第六章　LR分析法及分析程序自动构造

廖力
xobjects@seu.edu.cn
3793235

第一节　概述

一、LR方法

1、 LR：自左至右扫描，最右推导的逆过程。

注：一般地说，大多数用上下文无关文法描述的程序设计语言均可用LR分析器予以识别。

第一节　概述
一、LR方法

2、LR分析法的优点

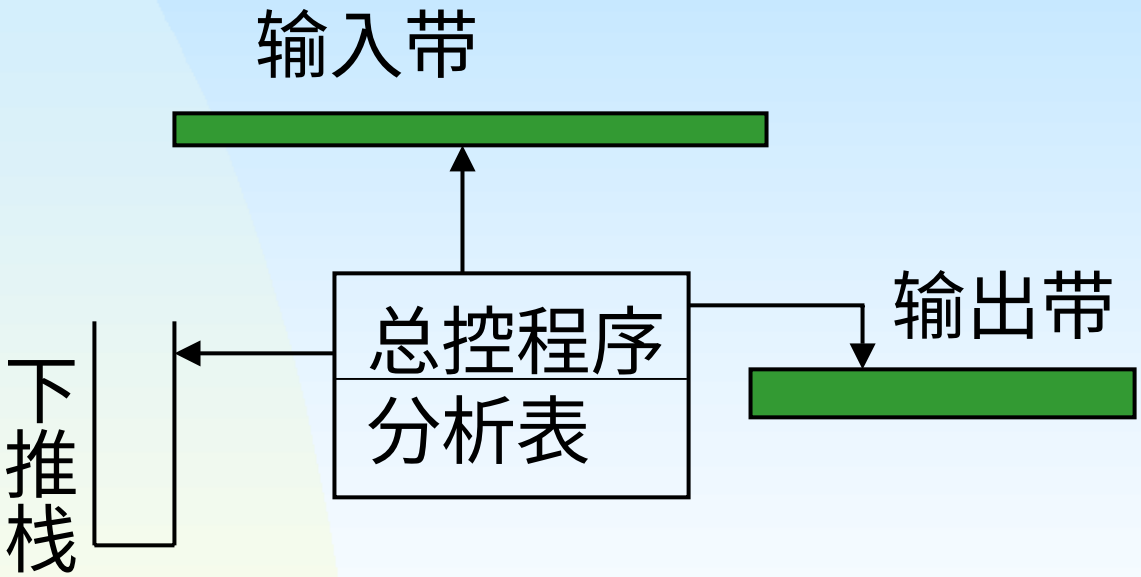
- 与算符优先分析法或其它的“移进-归约”技术相比，适应文法范围更广，能力更强，识别效率相当。
- 与普通不带回溯的自上而下预测技术相比也要好些。
- 在自左向右扫描输入串时就能发现其中错误，并能准确指出出错位置。

3、 LR分析法的缺点

- 若用手工构造分析程序，工作量太大，且容易出错，所以必须使用自动产生这种分析程序的产生器。

第一节　概述
二、LR分析器

- LR分析法通过LR分析器来实现。



- LR分析器包括两部分：总控程序和分析表；

6.1 LR分析器

一、LR分析法的基本思想

1、基本思想

在规范归约过程中，一方面记住已移进和归约出的整个符号串，另一方面根据所用的产生式推测未来可能碰到的输入符号。

2、 LR分析法寻找可归约句柄的依据

- 历史：
  - 记录在栈内的符号串移进、归约的历史情况
- 展望：
  - 预测句柄之后可能出现的信息；
- 现实：
  - 读头下的符号。

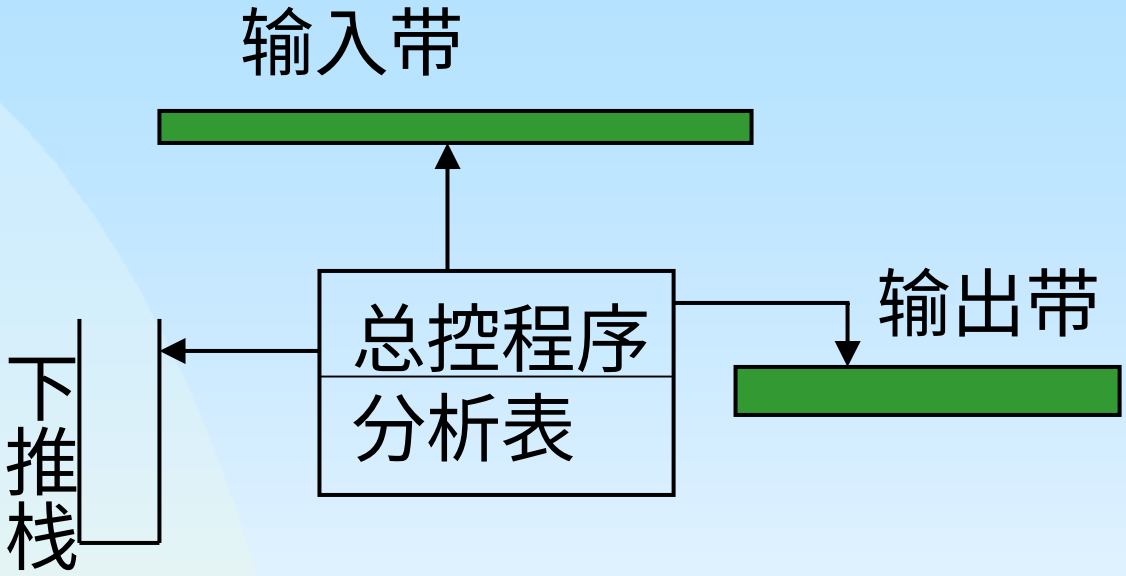
6.1 LR分析器

一、LR分析法的基本思想

1、 LR分析法寻找可归约句柄的依据

- 注：1）LR分析法是规范归约，而规范归约的关键问题是找句柄。
- 2）LR分析法的基本思想符合人的思维习惯，但实现上有困难，因为基于“历史”对未来的“展望”可能存在多种可能，当把“历史”和“展望”材料综合在一起时复杂性就大大增加。
- 3）一般会简化对“展望”的要求，以获得切实可行的分析算法。

6.1 LR分析器



二、LR分析器

- 一个LR分析器实际上是带有下推栈的确定的有限状态自动机。可将一个“历史”与这个“历史”下的展望信息综合为抽象的一个状态。

6.1 LR分析器

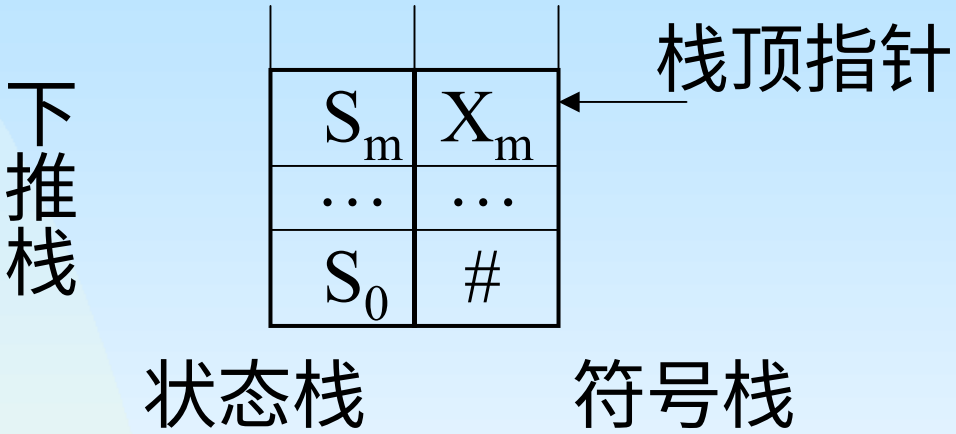
二、LR分析器

- 1、下推栈：
  - 下推栈用于存放分析输入串过程中的所有由“历史”及相应“展望”信息形成的抽象状态。
  - 由于每个状态都概括了从分析开始到归约阶段的全部“历史”和“展望”信息，所以 LR分析器的每步动作可由栈顶状态和读头下符号唯一确定。

6.1 LR分析器

二、LR分析器

- 1、下推栈：



  - 为了便于了解栈顶状态对正规分析过程的作用，将栈分为两栏：状态栏和符号栏。符号栈仅记录迄今移进—归约所得到的文法符号，由于它们已经被概括在“状态”里了，所以对语法分析不起作用。
  - 初始时符号栈放“#”（栈底符），状态栈放S_0（初态），栈顶S_m代表符号栈内的符号串$X_mX_{m-1}\ldots X_1$及其展望信息。

6.1 LR分析器

二、LR分析器

2、分析表——LR分析器的核心

	a_1 a_2 … a_n	a_1 a_2 … a_n A_1…A_m
s_0 … s_k		

Action goto

- 1）分析表构成：
  - 动作表(ACTION)和转向表(GOTO)
- 注：S_i表示状态， a_i表示终结符，A_i表示非终结符。

6.1 LR分析器

二、LR分析器

2、分析表

- 2） 动作表ACTION
  - ACTION[S,a]表示在当前状态S下，面临读头下的符号a所应采取的动作。
  - 注：该动作有四种可能：移进、归约、出错、接受
- 3）转向表GOTO
  - GOTO[S,X]：若X $\in V_T$，表示在当前状态下，读入X应转向什么状态；若X $\in V_N$，表示当前栈顶句柄归约成X后，应转向什么状态。
  - 注：对终结符的移进动作和转向动作可以合并在一起填在动作表中，这样，转向表可以只保留非终结符转向部分。

6.1 LR分析器

二、LR分析器

3、总控程序

- 总控程序的动作根据当前栈顶状态S_m和读头下符号a_i查表决定。
- 1）移进 把(S_m,a_i)的下一个状态S`=GOTO$(S_m,a_i)$连同读头下符号推进栈内，栈顶成为$(S`,a_i)$,读头前进一格。

