课时5 语法分析-自下而上

高数帮

考点	重要程度	占分	题型
1.短语、直接短语、句柄、素短语、最左素短语的概念及判定。	★★★	5～8	问答题
2.算符文法和算符优先文法的概念	★★★★	3～5	简答题
3.算符优先分析算法的步骤	★★★★★	10～20	问答题
4.LR分析法的概念及特点	★★★	8～10	问答题
5.LR(0)分析法的流程	★★★★★	10～20	问答题
6.SLR(1)分析法的流程	★★★★★	10～20	问答题
7.LR(0)和SLR(1)的对比	★★★★	8～10	问答题

一、 自下而上分析法(Bottom-up)

基本思想：

从输入串开始，“归约”，直到文法的开始符号。所谓归约，是指根据文法的产生式规则，把产生式的右部替换成左部符号。

从语法树的角度看：从语法树的树叶开始，逐步向上归约构造分析树，直到形成根结点。是推导的逆过程。

算符优先分析法：按照算符的优先关系和结合性质进行语法分析。适合分析表达式。

LR分析法：规范归约

1.1 自下而上分析基本问题

（1）规约

采用“移进—归约”思想进行自下而上分析。

基本思想：从输入符号串开始，从左到右进行扫描，将输入符号逐个移入一个栈中，边移入边分析，一旦栈顶符号串形成某个产生式的右部时，就用该产生式的左部非终结符代替，称为归约。重复这一过程，直到归约到栈中只剩下文法的开始符号时，则分析成功，称为**“移进-归约”方法**。

【题1】：设文法G(S)：

(1) S → aAcBe

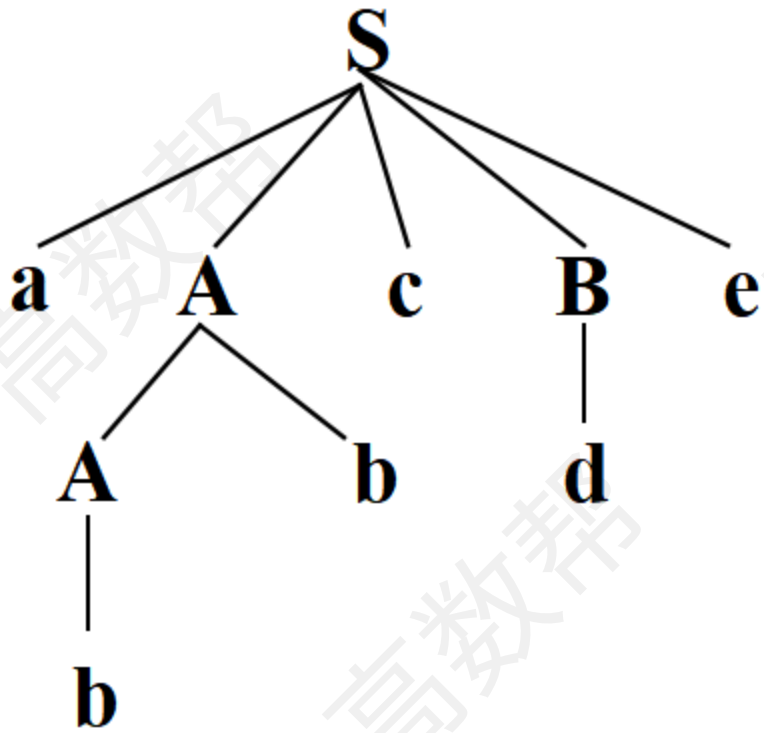
(2) A → b

(3) A → Ab

(4) B → d

试对abbcde进行“移进－归约”分析。

视频讲解更清晰



自下而上分析过程：边输入单词符号，边归约。

核心问题：识别可归约串

1.2 规范归约

1.定义：令G是一个文法，S是文法的开始符号，假定αβδ是文法G的一个句型，如果有

$$S \overset{*}{\Rightarrow} \alpha A \delta \text{ 且 } A \overset{+}{\Rightarrow} \beta$$

则β称是句型αβδ相对于非终结符A的短语。

特别是，如果有$A \Rightarrow \beta$,则称β是句型αβδ相对于规则$A \rightarrow \beta$的直接短语。

一个句型的最左直接短语称为该句型的句柄。

考虑文法G(E)： $E \to T \mid E+T$

$T \to F \mid T*F$

$F \to (E) \mid i$

和句型$i_1*i_2+i_3$：

$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+i_3 \Rightarrow T+i_3$

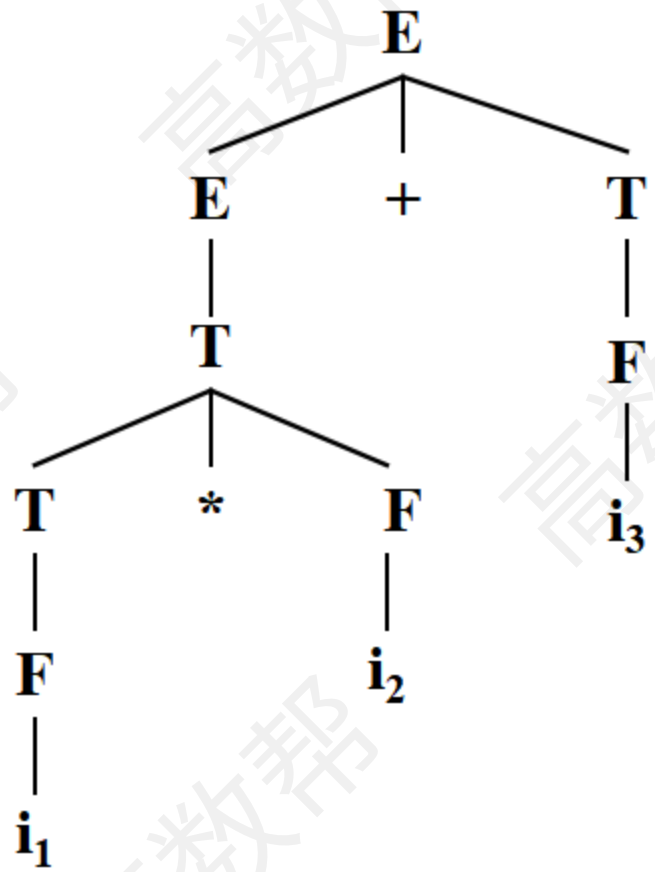
$\Rightarrow T*F+i_3 \Rightarrow T*i_2+i_3 \Rightarrow F*i_2+i_3$

