编译原理　第四章　语法分析(Syntax Analysls)

内容回顾

提出问题：其它上下文无关文法？

4.1 自上而下的语法分析(TOP-DOWN PARSING)

4.2 自下而上的语法分析(BOTTOM-UP PARSING)

1、 一般分析方法　移进-归约（Shift-Reduce）　最左归约

2、 简单优先分析

简单优先文法　最左归约　句柄

3 、算符优先分析

算符优先文法　“最左归约”　最左素短语

4、 LR分析法

4.2.4 LR分析法 (LR PARSERS)

L R （K） LR(0)、SLR(1)、LR(1)……

- 向前看K个符号能确定用哪个产生式归约
- 产生一个最右推导(Rightmost)的逆过程
- 自左向右扫描输入符号(Left-to-right)

- 解决所有无二义性的上下文无关文法
- 严格的最左归约-----句柄
- 语法分析程序自动生成器YACC(Yet Another Compiler-Compiler)

4.2.4 LR分析法 (LR PARSERS)

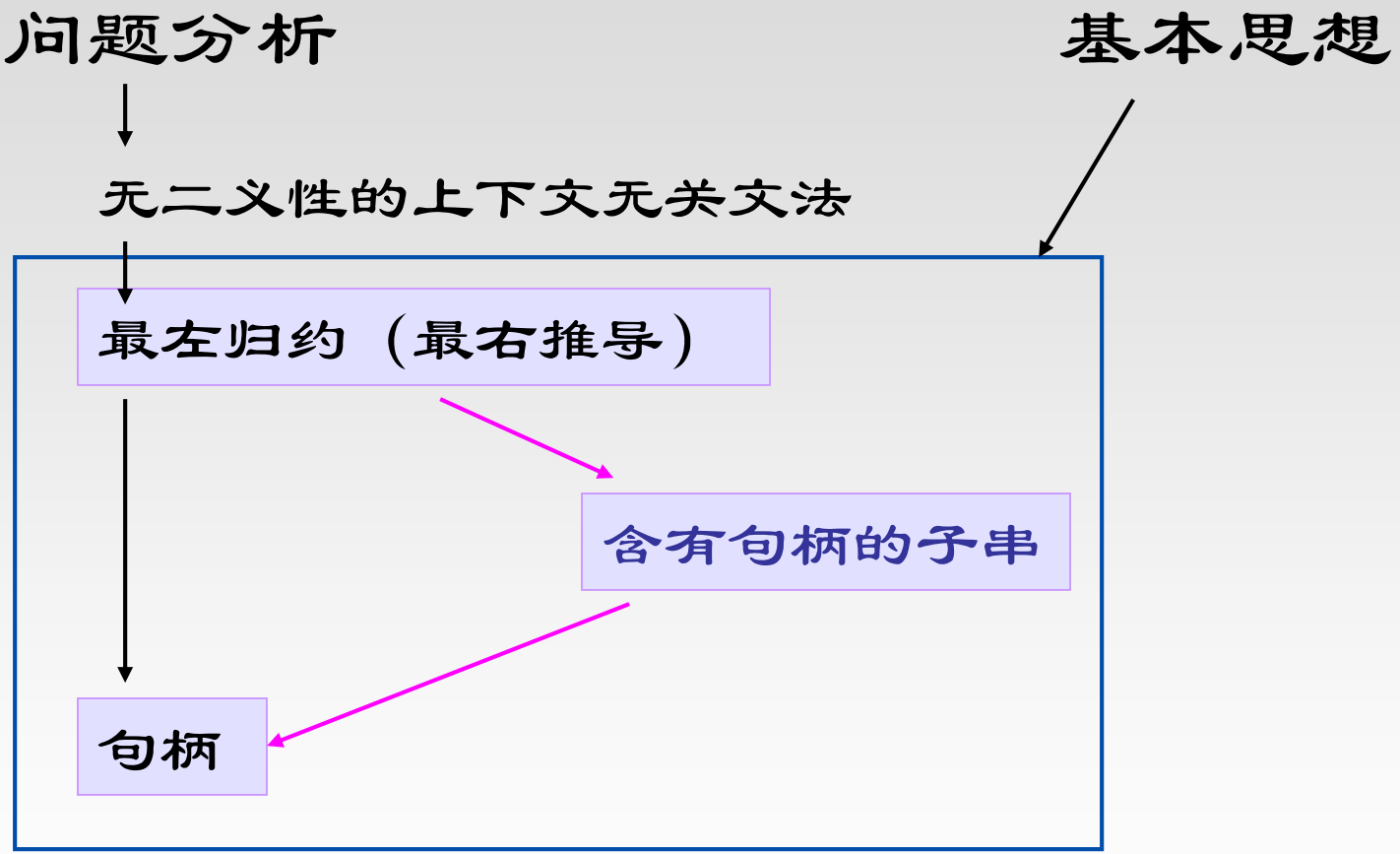
主要内容

- 基本概念
- LR(0) 分析法
- SLR(1) 分析法
- LR(1) 分析法
- LALR(1) 分析法

重点掌握

- LR(0)、SLR(1)、LR(1)等文法定义
- LR(0)和SLR(1) 分析器的构造

4.2.4 LR分析法 (LR PARSERS)



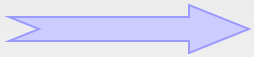
4.2.4 LR分析法　❑基本概念

一、基本概念

❑可归前缀

❑活前缀

例：G[S]:
S→aBEb
B→BaE|c
E→e



S→aBEb[1]
B→BaE[2]
B→c[3]
E→e[4]

问题：将规则编上号,并将序号人为带入句型分析中，给出句子 acaeeb 的规范推导过程；

4.2.4 LR分析法 ❑基本概念

S→aBEb[1]
B→BaE[2]
B→c[3]
E→e[4]

S=>aBEb[1]
=>aBe[4]b[1]
=>aBaE[2]e[4]b[1]
=>aBae[4][2]e[4]b[1]
=>ac[3]ae[4][2]e[4]b[1]

规范推导

acaeeb

4.2.4 LR分析法

❑基本概念

ac[3]ae[4][2]e[4]b[1]

