南京理工大学

编译原理

项欣光

计算机科学与工程学院

第6章 LR分析法

- LR分析概述
- LR(0)分析
- SLR(1)分析
- LR(1)分析
- LALR(1)分析
- 二义性文法在LR分析中的应用

6.1 LR分析概述

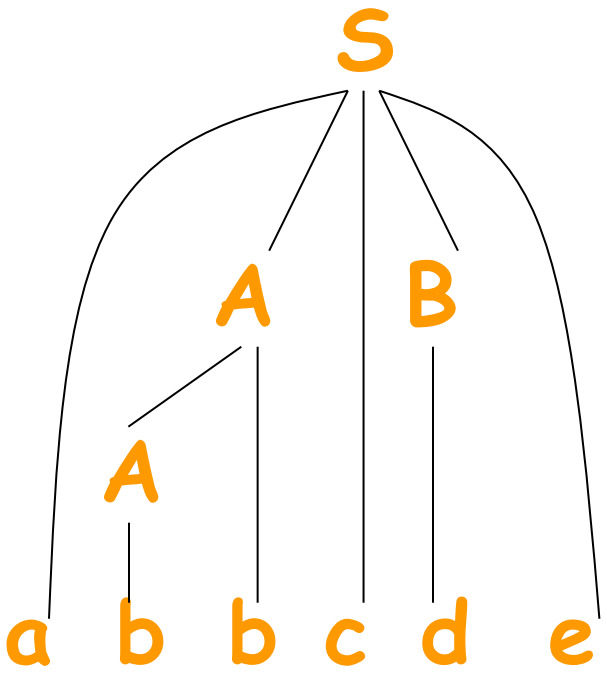
- LR分析法
  - 自底向上分析法，即移进-归约的过程
- 为什么提出LR分析法

  比LL(k)和优先分析法
  - 对文法的限制少
  - 速度快
  - 能够准确、及时地指出出错位置
- 缺点：构造文法分析器的工作量大

回顾：移进-归约过程（对输入串abbcde进行移进-归约分析）

例1：文法G[S]:

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



步骤	符号栈	输入符号串	动作
1）	#	abbcde#	移进
2）	#a	bbcde#	移进
3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进
7）	#aAc	de#	移进
8）	#aAcd	e#	归约(B→d)
9）	#aAcB	e#	移进
10）	#aAcBe	#	归约(S→aAcBe)
11）	#S	#	接受

- 问题：何时移进？何时归约？用哪个产生式归约？

LR分析的基本思想

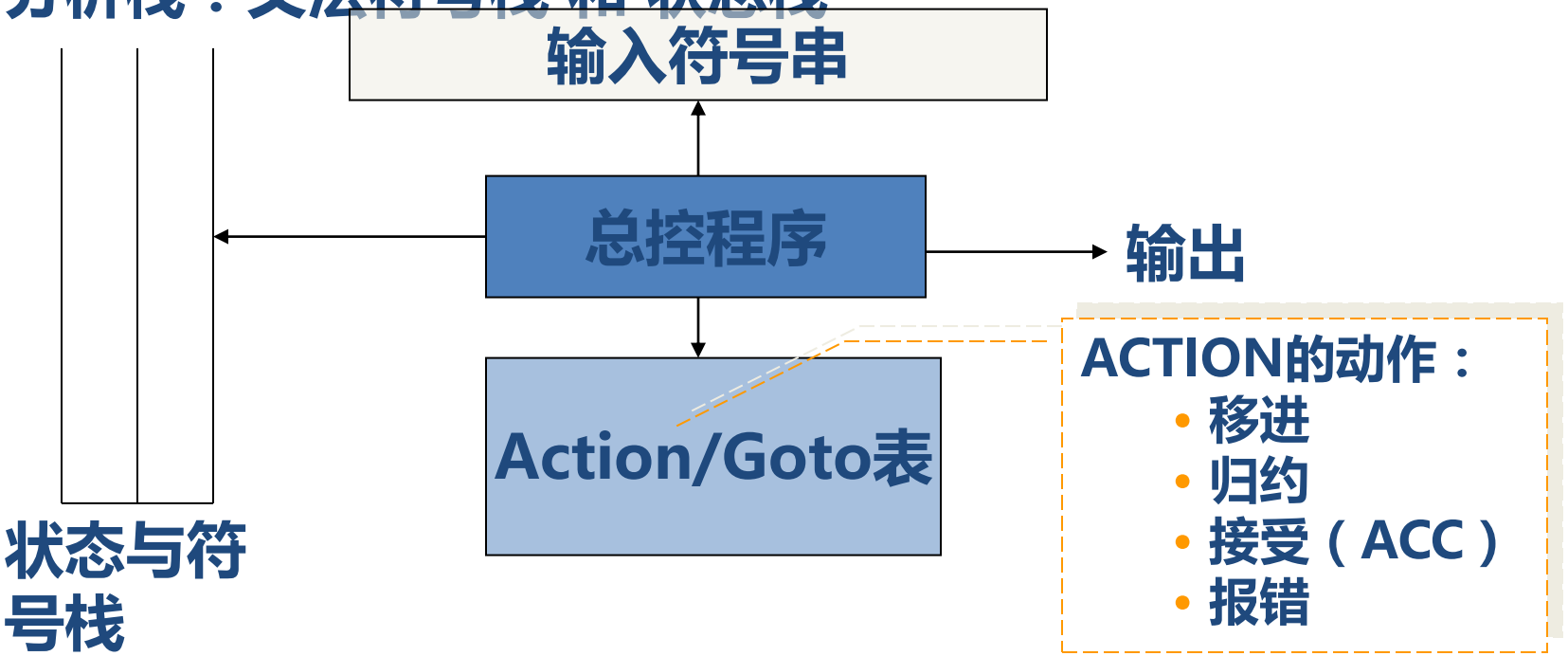
根据当前分析栈中的状态，向右顺序查看输入串中k个符号（k>=0），就可以唯一确定分析器的动作是移进还是归约。若是归约，确定用哪个产生式进行归约。

LR分析器

- LR(0)：无需向右查看输入符号（不适于高级语言）
  ——改进：SLR(1)
- LR(1)：向右查看一个输入符号（适于高级语言）
  ——改进：LALR(1)

LR分析器的组成

- 总控程序
- 分析表(或分析函数)：动作表ACTION 和 状态转换表GOTO
- 分析栈：文法符号栈 和 状态栈



LR分析器的工作过程（参见书）

LR分析

- （一）分析过程使用的栈和表：
  - 符号栈：存储已经移入或已经归约的符号串
  - 状态栈：存储状态
  - 输入符号串：存储待移入或待归约的符号串
  - ACTION：指明遇到输入符号应执行的动作：移入（及移入后应转向的下一个状态）或 归约（及所应用的产生式）或 接受 或 出错
  - GOTO：指明遇到当前文法符号（归约）后应转向的下一个状态

LR分析

- （二）状态栈顶 与 当前输入符号 查Action表：
  - 1）遇到Si：表示“移进”，并转向下一个状态。
    - 状态栈： i入栈
    - 符号栈：当前输入符号入栈
  - 2）遇到ri：表示使用第i个产生式“归约”。（所用产生式的右部符号的个数为k）
    - 状态栈：弹出k个状态，栈顶状态 与 所用产生式的左部 查GOTO表并将相应的状态入栈
    - 符号栈：弹出k个符号，并将所用产生式的左部入栈
  - 3）遇到acc：表示接受该输入串。
  - 4）遇到空白：表示出错。

LR(0)分析举例

文法G[S]：
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

对输入串 abbcde# 进行LR分析。

	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S_2						1		
1						acc			
2				S_4				3	
3		S_5		S_6					
4	r_2	r_2	r_2	r_2	r_2	r_2			
5					S_8				7
6	r_3	r_3	r_3	r_3	r_3	r_3			
7			S_9						
8	r_4	r_4	r_4	r_4	r_4	r_4			
9	r_1	r_1	r_1	r_1	r_1	r_1			

对输入串 abbcde# 的LR分析过程

步骤	符号栈	状态栈	输入符号串	动作	ACTION	GOTO
1）	#	0	abbcde#	移进	S_2	
2）	#a	02	bbcde#	移进	S_4	
3）	#ab	024	bcde#	归约(A→b)	r_2	3
4）	#aA	023	bcde#	移进	S_6	
5）	#aAb	0236	cde#	归约(A→Ab)	r_3	3
6）	#aA	023	cde#	移进	S_5	
7）	#aAc	0235	de#	移进	S_8	
8）	#aAcd	02358	e#	归约(B→d)	r_4	7
9）	#aAcB	02357	e#	移进	S_9	
10）	#aAcBe	023579	#	归约(S→aAcBe)	r_1	1
11）	#S	01	#	接受	acc	

可见：LR分析表非常重要！！

问题：

- 对于一个文法，状态集是如何确定的？
- LR分析表是如何得到的？

构造LR分析表之前，要了解一些概念、术语和方法：

- 可归前缀、活前缀
- 得到可归前缀、活前缀的一般方法：构造识别活前缀的DFA
- 利用DFA构造LR分析表的方法

NEXT

可归前缀与活前缀

文法G[S]：
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

LR分析过程：

…

3）	#ab	024	bcde#	归约(A→b)	r_2	3

…

5）	#aAb	0236	cde#	归约(A→Ab)	r_3	3
8）	#aAcd	02358	e#	归约(B→d)	r_4	7
10）	#aAcBe	023579	#	归约(S→aAcBe)	r_1	1

可归前缀

每次归约之前符号栈中的内容。如：ab、aAb、aAcd、

活前缀

形成可归前缀之前包括可归前缀在内的所有规范句型的前缀。

如： ε , a , ab
ε , a , aA , aAb
ε , a , aA , aAc , aAcd
ε , a , aA , aAc , aAcB , aAcBe

活前缀

- 活前缀 定义：

  $S' \overset{*}{\Rightarrow} \alpha A\omega \Rightarrow \alpha\beta\omega$ 是文法G中的一个规范推导，

  若符号串γ是αβ的前缀，则称γ是G的一个活前缀

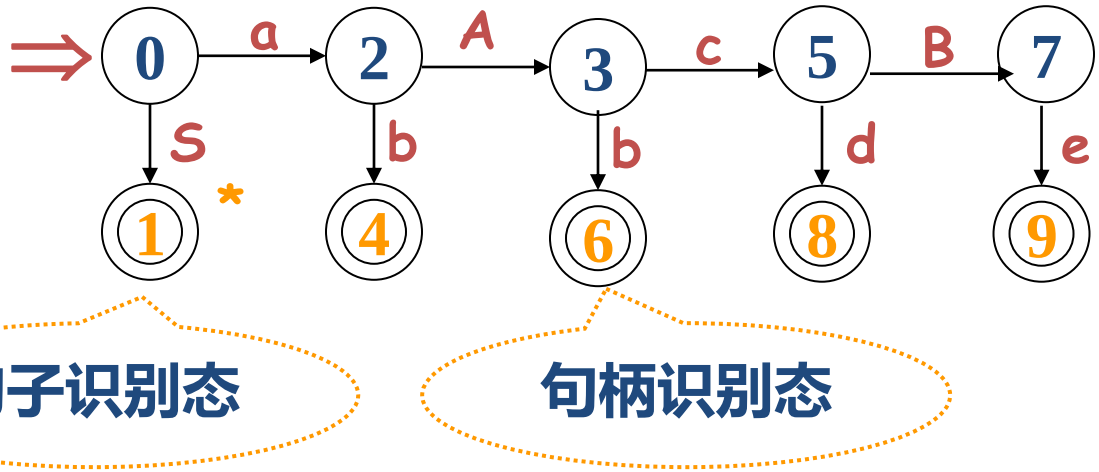
  。

返 回

识别活前缀的有穷自动机

- 把文法的终结符和非终结符都看成有穷自动机的输入符号，每次把一个符号进栈看成已识别过了该符号，同时状态进行转换。当识别到可归前缀时，相当于在栈中形成句柄，认为达到了识别句柄的终态。
- 如文法G[S]对应的识别活前缀的DFA