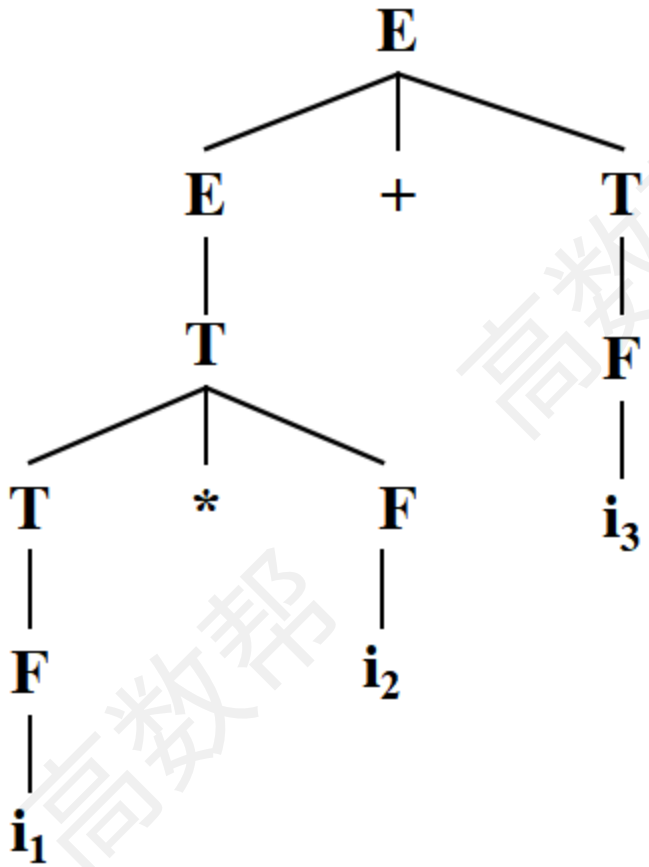
$\Rightarrow i_1*i_2+i_3$

短语： i_1，i_2，i_3，i_1*i_2，$i_1*i_2+i_3$

直接短语： i_1，i_2，i_3

句柄： i_1





短语： i_1，i_2，i_3，i_1*i_2，$i_1*i_2+i_3$

直接短语： i_1，i_2，i_3

句柄： i_1

1.短语： 在一个句型对应的语法树中，以某非终结符为根的一棵子树的所有叶子自左至右排列起来形成一个相对于子树根的短语。

2.直接短语： 仅有父子两代的一棵子树，它的所有叶子自左至右排列起来所形成的符号串。

3.句柄： 一个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列。

1.2 规范归约

2.定义：假定α是文法G的一个**句子**，我们称序列

$$\alpha_n,\ \alpha_{n-1},\ \ldots,\ \alpha_0$$

是一个规范归约，如果此序列满足：

1.$\alpha_n = \alpha$

2.α_0为文法的开始符号，即$\alpha_0 = S$

3.对任何i，$0 \le i \le n$，α_{i-1}是从α_i经把句柄替换成为相应产生式左部符号而得到的。

把上例倒过来写，则得到：

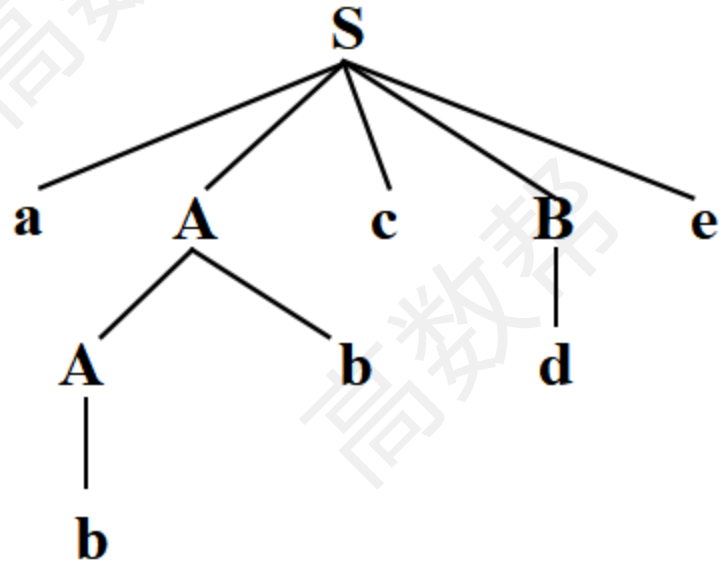
S ⇒ aAcBe⇒ aAcde ⇒ aAbcde ⇒ abbcde

显然这是一个最右推导。

规范归约是一个**最右推导**的逆过程

最左归约　　**规范推导**

由规范推导推出的句型称为**规范句型**。



1.3 符号栈的使用和分析树的表示

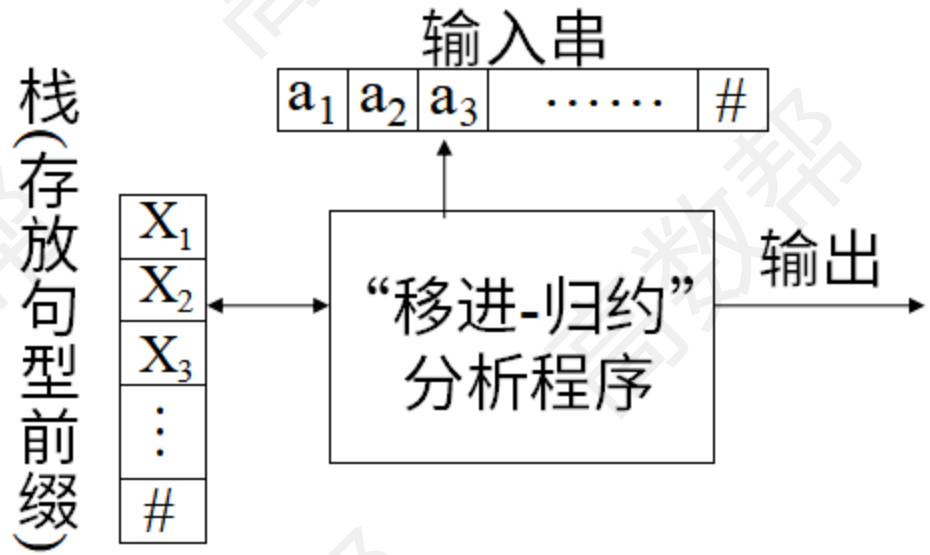
栈是语法分析的一种基本数据结构。’ #’作为栈底符号
移进-归约分析器使用了一个分析栈和一个输入缓冲区。

1.句型表示

一般形式	分析栈的内容	剩余输入串
初态：	#	输入串#
终态：	#S	#

即：分析栈内容 + 输入缓冲区内容 = # 当前句型 #

2.分析器结构



3. “移进- 归约”分析法的栈实现

“移进–归约”分析器使用一个栈和一个存放输入符号串w的缓冲器。分析器的**初始状态**为:

栈	输入
#	w#

工作过程：自左至右把串w 的符号一一移进栈里，一旦栈顶形成可归约的串（如句柄）时，就进行归约。这种归约可能持续多次，直至栈顶不再呈现句柄为止。然后，继续向栈里移进符号，重复这个过程，直至最终形成如下格局：

栈	输入
#S	#

【题2】考虑文法G(E)：

$E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow (E) \mid I$

输入串为$i_1*i_2+i_3$，分析步骤为：

步骤	符号栈	输入串	动作
0	#	$i_1*i_2+i_3$#	预备
1	#i_1	$*i_2+i_3$#	移进
2	#F	$*i_2+i_3$#	归约，用F→i
3	#T	$*i_2+i_3$#	归约，用T→F
4	#T*	i_2+i_3#	移进

G(E)：

$E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow (E) \mid i$

步骤	符号栈	输入串	动作
4	#T*	$i_2+i_3\#$	移进
5	#T*i_2	$+i_3\#$	移进
6	#T*F	$+i_3\#$	归约，用F→i
7	#T	$+i_3\#$	归约，用T→T*F
8	#E	$+i_3\#$	归约，用E→T
9	#E+	$i_3\#$	移进

G(E)：

$E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow (E) \mid i$

步骤	符号栈	输入串	动作
9	#E+	i3#	移进
10	#E+i3	#	移进
11	#E+F	#	归约，用F→i
12	#E+T	#	归约，用T→F
13	#E	#	归约，用E→E+T
14	#E	#	接受

分析器的四种动作

1) 移进：读入下一个输入符号并把它下推进栈。

2) 归约：当栈顶符号串形成一个可归约的串（如：句柄）时，直接进行归约，即用产生式左侧的非终结符替换栈顶的句柄。

3) 接受：当栈底只有“#”和开始符号，而输入也已经到达右端标志符号“#”时，识别出符号串是句子，执行该动作，表示分析成功，是归约的一种特殊情况。

4) 出错：栈顶的内容与输入符号相悖，即当识别程序发现输入符号串不是句子时，进行出错处理。

注意：决定移进和归约的依据是什么？

栈顶是否出现了可归约的符号串。

关键：如何识别可归约的符号串？

通过不同的自下而上的分析算法来解释，不同的算法对可归约串的定义是不同的，但分析过程都有一个共同的特点：边移进边归约

- 规范归约：使用句柄来定义可归约串
- 算符优先：使用最左素短语来定义可归约串

【题3】

有文法如下:

（1）E→E+T|T

（2）T→T*F|F

（3）F→(E)|id

分析输入串 id+id*id，给出对该句子进行“移进-归约”语法分析的过程。

(1)E→E+T|T

(2)T→T*F|F

(3)F→(E)|id

步骤	栈	输入缓冲区	动作
1）	#	id+id*id#	移进
2）	#id	+id*id#	归约 F→id
3）	#F	+id*id#	归约 T→F
4）	#T	+id*id#	归约 E→T
5）	#E	+id*id#	移进
6）	#E+	id*id#	移进
7）	#E+id	*id#	归约F→id
8）	#E+F	*id#	归约 T→F
9）	**#E+T**	***id#**	**移进**
10）	#E+T*	id#	移进
11）	#E+T*id	#	归约 F→id
12）	**#E+T*F**	**#**	**归约 T→T*F**
13）	#E+T	#	归约 E→E+T
14）	#E	#	接受