<= aBae[4][2]e[4]b[1]

<= aBaE[2]e[4]b[1]

<= aBe[4]b[1]

<= aBEb[1] <= S

最左归约

当前句型的句柄

S→aBEb[1]

B→BaE[2]

B→c[3]

E→e[4]

acaeeb

aBEb[1]

aBe[4]

aBaE[2]

aBae[4]

ac[3]

可归前缀

4.2.4 LR分析法 ❑基本概念

S→aBEb[1]
B→BaE[2]
B→c[3]
E→e[4]

特点：❑可归前缀的后半部分总是包含当前句型的句柄；

❑可归前缀含有用哪一个产生式进行归约的信息；

aBEb[1]
aBe[4]
aBaE[2]
aBae[4]
ac[3]

} 可归前缀

4.2.4 LR分析法　　❑基本概念

1、可归前缀

形式为 $\beta\omega$ [p]

其中: $\beta \in V^*$

p为规则序号,

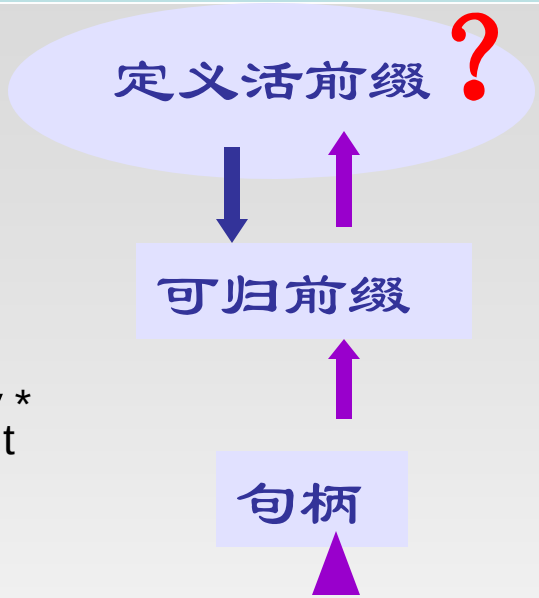
ω为第p条规则右部, $B \rightarrow \omega$

可归前缀中应包含的信息

- ❑句柄（在最后）
- ❑用哪条产生式进行归约

4.2.4 LR分析法 ❑基本概念

2、活前缀(Viable Prefix)



对于最右推导过程

$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots\cdots \Rightarrow \alpha_m = x$

若$\alpha_i = \varphi B t$　　且$B \rightarrow \beta$, $\varphi \in V^*$,$t \in V_t^*$

则存在最右推导$\varphi B t \Rightarrow \varphi \beta t$

令$\varphi\beta = u_1 u_2 \cdots\cdots u_r$　　$u_i \in V$

则$u_1 u_2 \cdots\cdots u_i$ (1<=i<=r)为句型$\varphi\beta t$ 的活前缀

❖最长的活前缀就是可归前缀

❖ ε 是句型$\varphi\beta t$的活前缀

4.2.4 LR分析法 ❑基本概念

例：G[E]:E→E+T|E-T|T
T →i|(E)

拓广文法：G′[S]
S →E#
E→E+T|E-T|T
T →i|(E)

句型：E-(i+i)#

有S=+>E-(T+i)#=>E-(i+i)#

活前缀：E　E-　E-(　E-(i　(可归前缀)

问题：用什么样的方法来识别活前缀？　FA M

❖一个文法所有规范句型的活前缀（可归前缀），

能够为有限自动机所识别。

二、LR(0)分析法

在归约时不向前看任何一个符号就能确定用哪一个产生式

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]

其识别可归前缀的有限自动机如图 DFA M

4.2.4 LR分析法　　❑LR(0)分析法

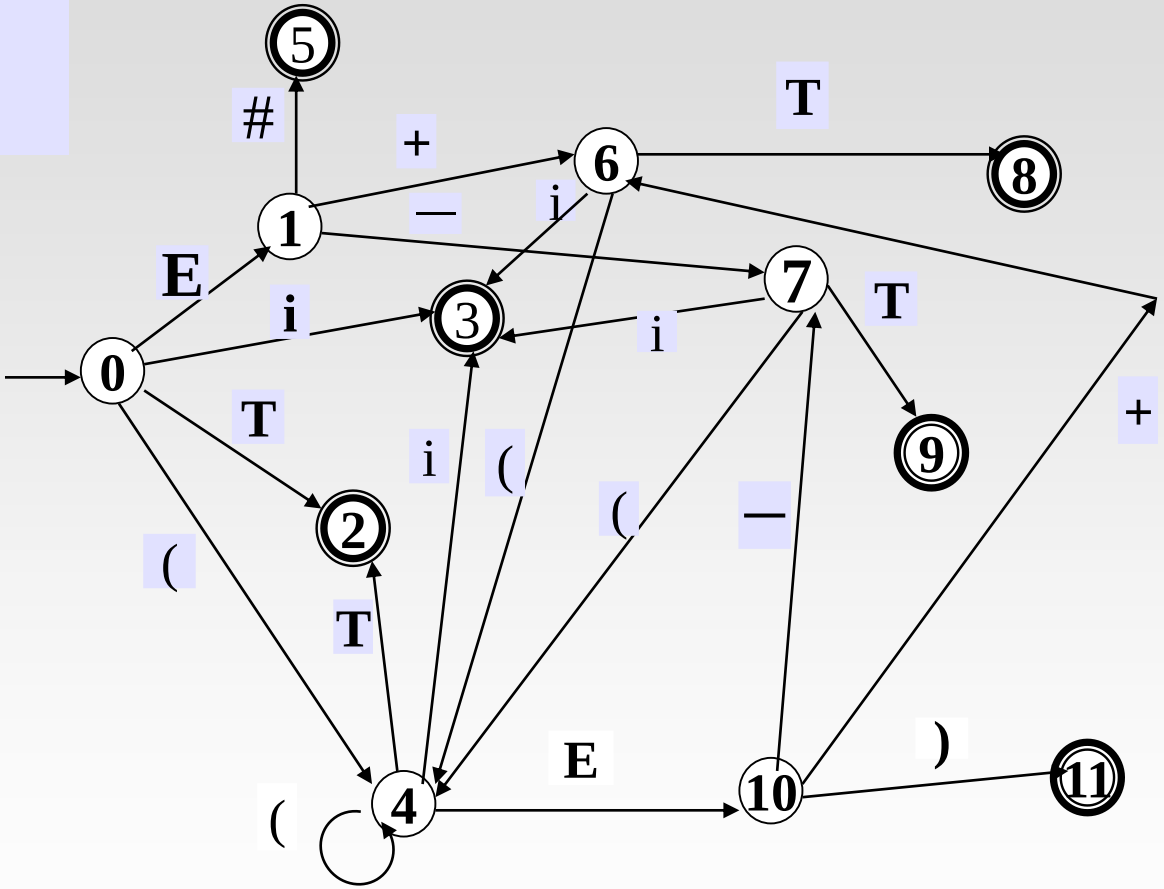
G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]