6.1 LR分析器

二、LR分析器

3、总控程序

- 1）移进
- 2）归约　指用某产生式$A \rightarrow \beta$进行归约。若β的长度为γ，则弹出栈顶的γ个元素，使得栈顶的状态变成$S_{m-\gamma}$，然后把（$S_{m-\gamma}$，A）的下一个状态S`=GOTO($S_{m-\gamma}$，A)连同非终结符A一起推进栈，栈顶变成(S`,A),读头不动。
- 3）接受　分析成功，退出总控程序。
- 4）报错　输入串出错，调用相应出错程序。
- 注：不管哪一类分析程序，总控程序的动作都一样。程序本身很简单，按动作表中填的内容具体实施而已。

6.1 LR分析器

- 例：根据表达式文法的LR分析表分析输入串 i*i＋i的LR动作过程。
- $E \rightarrow E+T$
- $E \rightarrow T$
- $T \rightarrow T*F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow i$

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4				9	3
7	S_5			S_4					10
8		S_6			S_{11}				
9		r_1			r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

6.1 LR分析器

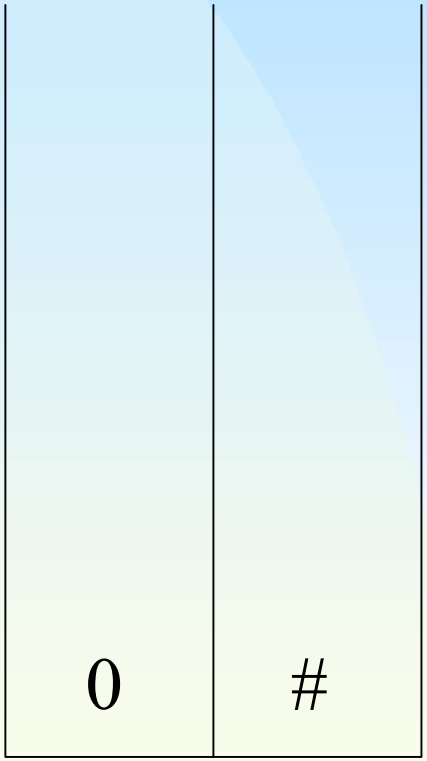
- 注：表中符号的含义：
  - S_j Shift j，指将读入符号a移进栈内并转到j状态，栈顶变成（j,a）；
  - r_j Reduce j,按第j号产生式进行归约；
  - acc accept ,分析成功；
  - 空白格 出错标志，若填入相应出错处理程序的编号，便转到相应程序处理。
  - 分析过程见书P103.

*i+i#

状态	符号
5	i
0	#

状态、符号栈

- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i



*i+i#

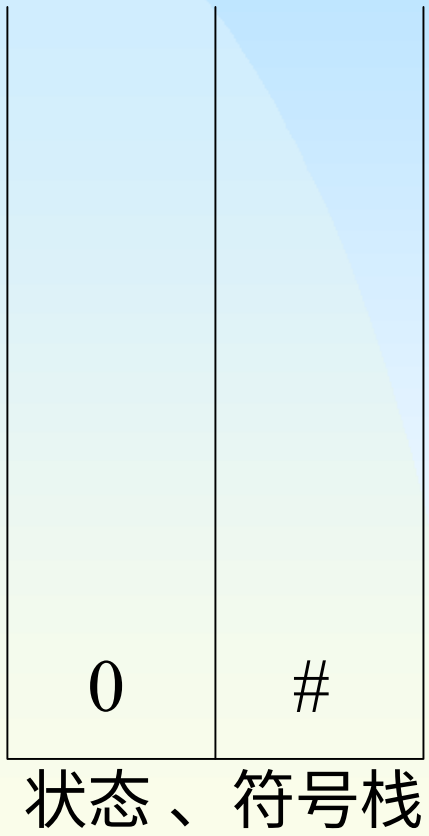
- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i

*i+i#

状态	符号
3	F
0	#

状态、符号栈

- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i



*i+i#

3 F

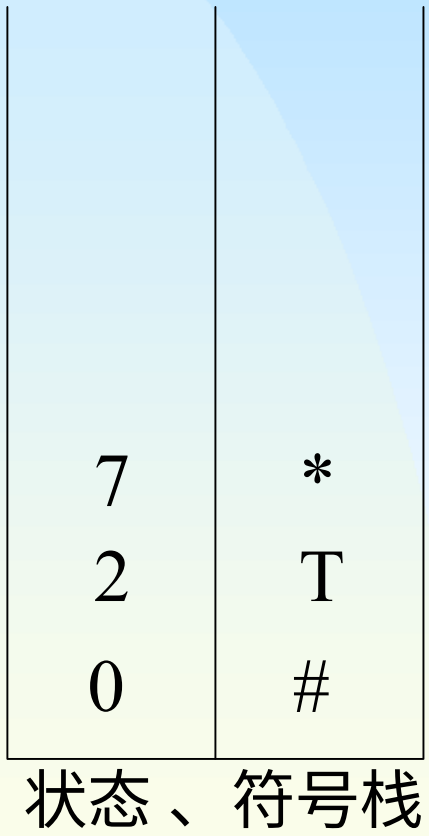
- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i

*i+i#

2	T
0	#

状态、符号栈

- $E \rightarrow E+T$
- $E \rightarrow T$
- $T \rightarrow T*F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow i$



i+i#

- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i

5	i
7	*
2	T
0	#

状态、符号栈

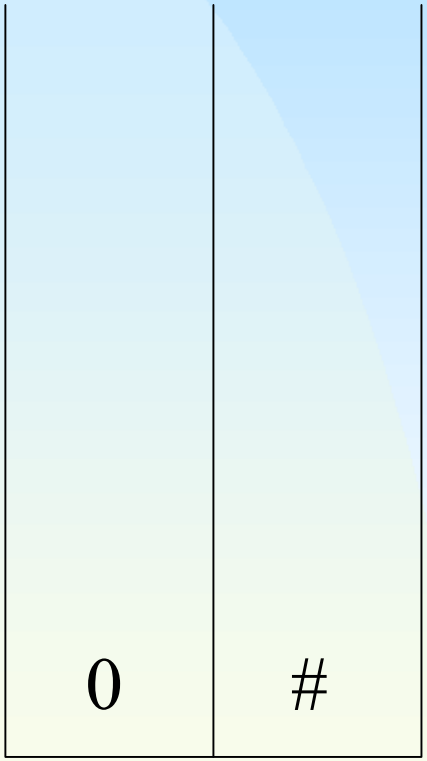
+i#

- $E \rightarrow E+T$
- $E \rightarrow T$
- $T \rightarrow T*F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow i$

状态	符号栈
10	F
7	*
2	T
0	#

+i#

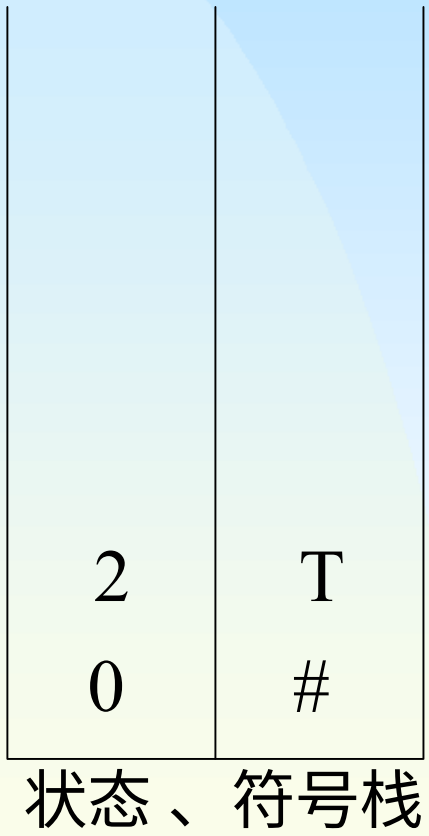
- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i



+i#

10	F
7	*
2	T

- E → E+T
- E →T
- T →T*F
- T →F
- F →(E)
- F → i



+i#

- $E \rightarrow E+T$
- $E \rightarrow T$
- $T \rightarrow T*F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow i$

状态栈	符号栈	输入串	动 作
0	#	i*i+i#	
05	#i	*i+i#	移进
03	#F	*i+i#	R6归约
02	#T	*i+i#	R4归约
027	#T*	i+i#	移进
0275	#T*i	+i#	移进
02710	#T*F	+i#	R6归约
02	#T	+i#	R3归约
01	#E	+i#	R2归约
016	#E+	i#	移进
0165	#E+i	#	移进
0163	#E+F	#	R6归约
0169	#E+T	#	R4归约
01	#E	#	R1归约
分析成功			

6.1 LR分析器

二、LR分析器

4、LR文法

- 定义：如果某一文法能够构造一张分析表，使得表中每一元素至多只有一种明确动作，则该文法称为LR文法。
- 注：1）并非所有CFG都是LR文法，但对于多数程序设计语言来说，一般都可以用LR文法描述。

  2）和LL(1)分析法相比，LR分析法适应的文法范围要广一些。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造基本思想

- 构造LR(0)分析表的基本思想：
  - 只根据历史信息识别呈现于栈顶的句柄。
- 注：LR(0)分析表构造的思想和方法是构造其它LR分析表的基础。
- 构造LR(0)分析表的基本策略：
  - 构造文法G的一个有限自动机，它能识别文法中的所有活前缀。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

1、活前缀

- 字的前缀是指该字的任意首部。
  - 例如：字ABC的前缀有ε，A,AB,ABC。
- 活前缀的概念：
  - 指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。

例如：根据下面的文法识别输入串abbcde。

1) S →aAcBe

2) A→b

3)A →Ab

4)B →d

为每条产生式的尾部加上用[I]表示的产生式序号

1) S →aAcBe[1]

2) A→b[2]

3)A →Ab[3]

4)B →d[4]