移进-归约分析过程中存在的问题

①移进-归约冲突

在分析到某一步时，既可以移进，又可以归约。

上题第9步可以移进*，也可以按产生式E→E+T进行归约。

②归约-归约冲突

存在两个可选的句柄可对栈顶符号进行归约。

例如上述第12)步，可以用T→F进行归约，又可以按T→T*F进行归约。

各种分析方法中处理冲突的技术不同

二、算符优先分析

1.四则运算的优先规则：

先乘除后加减，同级从左到右

2.考虑二义文法G(E)：

$$G(E)：\quad E \rightarrow i|\ E+E|E-E|E*E|E/E|(E)$$

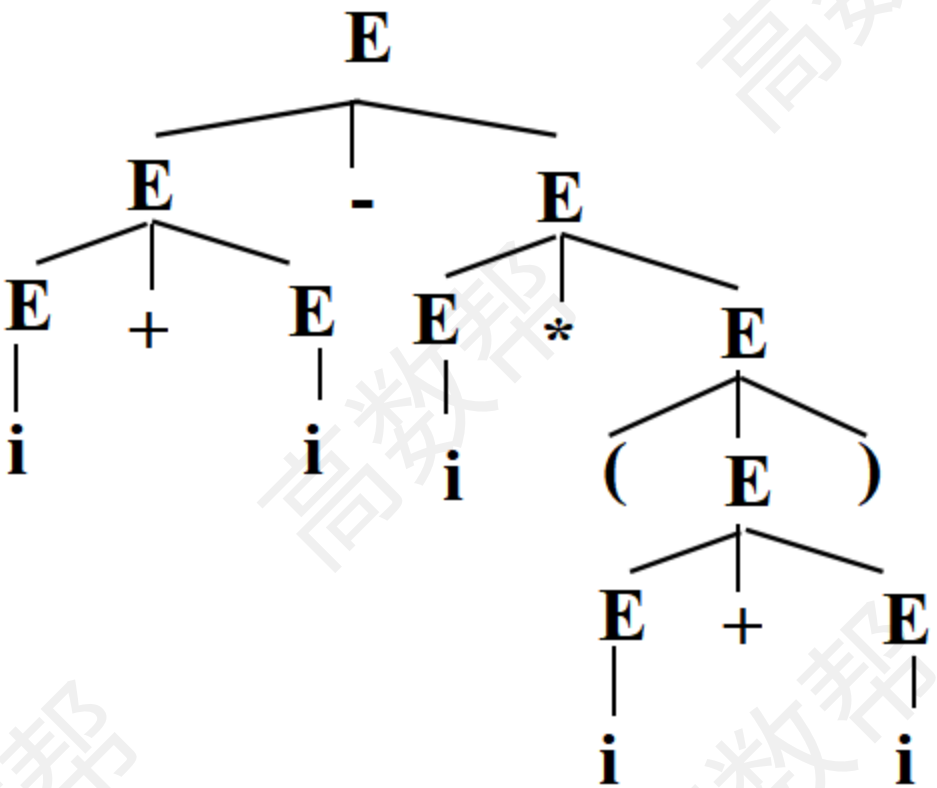
它的句子有几种不同的规范归约

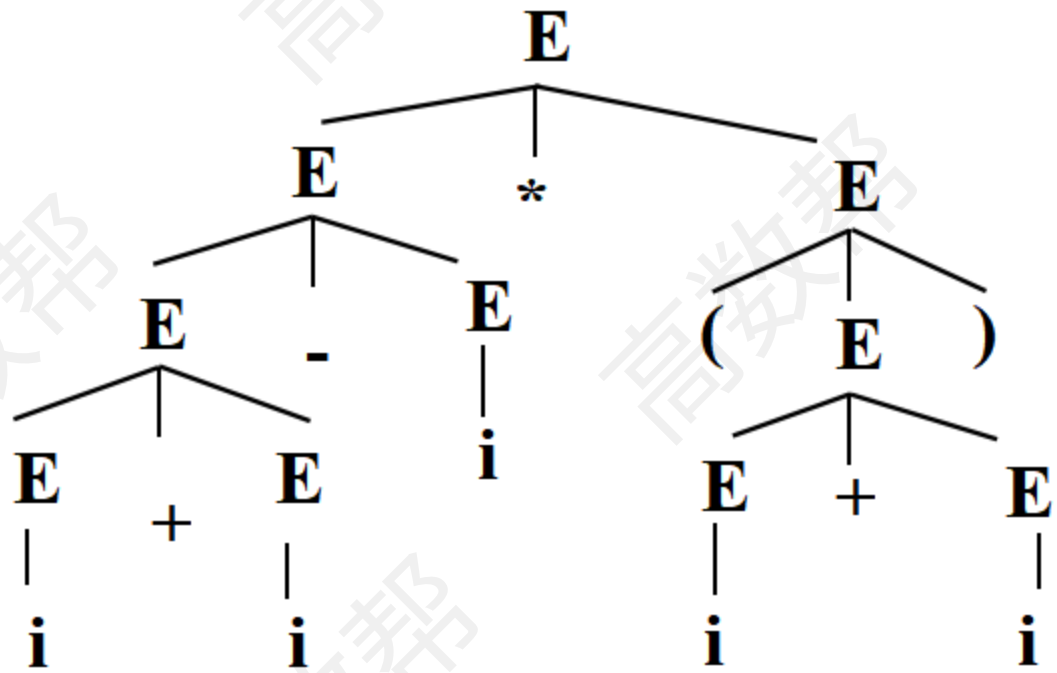
3.归约即计算表达式的值。归约顺序不同，则计算的顺序也不同，结果也不一样。

4.如果规定算符的优先次序，并按这种规定进行归约，则归约过程是唯一的。

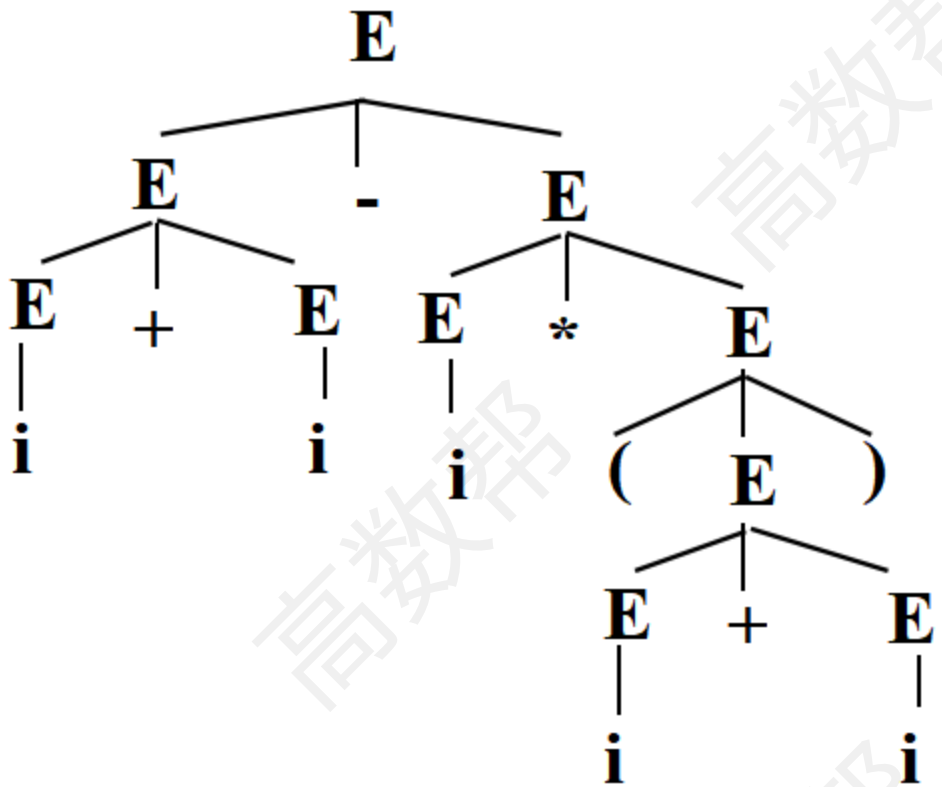
【题4】句子i+i-i*(i+i)

G(E)： E → i| E+E|E-E|E*E|E/E|(E)





句子i+i-i*(i+i)的归约过程是：



(1) i+i-i*(i+i)

(2) E+i-i*(i+i)

(3) E+E-i*(i+i)

(4) E-i*(i+i)

(5) E-E*(i+i)

(6) E-E*(E+i)

(7) E-E*(E+E)

(8) E-E*(E)

(9) E-E*E

(10) E-E

(11) E

算符优先分析中：

1.起决定作用的是相邻的两个算符之间的优先关系。

2.所谓算符优先分析法就是定义算符之间的某种优先关系，借助于这种关系寻找“可归约串”和进行归约。

3.算符优先分析法的思想源于表达式的分析，利用相邻终结符号之间的关系来寻找可归约串。

4.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型的“可归约串”。

首先必须定义任何两个可能相继出现的终结符a与b的优先关系

三种关系

$a \lessdot b$ a的优先级低于b

$a \doteq b$ a的优先级等于b

$a \gtrdot b$ a的优先级高于b

注意：与数学上的<>=不同

$a \doteq b$并不意味着$b \doteq a$，$a \doteq b$并不意味着$b \doteq a$。

2.1 算符优先文法及优先表构造

定义：

一个文法，如果它的任一产生式的右部都不含两个相继(并列)的非终结符，即不含如下形式的产生式右部：

$$\ldots QR\ldots$$

则我们称该文法为算符文法。

约定：

a、b代表任意终结符；

P、Q、R代表任意非终结符；

‘…’代表由终结符和非终结符组成的任意序列，包括空字。

假定G是一个不含ε-产生式的算符文法。对于任何一对终结符a、b，我们说：

1. $a \doteq b$ 当且仅当文法G中含有形如P→…ab…或P→…aQb…的产生式；

2. $a \lessdot b$ 当且仅当G中含有形如P→…aR…的产生式，而$R \overset{+}{\Longrightarrow} b\ldots$或$R \overset{+}{\Longrightarrow} Qb\ldots$；

3. $a \gtrdot b$ 当且仅当G中含有形如P→…Rb…的产生式，而$R \overset{+}{\Longrightarrow} \ldots a$或$R \overset{+}{\Longrightarrow} \ldots aQ$

如果一个算符文法G中的任何终结符对(a，b)至多只满足下述三关系之一：

$a \doteq b$，$a \lessdot b$，$a \gtrdot b$

则称G是一个算符优先文法。

【题5】考虑下面的文法G(E)：

(1) E→E+T | T

(2) T→T*F | F

(3) F→P↑F | P

(4) P→(E) | i

由第(4)条规则，有 '(' = ')'；

由规则E→E＋T和T⇒T*F，有 ＋ ⋖ *；

由(2) T→T*F 和(3) F→P↑F，可得* ⋖ ↑；

由(1)E→E＋T和E ⇒ E+T，可得+ ⋗ +；

由(3)F→P↑F和F ⇒ P↑F，可得↑ ⋖ ↑。

由(4)P→(E)和 E⇒E+T⇒T+T⇒T*F+T⇒F*F+T

⇒P↑F*F+T⇒i↑F*F+T

有 (⋖ +、(⋖ *、(⋖ ↑和(⋖ i

2.1 算符优先文法及优先表构造

简单算符优先关系表如下

	+	−	*	/	↑	(	)	i	#
+	>	>	<	<	<	<	>	<	>
−	>	>	<	<	<	<	>	<	>
*	>	>	>	>	<	<	>	<	>
/	>	>	>	>	<	<	>	<	>
↑	>	>	>	>	<	<	>	<	>
(	<	<	<	<	<	<	=	<	
)	>	>	>	>	>		>		>
i	>	>	>	>	>		>		>
#	<	<	<	<	<	<		<	=

2.1 算符优先文法及优先表构造

从算符优先文法G构造优先关系表

通过检查G的每个产生式的每个候选式，可找出所有满足a $\doteq$ b的终结符对

确定满足关系 $\lessdot$ 和 $\gtrdot$ 的所有终结符对：

首先需要对G的每个非终结符P构造两个集合FIRSTVT(P)和LASTVT(P)：

$a \lessdot b$　　当且仅当G中含有形如$P \to \ldots aR \ldots$的产生式，而$R \Longrightarrow b\ldots$或$R \Longrightarrow Qb\ldots$；

$a \gtrdot b$ 当且仅当G中含有形如$P \to \ldots Rb \ldots$的产生式，而$R \Longrightarrow \ldots a$或$R \Longrightarrow \ldots aQ$

有了这两个集合之后，就可以通过检查每个产生式的候选式确定满足关系$\lessdot$ 和 $\gtrdot$ 的所有终结符对。

假定有个产生式的一个候选形为

$$\ldots aP\ldots$$

那么，对任何$b\in FIRSTVT(P)$，有 $a \lessdot b$

假定有个产生式的一个候选形为

$$\ldots Pb\ldots$$

那么，对任何$a\in LASTVT(P)$，有 $a \gtrdot b$

按其定义，可用下面两条规则来构造集合FIRSTVT(P)：

1. 若有产生式P→a…或P→Qa…，则a∈FIRSTVT(P)；

2. 若a∈FIRSTVT(Q)，且有产生式P→Q…，则a∈FIRSTVT(P)。

按其定义，可用下面两条规则来构造集合LASTVT(P)：

1. 若有产生式P→… a或P→ … aQ，则a∈ LASTVT(P)；

2. 若a∈ LASTVT(Q)，且有产生式P→… Q ，则a∈ LASTVT(P)。

【题6】文法G[E]:

E→E+T|T

T→T*F|F

F→(E)|i

试构造其算符优先表。

1.构造FIRSTVT集。

(1)根据规则①可知:

由E→E+…得FIRSTVT(E)={+};

由T→T*…得FIRSTVT(T)={*};

由F→(…和F→i得FIRSTVT(F)={(,i};

(2)根据规则②可知:

由FIRSTVT(F)={(,i}和T→F得FIRSTVT(T)={*,(,i};

由FIRSTVT(T)={*,(,i}和E→T得FIRSTVT(E)={+,*,(,i}。

E→E+T | T

T→T*F | F

F→ (E) | i

2.构造LASTVT集。

(1)根据规则①可知：

由E→…+T得LASTVT(E)={+};

由T→…*F得LASTVT(T)={*};

由F→…)和F→i得LASTVT(F)={),i};

(2)根据规则②可知：

由LASTVT(F)={),i}和T→F得LASTVT(T)={*,),i};

由LASTVT(T)={*,),i}和E→T得LASTVT(E)={+,*,),i}。

3.构造优先关系表。

(1)根据规则①可知,由“(E)”得（ $\doteq$ ）。

(2)根据规则②可知：

由E→…+T得+ $\lessdot$ FIRSTVT(T)，即+ $\lessdot$ {*,(,i};

由T→…*F得* $\lessdot$ FIRSTVT(F)，即* $\lessdot$ {(,i};

由F→(E…得($\lessdot$ FIRSTVT(E)，即($\lessdot$ {+,*,(,i};

(3)根据规则③可知：

由E→E+…得LASTVT(E) $\gtrdot$ +，即{+,*,),i} $\gtrdot$ +;

由T→T*…得LASTVT(T) $\gtrdot$ *，即{*,),i} $\gtrdot$ *;

由F→…E)得LASTVT(E) $\gtrdot$)，即{+,*,),i} $\gtrdot$)。

此外，由#E#得# $\doteq$ #;

$\lessdot$ FIRSTVT(E)，即# $\lessdot$ {+,*,(,i};

LASTVT(E) $\gtrdot$ #，即{+,*,),i} $\gtrdot$ #。

得G[E]的算符优先关系表：

2.2 算符优先分析算法

算符优先分析法的设计

通过比较终结符间的优先关系来实现；

根据优先分析的原理：语法分析程序的任务是不断移进输入符号，识别可归约串并进行归约。

问题：如何识别可归约的串？

分析的基本方法：根据优先关系“高于”来识别可归约串的尾，根据优先关系“低于”来识别可归约串的头。

分析栈存放已识别部分，比较栈顶和下一输入符号的关系，如果是可归约串的尾，则沿栈顶向下寻找可归约串的头，找到后弹出可归约串，归约为非终结符。

注:各种优先关系已经存于优先关系表中。

可归约串，句型，短语，直接短语，句柄，规范归约。

一个文法G的句型的素短语是指这样一个短语，它至少含有一个终结符，并且，除它自身之外不再含任何更小的素短语。

最左素短语是指处于句型最左边的那个素短语。

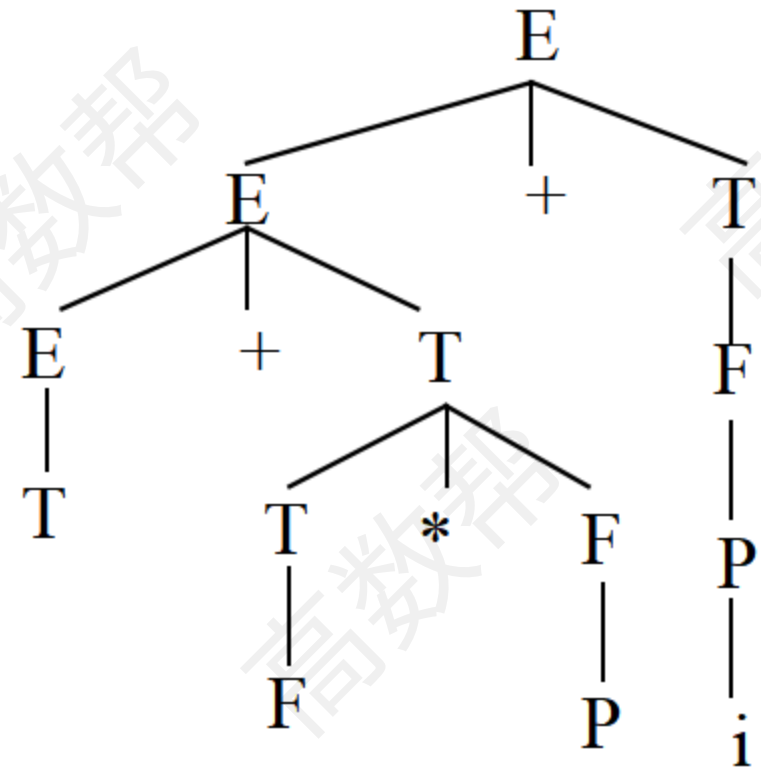
【题7】考虑下面的文法G(E)：

(1) E→E+T | T

(2) T→T*F | F

(3) F→P↑F | P

(4) P→(E) | i



句型：T+F*P+i

短语：T，F，P，i，F*P，T+F*P，T+F*P+i

直接短语：T，F，P，i

句柄：T

素短语：F*P，i

最左素短语：F*P

算符优先文法句型(括在两个＃之间)的一般形式写成：

$$\#N_1a_1N_2a_2\ldots N_na_nN_{n+1}\#$$

其中，每个a_i都是终结符，N_i是可有可无的非终结符。

定理：一个算符优先文法G的任何句型的最左素短语是满足如下条件的最左子串 $N_ja_j\ldots N_ia_iN_{i+1}$，

$$a_{j-1} \lessdot a_j$$

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

$$a_i \gtrdot a_{i+1}$$

算符优先分析过程小结:

1.句型的一般形式: $\#N_1a_1N_2a_2...N_na_nN_{n+1}\#$

(其中a_i为终结符,N_i为可有可无的非终结符)

2.从左向右扫描各符号,依次查看算符优先表,一直移进，直至找到满足$a_i > a_{i+1}$的终结符为止。再从a_i开始往左扫描,直至找到满足关系$a_{j-1} \lessdot a_j$的终结符为止,进行归约。

3.此时,形如$N_j\ a_j\ N_{j+1}\ a_{j+1}.....N_i\ a_i\ N_{i+1}$的子串即为最左素短语，应被归约。

4.分析过程的结束：分析栈中为#N，输入串为#。

算符优先分析一般并不等价于规范归约。

算符优先分析法特点：

优点: 简单，快速

缺点: 可能错误接受非法句子，能力有限.

【题8】已知文法G[E]和优先关系表如下，G[E]:

$E \rightarrow E+T|T$

$T \rightarrow T*F|F$

$F \rightarrow (E)|i$

试给出输入串i+i*i的算符优先分析过程。

视频讲解更清晰

i+i*i算符优先分析过程

符号栈	输入串	动作
#	i+i*i#	#<· i，移进
#i	+i*i#	#<· i·>+，归约
#F	+i*i#	#<· +，移进
#F+	i*i#	#<· +<· i，移进
#F+i	*i#	#<· +<· i·>*，归约

i+i*i算符优先分析过程（续）

#F+F	*i#	#<· +<· *，移进
#F+F*	i#	#<· +<· *<· i，移进
#F+F*i	#	#<· +<· *<· i·>#，归约
#F+F*F	#	#<· +<· * ·>#，归约
#F+T	#	#<· + ·>#，归约
#E	#	#E#，结束

注意:

1.算符优先分析只关心句型中自左向右的终结符序列的优先关系,不涉及终结符之间可能存在的非终结符,即可认为这些非终结符是同一个符号。

2.算符优先分析比规范归约要快，跳过了所有形如P→Q的产生式所对应的归约步骤。（这也有可能会出错）

三、LR分析法

LR分析法：是另一种有效的自底向上的分析方法，仍然是一种“移进-归约”分析方法， 1965年 由Knuth提出。

L——从左向右扫描输入串（read from Left to right）；

R——构造最右推导的逆过程（for constructing a Rightmost derivation in reverse）。

大多数用上下文无关文法描述的高级语言的语法成分可以用LR分析器来识别。

LR分析根据当前分析栈中的符号串和向右顺序查看输入串的K(K≥0)个符号就可以唯一确定分析的动作是移进还是归约。

LR分析法的优缺点

优点：

1.比其他“移进-归约”分析法，如算符优先分析法使用更加广泛，识别效率高。

2.能用LL(1)分析法分析的所有文法都能使用LR方法来进行分析。

3.LR分析法在自左至右扫描输入串的过程中就能发现其中的任何错误，并能准确地指出出错位置。

缺点：

手工构造分析表工作量太大。必须使用自动生成器。如使用专用工具YACC。

LR分析法的关键问题及解决方案

1.自底向上分析法的关键问题是如何确定可归约的串。

LR分析法与算符优先方法一样，LR方法是通过求句柄逐步归约来进行语法分析。

2.在算符优先分析中，通过算符的优先关系得到可归约串（最左素短语），

LR方法中句柄是通过求活前缀而求得。

视频讲解更清晰

3.1 LR分析器

1.规范归约的关键问题是寻找**句柄**.