非终态:识别到活前缀

终态: 识别到可归前缀



识别活前缀和可归前缀的FA

4.2.4 LR分析法　　❑LR(0)分析法

1、LR(0) 分析器

❑总控程序

❑DFA M (LR(0)分析表)

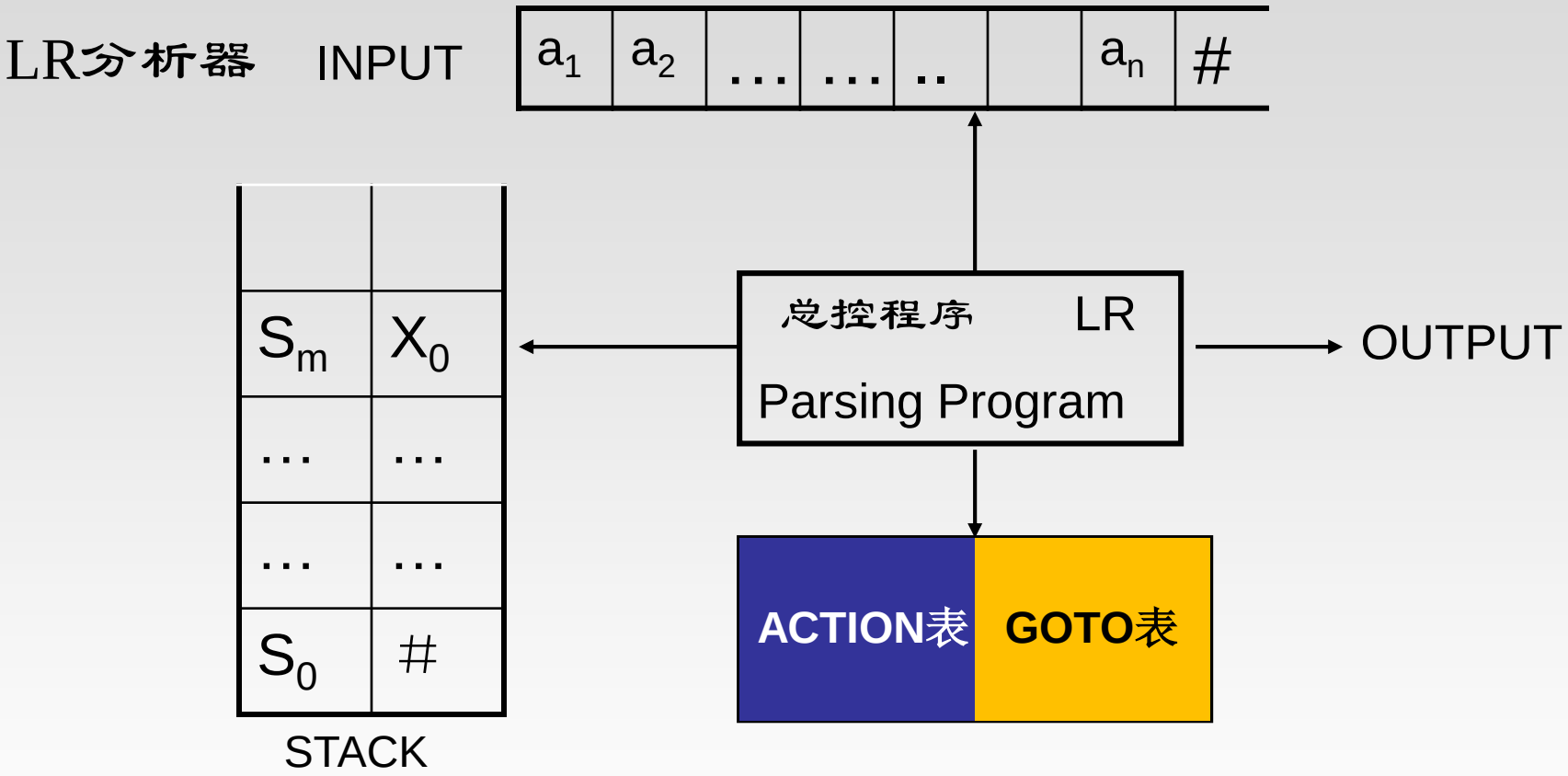
0状态：开始状态

读状态：非终态，识别到活前缀

归约状态：终态，识别到可归前缀

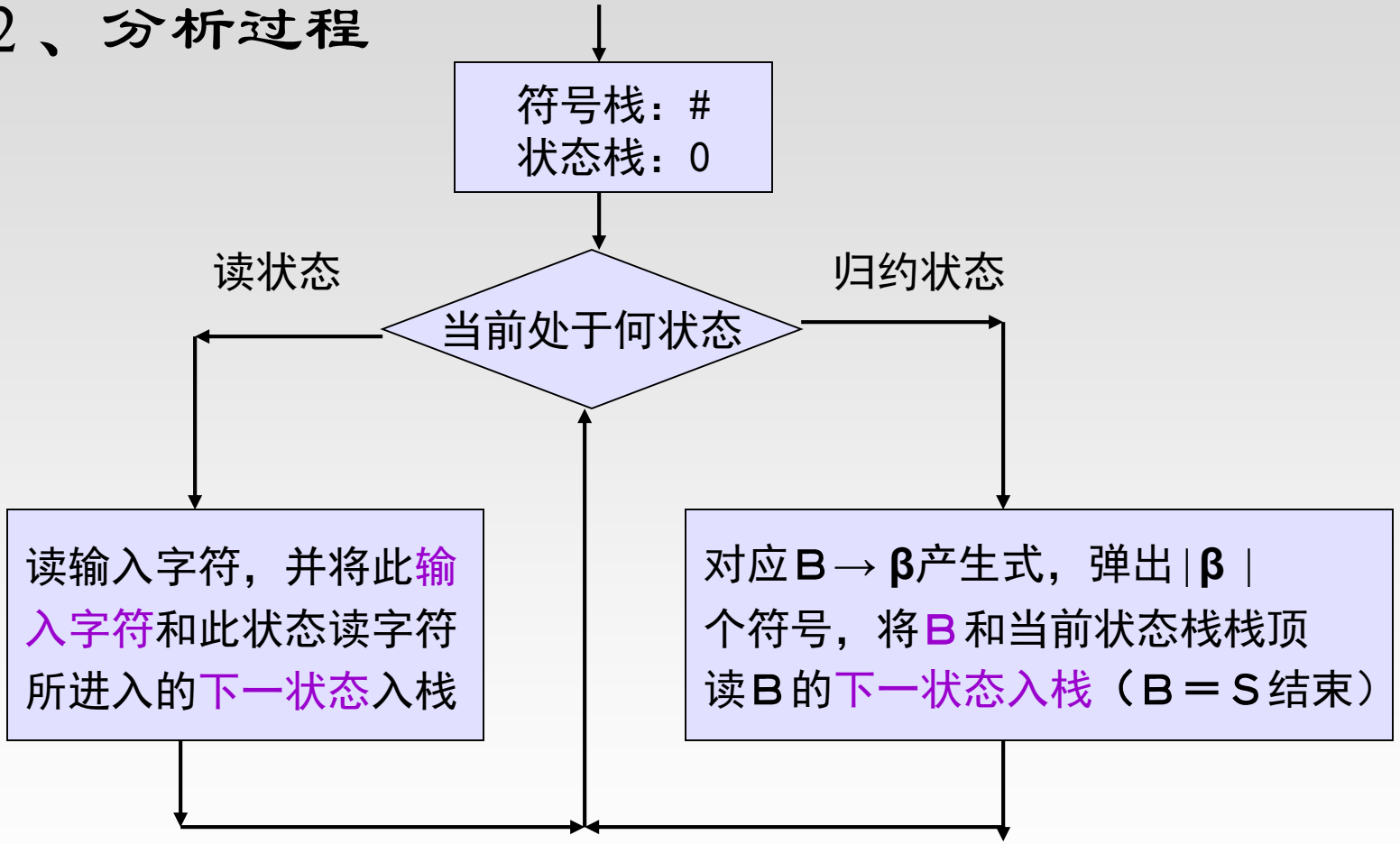
❑对偶栈
- 符号栈：放V中的字符
- 状态栈：扫描V上的字符后进入的状态(DFA)