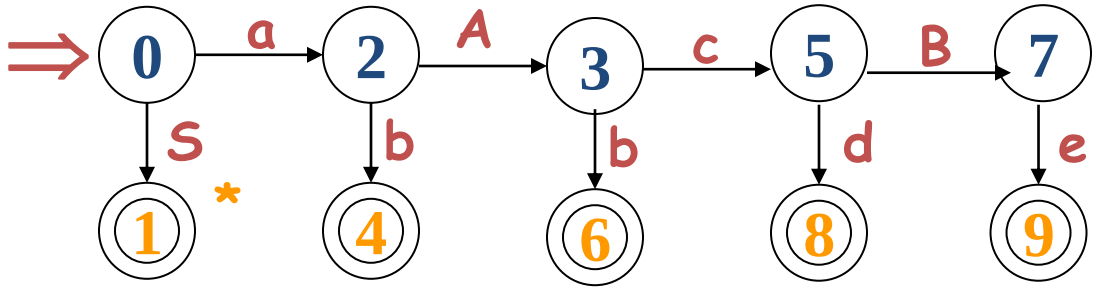
(1) S→aAcBe
(2) A→b
(3) A→Ab
(4) B→d



识别活前缀的有穷自动机

例如文法G[S]对应的识别活前缀的DFA如图所示：

(1) S→aAcBe
(2) A→b
(3) A→Ab
(4) B→d



利用DFA对输入串 abbcde# 的分析过程：

步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1	#	abbcde#	移进	0	S2	
2	#a	bbcde#	移进	02	S4	
3	#ab	bcde#	归约(A→b)	024	r2	3
4	#aA	bcde#	移进	023	S6	
5	#aAb	cde#	归约(A→Ab)	0236	r3	3
6	#aA	cde#	移进	023	S5	
7	#aAc	de#	移进	0235	S8	
8	# aAcd	e#	归约(B→d)	02358	r4	7
9	#aAcB	e#	移进	02357	S9	
10	#aAcBe	#	归约(S→aAcBe)	023579	r1	1
11	#S	#	接受	01	acc	

构造识别活前缀的有穷自动机

方法一：形式定义法

- 拓广文法　　（S'→S）
- 根据形式定义求活前缀的正规表达式和可归前缀
- 根据可归前缀构造NFA
- NFA → DFA

方法二：项目法

- 拓广文法　　（S'→S）
- 求出文法的所有项目
- 按照一定规则构造NFA
- NFA → DFA　(工作量大，所以引入方法三)

方法三：项目集规范族法

- 拓广文法　　（S'→S）
- 通过核的闭包和转换函数，求出LR(0)项目集规范族
- 由转换函数建立状态之间的连接关系，即得到DFA

返 回

构造识别活前缀的有穷自动机（形式定义法）

形式定义一（不包含句柄在内的所有活前缀的集合）：

文法G，$A \in V_N$，$LC(A)=\{\alpha \mid S' \overset{*}{\underset{R}{\Rightarrow}} \alpha A\omega, \alpha \in V^*, \omega \in V_T^*\}$

即：规范推导中，在A左边所有可能出现的符号串的集合

推论：若文法G中有产生式$B \to \gamma A \delta$，则 $LC(A) \supseteq LC(B) \cdot \{\gamma\}$

形式定义二（包含句柄在内的所有活前缀的集合）：

$LR(0)C(A \to \beta) = LC(A)\{\beta\}$

例如：
S'→S
S→aAcBe
A→b
A→Ab
B→d

求：
LC(S')
LC(S)
LC(A)
LC(B)

LR(0)C(S'→S)
LR(0)C(S→aAcBe)
LR(0)C(A→b)
LR(0)C(A→Ab)
LR(0)C(B→d)

S'→S S→aAcBe A→b A→Ab B→d

- LC(S') = {ε}
- LC(S) = LC(S') {ε} = {ε}
- LC(A) = LC(S) {a} ∪ LC(A) {ε}= {a}
- LC(B)= LC(S) {aAc}= {aAc}

这样我们求出了规范归约过程中用句柄归约成该非终结符之前不包括句柄在内的活前缀。

- LR(0)C(S'→S) = LC(S') {S} =S
- LR(0)C(S→aAcBe) = LC(S) {aAcBe} =aAcBe
- LR(0)C(A→b) = LC(A) {b} =ab
- LR(0)C(A→Ab) = LC(A) {Ab} =aAb
- LR(0)C(B→d) = LC(B) {d} =aAcd
- **这样我们构造出了文法的所有可归前缀。**
- **如何根据可归前缀构造识别文法活前缀的有限自动机？**

文法G[S]：

S' → S

S → aAcBe

A → b

A → Ab

B → d

根据定义求得的可归前缀如下：

S

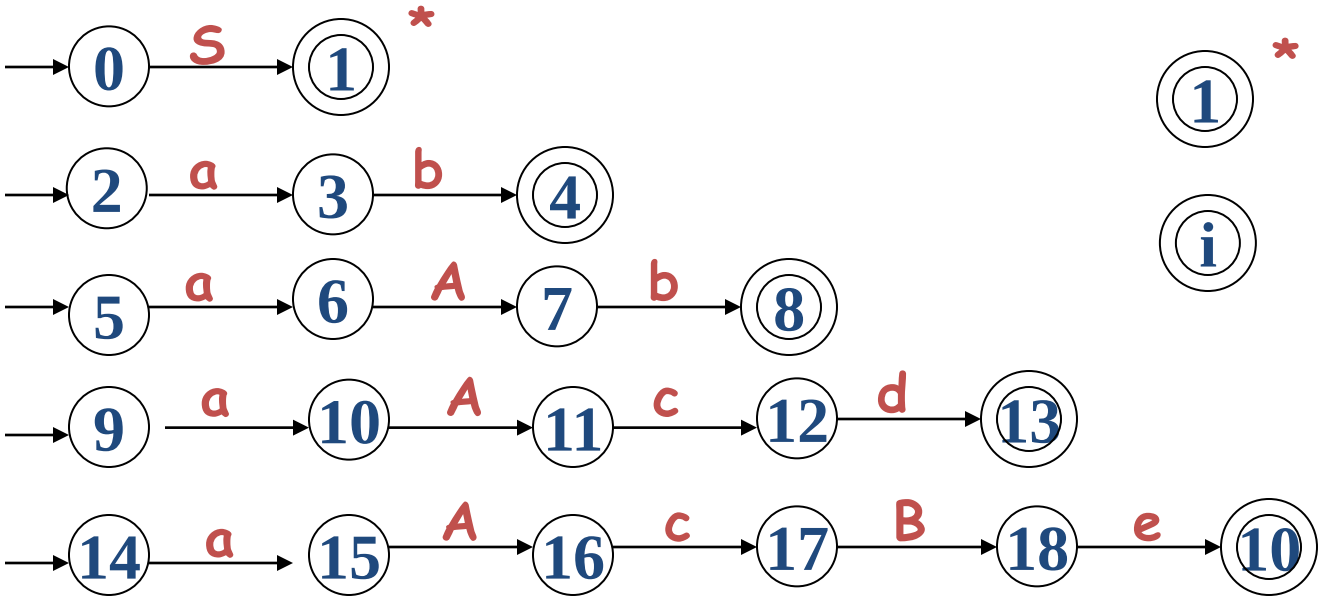
ab

aAb

aAcd

aAcBe

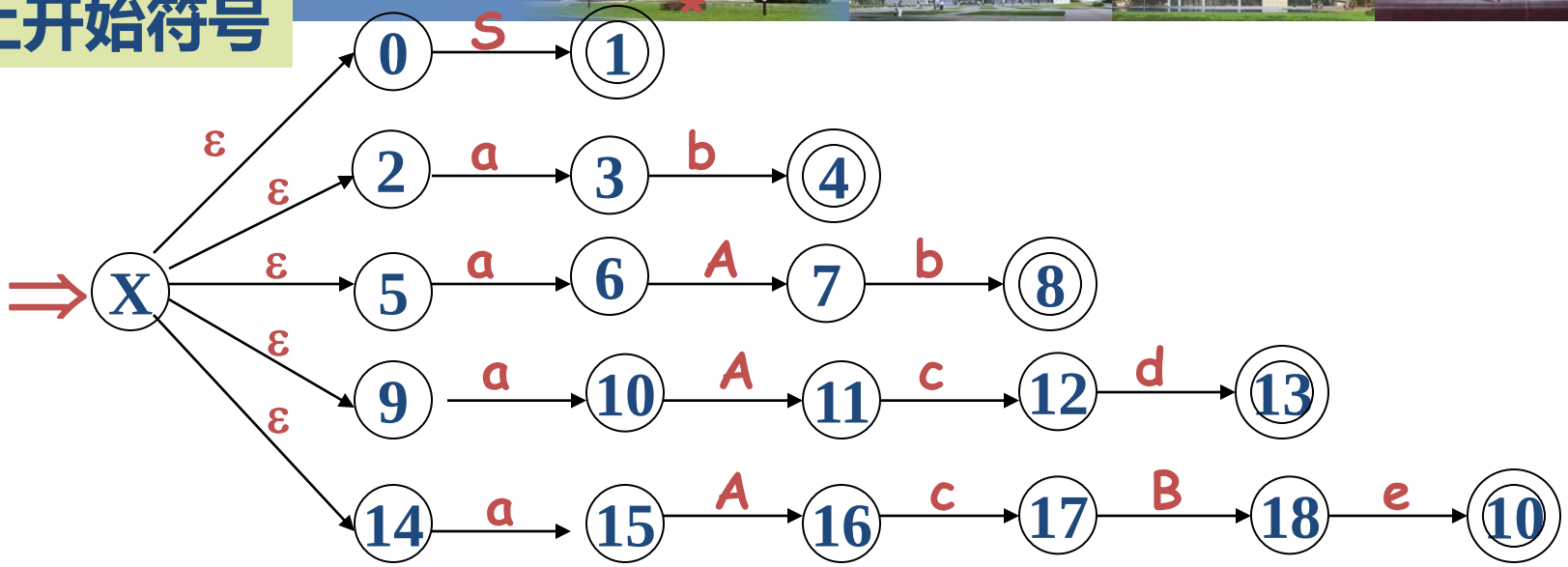
构造识别其活前缀及可归前缀的有限自动机如下：



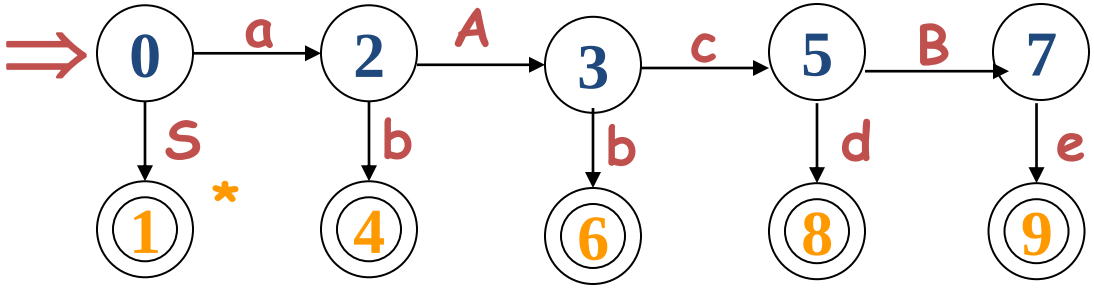
句子识别态

句柄识别态

加上开始符号


0
S
1
ε
2
a
3
b
4
ε
ε
X
5
a
6
A
7
b
8
ε
9
a
10
A
11
c
12
d
13
ε
14
a
15
A
16
c
17
B
18
e
10

确定化


0
a
2
A
3
c
5
B
7
S
b
b
d
e
1
*
4
6
8
9




形式定义法构造识别活前缀的有限自动机

从理论的角度讲是比较严格的，但实现起来却很复杂。

是否存在一种比较实用的方法？

返 回

构造识别活前缀的有穷自动机（项目法）

1. **LR(0)项目（item）：在每个产生式的右部适当位置添加一个圆点，构成项目。**

例如：产生式 S→aAcBe 对应有6个项目：

[0] S → • aAcBe
[1] S → a • AcBe
[2] S → aA • cBe
[3] S → aAc • Be
[4] S → aAcB • e
[5] S → aAcBe •