LR分析方法的**基本思想**是：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(历史)，另一方面又根据所用产生式推测未来可能碰到的输入符号(对未来的展望)。

当某一符号串类似于句柄出现在栈顶时，需要根据已记载的“历史”、“展望”和“现实”的输入符号三方面的内容来决定栈顶的符号串是否构成了真正的句柄，是否 需要归约。

2.“历史”：已移入符号栈的内容

3.“展望”：根据产生式推测未来可能遇到的输入符号

4.“现实”：当前的输入符号

LR分析器结构

文法符号：X1X2…Xm是目前已移进并归约出的句型部分。其实它是多余的，已经概括到状态里。

状态栈：(S0,#)为预先放到栈中的初始状态和符号。

一个LR分析器由3个部分组成：

1.LR分析程序：又称总控程序。所有的LR分析器都是相同的。

2.分析表(分析函数)：不同的文法分析表不同，同一个文法采用的LR分析器不同时，分析表也不同，分析表又可分为动作表(ACTION)和状态转换(GOTO)表两个部分，它们都可用二维数组表示。

3.分析栈：包括文法符号栈和相应的状态栈，它们均是先进后出栈。

LR分析器的核心是一张分析表：

ACTION[s，a]：当状态s面临输入符号a时，应采取什么动作

GOTO[s，X]：状态s面对文法符号X时，下一状态是什么

每一项ACTION[s，a]所规定的四种动作:

1. **移进** 把(s，a)的下一状态s'和输入符号a推进栈，下一输入符号变成现行输入符号.
2. **归约** 指用某产生式A→β进行归约. 假若β的长度为r， 归约动作是，去除栈顶r个项，使状态s_{m-r}变成栈顶状态，然后把(s_{m-r}, A)的下一状态s'=GOTO[s_{m-r}, A]和文法符号A推进栈.
3. **接受** 宣布分析成功，停止分析器工作.
4. **报错**

LR分析过程

分析开始时:

状态	已归约串	输入串
$(s_0,$	$\#,$	$a_1a_2 \ldots a_n \#)$

以后每步的结果可以表示为:

$(s_0\ s_1 \ldots s_m\ ,\ \# X_1 \ldots X_m\ ,\ a_ia_{i+1} \ldots a_n \#)$

LR分析过程举例

设文法为G[E]：

(1) E →E+T
(2) E →T
(3) T →T*F
(4) T →F
(5) F →(E)
(6) F →i

为了介绍LR分析过程，直接给出文法的分析表，以后生成。

产生式的序号

状态	ACTION						GoTo		
	i	+	*	(	)	#	E	T	F
0	S_5			S4			1	2	3
1		S_6							
2		r_2							
3									
4	S_5			S_4			8	2	3
5		r_6							
6	S_5							9	3
7	S_5								10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3				
11					r_5				

S_i表示把当前输入符号移进栈，第i个状态进状态栈。

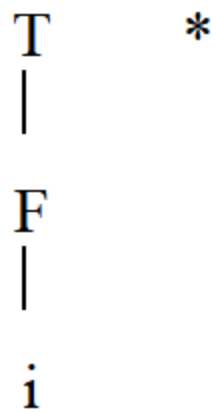
i表示转第i个状态，即第i个状态进状态栈。

ri表示按第i个产生式进行归约

空白表示分析动作出错

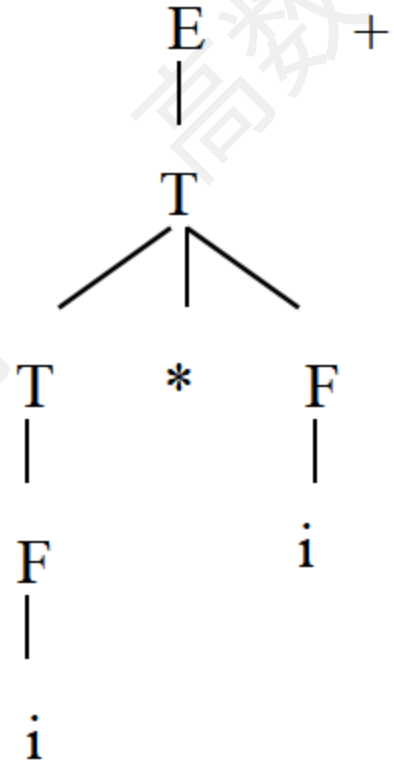
假定输入串为i*i+i， LR分析器的工作过程:

步骤	状态	符号	输入串
(1)	0	#	i*i+i#
(2)	05	#i	*i+i#
(3)	03	#F	*i+i#
(4)	02	#T	*i+i#
(5)	027	#T*	i+i#

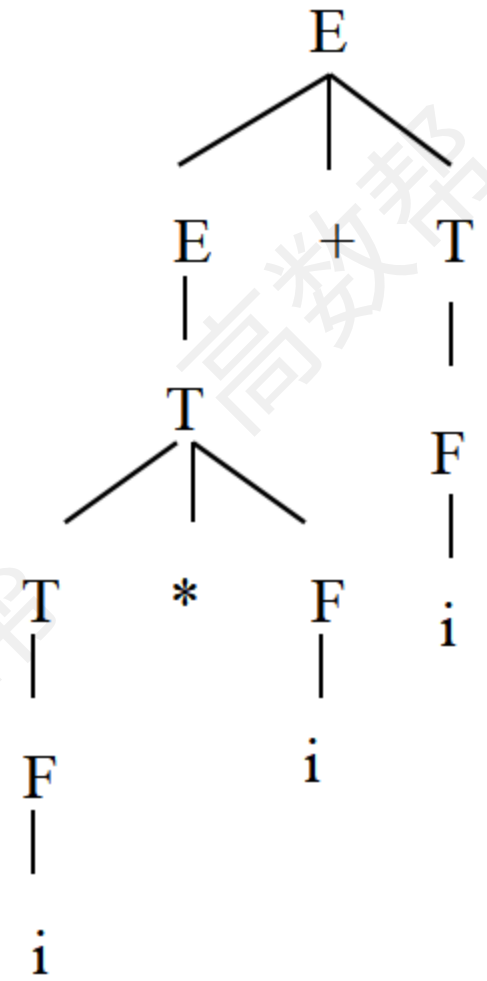


假定输入串为i*i+i， LR分析器的工作过程:

步骤	状态	符号	输入串
(5)	027	#T*	i+i#
(6)	0275	#T*i	+i#
(7)	02710	#T*F	+i#
(8)	02	#T	+i#
(9)	01	#E	+i#
(10)	016	#E+	i#



步骤	状态	符号	输入串
(10)	016	#E+	i#
(11)	0165	#E+i	#
(12)	0163	#E+F	#
(13)	0169	#E+T	#
(14)	01	#E	#
(15)	接受		



3.1 LR分析器

1.定义：对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则这个文法就称为LR文法。

2.定义：一个文法，如果能用一个每步顶多向前检查k个输入符号的LR分析器进行分析，则这个文法就称为LR(k)文法.

3.非LR结构

LR文法不是二义的，二义文法肯定不会是LR的。

$S \rightarrow iCtS \mid iCtSeS$

栈	输入
…iCtS	e…#

3.2 LR(0)项目集族和LR(0)分析表的构造

假定α是文法G的一个句子，我们称序列

α_n， α_{n-1}， … ， α_0

是的一个规范归约，如果此序列满足：

1 $\alpha_n = \alpha$

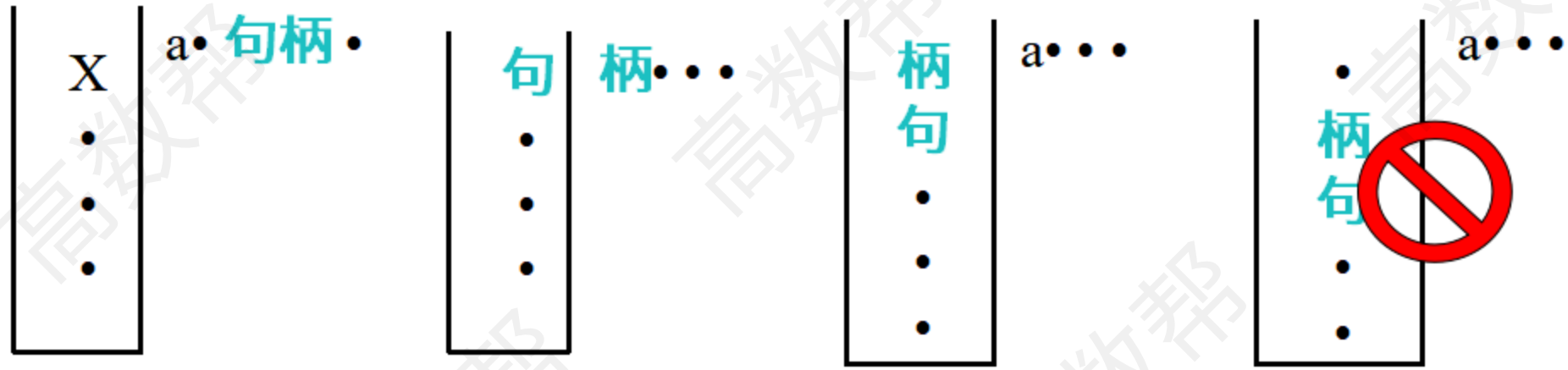
2 α_0为文法的开始符号，即α_0=S

3 对任何i，$0 \le i \le n$， α_{i-1}是从α_i经把句柄替换成为相应产生式左部符号而得到的。

规范归约过程中

栈内的符号串和扫描剩下的输入符号串构成了一个规范句型

栈内的如果出现句柄，句柄一定在栈的顶部



栈内永远不会出现句柄之后的符号

核心概念：活前缀

1.字的前缀：是指字的任意首部，如字*abc*的前缀有ε，*a*，*ab*，*abc*

2.活前缀：是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。

即，对于规范句型αβδ，β为句柄，如果$\alpha\beta=u_1u_2\ldots u_r$，则符号串$u_1u_2\ldots u_i(1\leq i\leq r)$是αβδ的活前缀。(δ必为终结符串)

3.对于一个文法G, 可以构造一个DFA,它能识别G的所有活前缀.

识别活前缀的本质就是识别句柄!