4.2.4 LR分析法 ❑LR(0)分析法



4.2.4 LR分析法　　❑LR(0)分析法

2、分析过程



4.2.4 LR分析法 ❑LR(0)分析法

#		
0		

#	i	
0	3	

#	T	
0	2	

#	E	
0	1	

#	E	-	
0	1	7	

输入串i-(i+i)#

i

规约i 规则[5]

规约T 规则[4]

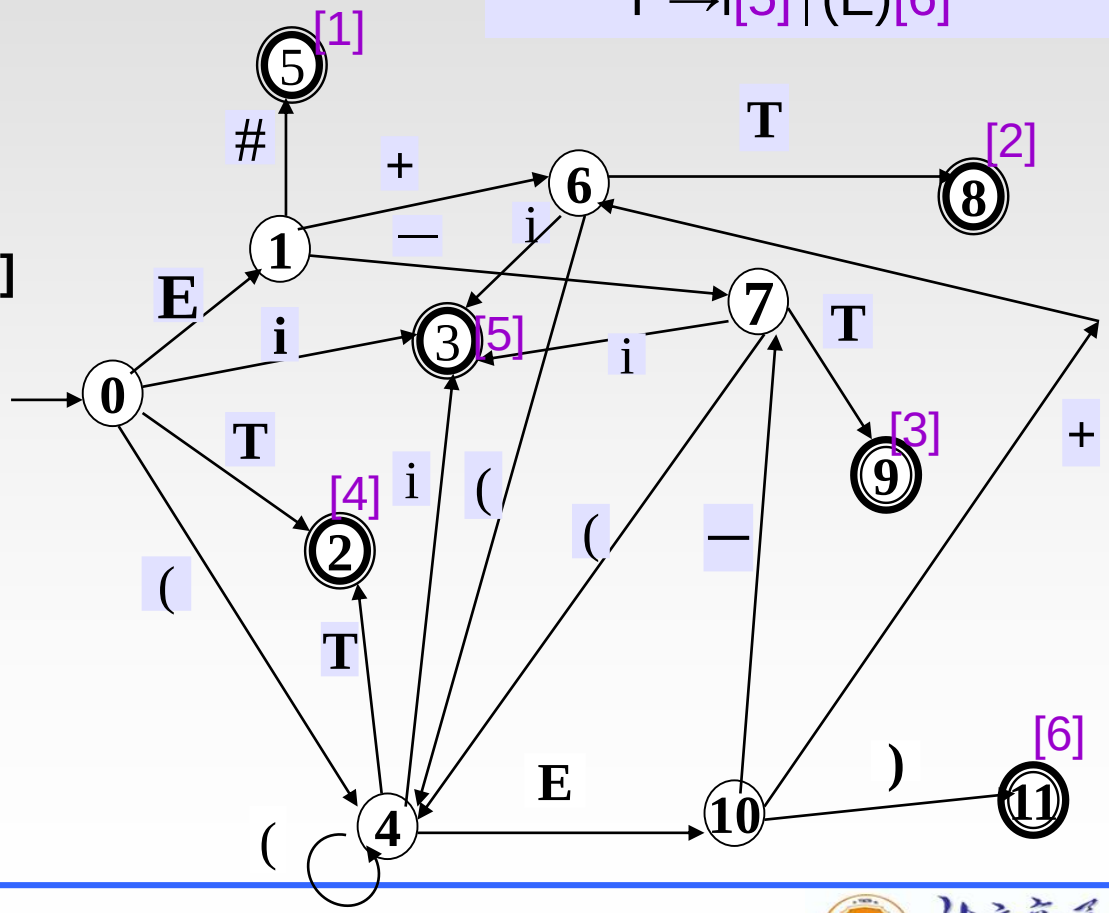
-

(

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]