- 用最右推导方式来识别,推导时把序号也带上。
- S →aAcBe[1]→aAcd[4]e[1] →aAb[3]cd[4]e[1]
  →ab[2]b[3]cd[4]e[1]
- 若用最左归约的方式进行识别，则完全是上面的逆过程。
- 规范句型abbcde的前缀有：ε，a， ab
- 规范句型aAbcde的前缀有：ε，a， aA， aAb
- 规范句型aAcde的前缀有：ε，a， aA，aAc, aAcd
- 规范句型aAcBe的前缀有：ε，a, aA，aAc, aAcB, aAcBe

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

1、活前缀

- 活前缀有两种类型：
  - 1）归态活前缀　活前缀的尾部正好是句柄之尾，这时可以进行归约。归约之后又成为另一句型的活前缀。
  - 2）非归态活前缀　句柄尚未形成，需要继续移进若干符号之后才能形成句柄。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

- 对于一个文法G，我们可以构造一个有限自动机，它能识别G的所有活前缀。
- 由于产生式右部的符号串就是句柄，若这些符号串都已进栈，则表示它已处于归态活前缀，若只有部分进栈，则表示它处于非归态活前缀。要想知道活前缀有多大部分进栈了，可以为每个产生式构造一个自动机，由它的状态来记住当前情况，此“状态”称为“项目”。这些自动机的全体就是能识别所有活前缀的有限自动机。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

1）项目:

– 在文法的每个产生式右部添加一个圆点，就成为G的一个LR(0)项目（简称项目）。

– 注：圆点在产生式中的位置不同则是不同项目。

注：（1）可以把圆点理解为栈内外的分界线。

（2）产生式右部符号串的长度为n，则可以分解为n+1个项目。

（3）产生式A→ε只有一个项目A→•。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

1）项目:

- 例如，产生式A → XYZ对应四个项目
  - A→•XYZ　预期要归约的句柄是XYZ，但都未进栈
  - A→X•YZ　预期要归约的句柄是XYZ，仅X进栈
  - A→XY•Z　预期要归约的句柄是XYZ，仅XY进栈
  - A→X YZ•　已处于归态活前缀，XYZ可进行归约，这个项目是归约项目。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

2）由项目构造NFA的构造方法

（1）将文法进行拓广，保证文法开始符号不出现在任何产生式右部，即增加产生式S`→S，并令S`→ • S作为初态项目；

（2）凡圆点在串的最右边的项目称终态项目或称归约项目，而S`→ S • 称为接受项目；

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

2）由项目构造NFA的构造方法

（3）设项目i为$X \rightarrow X_1 \ldots X_{i-1} \bullet X_i \ldots X_n$，项目j为$X \rightarrow X_1 \ldots X_i \bullet X_{i+1} \ldots X_n$，则从项目i画一弧线射向j，标记为$X_i$，若$X_i$是终结符则项目i称为移进项目，$X_i$是 非终结符则称项目i为待约项目；

（4）若项目i为$X \rightarrow \alpha \bullet A\beta$，其中A是非终结符，则从i项目画ε弧射向所有$A \rightarrow \bullet \gamma$的项目，$\gamma \in V^*$

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

2）由项目构造NFA的构造方法

注：1)构造出的NFA是包含有ε串的NFA，可以使用子集法使之确定化，使之成为一个以项目集为状态的DFA，这个DFA就是建立LR分析算法的基础。

2)相应DFA的每个状态是一个项目集，称作LR(0)项目集，整个状态集称为LR(0)项目集规范簇。

3)在DFA的一个状态对应的项目集内，每个项目是“等价”的，即从期待归约的角度看相同。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、 LR(0)分析表构造