活前缀与句柄的关系

1.活前缀已含有句柄的全部符号，表明该句柄对应的产生式 A→α的右部α已出现在栈顶（即活前缀的尾部正好是句柄之尾），此时可以进行归约，该活前缀称为可归约（或归态）活前缀。

2.活前缀只含句柄的一部分符号，表明该句柄对应的产生式 A→α1α2 的右部子串α1已出现在栈顶，期待从输入串中看到α2 推导出的符号串。

3.活前缀不含有句柄的任何符号，此时期待从输入串中看到该句柄对应的产生式A→α的右部所推导出的符号串。

构造自动机识别活前缀

1.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。

2.由于产生式右部的符号就是句柄，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，由它的状态来记住这些情况，此“状态”称为“**项目**”。这些自动机的全体就是识别所有活前缀的有限自动机。

视频讲解更清晰

文法G的每个产生式的右部添加一个圆点称为G的LR(0)项目

如:$A\rightarrow XYZ$有四个项目：

$A\rightarrow\cdot XYZ$　　$A\rightarrow X\cdot YZ$　$A\rightarrow XY\cdot Z$　$A\rightarrow XYZ\cdot$

$A\rightarrow\alpha\cdot$称为"归约项目"

归约项目 $S'\rightarrow A\cdot$ 称为"接受项目"

$A\rightarrow\alpha\cdot a\beta\ (a\in V_T)$ 称为"移进项目"

$A\rightarrow\alpha\cdot B\beta\ (B\in V_N)$ 称为"待约项目".

【题9】：文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d

该文法的项目有：

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·

构造识别文法所有活前缀的NFA方法

1. 若状态i为$X \to X_1 \ldots X_{i-1}.X_i \ldots X_n$，

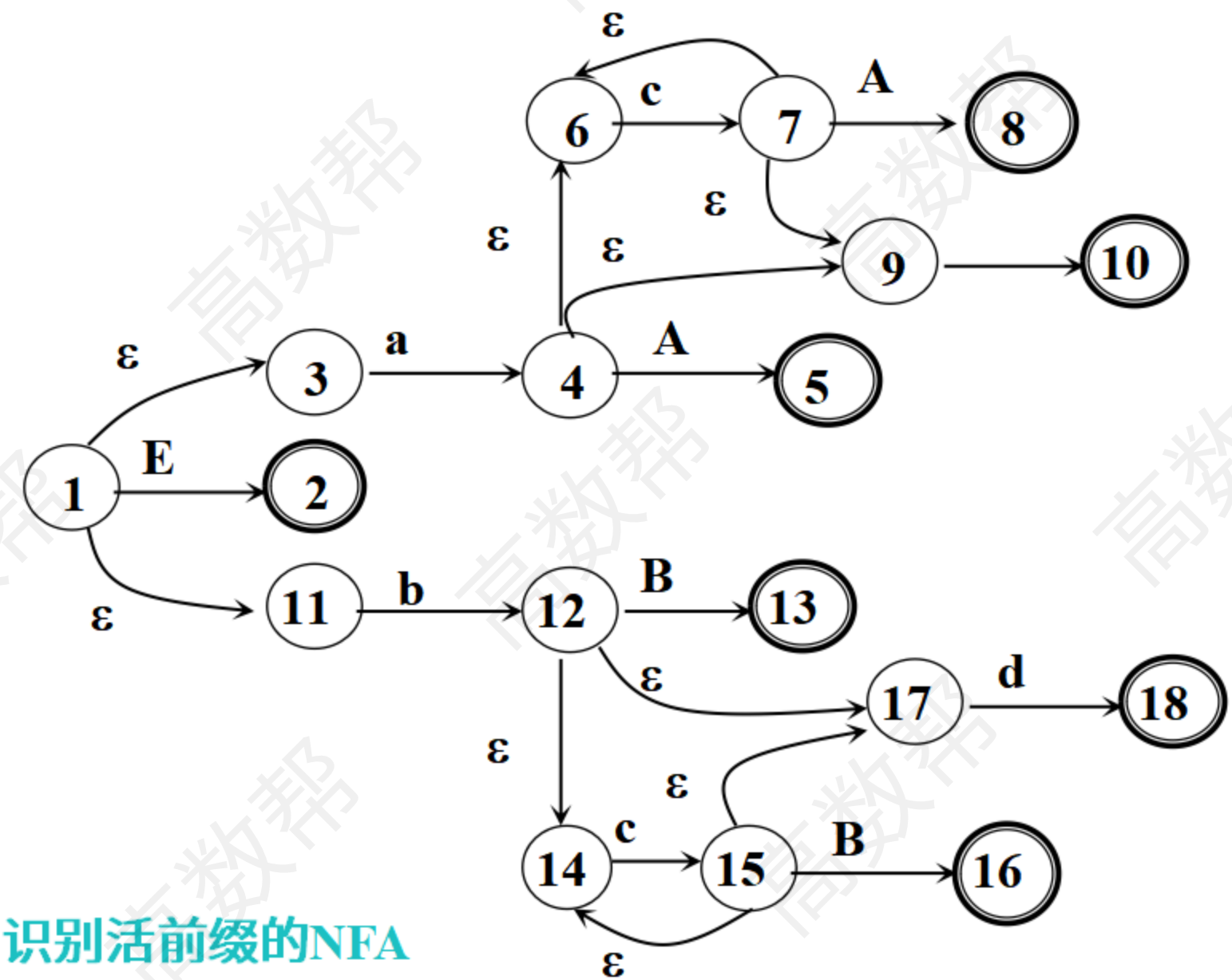
   状态j为$X \to X_1 \ldots X_{i-1}X_i .X_{i+1} \ldots X_n$，

   则从状态i画一条标志为X_i的有向边到状态j；

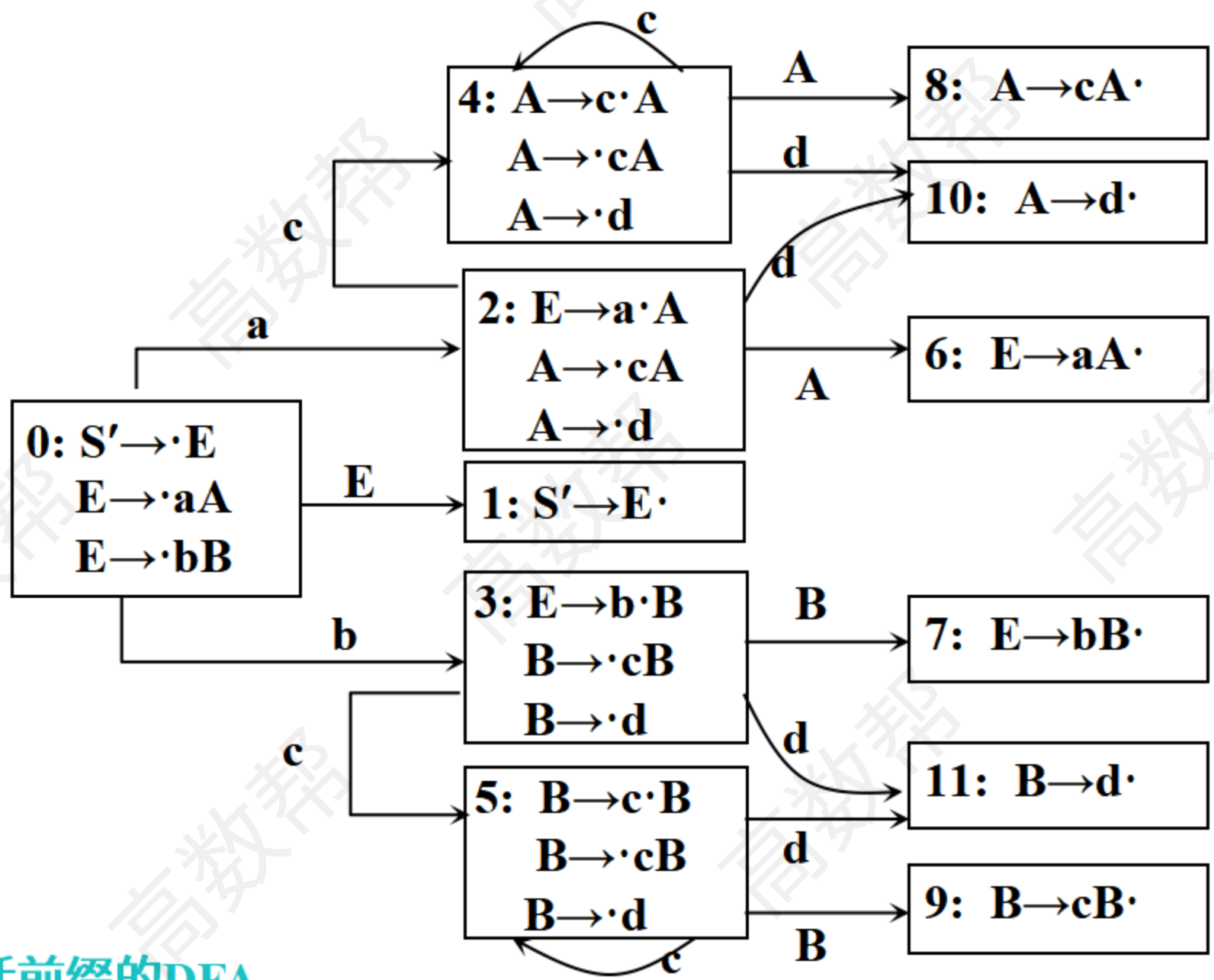
2. 若状态i为$X \to \alpha .A\beta$，A为非终结符，

   则从状态i画一条ε边到所有状态$A \to .\gamma$。

把识别文法所有活前缀的NFA确定化。



识别活前缀的NFA


0: S′→·E
E→·aA
E→·bB
1: S′→E·
2: E→a·A
A→·cA
A→·d
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

识别活前缀的DFA

构成识别一个文法活前缀的DFA的项目集(状态)的全体称为文法的LR(0)项目集规范族。

有效项目

我们说项目$A\rightarrow \beta_1.\beta_2$对活前缀$\alpha\beta_1$是有效的，其条件是存在规范推导

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

LR(0)项目集规范族的构造

1.假定文法G是一个以S为开始符号的文法，我们构造一个G′，它包含了整个G，但它引进了一个不出现在G中的非终结符S′，并加进一个新产生式S′→S，而这个S′是G′的开始符号。那么，我们称G′是G的拓广文法。这样，便会有一个仅含项目S′→S.的状态，这就是唯一的“接受”态。