4.2.4 LR分析法　　❑LR(0)分析法

#	E	–	(	
0	1	7	4	

#	E	–	(	i	
0	1	7	4	3	

#	E	–	(	T	
0	1	7	4	2	

#	E	–	(	E	
0	1	7	4	10	

#	E	–	(	E	+	
0	1	7	4	10	6	

i

规约i

规约T

+

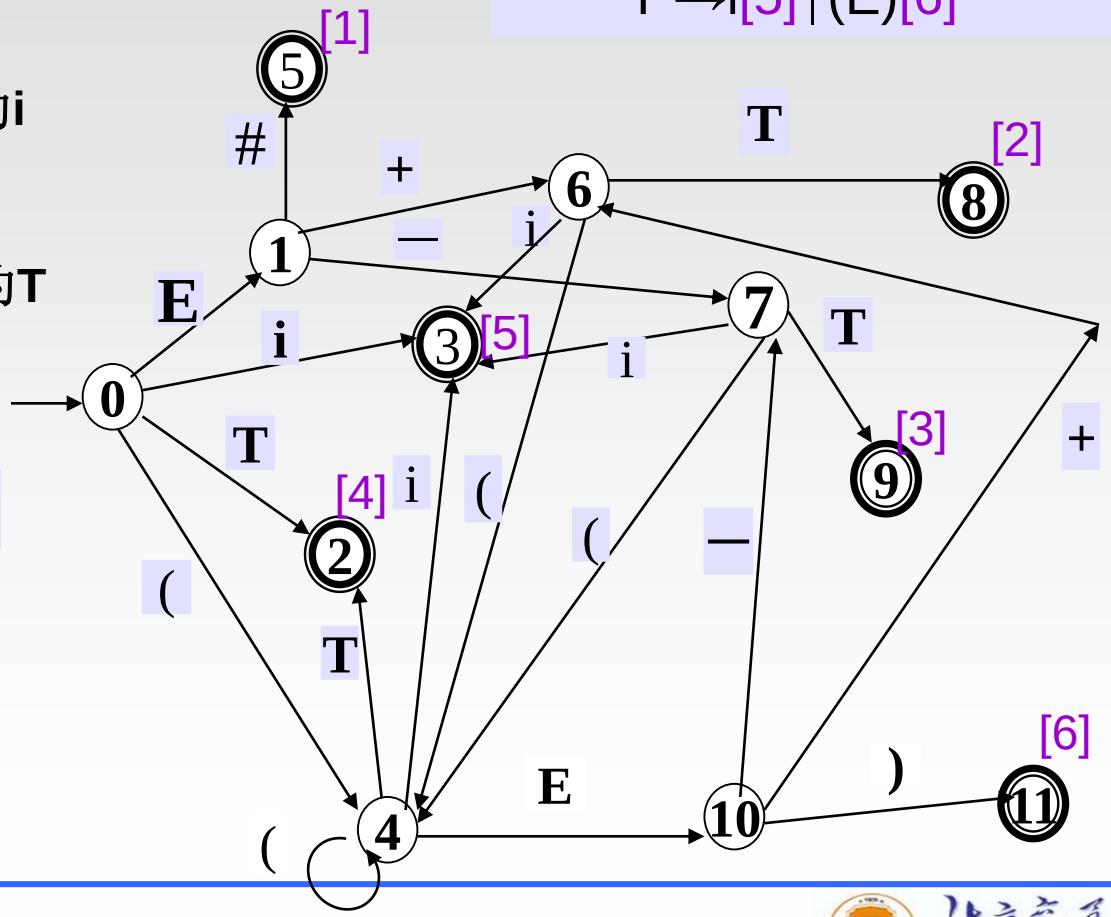
i

输入串i-(i+i)#

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]