2、构造自动机识别活前缀

2）由项目构造NFA的构造方法

注： 4)有一个唯一的初态和一个唯一的接受态，但有若干个归约态，表示有若干种活前缀的识别状态。

5)状态反映了识别句柄的情况，即句柄的多大部分已进栈，即知道了历史情况。

6)手工构造文法的项目集规范簇很困难。

例如：有一已拓广的文法G6.1构造识别活前缀的NFA

S` →E　E →aA|bB　A →cA|d　B →cB|d

解：1、这个文法的项目有：

1. S` →•E　2. S` →E•　3. E → •aA

4. E → a•A　5. E → aA•　6. A → •cA

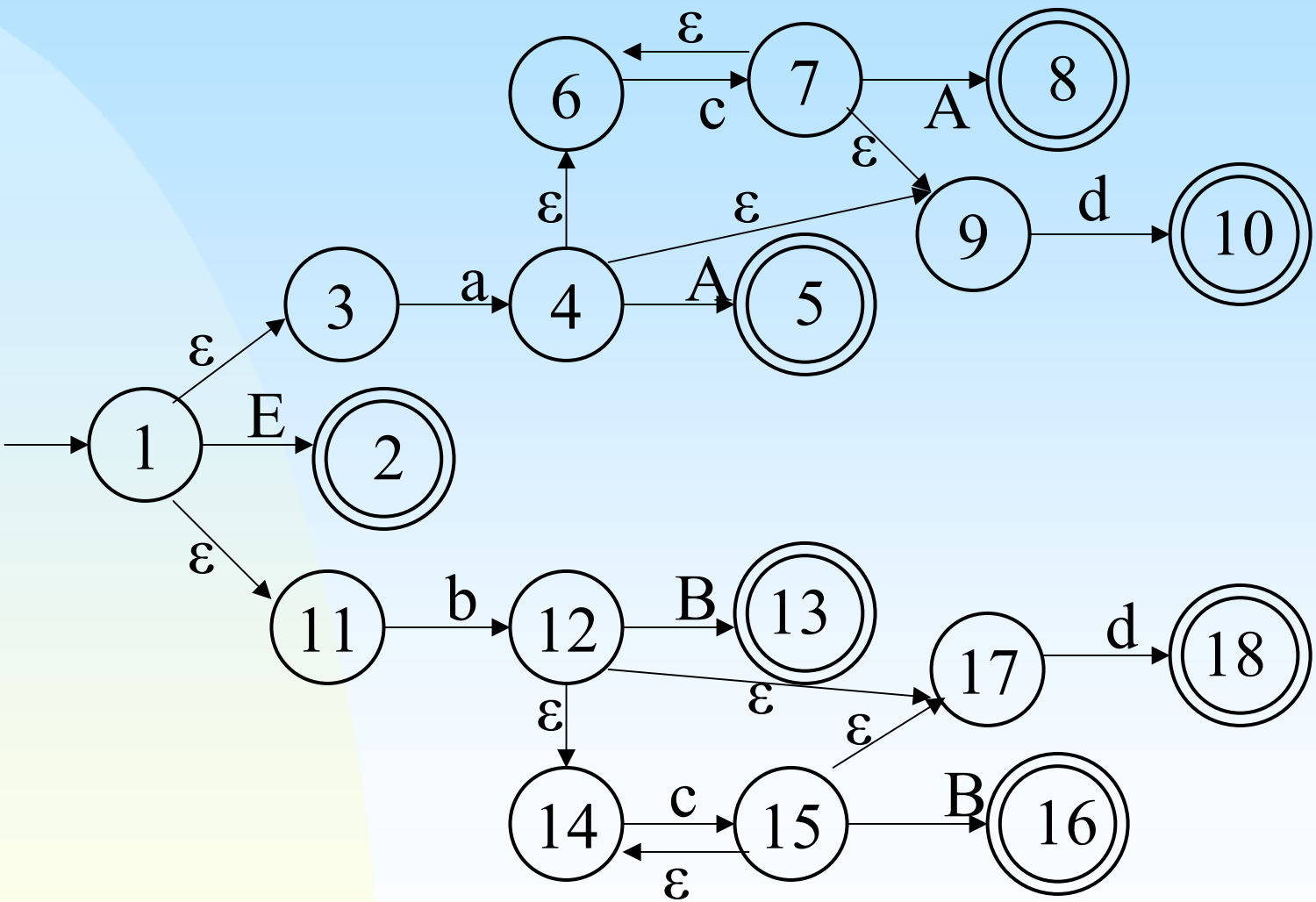
7. A → c•A　8. A → cA •　9. A → •d

10. A → d•　11. E → •bB　12. E → b•B

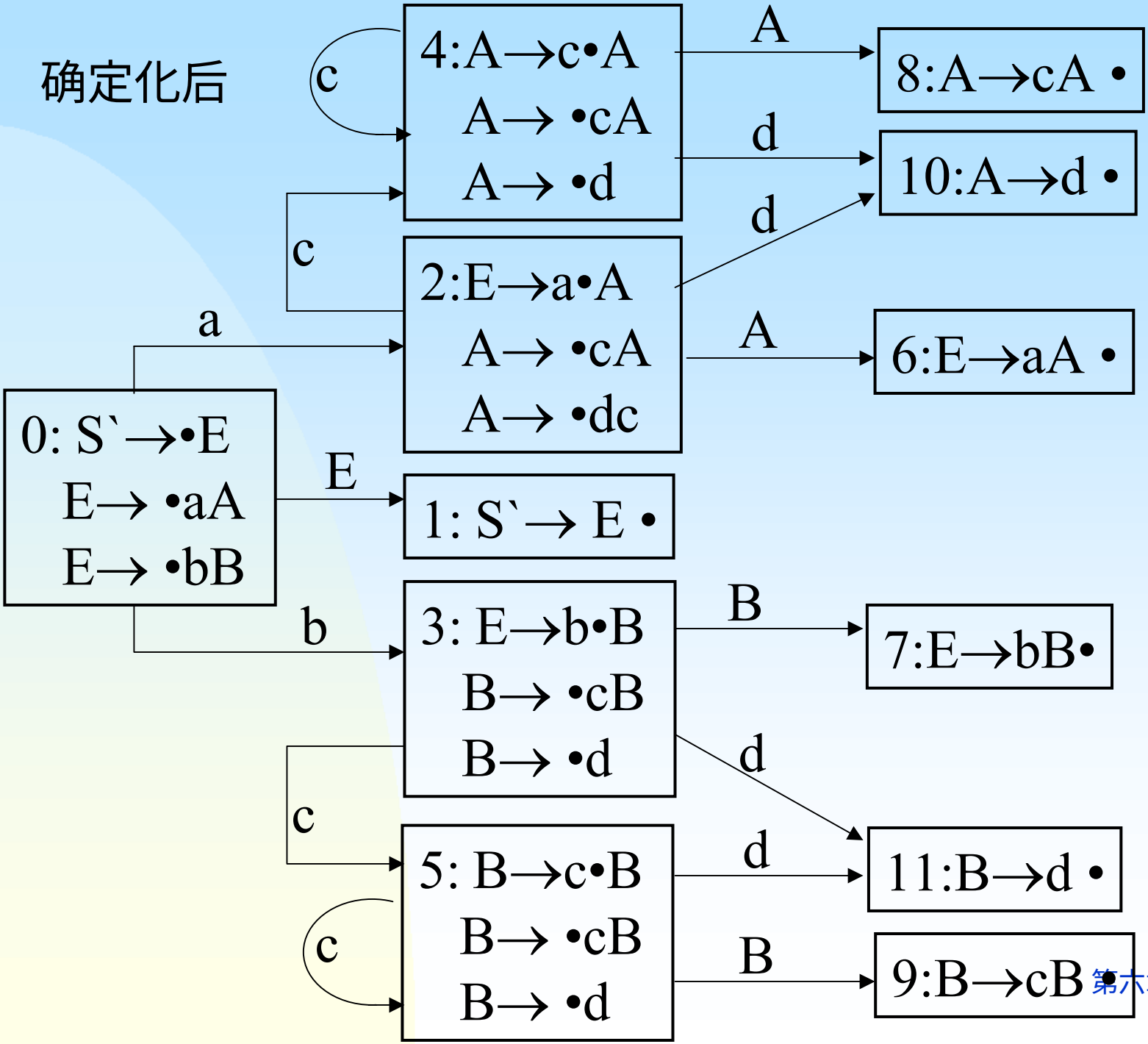
13. E → bB•　14. B → •cB　15. B → c•B

16.B → cB•　17. B → •d　18. B → d•

2、构造识别活前缀的NFA



确定化后


0: S`→•E
E→ •aA
E→ •bB
1: S`→ E •
2:E→a•A
A→ •cA
A→ •dc
3: E→b•B
B→ •cB
B→ •d
4:A→c•A
A→ •cA
A→ •d
5: B→c•B
B→ •cB
B→ •d
6:E→aA •
7:E→bB•
8:A→cA •
9:B→cB
10:A→d •
11:B→d •
a
b
c
E
A
B
d

6.2 LR(0)项目集族和LR(0)分析表的构造

一、LR(0)项目集规范族的构造

3、LR(0)项目集规范族的自动构造

- 1）拓广文法：增加S` →S 产生式，使文法的开始符号不出现在任何产生式右部，从而保证有唯一的接受项目。
  - 注：即使原开始符号S不出现在任何产生式右部,为了统一起见也要增加该产生式。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、LR(0)项目集规范族的构造

3、LR(0)项目集规范族的机器构造

2）定义和构造项目集的闭包

- 设I是拓广文法G`的一个项目集，定义和构造I的闭包CLOSURE(I)。
- 构造方法：
  - A. I的任何项目都属于CLOSURE(I)；
  - B. 若A→α•Bβ属于CLOSURE(I),B是非终结符，那么，对于任何关于B的产生式B→γ，项目B→•γ也属于CLOSURE(I);
  - C. 重复执行步骤B.直到CLOSURE(I)不再扩大为止。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、LR(0)项目集规范族的构造

3、LR(0)项目集规范族的机器构造

3)状态转换函数GO

- GO(I,X)定义为CLOSURE(J)，其中I,J都是项目集，X$\in(V_N \cup V_T)$,J={任何形如A→αX•β的项目| A→α• Xβ$\in$I}。
  - 注：其含义是对于任意项目集I转换到项目J，是由于I中有A→α•Xβ项目，J中有A→αX•β项目，表示识别活前缀又移进一个符号X。

6.2 LR(0)项目集族和LR(0)分析表的构造

一、LR(0)项目集规范族的构造

3、LR(0)项目集规范族的机器构造

- 4）构造LR(0)项目集规范族的算法
  - 过程如下：
  - { C={CLOSURE(S` →•S)} /*初态项目集*/
  - DO
  - { FOR (对C中每个项目集I和G`中每个文法符号X)
  - IF (GO(I,X)非空且不属于C)
  - {把GO(I,X)加入C中}
  - }WHILE C仍然在扩大
  - } 注：其中C是集合，用于存放全部的项目集

6.2 LR(0)项目集族和LR(0)分析表的构造

- 注：1）此算法是迭代算法，置了C的初态（初态仅包含第一个项目集）后，每经过一次FOR语句，就扩大一次C中的项目集数，直到项目集数不再增加为止。

  2）此算法是从I_0开始，按该项目集内的项目顺序依次求出所有后继项目集。由这样一层一层向下生成的所有项目集的方法避免了项目集的遗漏。

  3）若在项目集中存在A→ •ε项目，不应再做GO函数转向其他项目集，因为A→ε•和A→•ε是同一项目，都是A→•项目，它本身是归约项目，不存在后继项目。

  4）由这个项目集规范族C中各个状态及状态转换函数GO ，可构造一张识别活前缀的DFA图。

例如：文法如下：

S` →E　E →aA|bB　A →cA|d　B →cB|d

这个文法的项目有：

1. S` →•E　2. S` →E•　3. E → •aA

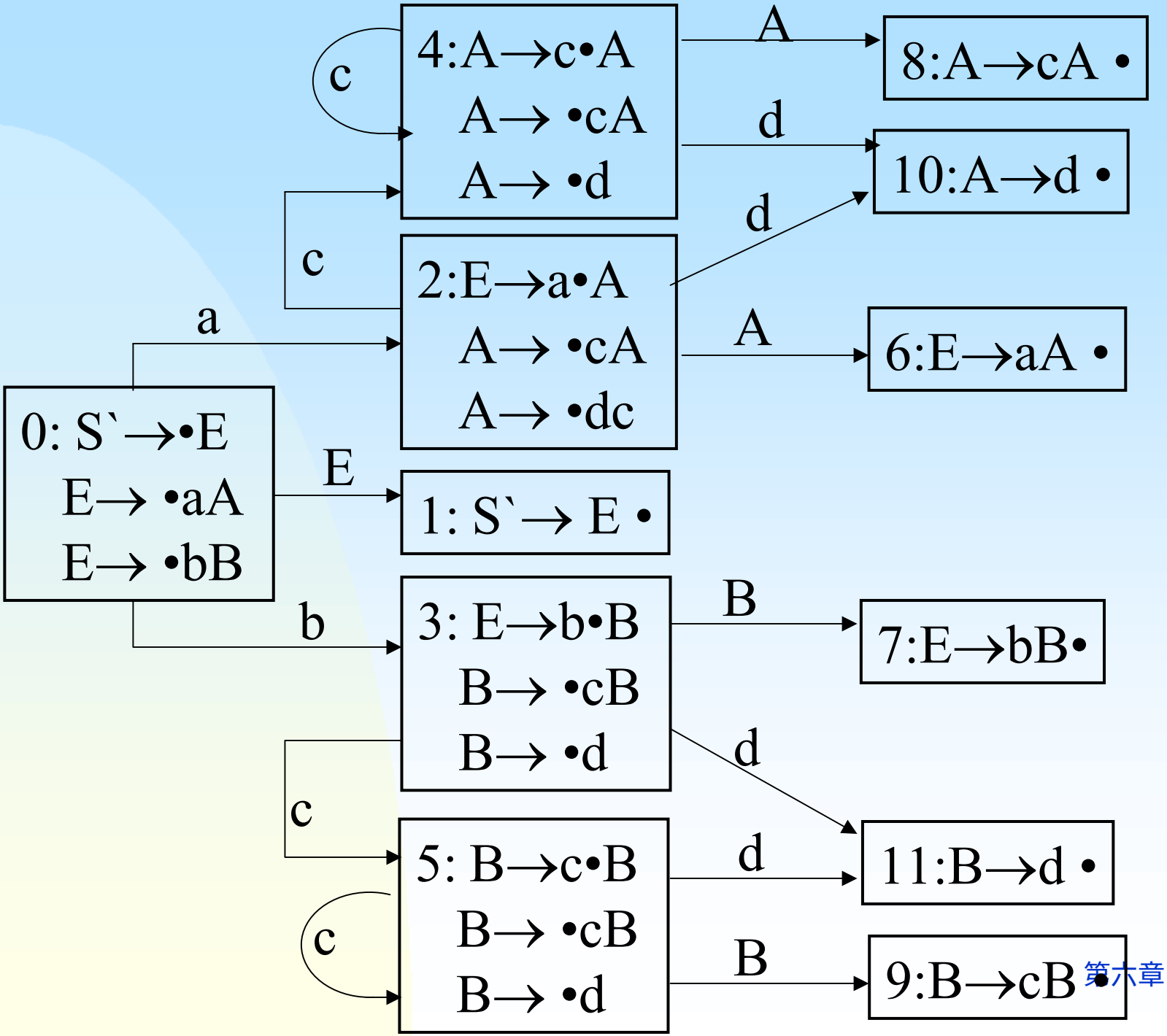
4. E → a•A　5. E → aA•　6. A → •cA

7. A → c•A　8. A → cA •　9. A → •d

10. A → d•　11. E → •bB　12. E → b•B

13. E → bB•　14. B → •cB　15. B → c•B

16.B → cB•　17. B → •d　18. B → d•


0: S`→•E
E→ •aA
E→ •bB
1: S`→ E •
2:E→a•A
A→ •cA
A→ •dc
3: E→b•B
B→ •cB
B→ •d
4:A→c•A
A→ •cA
A→ •d
5: B→c•B
B→ •cB
B→ •d
6:E→aA •
7:E→bB•
8:A→cA •
9:B→cB •
10:A→d •
11:B→d •
a
b
c
d
E
A
B