左部 • 右部

已识别部分　　待识别部分

特例：空产生式 A → ε 只有一个项目 A → •

2. 根据项目构造识别活前缀的 DFA

1) 构造文法的所有产生式的所有项目，每个项目都为NFA的一个状态

- 移进项目：A→α•aβ （a∈VT)分析时把a移进符号栈
- 待约项目：A→α•Bβ （B∈VN)期待着先归约为B再归约为A
- 归约项目：A→α• 表明句柄形成，可以归约
- 接受项目：S’→α• 表明接受句子，分析成功

2) 确定初态、句柄识别态、句子识别态

- 初态：S’ → • S
- 句柄识别态：所有的归约项目
- 句子识别态：接受项目

2. 根据项目构造识别活前缀的 DFA

3) 确定状态之间的转换关系

- 若 项目 i 为 $X \rightarrow X_1X_2...X_{i-1} \cdot X_i...X_n$
  项目 j 为 $X \rightarrow X_1X_2...X_{i-1}\ X_i \cdot X_{i+1}...X_n$
  则从状态i到状态j连一条标记为 X_i 的箭弧
- 若 项目i为$X \rightarrow \gamma \cdot A\delta$ ，项目k为$A \rightarrow \cdot\beta$
  则从状态i画标记为 ε 的箭弧到状态k

4) NFA→DFA

例如文法G：

$S' \rightarrow E$

$E \rightarrow T + E$

$E \rightarrow T$

$T \rightarrow i * T$

$T \rightarrow i$

$T \rightarrow (E)$

文法G：

S’ → E
E → T + E
E → T
T → i * T
T → i
T → (E)

① 文法的项目有：

[1] S’ → • E
[2] S’ → E •
[3] E → • T + E
[4] E → T • + E
[5] E → T + • E
[6] E → T + E •
[7] E → • T
[8] E → T •
[9] T → • i * T
[10] T → i • * T
[11] T → i * • T
[12] T → i * T •
[13] T → • i
[14] T → i •
[15] T → •(E)
[16] T → (• E)
[17] T → (E •)
[18] T → (E) •