4.2.4 LR分析法　　❑LR(0)分析法

#	E	–	(	E	+	i	
0	1	7	4	**10**	6	3	

#	E	–	(	E	+	T	
0	1	7	4	**10**	6	8	

#	E	–	(	E	
0	1	7	4	10	

#	E	–	(	E	)	
0	1	7	4	**10**	11	

输入串i-(i+i)#

规约i

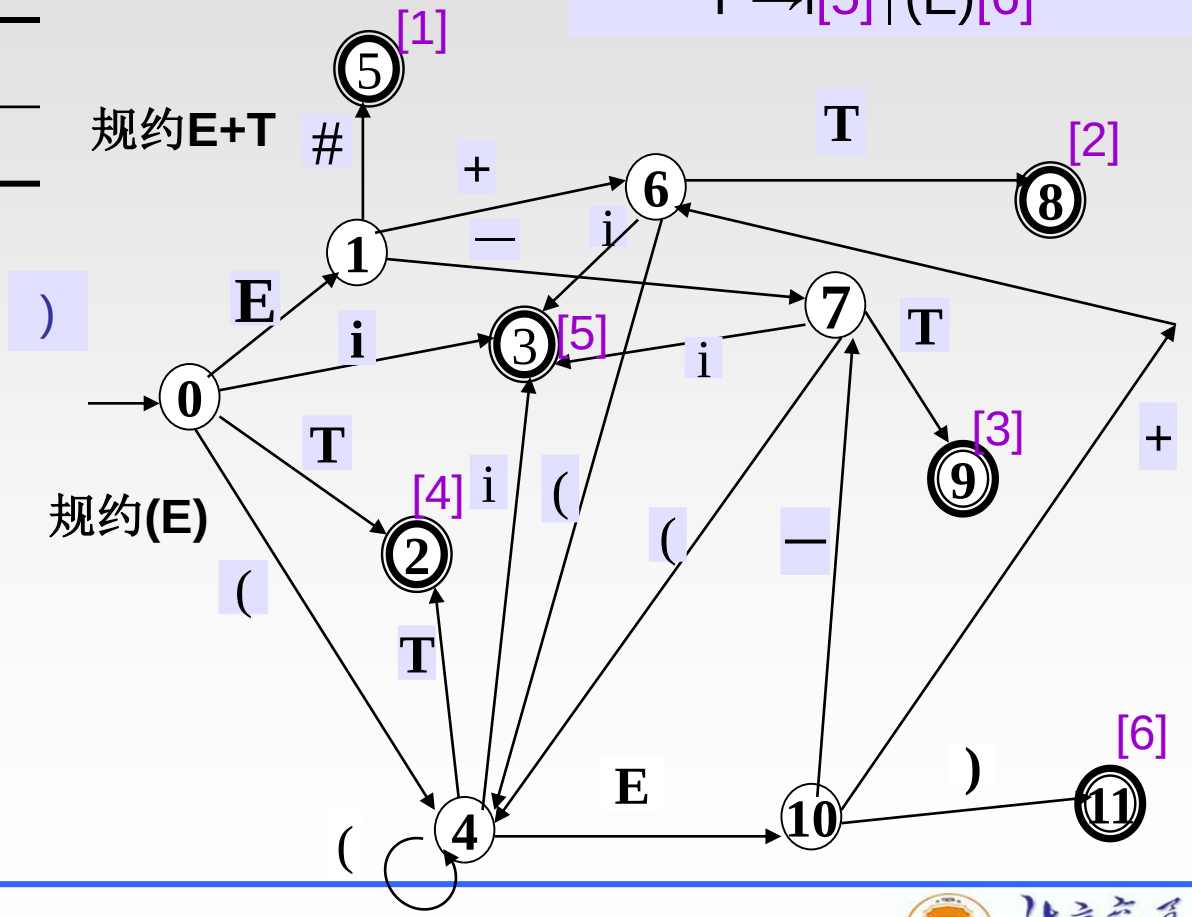
规约E+T

规约(E)

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]



4.2.4 LR分析法

❑LR(0)分析法

#	E	–	(	E	)	
0	1	7	4	**10**	11	

输入串i-(i+i)#

规约(E)

#	E	–	T	
0	1	7	9	

规约E-T

#	E	
0	1	

#

#	E	#	
0	1	5	

规约E#

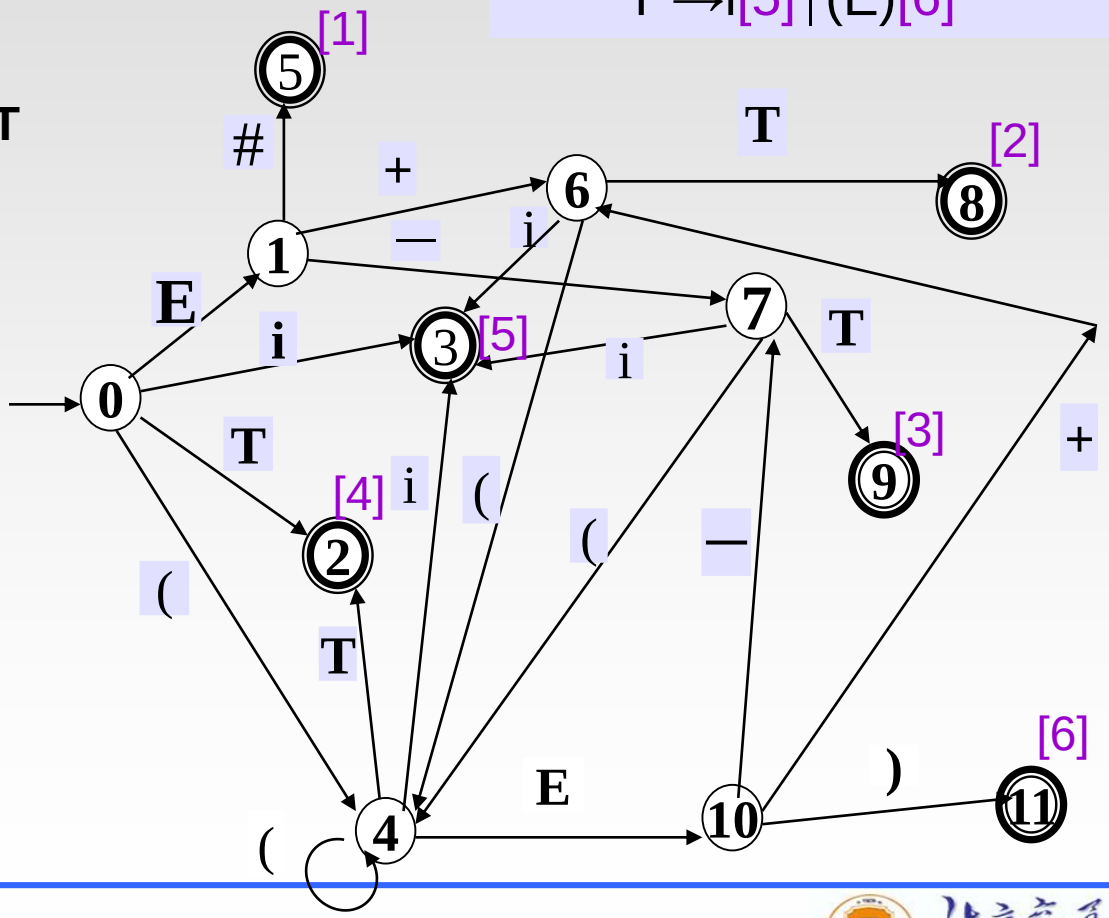
#	S		
0			

接受

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]