6.2 LR(0)项目集族和LR(0)分析表的构造

二、LR(0)分析表的构造算法

- 1、构造LR(0)分析表
  - 设C={I_0,I_1,…I_n},以各项目集I_k(k＝0，…,n)的k作为状态序号，并以包含S`→•S的项目集作为初始状态，同时将G`文法的产生式进行编号。然后按下列步骤填写ACTION表和GOTO表：

1)若项目A→α•aβ属于I_k状态且GO(I_k,a)= I_j,a为终结符，则置ACTION[k,a]=S_j; 即：移进a，并转向I_j状态。

2)若项目A→α•∈I_k,则对任何终结符a(包括语句结束符＃),置ACTION[k,a]=r_j; 其中，j为产生式A→α 的编号，即根据j号产生式进行归约。

6.2 LR(0)项目集族和LR(0)分析表的构造

二、LR(0)分析表的构造算法

1、构造LR(0)分析表

3)若项目S` → S•属于I_k，则置ACTION[k,＃]=accept,简写为acc;

4)若GO(I_k，A)= I_j,A是非终结符，则置GOTO[k,A]=j;

5)分析表中凡不能用步骤1至4填入信息的空白项，均置上“出错标志”。

6.2 LR(0)项目集族和LR(0)分析表的构造

二、LR(0)分析表的构造算法

2.LR(0)文法

- 定义：若文法G按上面算法构造出来的分析表不包含多重定义项，则该文法G是LR(0)文法。
- 注：1）显然，LR(0)文法的每个项目集中不包含有任何冲突项目。

  2）LR(0)文法的能力很弱，甚至连表达式文法也不属于LR(0)文法，所以没有实用价值，但可以利用它的构造算法来构造其他LR分析表。

6.2 LR(0)项目集族和LR(0)分析表的构造

二、LR(0)分析表的构造算法

2.LR(0)文法

- 注：3)下推栈栈顶状态实质上是DFA的一个状态，它反映了识别活前缀的进程，反映了找句柄的历史情况（即句柄的多大部分已经进栈），而不必到栈里去找过去的移进－归约历史。这种下推自动机的分析过程实际上可看作是有限自动机与下推栈结合的过程。由有限自动机表示分析活前缀的过程，而下推栈是记住已分析的历史情况。

6.2 LR(0)项目集族和LR(0)分析表的构造

二、LR(0)分析表的构造算法

2.LR(0)文法

- 注： 4) 下推自动机的能力比有限自动机能力强。

  5)由于LR(0)文法作归约时只看历史情况，与当前读头下的符号无关，所以它的能力很弱，需要改进。

例如：下面的文法是LR(0)文法，假定对这个文法的各个产生式给予编号并写成：

0. S` → E　　4.A →d

1.E →aA　　5.B →cB

2.E →bB　　6.B →d

3.A →cA

状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0	S_2	S_3				1		
1					acc			
2			S_4	S_{10}			6	
3			S_5	S_{11}				7
4			S_4	S_{10}			8	
5		r_6	S_5	S_{11}				9
6	r_1	r_1	r_1	r_1	r_1			
7	r_2	r_2	r_2	r_2	r_2			
8	r_3	r_3	r_3	r_3	r_3			
9	r_5	r_5	r_5	r_5	r_5			
10	r_4	r_4	r_4	r_4	r_4			
11	r_6	r_6	r_6	r_6	r_6			

- 例如：设文法G的LR(0)项目集规范族中含有如下一个项目集（状态）I：
  - I={X→δ•bβ　　/*移进项目*/
  - 　　A→α•　　/*归约项目*/
  - 　　B→α•　　/*归约项目*/
  - 这三个项目告诉我们应做的动作各不相同，出现了移进－归约冲突和归约－归约冲突。这个文法一定不是LR(0)文法。

6.3 SLR分析表的构造

- SLR是LR(0)的一种改进，它在归约时除了考虑历史情况之外还考虑了一点现实。

一、消除冲突

- 1、一个有冲突的项目集，可以根据读头下符号的不同来消除冲突。
- 一般而言，对于任何形如$I=\{X\rightarrow\delta\bullet b\beta\ ,\ A\rightarrow\alpha\bullet,\ B\rightarrow\alpha\bullet\}$的LR(0)项目集，若$Follow(A)\cap Follow(B)=\Phi$且$b\notin Follow(A)$, $b\notin Follow(B)$,则可以根据当前读头下符号a来消除冲突。即在构造LR分析表的算法中做一些改变：

6.3 SLR分析表的构造

一、消除冲突

1）若当前输入符a=b，做移进；

2）若当前输入符a ∈ Follow(A)，按A→α产生式归约；

3）若当前输入符a ∈ Follow(B)，按B→α产生式归约；

4）其他，报错。

6.3 SLR分析表的构造

二、构造SLR分析表的算法

- 设C={I_0,I_1,…I_n},以各项目集I_k(k＝0，…,n)的k作为状态序号，并以包含S` →•S的项目集作为初始状态，同时将产生式进行编号。然后按下列步骤填写ACTION表和GOTO表：

1)若项目A→α•aβ属于I_k状态且GO(I_k,a)= I_j,a为终结符，则置ACTION[k,a]=S_j; 即：移进a，并转向I_j状态。

2)若项目A→α• ∈I_k,则对任何终结符a ∈ Follow(A),置ACTION[k,a]=r_j;其中j为产生式A→α的编号，即根据j号产生式A→α进行归约。

6.3 SLR分析表的构造

二、构造SLR分析表的算法

3)若项目S` → S•属于I_k，则置ACTION[k,＃]=acc;

4)若GO(I_k，A)=I_j,A是非终结符，则置GOTO[k,A]=j;

5)分析表中凡不能用步骤1至4填入信息的空白项，均置上“出错标志”。

注：1）若文法G按上面算法构造出来的分析表不包含多重定义项，则该文法G是SLR文法。

2)二义文法决不是LR文法。

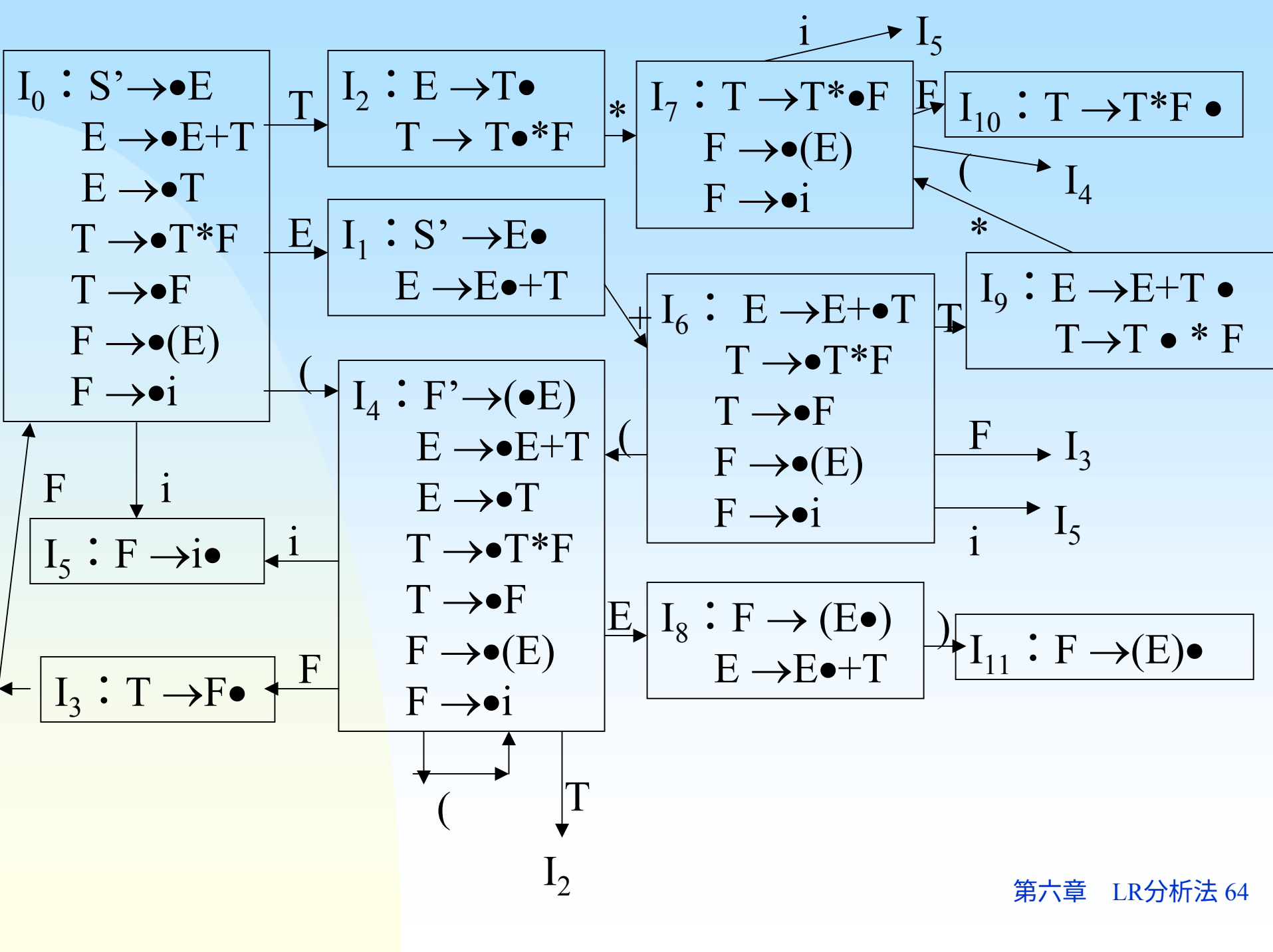
3) SLR分析法包含的展望信息是体现在利用了Follow(A)信息，可以解决“归约-归约”冲突