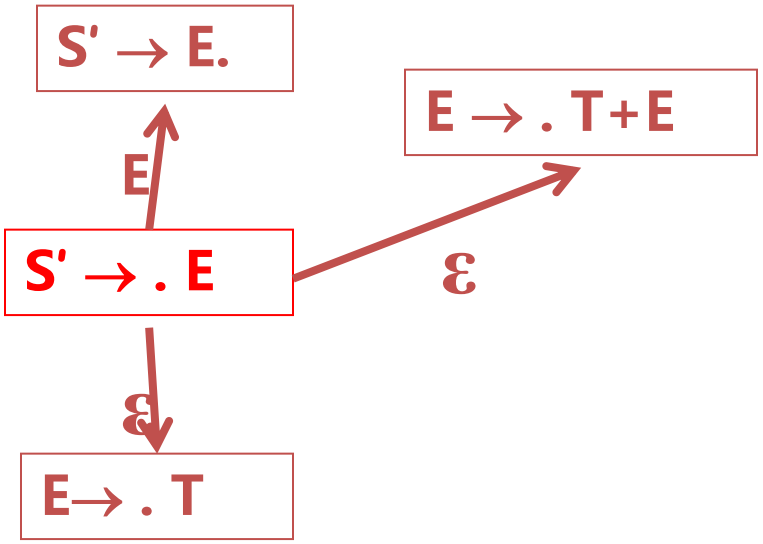
② 确定初态、句柄/句子识别态

[1]（初态）
[2]（句子识别态）
[6]（句柄识别态）
[8]（句柄识别态）
[12]（句柄识别态）
[14]（句柄识别态）
[18]（句柄识别态）

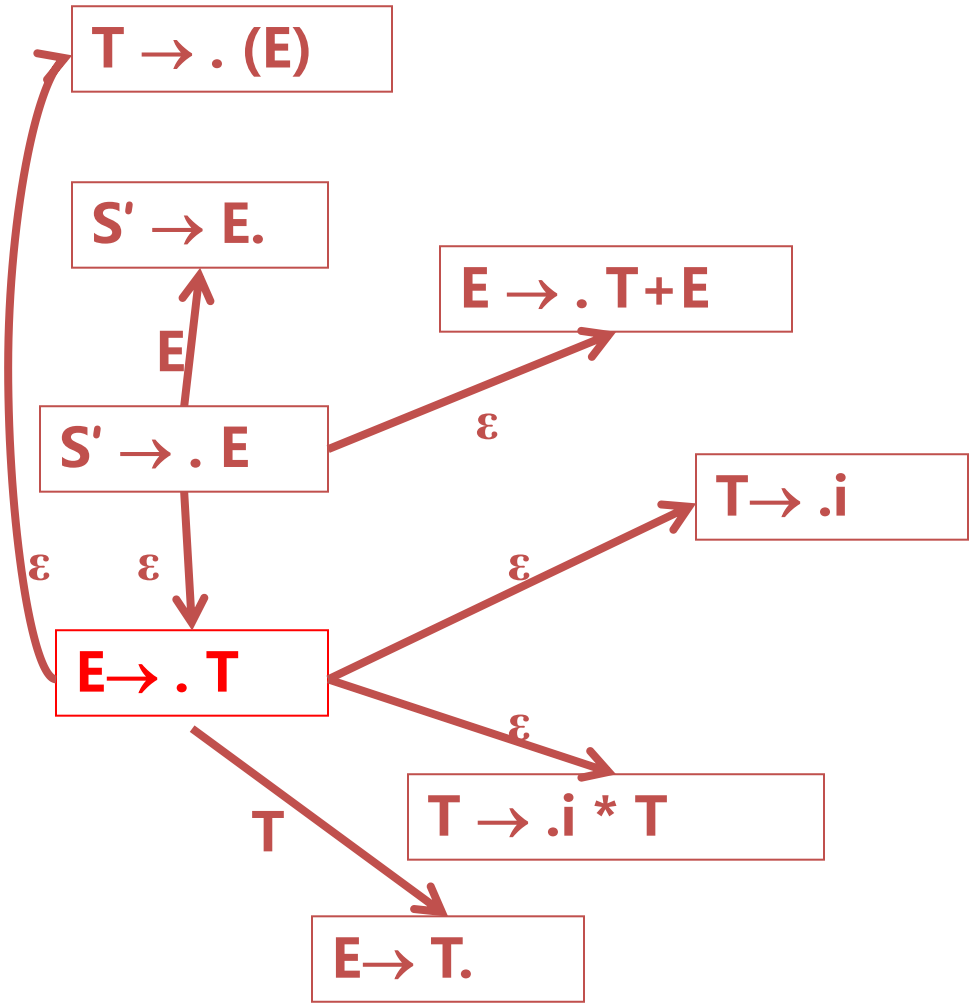
③ 确定状态之间的转换关系

$S' \rightarrow .\ E$

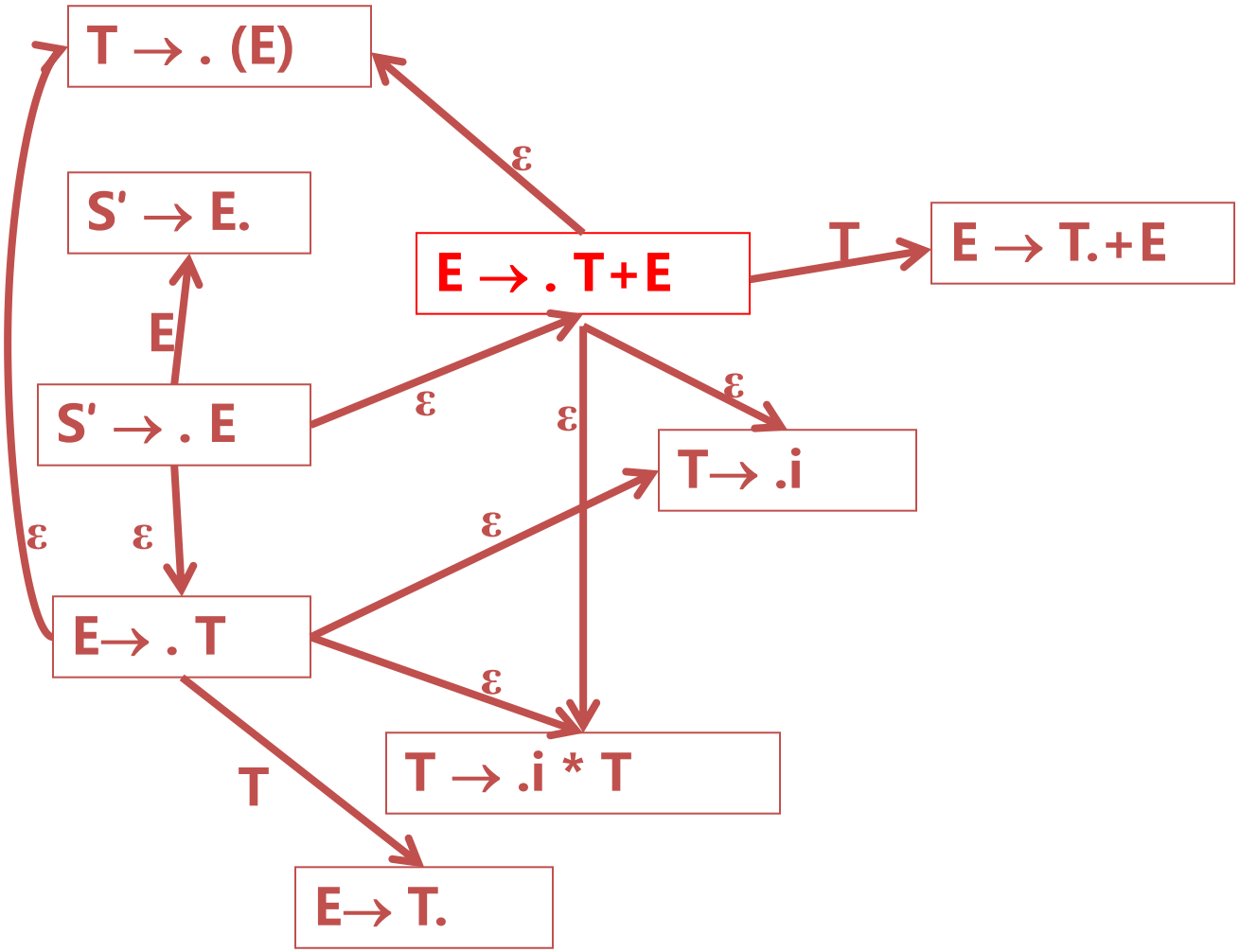
③ 确定状态之间的转换关系



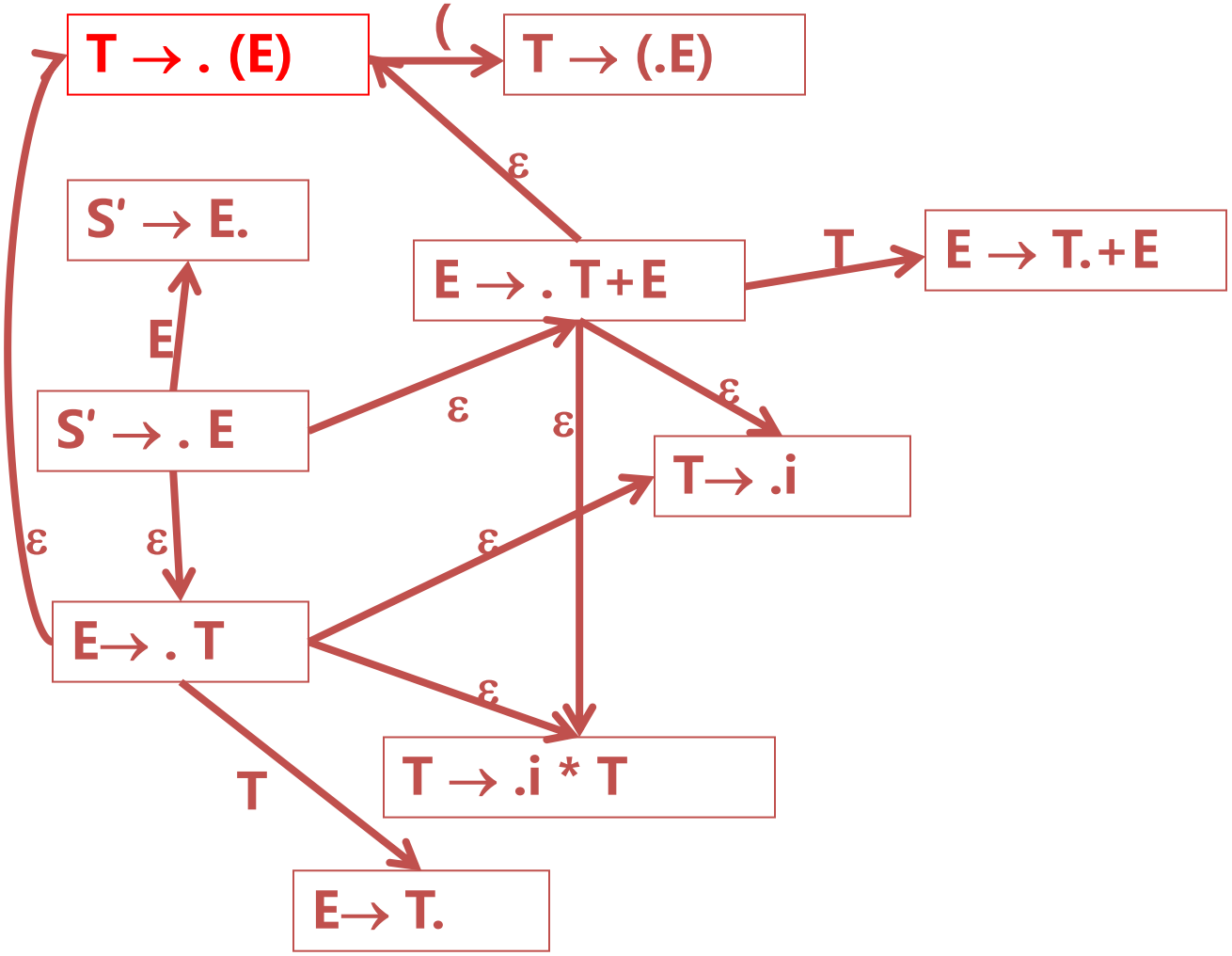
③ 确定状态之间的转换关系



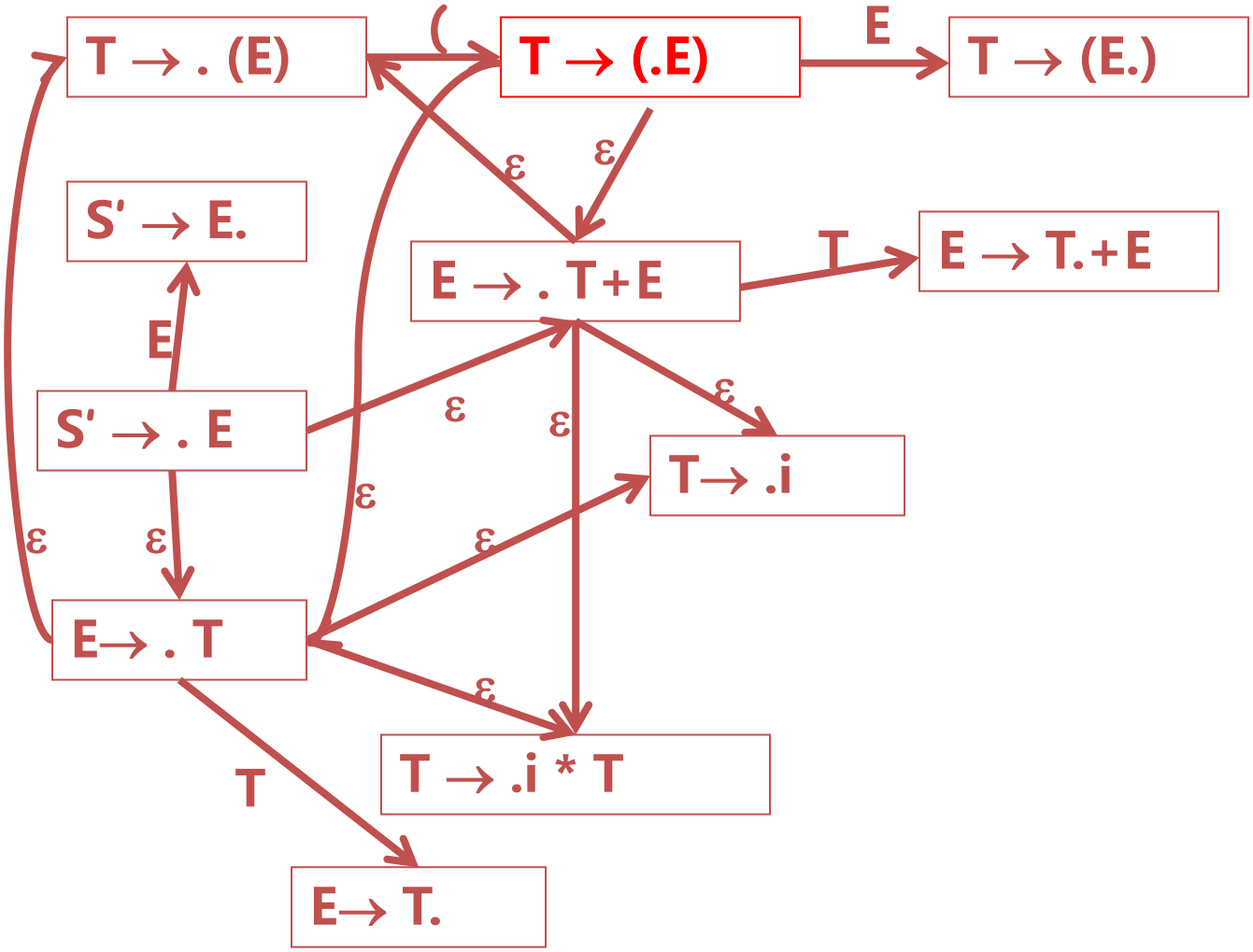
③ 确定状态之间的转换关系



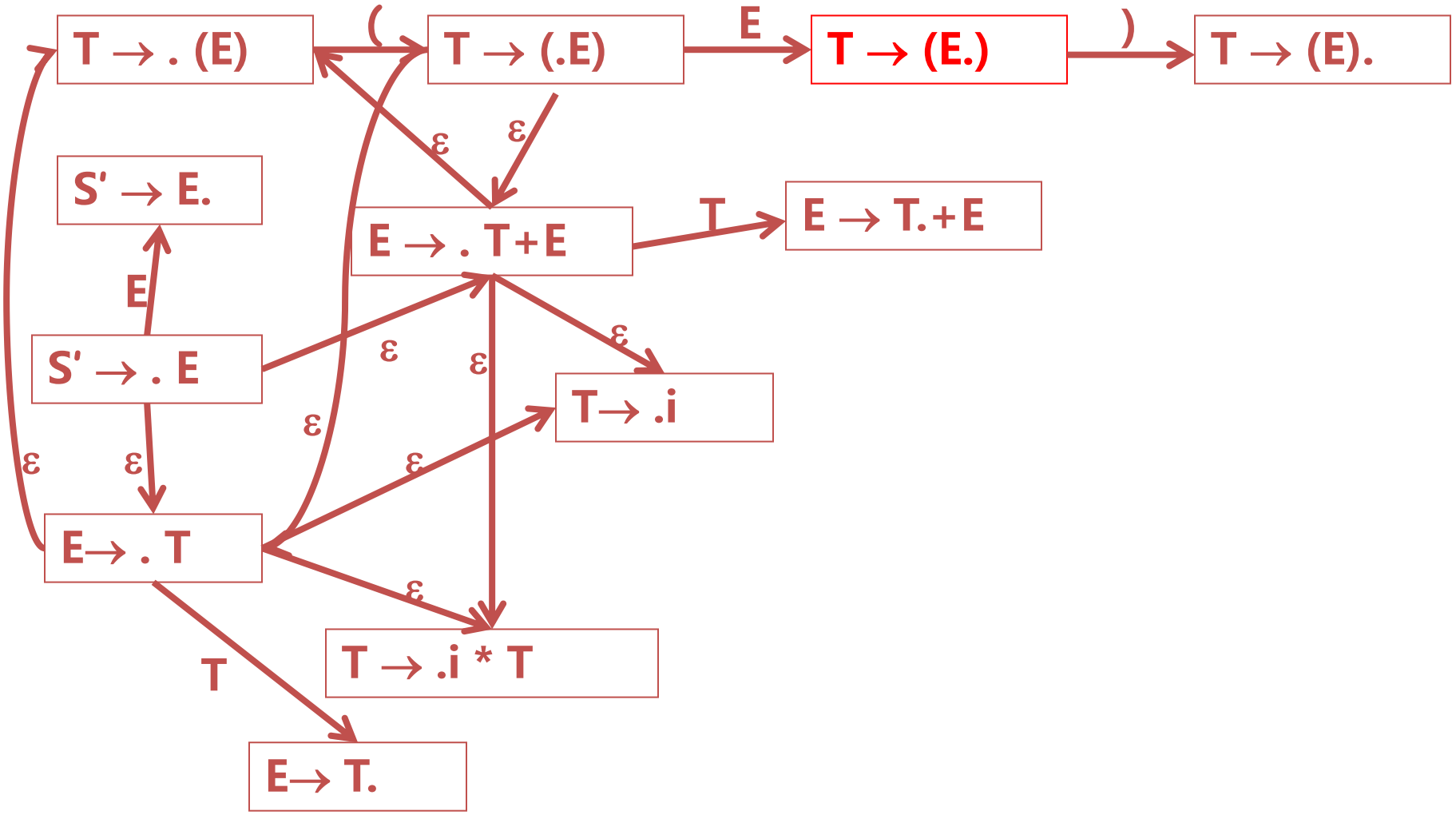
③ 确定状态之间的转换关系



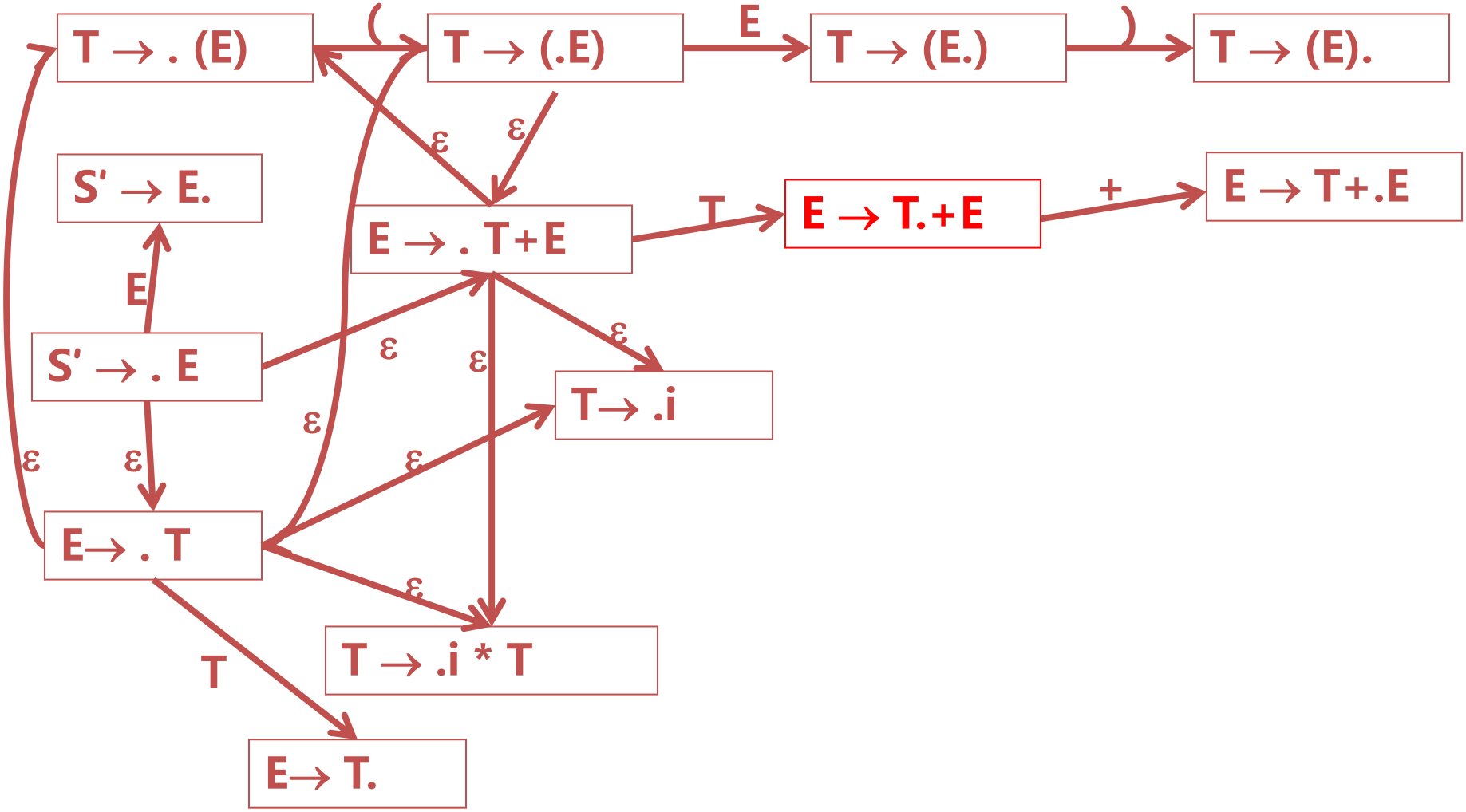
③ 确定状态之间的转换关系



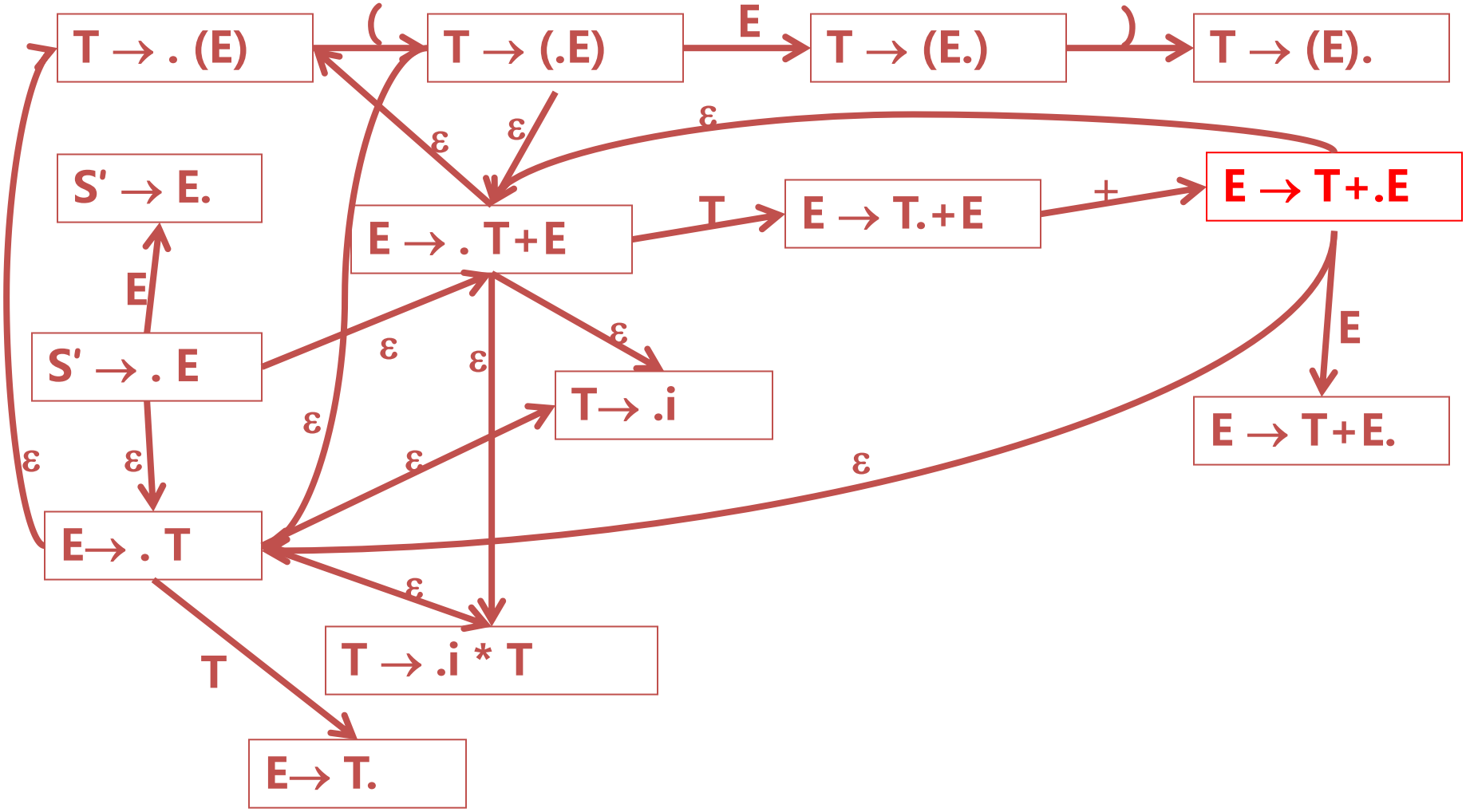
③ 确定状态之间的转换关系



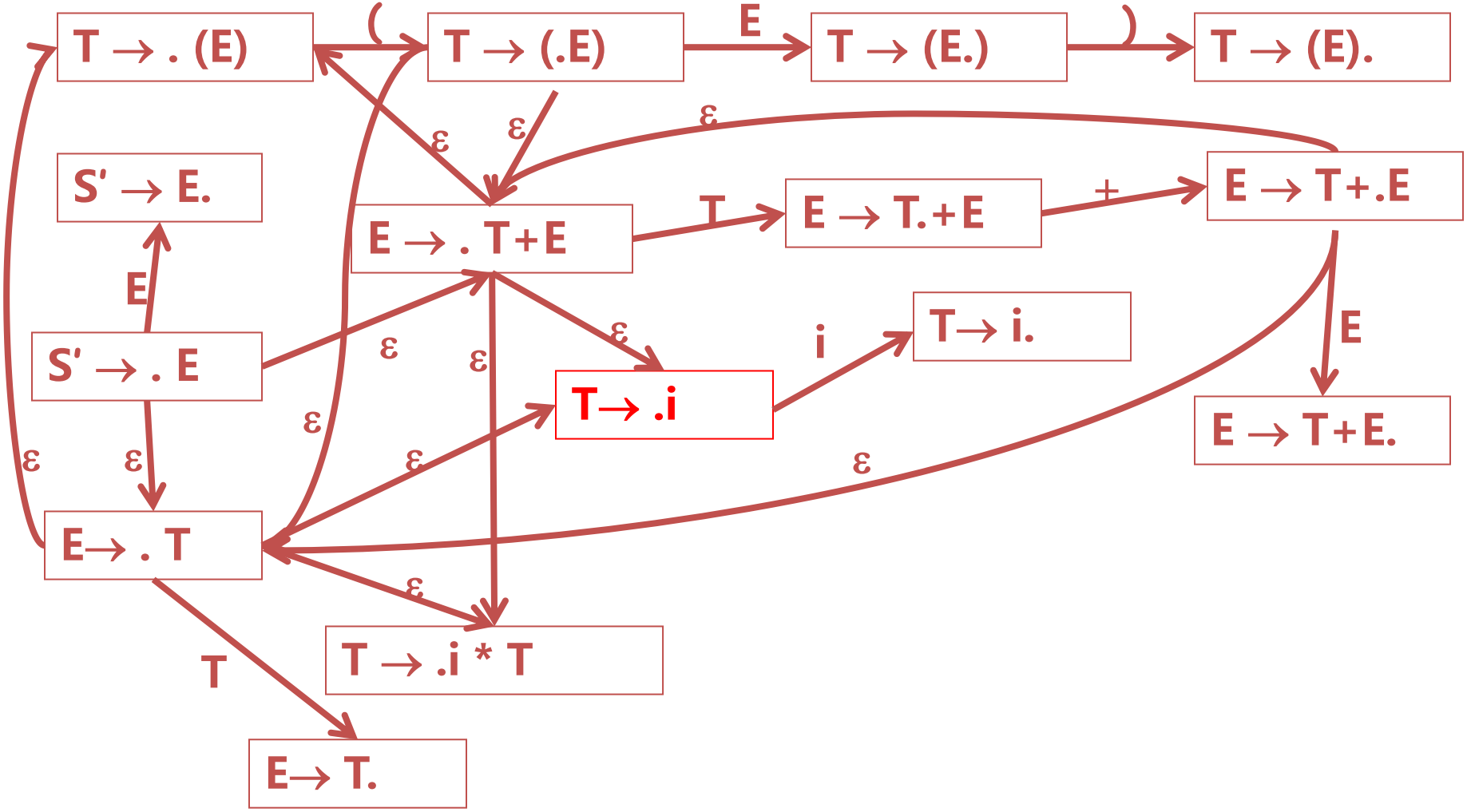
③ 确定状态之间的转换关系



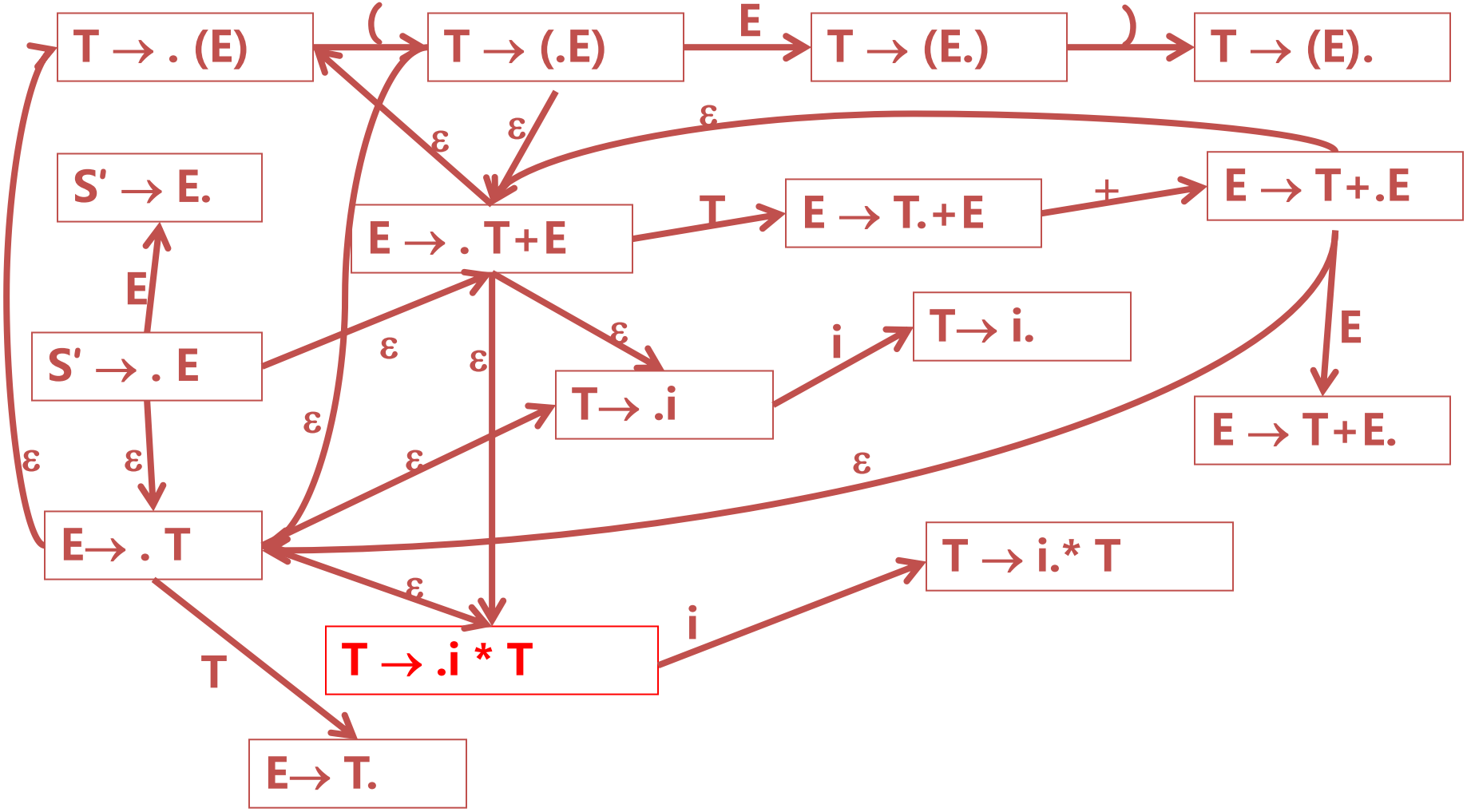