2.假定I是文法G′的任一项目集，定义和构造I的闭包CLOSURE(I)如下：

1)I的任何项目都属于CLOSURE(I)；

2) 若A→α·Bβ属于CLOSURE(I)，那么，对任何关于B的产生式B→γ，项目B→·γ也属于CLOSURE(I)；

3)重复执行上述两步骤直至CLOSURE(I) 不再增大为止。

3.为了识别活前缀，我们定义一个状态转换函数GO是一个状态转换函数。I是一个项目集，X是一个文法符号。函数值GO(I，X)定义为：

$$GO(I，X)=CLOSURE(J)$$

其中

J＝{任何形如$A\rightarrow\alpha X\cdot\beta$的项目| $A\rightarrow\alpha\cdot X\beta$属于I}。

直观上说，若I是对某个活前缀 γ 有效的项目集，那么，GO(I，X)便是对 γX 有效的项目集。

【题10】：文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d

I_0={S′→·E, E→·aA, E→·bB}

GO(I_0, E)= closure(J)=closure({S’→E·})

= {S’→E·} = I_1

GO(I_0, a)= closure(J)=closure({E→a·A})

={ E→a·A, A→·cA, A→·d})=I_2

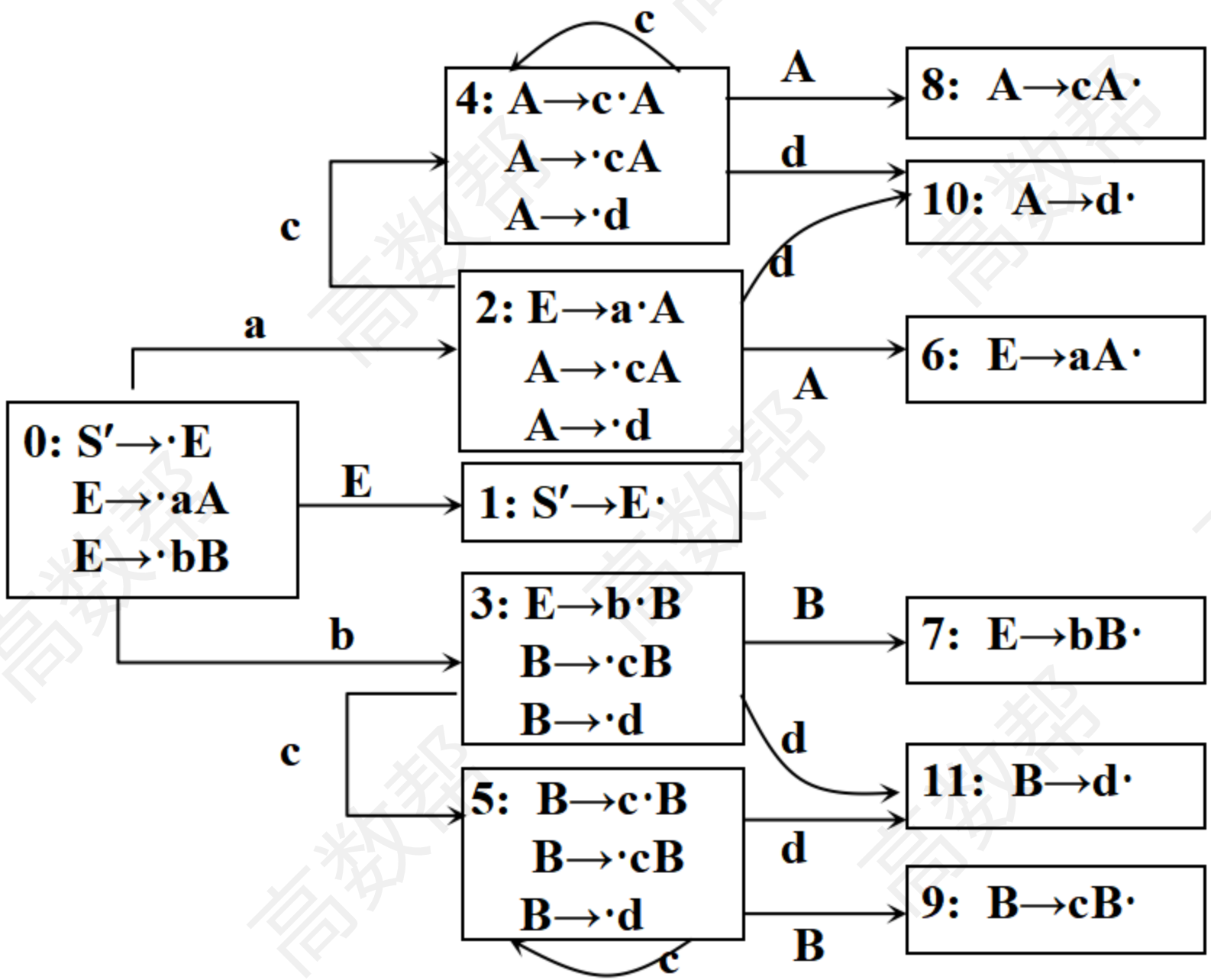
GO(I_0, b)= closure(J)=closure ({E →b.B})

={E →b.B, B→.cB, B→.d}= I_3

构造文法G的拓广文法G′的LR(0)项目集规范族算法：

```
PROCEDURE  ITEMSETS(G′);
BEGIN
  C:={CLOSURE({S′→·S})};
  REPEAT
     FOR  C中每个项目集I和G′的每个符号X  DO
          IF  GO(I，X)非空且不属于C   THEN
            把GO(I，X)放入C族中;
  UNTIL C     不再增大
END
```

转换函数GO把项目集连接成一个DFA转换图


0: S′→·E
E→·aA
E→·bB
1: S′→E·
2: E→a·A
A→·cA
A→·d
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

LR(0)分析表的构造

假若一个文法G的拓广文法G′的活前缀识别自动机中的每个状态(项目集)不存在下述情况：

1) 既含移进项目又含归约项目，

2) 含有多个归约项目

则称G是一个**LR(0)文法**。

视频讲解更清晰

构造LR(0)分析表的算法

令每个项目集I_k的下标k作为分析器的状态，包含项目$S'\rightarrow\cdot S$的集合I_k的下标k为分析器的初态。

分析表的ACTION和GOTO子表构造方法：

1. 若项目$A\rightarrow\alpha\cdot a\beta$属于$I_k$且$GO(I_k, a)=I_j$，a为终结符，则置ACTION[k,a] 为 “sj”。
2. 若项目$A\rightarrow\alpha\cdot$属于I_k，那么，对任何终结符a(或结束符#)，置ACTION[k,a]为“rj”(假定产生式$A\rightarrow\alpha$是文法G′的第j个产生式)。
3. 若项目$S'\rightarrow S\cdot$属于I_k，则置ACTION[k,#]为 “acc”。
4. 若$GO(I_k,A)=I_j$，A为非终结符，则置GOTO[k,A]=j。
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。

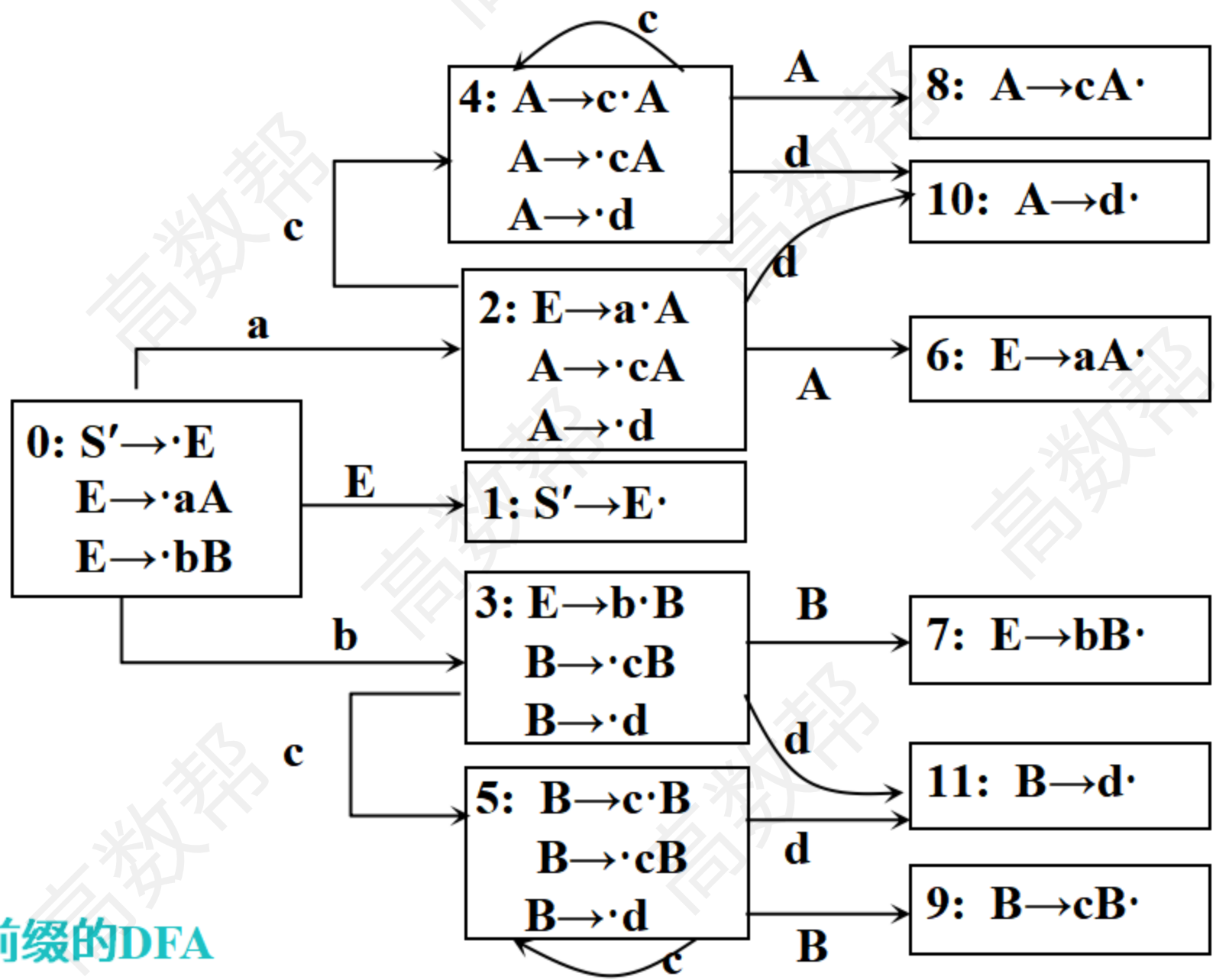
【题11】：文法G(S′)

$S' \rightarrow E$

$E \rightarrow aA|bB$

$A \rightarrow cA|d$

$B \rightarrow cB|d$


0: S′→·E
E→·aA
E→·bB
1: S′→E·
2: E→a·A
A→·cA
A→·d
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

识别活前缀的DFA

LR(0)分析表为

	ACTION					GOTO		
状态	a	b	c	d	#	E	A	B
0	s2	s3				1		
1					acc			
2			s4	s10			6	
3			s5	s11				7
4			s4	s10			8	
5			s5	s11				
6	r1	r1	r1	r1	r1			9
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			
9	r5	r5	r5	r5	r5			
10	r4	r4	r4	r4	r4			
11	r6	r6	r6	r6	r6			

【题12】：按上表对acccd进行分析

步骤	状态	符号	输入串
1	0	#	acccd#
2	02	#*a*	cccd#
3	024	#ac	ccd#
4	0244	#acc	cd#
5	02444	#accc	d#
6	0244410	#acccd	#
7	024448	#acccA	#
8	02448	#accA	#
9	0248	#acA	#
10	026	#aA	#
11	01	#E	#

3.3 SLR分析表的构造

1.LR(0)文法太简单，没有实用价值.