4.2.4 LR分析法　　❑LR(0)分析法

讨论

❑算符优先分析与LR分析法比较

相同点：通过分析栈的栈顶项和当前输入符号找当前句型句柄的右端；

不同点：优先分析法为找句柄的头必须对栈进行搜索；

LR分析法只根据栈顶状态和当前输入符号就可判断；

问题：LR(0)分析器如何构造？

LR(0)分析器如何转换成LR(0)分析表？

4.2.4 LR分析法

总结

- 基本概念：活前缀、可归前缀
  - 规范句型的活前缀可以为FA M所识别
- LR(0)分析法
  - LR(0)分析器
  - LR(0)分析方法
- 进一步要解决的问题
  - 由LR(0)分析器构造LR(0)分析表
  - 构造LR(0)分析器

4.2.4 LR分析法　　❑LR(0)分析法

3、 LR(0)分析表:将DFA的信息放入一张表中

ACTION[S,a]函数:栈顶状态S面临输入符号a时应采取的动作

S_j:移进,把下一状态j和现输入符号a移入栈

R_j:归约,按第j产生式归约

acc:接受

空白:出错

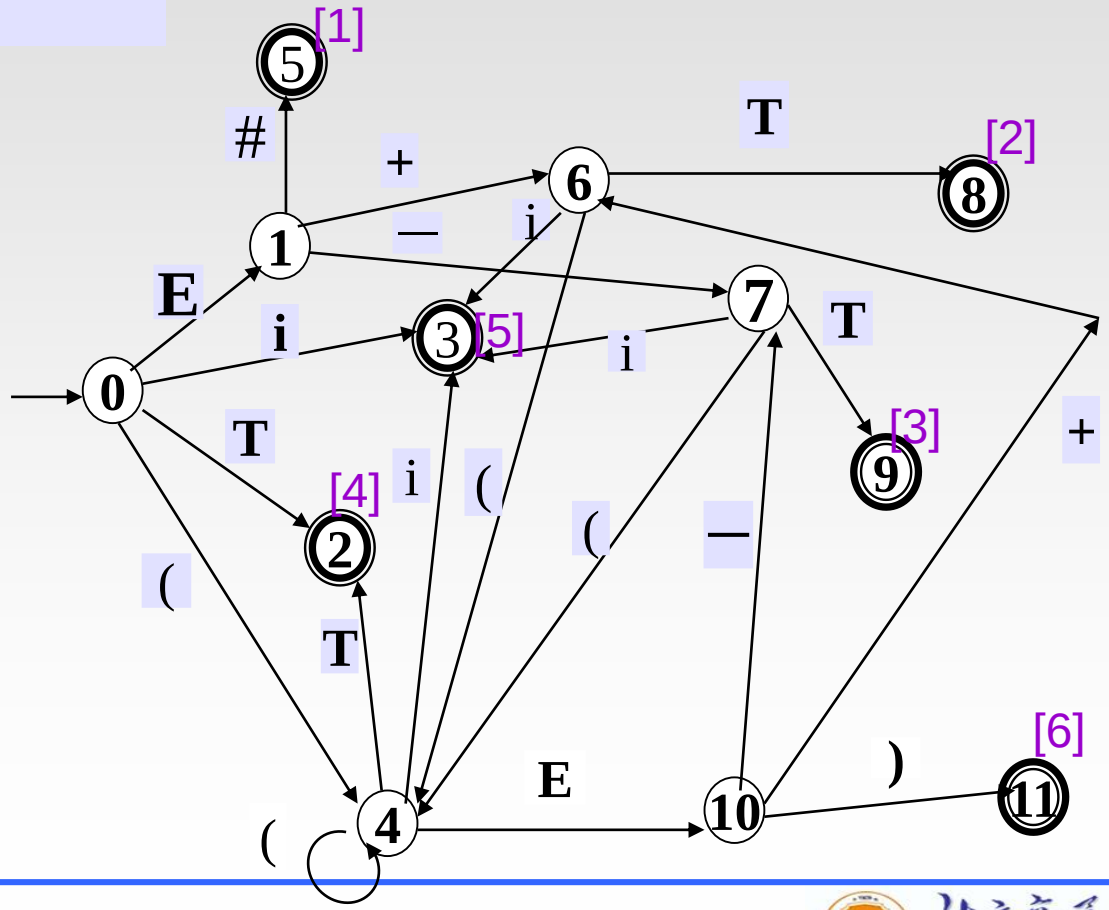
GOTO[S,x]函数: 栈顶状态S遇到当前文法符号x($x \in V$)时应转向的下一状态.

由识别活前缀的DFA构造LR(0)分析表？

G[S]:S →E#[1]

E→E+T[2]|E-T[3]|T[4]

T →i[5]|(E)[6]



文法G的LR(0)分析表

状态	ACTION						GOTO								
	i	+	—	(	)	#	S	E	T	i	+	—	(	)	#
0	S_3			S_4				1	2	3			4		
1		S_6	S_7			S_5									
2	r_4	r_4	r_4	r_4	r_4	r_4									
3	r_6	r_6	r_6	r_6	r_6	r_6									
4	S_3			S_4				10	2	3			4		
5						ac									
6	S_3			S_4					8	3			4		
7	S_3			S_4					9						
8	r_2	r_2	r_2	r_2	r_2	r_2									
9	r_3	r_3	r_3	r_3	r_3	r_3									
10		S_6	S_7		S_{11}						6	7		11	
11	r_5	r_5	r_5	r_5	r_5	r_5									

问题：

如何体现是LR(0)分析？

通过上述LR(0)分析表可见, 每一行不存在下述项目---冲突项目

(1)既含移进项目,又含归约项目

(2)含有多个归约项目

这种文法称为LR(0)文法.

不满足条件可采用SLR(1), LR(1)分析法.

4、LR(0)分析器的构造 --DFA构造

LR(0)项目：给定文法的一个项目是一个在右部符号串中标有一圆点的产生式

形式：$A \rightarrow \alpha_1 \cdot \alpha_2$

$A \rightarrow \alpha_1 \alpha_2$ 为一个产生式

表示：已从输入串中看到了能由α_1推导出的符号串，希望进一步看到由α_2推导出的符号串.