③ 确定状态之间的转换关系



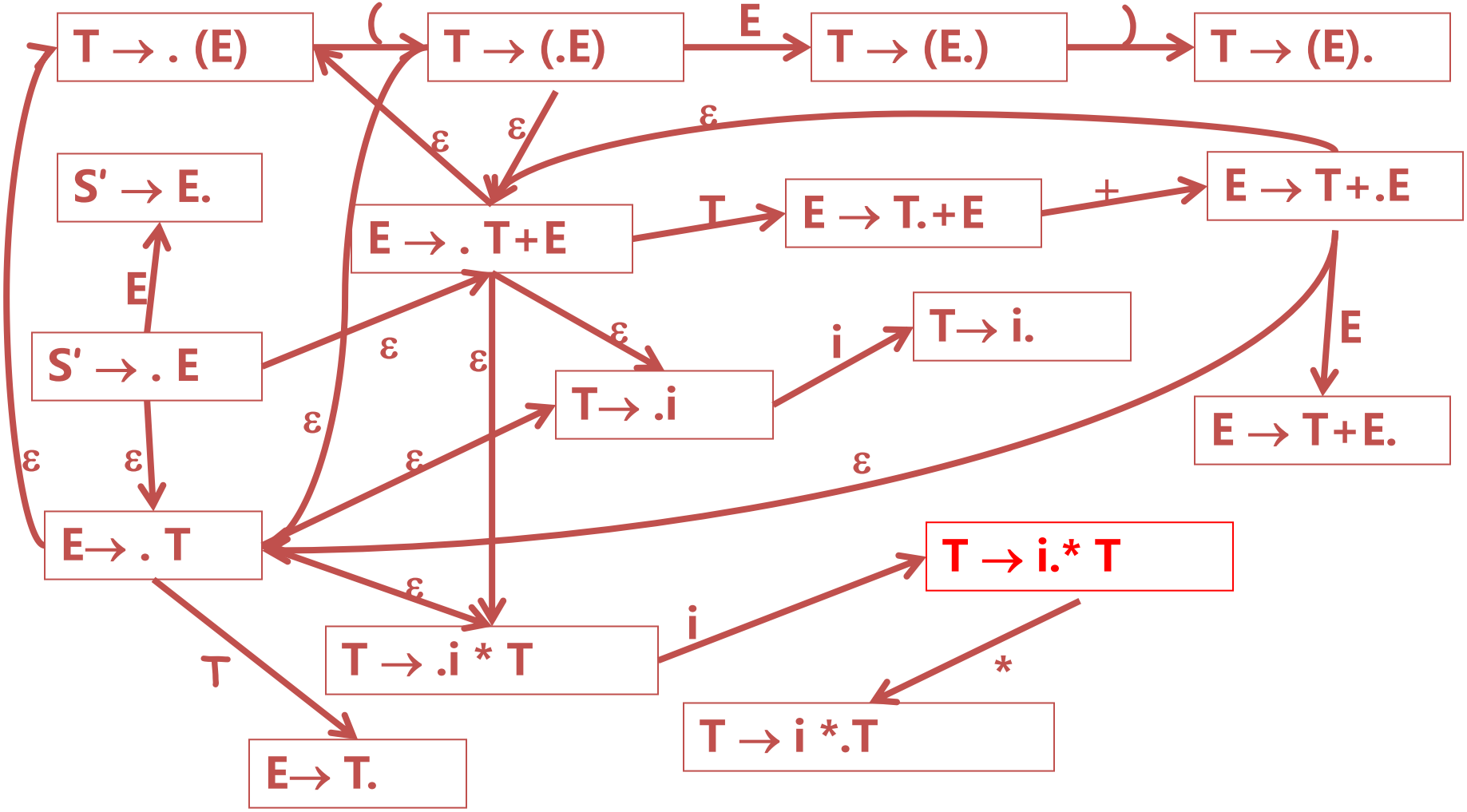
③ 确定状态之间的转换关系



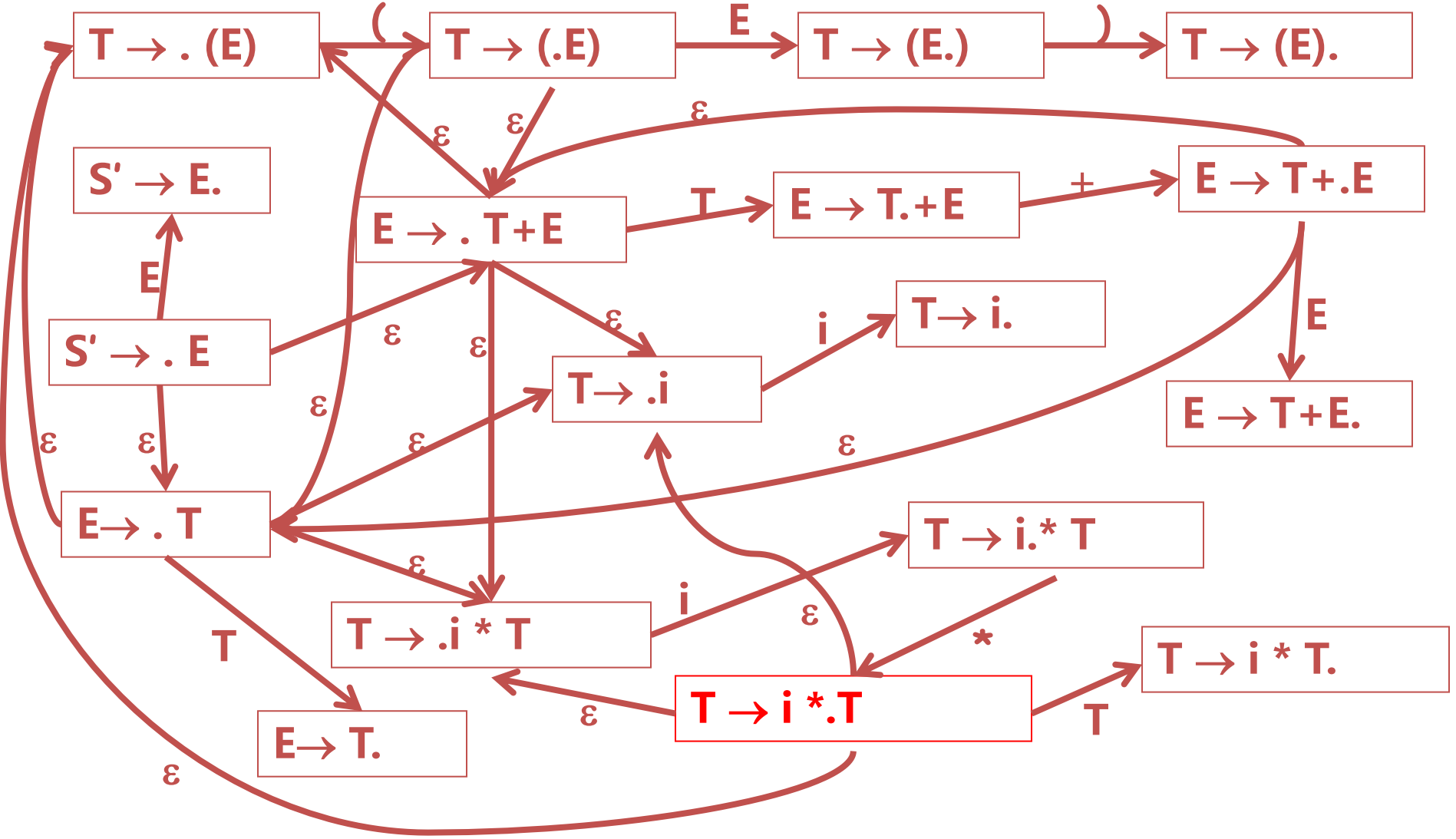
③ 确定状态之间的转换关系



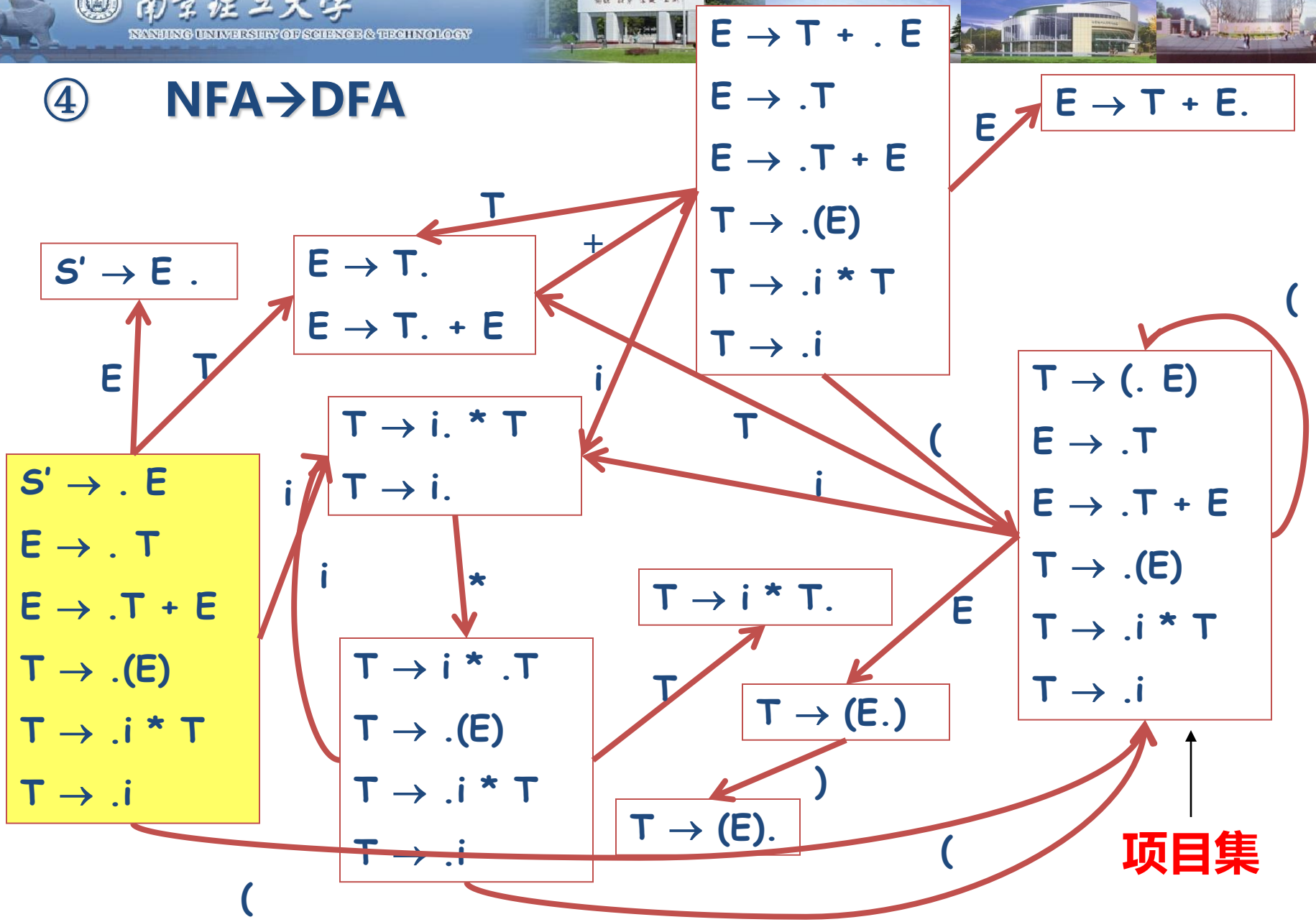
③ 确定状态之间的转换关系



③ 确定状态之间的转换关系（全）



④ NFA→DFA


E → T + . E
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
E → T + E.
E
T
S' → E .
E → T.
E → T. + E
+
E
T
i
T → i. * T
T → i.
T
(
T → (. E)
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
(
S' → . E
E → . T
E → .T + E
T → .(E)
T → .i * T
T → .i
i
i
i
*
T → i * .T
T → .(E)
T → .i * T
T → .i
T → i * T.
T
E
T → (E.)
)
T → (E).
(
(
项目集

- **LR(0)项目集规范族：识别文法活前缀的DFA**

项目集（状态）的全体。

项目法的缺点：NFA确定化为DFA的工作量较大。