4) SLR分析法没有包含足够的展望信息，不能解决“移进-归约”冲突，需要改进。

例如：试构造表达式文法G(E)的SLR分析表

- 0. S` →E　　1. E → E+T
- 2. E →T　　3. T →T*F
- 4.T →F　　5. F →(E)
- 6. F → i

解：按照求LR(0)项目集规范族的算法，求得G(E)文法得项目集族如下图：


I0 ：S’→•E
E →•E+T
E →•T
T →•T*F
T →•F
F →•(E)
F →•i
T
I2 ：E →T•
T → T•*F
*
I7 ：T →T*•F
F →•(E)
F →•i
i
I5
F
I10 ：T →T*F •
(
I4
*
E
I1 ：S’ →E•
E →E•+T
+
I6 ： E →E+•T
T →•T*F
T →•F
F →•(E)
F →•i
T
I9 ：E →E+T •
T→T • * F
(
I4 ：F’→(•E)
E →•E+T
E →•T
T →•T*F
T →•F
F →•(E)
F →•i
(
F
I3
i
I5
F
i
I5 ：F →i•
i
E
I8 ：F → (E•)
E →E•+T
)
I11 ：F →(E)•
F
I3 ：T →F•
(
T
I2

构造SLR分析表时要注意项目集族中I_1,I_2,I_9三个项目集，其中含有冲突：

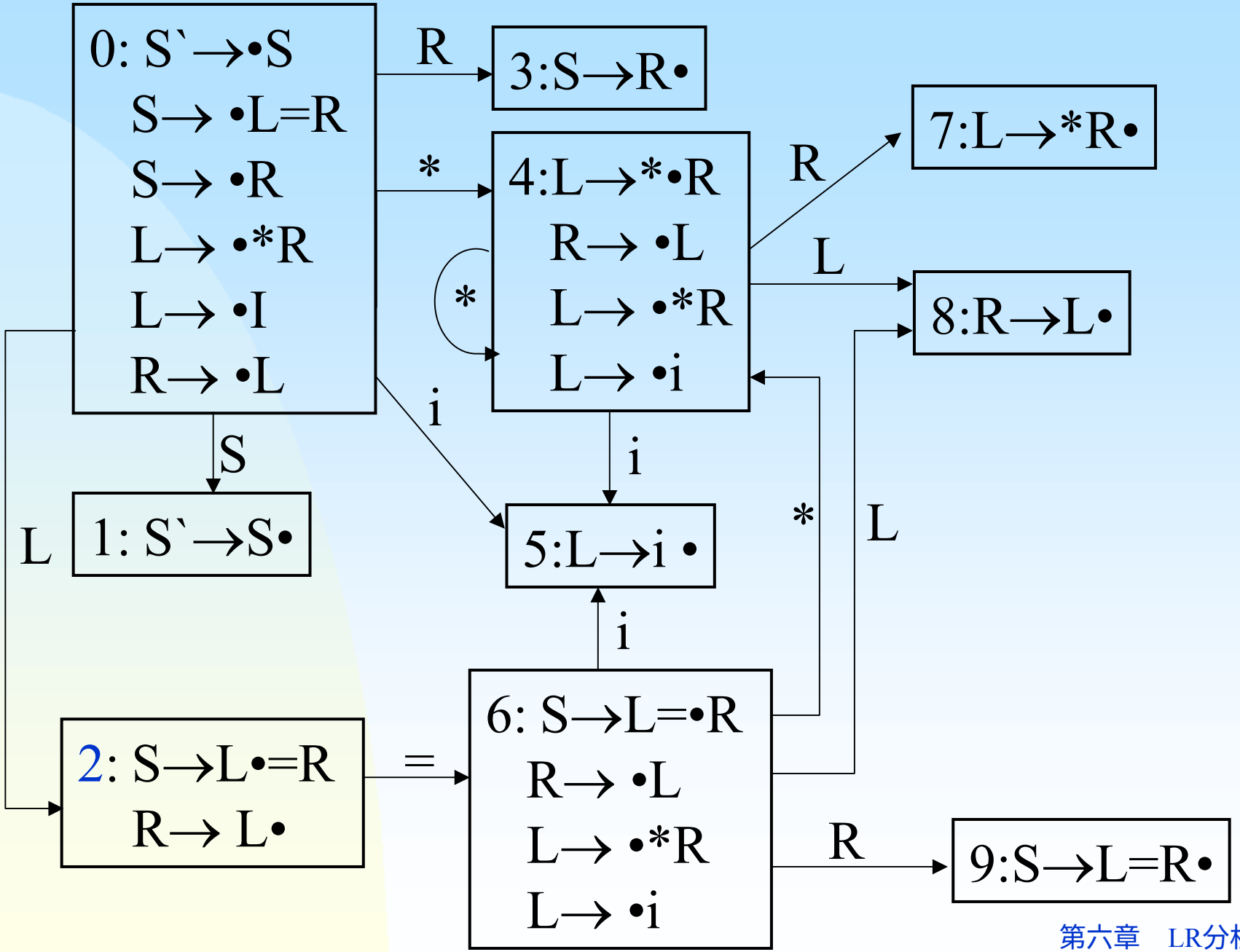
I_1 ： S` →E•　　I_2：E →T•　　I_9：E →E+T •

E → E•+T　　T → T•*F　　T→T • * F

根据原来的文法分别求S`,E的follow集，按SLR方法进行分析，得：

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4				9	3
7	S_5			S_4					10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

- 例：一个非SLR文法的例子。有如下文法:
- 1. S` →S　　2. S →L=R
- 3. S →R　　4. L →*R
- 5. L → i　　6. R →L
- 按照求LR(0)项目集规范族的算法，求得G(S)文法得项目集族如下图


0: S`→•S
S→ •L=R
S→ •R
L→ •*R
L→ •I
R→ •L
R
3:S→R•
*
4:L→*•R
R→ •L
L→ •*R
L→ •i
*
R
7:L→*R•
L
8:R→L•
i
S
i
L
1: S`→S•
5:L→i •
*
L
i
2: S→L•=R
R→ L•
=
6: S→L=•R
R→ •L
L→ •*R
L→ •i
R
9:S→L=R•

状态	ACTION				GOTO		
	=	i	*	#	S	L	R
0		S_5	S_4		1	2	3
1				acc			
2	S_6/ r_6			r_6			
3				r_3			
4		S_5	S_4			8	7
5	r_5			r_5			
6		S_5	S_4			8	9
7	r_4			r_4			
8	r_6			r_6			
9				r_2			

6.4 规范LR分析表的构造

- 对SLR分析的思考
  - 在构造SLR分析表的方法中，若项目集I_k中含有$A\rightarrow\alpha\bullet$，那么在状态k时，只要面临 输入符号$a \in Follow(A)$，就确定采用$A\rightarrow\alpha$产生式进行归约。但是，在某种情况下，当状态k呈现于栈顶时，栈里的符号串所构成的活前缀$\beta\alpha$未必允许把α归约为A。因为可能没有一个规范句型含有前缀βAa。因此此时用$A\rightarrow\alpha$产生式进行归约未必有效。
  - 我们可以从上例中看到这个问题。

6.4 规范LR分析表的构造

- 结论：
  - 由此看出，并非随符都出现在规范句型中。
- 对策：
  - 给每个LR(0)项目添加展望信息，即：添加句柄之后可能跟的终结符，因为这些终结符确实是规范句型中跟在句柄之后的。这就是LR(1)的方法。
- 可能引起的问题
  - 故，LR(1)项目是对LR(0)项目的分裂，若文法中终结符的数目为n，则每个LR(0)项目都可以分裂成n个LR(1)项目。这可能会引起分析表的膨胀。

6.4 规范LR分析表的构造

一、相关定义

- 1、LR(1)项目：形如（A→α•β,a）的二元式称为LR(1)项目。其中，A→αβ是文法的一个产生式，a是终结符，称为搜索符。
- 注：
  - 1）LR(1)项目是对LR(0)项目的分裂，若文法中终结符的数目为n，则每个LR(0)项目可以分裂成n个LR(1)项目。
  - 2）(A→α•β,a)的含义：预期当栈顶句柄αβ形成后，在读头下读到a。此时，α在栈内，β还未入栈，即它展望了句柄后的一个符号。

6.4 规范LR分析表的构造

- 2、有效项目：
  - 若存在规范推导$S` \rightarrow \delta A\omega \rightarrow \delta\alpha\beta\omega$，其中$\delta\alpha$称规范句型$\delta\alpha\beta\omega$的活前缀（记作$\gamma$），$a \in First(\omega)$，则LR(1)项目（$A \rightarrow \alpha \bullet \beta, a$）对于活前缀$\gamma$是有效的。如果$a \notin First(\omega)$，即使$a \in Follow(A)$，项目（$A \rightarrow \alpha \bullet \beta, a$）也是无效的。
- 注：1)规范LR分析法仅考虑有效的LR(1)项目。在LR(1)项目中有效的项目并不多。
- 2)对于多数程序设计语言，向前展望一个符号就足以决定归约与否，所以只研究LR(1)。

- 例：非SLR文法的例子。有如下文法:
  - 1. S` →S　　2. S →aAd
  - 3. S →bAc　　4. S →aec
  - 5. S →bed　　6. A →e
- 按照求LR(0)项目集规范族的算法，求得G(S)文法的项目集族，其中状态5中发生移进－归约冲突：

  5: S→ae•c

  A→ e•
- 由于规范推导为：S` →S →aAd→aed
  - S` →S →aec

- 这两个最右推导中已包含了活前缀为ae的所有句型，可见“aAc”决不会是规范句型，即：归约成非终结符“A”之后，其后决不会跟“c”。
- 故：虽然FOLLOW(A)＝{c,d}，但是（ A→ e•，c）对活前缀ae是无效的，仅（ A→ e•，d）是有效的。

6.4 规范LR分析表的构造

二、构造LR(1)项目集规范族的算法

1、函数CLOSURE(I)－I的项目集

(1)I的任何项目都属于CLOSURE(I)；

(2)若项目（A→α•Bβ,a）属于CLOSURE(I)， B →γ是一个产生式，那么对于FIRST(βa)中的每个终结符b，如果（B→• γ,b）原来不在CLOSURE(I)中，则把它加进去；

