)闭包/状态
- (lookahead-LR)
  - 在不带来移进归约冲突的条件下，合并状态，重构分析表

LALR(1) 的分析能力

- 强于 SLR(1)
  - 合并的后继符仍为 FOLLOW 集的子集
- 局限性
  - 合并中不出现归约-归约冲突
  - 如果CFG G的LALR(1)分析表无冲突则称G为LALR(1)文法

5.3.6 二义性文法的应用

I_1:
E'→E.
E→E . +E
E→E.*E

I_7:
E→E+E.
E→E.+E
E→E.*E

I_8:
E→E*E.
E→E.+E
E→E.*E

- 采用二义性文法，可以减少结果分析器的状态数，并能减少对单非终结符（E→T）的归约。
- 在构造分析表时采用消除二义性的规则(按优先级)

5.3.6 二义性文法的应用

I_4:
S→iS.eS
S→iS .

选择移进else，以便让它与前面的then配对

5.3.7 LR分析中的出错处理

- 当分析器处于某一状态S，且当前输入符号为a时，就以符号对(S, a)查LR分析表，如果分析表元素$action[S, a]$为空(或出错)，则表示检测到了一个语法错误。
- 紧急方式的错误恢复：从栈顶开始退栈，直至发现在特定语法变量A上具有转移的状态S为止，然后丢弃零个或多个输入符号，直至找到符号$a \in \mathrm{FOLLOW}(A)$为止。接着，分析器把状态$goto[S, A]$压进栈，并恢复正常分析。

LR分析的基本步骤

1、编写拓广文法，求Follow集

2、求识别所有活前缀的DFA

3、构造LR分析表

5.4 语法分析程序的自动生成工具Yacc

YSP(Yacc Specification)

%{变量定义：头文件和全局变量

%开始符号

词汇表：%Token n_1,n_2,…(自动定义种别码)

%Token n_1,i_1（用户指定种别码）

……

%Token n_h,i_h（用户指定种别码）

类型说明　%type

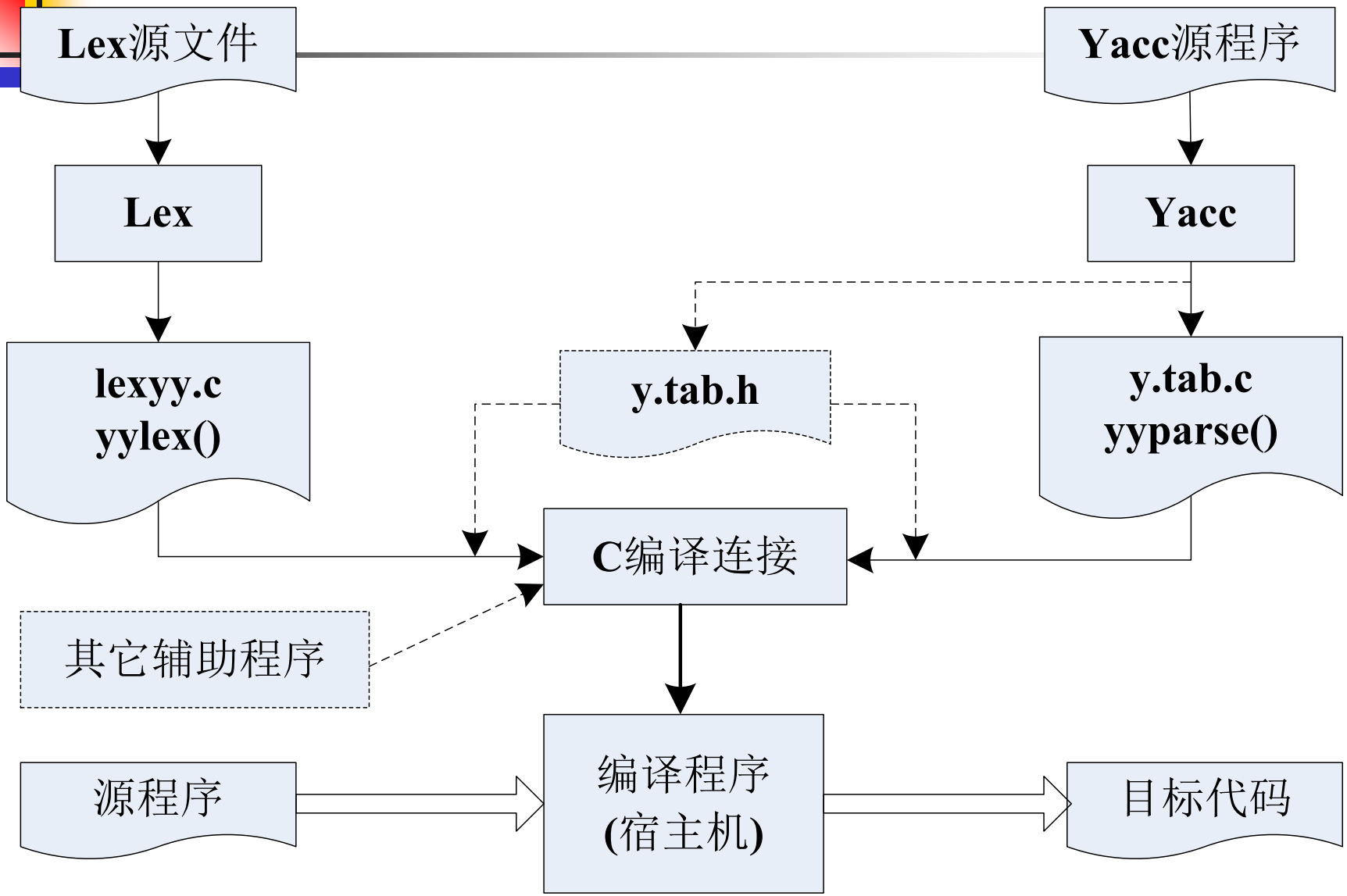
其它说明%}

%%规则部分　给出文法规则的描述

%%程序部分　扫描器和语义动作程序

- **输出：LALR(1)分析器**

用Yacc和Lex合建编译程序



本章小结

- 自底向上的语法分析从给定的输入符号串w出发，自底向上地为其建立一棵语法分析树。
- 移进-归约分析是最基本的分析方式，分为优先法和状态法。
- 算符优先分析法是一种有效的方法，通过定义终结符号之间的优先关系来确定移进和归约。
- *LR*分析法有着更宽的适应性。该方法通过构建识别规范句型活前缀的DFA来设计分析过程中的状态。可以将*LR*分析法分成*LR*(0)、*SLR*(1)、*LR*(1)、*LALR*(1)。
- 通过增加附加的信息可以解决一些二义性问题。
- Yacc是*LALR*(1)语法分析器的自动生成工具。