是否能直接构造出 LR(0)项目集规范族，并直接构造出 识别活前缀的DFA？

返 回

构造识别活前缀的有穷自动机（项目集规范族法）

1、概念

- 核：圆点不在产生式最左边的项目，称为核。特例：S'→•S也是核。
- 闭包：CLOSURE(J)
  a) J的项目均在CLOSURE(J)中
  b) 若 A→α•Bβ 属于CLOSURE(J)，则每一形如 B→•γ 的项目也属于 CLOSURE(J)
  c) 重复b)直到 CLOSURE(J)不再扩大

  对核求闭包就构成了新状态的项目集
- 转换函数：GO（I，X）

  GO（I，X）= CLOSURE（J）

  其中：I为包含某一项目集的状态，X为一文法符号

  J={ 任何形如 A→αX•β 的项目，A→α•Xβ 属于I }

构造识别活前缀的有穷自动机（项目集规范族法）

2、利用闭包和转换函数构造文法的LR(0)项目集规范族

1) 置项目S'→.S为初态集的核，之后对核求闭包，得到初态的项目集
2) 对项目集应用转换函数GO(I,X)=CLOSURE(J)求出新状态J的项目集
3) 重复2），直至不出现新的项目集为止

例如 文法G：(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

构造对应的LR(0)项目集规范族。

步骤一：拓广文法

(0) S’ → S
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

步骤二：构造项目集规范族