2.假定一个LR(0)规范族中含有如下的一个项目集(状态)I＝{X→α·bβ，A→α·，B→α·}。FOLLOW(A)和FOLLOW(B)的交集为∅，且不包含b，那么，当状态I面临任何输入符号*a*时，可以:

1. 若a=b，则移进；
2. 若a∈FOLLOW(A)，用产生式A→α进行归约；
3. 若a∈FOLLOW(B)，用产生式B→α进行归约；
4. 此外，报错。

3.假定LR(0)规范族的一个项目集I={$A_1\to\alpha\cdot a_1\beta_1$，$A_2\to\alpha\cdot a_2\beta_2$，…，$A_m\to\alpha\cdot a_m\beta_m$，$B_1\to\alpha\cdot$，$B_2\to\alpha\cdot$，…，$B_n\to\alpha\cdot$} 如果集合{$a_1$，…，$a_m$}，FOLLOW($B_1$)，…，FOLLOW($B_n$)两两不相交(包括不得有两个FOLLOW集合有#)，则：

1. 若a是某个a_i，i=1,2,…,m，则移进；
2. 若a∈FOLLOW(B_i)，i=1,2,…,n，则用产生式$B_i\to\alpha$进行归约；
3. 此外，报错。

4.冲突性动作的这种解决办法叫做SLR(1)解决办法。

构造SLR(1)分析表方法:

1.首先把G拓广为G′，对G′构造LR(0)项目集规范族C和活前缀识别自动机的状态转换函数GO.

2.然后使用C和GO，按下面的算法构造SLR分析表：令每个项目集I_k的下标k作为分析器的状态，包含项目$S'\rightarrow\cdot S$的集合I_k的下标k为分析器的初态。

分析表的ACTION和GOTO子表构造方法：

1. 若项目$A\rightarrow\alpha\cdot a\beta$属于$I_k$且$GO(I_k,a)=I_j$，a为终结符，则置ACTION[k,a]为 “sj”；
2. 若项目$A\rightarrow\alpha\cdot$属于I_k，那么，对任何终结符a，$a\in FOLLOW(A)$，置ACTION[k,a]为 “rj”；其中，假定$A\rightarrow\alpha$为文法G′的第j个产生式；
3. 若项目$S'\rightarrow S\cdot$属于I_k，则置ACTION[k,#]为 “acc”；
4. 若$GO(I_k,A)=I_j$，A为非终结符，则置GOTO[k,A]=j；
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

按上述方法构造出的ACTION与GOTO表如果不含多重入口，

则称该文法为**SLR(1)文法**。

使用SLR表的分析器叫做一个**SLR分析器**。

每个SLR(1)文法都是无二义的。但也存在许多无二义文法不是SLR(1)的.

【题13】：考察下面的拓广文法：

(0) S′→E

(1) E→E+T

(2) E→T

(3) T→T*F

(4) T→F

(5) F→(E)

(6) F→i

视频讲解更清晰

这个文法的LR(0)项目集规范族为：

I_0: S′→·E
E→·E+T
E→·T
T→·T*F
T→·F
F→· (E)
F→·i

I_1: S′→E·
E→E·+T

I_2: E→T·
T→T·*F

I_3: T→F·

I_4: F→(·E)
E→·E+T
E→·T
T→·T*F
T→·F
F→· (E)
F→·i

I_5: F→i·

I_6: E→E+·T
T→·T*F
T→·F
F→·(E)
F→·i

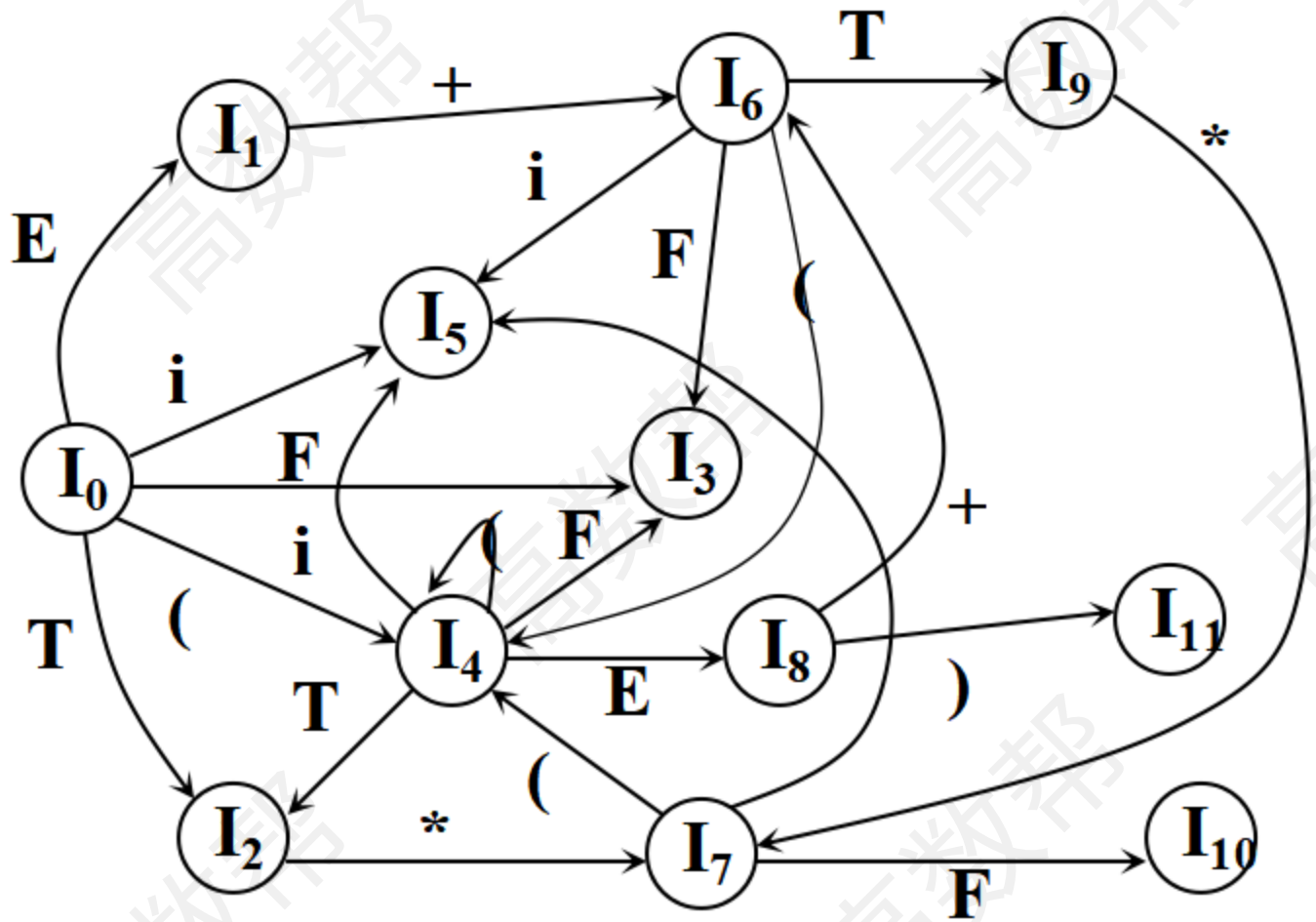
I_7: T→T*·F
F→·(E)
F→·i

I_8: F→(E·)
E→E·+T

I_9: E→E+T·
T→T·*F

I_{10}: T→T*F·

I_{11}: F→(E)·


I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
E
T
F
i
(
)
+
*

I_1、I_2和I_9都含有“移进－归约”冲突。

FOLLOW(E)＝{#,), +}，

I_1: $S' \rightarrow E\cdot$

$E \rightarrow E\cdot + T$

I_2: $E \rightarrow T\cdot$

$T \rightarrow T\cdot * F$

I_9: $E \rightarrow E + T\cdot$

$T \rightarrow T\cdot * F$

其分析表如下：

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			

SLR（1）的局限

SLR在方法中,如果项目集I_i含项目$A\rightarrow\alpha.$而且下一输入符号$a\in FOLLOW(A)$,则状态i面临a时,可选用“用$A\rightarrow\alpha$归约”动作。但在有些情况下,当状态i显现于栈顶时,栈里的**活前缀**未必允许把α归约为A，因为可能根本就不存在一个形如“βAa”的规范句型。因此，在这种情况下，用“ $A\rightarrow\alpha$ ”归约不一定合适。

所以需要扩充状态以包含更多的信息：Follow 集的哪些部分 才是进到该状态最恰当的归约依据。

FOLLOW集合提供的信息太泛

LR(1)、LALR(1)分析法自学

视频讲解更清晰