(3)重复步骤（2）直到CLOSURE(I)不再扩大为止。

注：因为（A→α•Bβ,a）属于CLOSURE(I)，那么(B→• γ,b)当然也属于CLOSURE(I)，其中b必定是跟在B后面的终结符，即b ∈ First(βa)。若β＝ ε，则b=a。

6.4 规范LR分析表的构造

二、构造LR(1)项目集规范族的算法

2、GO函数

- 令I是一个项目集，X是一个文法符号，函数GO(I,X)定义为： GO(I,X)=CLOSURE(J)
  - 其中 J={任何形如（$A\rightarrow\alpha X\bullet\beta,a$）的项目 | （$A\rightarrow\alpha\bullet X\beta,a$）$\in$I}
- 注：在执行转换函数GO时，搜索符并不改变。

6.4 规范LR分析表的构造

二、构造LR(1)项目集规范族的算法

3、构造拓广文法G`的LR(1)项目集族C的算法

```
PROC ITEMSET LR1
{   C={CLOSURE({(S` →•S,#)})};
      DO  {
            FOR C中的每个项目集I和G`的每个文法
                        符号X
                IF  GO(I,X)非空且不属于C
                    把GO(I,X)加入C中；
            }WHILE  C依然扩大；
}
```

- 例：有如下文法:
  - 1. S` →S　　2. S →L=R
  - 3. S →R　　4. L →*R
  - 5. L → i　　6. R →L
  - 按照求LR(1)项目集规范族的算法，求G(S)文法的项目集族
- 解：1、求初态项目集I_0：从(S` →•S,#)项目开始求闭包得：

I_0初态
S` →•S,#
S→•L=R,#
S→•R,#
L→•*R,=
L →•i,=
R→•L,#
L→•*R,#
L →•i,#

简化为

I_0初态
S` →•S,#
S→•L=R,#
S→•R,#
L→•*R,=|#
L →•i,=|#
R→•L,#

- 2、接着利用GO函数，对该项目集内得各项目求后继项目集，然后再对新求的项目集重复上面的过程，直到项目集不再增加为止。最后的LR(1)图见P113。

6.4 规范LR分析表的构造

三、构造LR(1)分析表算法

- 设C={I_0,I_1,…I_n},以各项目集I_k(k＝0，…,n)的下标k为分析表中的状态，并以包含（S`→•S，#）的项目的状态为分析表初态。按下列步骤填写ACTION表和GOTO表：

1)若项目(A→α•aβ,b)属于I_k，且GO(I_k,a)= I_j,a为终结符，则置ACTION[k,a]=S_j; 即：移进a，并转向I_j状态；

2)若项目(A→α•,a) ∈I_k，则置ACTION[k,a]=r_j ；其中j为产生式A→α的编号，即根据j号产生式A→α进行归约；

6.4 规范LR分析表的构造

三、构造LR(1)分析表的算法

3)若项目(S` → S•,#)属于I_k，则置ACTION[k,＃]=acc;

4)若GO(I_k，A)=I_j,A是非终结符，则置GOTO[k,A]=j;

5)分析表中凡不能用步骤1至4填入信息的空白项，均置上“出错标志”。

- 例如：根据书P113的LR(1)项目集族构造的规范LR(1)分析表如下：

状态	ACTION				GOTO		
	=	i	*	#	S	L	R
0		S_5	S_4		1	2	3
1				acc			
2	S_6			r_6			
3				r_3			
4		S_5	S_4			8	7
5	r_5			r_5			
6		S_{10}	S_{12}			11	9
7	r_4			r_4			
8	r_6			r_6			
9				r_2			
10				r_5			
11				r_6			
12		S_{10}	S_{12}			11	13

6.4　规范LR分析表的构造

四、LR（1）文法

- 定义：若文法G`按构造LR(1)分析表算法构造出来的分析表不包含多重定义项，则该文法G`是LR(1)文法。
- 注：1）每个SLR文法都是LR(1)文法，反之不一定成立；
- 　　2）一个SLR文法的规范LR分析表比其SLR分析表含有更多的状态。在严重的情况下，状态数可能成几倍增长。因此需要简化。

6.5 LALR分析表构造

一、基本思想

1、LALR分析法是一种简化的、具有预测能力(LookAhead)的LR(1)分析方法。是在LR(1)文法的基础上构造出来的。构造方法的基本思想是合并同心。

2、同心

若文法G`的LR(1)的两个项目集I_i和I_j在去掉各项目中搜索符之后是相同的，则称这两项目集为同心。

6.5 LALR分析表构造

一、基本思想

- 3、构造LALR分析表的基本思想
  - 合并同心，可得到一个与LR(0)状态数相同的项目集族。在合并同心的同时，以某种方式合并LR(1)分析表中的ACTION表和GOTO表的对应项，从而可以在LR(1)分析表的基础上构造一个尺寸与LR(0)分析表相同的分析表，若此表不含多重定义项，它就是LALR分析表。
- 注：
  - 仅当文法是LR(1)时，才有可能构造LALR分析表。
  - LALR分析法的分析能力比LR(1)弱，比SLR强。
  - LALR分析表的尺寸与SLR分析表相同。

6.5 LALR分析表构造

二、合并同心

合并ACTION和GOTO表的对应项。

1、合并GOTO表

- 方法：根据GO函数的定义：GO(I,X)=CLOSURE(J),其中：J={任何形如$(A\rightarrow\alpha X\bullet\beta,a)$的项目$|(A\rightarrow\alpha\bullet X\beta,a)\in I$}进行。
- 例如：设 I_m和I_n 同心，则：
  - $(A\rightarrow\alpha\bullet X\beta,a)\in I_m$ $GO(I_m,X)=>(A\rightarrow\alpha X\bullet\beta,a)$
  - $(A\rightarrow\alpha\bullet X\beta,b)\in I_n$ $GO(I_n,X)=>(A\rightarrow\alpha X\bullet\beta,b)$
- 注：根据GO函数定义，转向后，搜索符不变，即与搜索符无关，故合并之后不出现转向冲突，转向之后仍然同心。

6.5 LALR分析表构造

二、合并同心

2、合并ACTION表

- (1)出错与出错合并：结果仍为出错，无冲突；
- (2)移进与移进合并；
- (3)出错与移进合并：不会出现此情况，因为出错项目与移进项目不同心；
- (4)移进与归约合并：不会出现此情况；
- (5)归约与出错合并：规定它做归约；
  - 注：由此可见，LALR与LR(1)相比，放松了报错条件。但由于移进能力没有减弱，所以在下一个符号进栈之前总能报错，故它对错误的定向能力没减弱。

6.5 LALR分析表构造

二、合并同心

2、合并ACTION表

- (6)归约与归约合并：分两种情况讨论：
  - A) 按同一产生式归约，无冲突；
  - B) 按不同产生式归约，将造成冲突，因此LALR的能力弱于LR（1）。
- 综上所述，可得：只要合并同心之后，不存在按不同产生式的归约－归约冲突，由LR(1)项目集族总能构造出LALR分析表。

- 例如，文法
  - (0) S` →S
  - (1) S → aAd|bBd|aBe|bAe
  - (2) A →c
  - (3) B →c
  - 此文法是LR(1)文法，其项目集族不存在冲突性动作，但在合并时，将LR(1)中的两个同心项目{(A →c•,d) (B →c•,e)},{(A →c•,e) (B →c•,d)}进行合并，得：{(A →c•,d|e) (B →c•,e|d)}。当面临d或e这两个搜索符时，不知该用哪个产生式进行归约，出现了归约—归约冲突。

6.5 LALR分析表构造

三、构造LALR分析表的算法

1、构造文法G`的LR(1)项目集族C={$I_0, I_1 \dots I_n$}.

2、把所有同心集合并在一起，记作C`={$J_0, J_1 \dots J_m$}，为合并后的新族。将含有项目(S` →•S,#)的J_k作为分析表初态。

3、从C`构造ACTION表：

- （1）若 (A→α•aβ,b)∈ J_k ,且GO(J_k ， a)＝ J_j,a为终结符，则置ACTION[k,a]＝ S_j
- （2）若 (A→α•,b)∈ J_k ,则置ACTION[k,b]＝ r_j,其中j为产生式A→α的编号；

6.5 LALR分析表构造

三、构造LALR分析表的算法

3、从C`构造ACTION表：

- （3）若 $(S`\rightarrow S\bullet,\#)\in J_k$,则置ACTION[k,#]＝acc.

4、GOTO表的构造：

- 假定J_k是$I_{i1}, I_{i2} \ldots I_{it}$合并后的新集。由于这些$I_i$同心，因此，$GO(I_{i1},X), GO(I_{i2},X),\ldots. GO(I_{it},X)$ 也同心，并记为J_i.即：$GO(J_k,X)= J_i$。
- 若有$GO(J_k,A) = J_i$ ，A是非终结符，则置GOTO[k,A]=i.