例：$E \rightarrow E+T$ 项目：$E \rightarrow \cdot E+T$ $E \rightarrow E \cdot +T$

$E \rightarrow E+ \cdot T$ $E \rightarrow E+T \cdot$

4.2.4 LR分析法　　❑LR(0)分析法

- 归约项目:圆点在最后的项目

$$E \rightarrow E+T \cdot$$

- 接受项目:开始符号的归约项目

$$S \rightarrow E\# \cdot$$

- 移进项目:形如$A \rightarrow \alpha \cdot a\ \beta$项目 $a \in V_t$

$$E \rightarrow E \cdot +T$$

- 待约项目:形如$A \rightarrow \alpha \cdot B\ \beta$项目 $B \in V_n$

$$E \rightarrow E + \cdot T$$

4、LR(0)分析器的构造

- DFA M的一个状态 i

---由若干个LR(0)项目所组成的集合（项目集）C_i

- DFA M 的状态集Q

$Q=\{ C_0, C_1, C_2, \cdots\cdots, C_n \}=C$

- C 称为文法的LR(0)有效项目集规范族

(1)三种操作:

- 开始操作: S为开始符号, $S \rightarrow \delta$

  则 $S \rightarrow \cdot\delta \in C_0$

- 闭包操作: closure(C_i)　C_i的闭包

  ① C_i 的任何项目均属于closure(C_i)

  ②若$A \rightarrow \alpha \cdot X \beta$且$X \in V_n$属于closure($C_i$)

  则$X \rightarrow \cdot\lambda$ 属于closure(C_i)

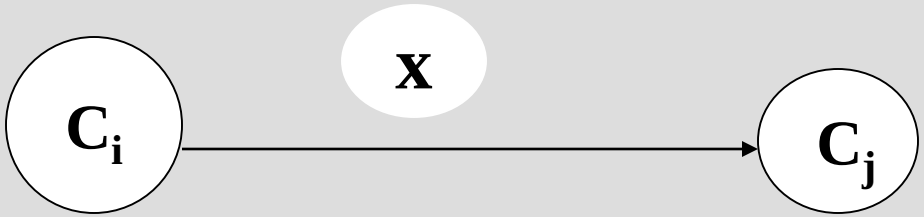
  重复,直至 closure(C_i)不再增大.

  ③ C_i=closure(C_i)

- 转移操作: $Go(C_i, x)$ $x \in V$

$$Go(C_i, x) = C_j = CLOSURE(J)$$

其中: $J=\{A \rightarrow \alpha x \cdot \beta \mid A \rightarrow \alpha \cdot x \beta \in C_i\}$



(2) 算法(求文法的LR(0)项目集规范族C)

①拓广文法，保证唯一初态.

②生成初态项目集C_0= closure(C_0)= closure(S→ ·δ)

③ C_i转移操作，Go(C_i , x) = C_j = CLOSURE(J)求出新状态C_j的项目集

重复以上过程,直至C不再增大为止。

(不出现新的项目集)

思考题

G[S]:S → A [1]

S → B [2]

A → aAb [3]

A → c [4]

B → aBb [5]

B → d [6]

构造识别G[S]全部活前缀的DFA？

G[S]:S → A [1] S → B [2] A → aAb [3] A → c [4] B → aBb [5] B → d [6]

构造识别G[S]全部活前缀的DFA？

1、拓广文法后所有项目为：

S’ → · S [1]
S’ → S ·[2]
S → · A [3]
S → A · [4]
A → · aAb [5]
A → a · Ab [6]
A → aA · b [7]
A → aAb · [8]
A → · c [9]
A → c ·[10]
S → · B [11]
S → B · [12]
B → · aBb [13]
B → a · Bb [14]
B → aB · b [15]
B → aBb · [16]
B → · d [17]
B → d · [18]

2、初态项目集C_0= closure(C_0)

= closure(S’→·S)

={S’ → · S， S → · A ， A → · aAb

A → · c， S → · B， B → · aBb， B → · d }

3、重复转移操作和闭包操作，至无新状态集：

C_1= GO(C_0, S)= closure(S’→S ·)={S’ → S ·}

C_2= GO(C_0, A)= closure(S→A ·)={S → A ·}

C_3= GO(C_0, B)= closure(S→B ·)={S→ B ·}

C_4= GO(C_0, a)= closure(A → a · Ab, B → a · Bb)

={A → a · Ab, A → · aAb, A → · c, B → a · Bb,
B → · aBb, B → · d }

C_5= GO(C_0, c)= closure(A → c ·)={A → c ·}

C_6= GO(C_0, d)= closure(B → d ·)={B → d ·}

G[S]:S → A [1] S → B [2] A → aAb [3] A → c [4] B → aBb [5] B → d [6]

构造识别G[S]全部活前缀的DFA？

1、所有项目为：

S’ → · S [1]
S’ → S ·[2]
S → · A [3]
S → A · [4]
A → · aAb [5]
A → a · Ab [6]
A → aA · b [7]
A → aAb · [8]
A → · c [9]
A → c ·[10]
S → · B [11]
S → B · [12]
B → · aBb [13]
B → a · Bb [14]
B → aB · b [15]
B → aBb · [16]
B → · d [17]
B → d · [18]

3、接上：

C_4= GO(C_0, a)= closure(A → a · Ab, B → a · Bb)

={A → a · Ab, A → · aAb, A → · c, B → a · Bb, B → · aBb, B → · d }

C_7= GO(C_4, A)= closure(A→ aA · b)={A→ aA · b}

GO(C_4, a)= closure(A → a · Ab, B → a · Bb)= C_4

GO(C_4, c)= closure(A → c ·)= C_5

C_8= GO(C_4, B)= closure(B → aB · b)={B → aB · b}

GO(C_4, d)= closure(B → d ·)= C_6

C_9= GO(C_7, b)= closure(A → aAb·)={A → aAb · }

C_{10}= GO(C_8, b)= closure(B → aBb·)={B → aBb · }

至此无新状态集，结束

识别活前缀的DFA构造与具有ε动作的NFA确定化的对比？

	识别活前缀的DFA构造	具有ε动作NFA确定化
每一个状态	项目集合	状态集合
初态	closure($S_0 \rightarrow \cdot\delta$)	ε_closure(S_0)
其余状态	转移，闭包，直至无新项目集合	转移，ε_闭包，直至无新状态集合
备注	首先拓广文法	