I_0
$S' \rightarrow \cdot S$
$S \rightarrow \cdot aAcBe$

$I_1[I_0 - S \rightarrow I_1]$
$S' \rightarrow S\cdot$

$I_2[I_0 - a \rightarrow I_2]$
$S \rightarrow a\cdot AcBe$
$A \rightarrow \cdot b$
$A \rightarrow \cdot Ab$

$I_3[I_2 - A \rightarrow I_3]$
$S \rightarrow aA\cdot cBe$
$A \rightarrow A\cdot b$

$I_4[I_2 - b \rightarrow I_4]$
$A \rightarrow b\cdot$

$I_5[I_3 - c \rightarrow I_5]$
$S \rightarrow aAc\cdot Be$
$B \rightarrow \cdot d$

$I_6[I_3 - b \rightarrow I_6]$
$A \rightarrow Ab\cdot$

$I_7[I_5 - B \rightarrow I_7]$
$S \rightarrow aAcB\cdot e$

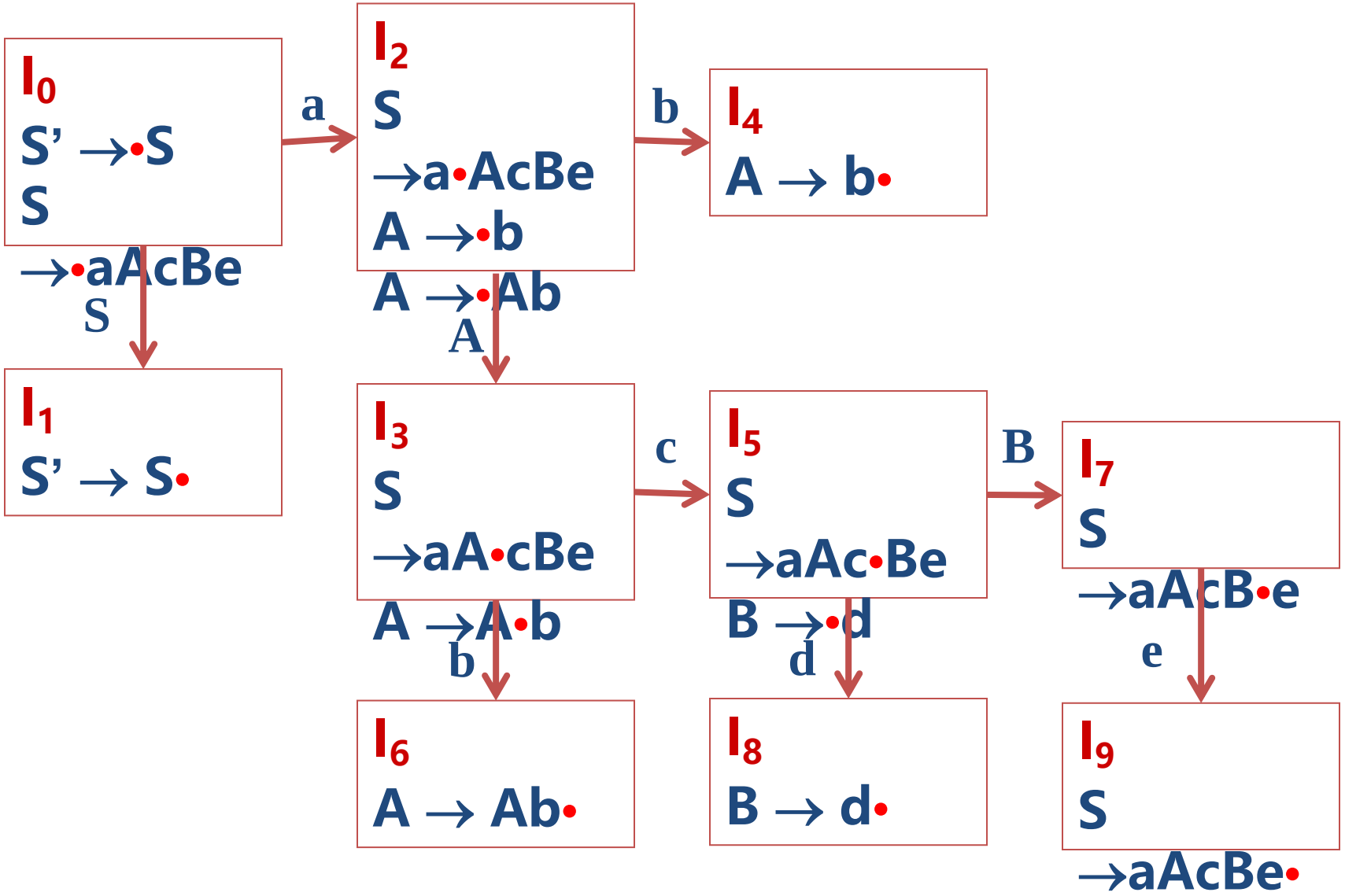
$I_8[I_5 - d \rightarrow I_8]$
$B \rightarrow d\cdot$