5、凡是不能用上述步骤填写的空白项均填上“报错”。

- 例：有如下文法:
  - 1. S` →S　　2. S →L=R
  - 3. S →R　　4. L →*R
  - 5. L → i　　6. R →L
  - 写出此文法的LALR分析表，并根据文法的LALR分析表分析输入串“i=*i=#”
- 解：1、写出此文法的LR(1)分析表，再根据该分析表写出其LALR分析表。

状态	ACTION				GOTO		
	=	i	*	#	S	L	R
0		S_5	S_4		1	2	3
1				acc			
2	S_6			r_6			
3				r_3			
4		S_5	S_4			8	7
5	r_5			r_5			
6		S_{10}	S_{12}			11	9
7	r_4			r_4			
8	r_6			r_6			
9				r_2			
10				r_5			
11				r_6			
12		S_{10}	S_{12}			11	13

状态	ACTION				GOTO		
	=	i	*	#	S	L	R
0		S_5	S_4		1	2	3
1				acc			
2	S_6			r_6			
3				r_3			
4		S_5	S_4			8	7
5	R_5			r_5			
6		S_5	S_4			8	9
7	r_4			r_4			
8	r_6			r_6			
9				r_2			

- “i=*i=#” 的LR(1)分析过程

步骤	状态栈	符号栈	输入串
0	0	#	i=*i=#
1	0，5	#i	=*i=#
2	0，2	#L	=*i=#
3	0，2，6	#L=	*i=#
4	0，2，6，12	#L=*	i=#
5	0，2，6，12，10	#L=*i	=#
	报错		

- “i=*i=#” 的LALR(1)分析过程

步骤	状态栈	符号栈	输入串
0	0	#	i=*i=#
1	0，5	#i	=*i=#
2	0，2	#L	=*i=#
3	0，2，6	#L=	*i=#
4	0，2，6，4	#L=*	i=#
5	0，2，6，4，5	#L=*i	=#
6	0，2，6，4，8	#L=*L	=#
7	0，2，6，4，7	#L=*R	=#
8	0，2，6，8	#L=L	=#
9	0，2，6，9	#L=R	=#
	报错		

6.5 LALR分析表构造

- 注：用LR(1)方法，遇到输入串有错就立即报错；而LALR方法没有立即报错，多做了几步归约之后才报错。但它们对错误的定位是相同的，故LALR方法的报错能力没减弱。

6.5 LALR分析表构造

四、LALR(1)文法

- 定义：用上述算法构造分析表，若不存在重定义项，则此文法是LALR(1)文法。

6.6 二义文法的应用

一、定理

- 任何二义文法决不是一个LR文法，故而也不是SLR或LALR文法。
- 注：虽然二义文法不是LR文法，但有些二义文法很有用，给它加上一些限制后，它可能成为描述某种语言的最简单的文法。
- 本节讨论如何使用LR分析法的基本思想，凭借其他一些条件，来分析二义文法所定义的语言。

– 例如：G1： E` →E
– E →E+E|E*E|(E)|i
– G2： E` →E
– E →E+T|T
– T →T*F|F
– F → (E)|i

6.6 二义文法的应用

- 两个文法进行对比可以发现G1有两个优点：
  - 1、若需要改变算符的优先级或结合规则，不需要改变文法G1本身；
  - 2、文法G1的分析表所包含的状态肯定比G2的状态要少得多。因为，在文法G2中含有单个非终结符的产生式，这些产生式用来定义算符的优先级和结合的规则，它们要占用不少状态和消耗不少时间。

6.6 二义文法的应用

二、二义文法分析

使用LR分析法的基本思想，凭借其他一些条件，来分析二义文法所定义的语言。可以根据二义文法构造出LR分析表。其步骤是：

- 1、构造LR(0)分析表；
- 2、对于发生冲突的项目用SLR方法解决；
- 3、对于SLR方法解决不了的冲突借助于其它条件解决。

例如：对文法G1，构造其LR分析表。

$E` \rightarrow E$

$E \rightarrow E+E|E*E|(E)|I$

- 1、构造LR(0)分析表；书P118图6.11

- 2、用SLR方法解决部分冲突
  - 例如：状态I_1，$\begin{cases} E` \rightarrow E\bullet \\ E \rightarrow E\bullet + E \\ E \rightarrow E\bullet * i \end{cases}$
  - 其中存在接受—移进冲突，它可以用SLR方法解决。
- 3、用SLR方法解决不了的冲突
  - 例如：状态I_7，$\begin{cases} E \rightarrow E + E\bullet \\ E \rightarrow E\bullet + E \\ E \rightarrow E\bullet * E \end{cases}$
  - 其中存在归约—移进冲突。

状态	ACTION						GOTO
	i	+	*	(	)	#	S
0	S_3			S_2			1
1		S_4	S_5			acc	
2	S_3			S_2			6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			7
5	S_3			S_2			8
6		S_4	S_5		S_9		
7		r_1/S_4	S_5/r_1		r_1	r_1	
8		r_2/S_4	r_2/S_5		r_2	r_2	
9		r_3	r_3		r_3	r_3	

- 此时，只能采取加入附加条件的办法。
  - 对状态7，遇到读头下为“*”，究竟应该先做加法的归约呢还是应该先做乘号的移进，由于我们认为乘法的优先级高于加法，所以这里应该做乘法的移进；
  - 对状态7，遇到读头下为“+”，究竟应该先做加法的归约呢还是应该先做加号的移进，由于我们认为相同优先级的算符服从左结合，所以这里应该做加法的归约；

状态	ACTION						GOTO
	i	+	*	(	)	#	S
0	S_3			S_2			1
1		S_4	S_5			acc	
2	S_3			S_2			6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			7
5	S_3			S_2			8
6		S_4	S_5		S_9		
7		$r_1(S_4)$	$S_5(r_1)$		r_1	r_1	
8		$r_2(S_4)$	$r_2(S_5)$		r_2	r_2	
9		r_3	r_3		r_3	r_3	

6.6 二义文法的应用

- 注：采用附加条件之后，发生冲突的表元素就只留下了一种操作；
- 在上面的规则中，我们是认为“*”的优先级大于“+”，若现在规则发生变化，认为“+” 的优先级大于“*”，那么对它的实现根本不需修改文法，只要处理冲突时改变一下就行。用这种方法改变算符的优先级是非常方便的。

- “i+i*(i)#” 的LR分析过程

步骤	状态栈	符号栈	输入串
0	0	#	i+i*(i)#
1	0，3	#i	+i*(i)#
2	0，1	#E	+i*(i)#
3	0，1，4	#E+	i*(i)#
4	0，1，4，3	# E+ i	*(i)#
5	0，1，4，7	#E+ E	*(i)#
6	0，1，4，7，5	#E+ E*	(i)#
….	……	……	……

6.7 分析表的自动生成-YACC

一、基本含义

- YACC(Yet Another Compiler-Compiler)的功能：

二、基本功能

- 接受用户提供的文法（可能是二义性的）、优先级、结合性质等附加信息，自动产生这个文法的LALR分析表。
- 注：有些YACC甚至可包括接受语义描述和目标机器描述，并完成源语言到目标代码的翻译。

6.7　分析表的自动生成

三、YACC的基本思想：

- 对用户提供的文法，YACC将首先产生它的LALR(1)状态，然后试图为每个状态选择适当的分析动作。如果不存在冲突，就得到了LALR分析表；若存在冲突，则利用用户提供的附加信息解决冲突。

6.7 分析表的自动生成

四、终结符和产生式的优先级

1、YACC解决移进－归约冲突的基本方法

- 赋予每个终结符和产生式以一定的优先级。
- 假定读头下为a时遇到移进－归约冲突(其中归约是对$A\rightarrow\alpha$进行归约)，就比较终结符a和产生式$A\rightarrow\alpha$的优先级：若产生式优先级高就执行归约，若a优先级高就执行移进。
- 注：并不是所有的项目集都会引起冲突，不涉及冲突的动作应不理睬终结符和产生式的优先级信息。此时若$A\rightarrow\alpha$没有特别赋予优先级，则$A\rightarrow\alpha$的优先级与出现在α中的最右终结符的优先级相同。

6.7 分析表的自动生成

- 例如：文法： S→iSeS|iS|a，这是一个二义性文法，它的项目集族如P120,图6.12

- 其中I_4: $\begin{cases} S \rightarrow iS \bullet eS \\ S \rightarrow iS \bullet \end{cases}$
  - 遇到移进－归约冲突。其中S→iS•的优先级与i相同。
- 若规定e的优先级高于i,则可以解决冲突。

6.7 分析表的自动生成

五、结合规则

1、结合规则的作用

– 许多算符具有相同的优先级，所以只给出终结符和产生式的优先级还不足以解决所有的冲突。只有给定了相同优先级算符的结合规则，这些冲突才能解决。

- 例如：二义性文法 $E \rightarrow E+E|E*E|(E)|I$
  - 项目集I_7，$\begin{cases} E \rightarrow E+E\bullet \\ E \rightarrow E\bullet+E \\ E \rightarrow E\bullet*E \end{cases}$
  - 其中存在归约－移进冲突

6.7 分析表的自动生成

六、YACC的规格说明

- 定义部分：
  - 用来说明文法节和程序中所使用的变量和函数类型；
- 文法节：
  - 由文法的各产生式和相关语义动作组成；
- 可选的用户子程序：
  - 辅助过程，由一些C语言函数组成，包含词法分析过程、出错过程等。

- 例如：二义性文法　E →E+E| E-E| E*E| E/E|(E)|i的YACC规格说明：
- %{　#include <ctype.h>
- 　　#include <stdio.h>　%}　　/*C语言程序正规说明*/
- Token　i
- %　left ‘*’’/’
- %　left ‘+’’-’
- %%　　/*定义部分*/
- Line:　expr’\n’{print(“%d\n”,$1);}/*再YACC源程序中，我们加入了一个新开始产生式，规定表达式后跟一换行符，其语义动作是打印相应非终结符值*/
- Expr:expr’+’expr　{$$=$1+$3;}
- 　　| expr’-’expr　{$$=$1-$3;}……

```
    |i
    ;
%%   /*文法节*/
Yylex(){
        int c;
        c=getchar();
        if (isdigit( c)){
            yylval=c-’0’;
            return I;
              }
          return c;}  /*子程序，词法分析程序*/
```

6.7 分析表的自动生成

- 移进—归约冲突解决：
  - 根据终结符的优先级与结合规则
- 归约—归约冲突解决：
  - 优先使用列在前面的产生式进行归约

小结

1)SLR分析表构造

除了历史，还考虑当前输入符号。

通过First和Follow，解决冲突。

2)规范LR分析表构造

除了历史、当前符号，还考虑展望信息——确实是规范句型中跟在句柄之后的终结符

LR(1)分析表构造。

3)LALR分析表构造

基于LR(1)，合并同心项

4)二义文法的应用

根据附加信息解决冲突。

附加信息：结合规则和优先级