$I_9[I_7 - e \rightarrow I_9]$
$S \rightarrow aAcBe\cdot$

步骤三：判别（无“移进-归约冲突”和“归约-归约冲突”）

- LR(0)项目类型：
  - 移进项目：A→α•aβ （a∈VT)
  - 待约项目：A→α•Bβ （B∈VN)期待着先归约为B再归约为A
  - 归约项目：A→α•　　句柄形成，可以归约
  - 接受项目：S’→α•　　接受句子，分析成功
- LR(0)的项目集中：
  - 不能有移进-归约冲突（移进项目与归约项目并存）
  - 不能有归约-归约冲突（归约项目与归约项目并存）
  - 若存在这些冲突，则不是LR(0)文法，不能采用LR(0)分析
- LR(0)文法：若其LR(0)项目集规范族不存在移进-归约，或归约-归约冲突，称为LR(0)文法。

步骤四：根据项目集规范族 构造 DFA



返 回

LR(0)分析表的构造

- **LR(0)**分析表相当于识别活前缀的有限自动机**DAF**的状态转换矩阵。
- **LR(0)**分析表的重要性：是总控程序的分析动作依据。
- **LR(0)**分析表的结构：

状态	ACTION（动作表）	GOTO（转换表）
	(V_T) . . . #	(V_N) . . .
状态号	（表示当前状态面临某输入符号时，应采取的动作：移进/归约/接受/报错）	（表示当前状态面临文法符号VN时应转向的下一个状态）

LR(0)分析表的构造

- LR(0)分析表的构造算法　　C={ I0, I1, …, In }

1) 若移进项目A→α•aβ属于Ik，且转换函数GO(Ik,a)=Ij
　　则 置ACTION[k,a] = Sj

2) 若GO(Ik,A)=Ij
　　则 置GOTO[k,A]为j

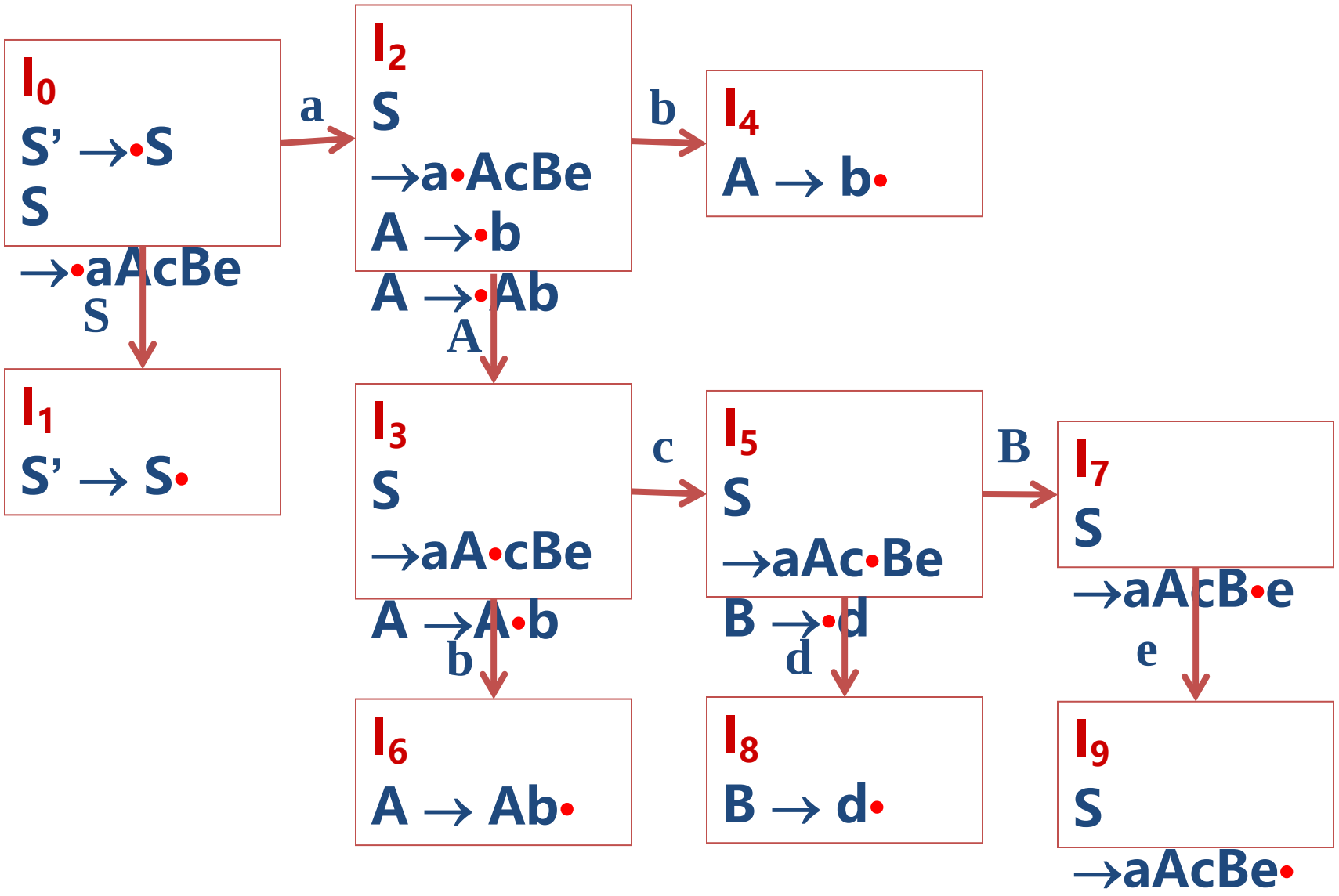
3) 若归约项目A→α• 属于Ik,
　　则 对任何 VT和#，置 ACTION[k,…]=r_j

j为A→α产生式的序号

4) 若接受项目S'→S.属于Ik
　　则 置ACTION[k,#] = acc

5) 凡是不能用上述方法填入的，即用空白表示报错

例如：构造如下DFA对应的 LR(0)分析表



LR(0)分析表

状态	ACTION						GOTO		
	a	b	c	d	e	#	S	A	B
0	S2						1		
1						acc			
2		S4						3	
3		S6	S5						
4	r2	r2	r2	r2	r2	r2			
5				S8					7
6	r3	r3	r3	r3	r3	r3			
7					S9				
8	r4	r4	r4	r4	r4	r4			
9	r1	r1	r1	r1	r1	r1			

另例

文法G[E]

(1) E → T + E

(2) E → T

(3) T → i * T

(4) T → i

(5) T → (E)

步骤一：拓广文法

(0) S’ → E

(1) E → T + E

(2) E → T

(3) T → i * T

(4) T → i

(5) T → (E)

步骤二：构造LR(0)项目集规范族

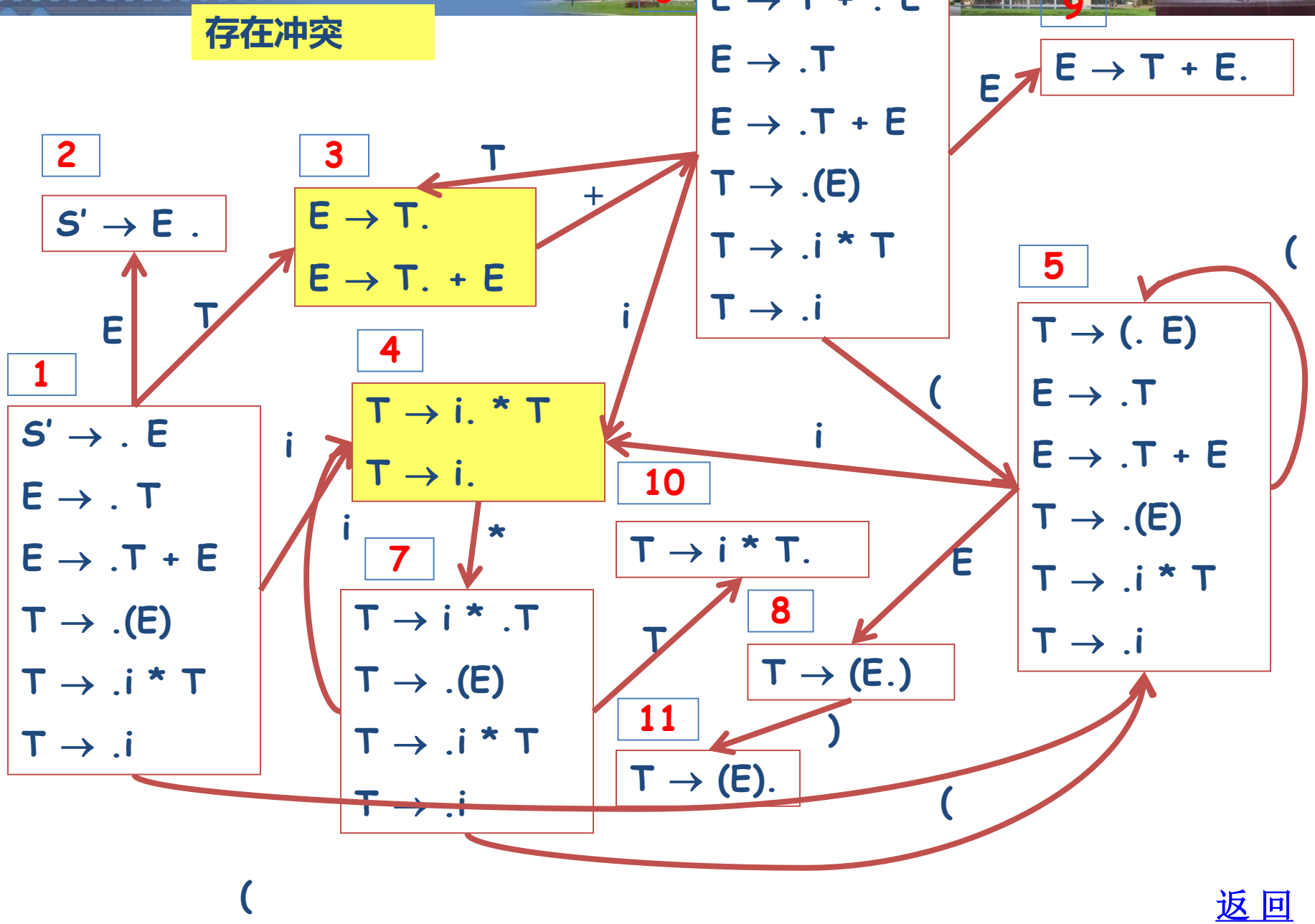
步骤三：判别LR(0)项目集中是否存在冲突（移进-归约冲突 和 归约-归约冲突）

步骤四：构造DFA

步骤五：构造 LR(0)分析表

步骤六：分析句子

存在冲突


1
S' → . E
E → . T
E → .T + E
T → .(E)
T → .i * T
T → .i
2
S' → E .
3
E → T.
E → T. + E
4
T → i. * T
T → i.
5
T → (. E)
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
6
E → T + . E
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
7
T → i * .T
T → .(E)
T → .i * T
T → .i
8
T → (E.)
9
E → T + E.
10
T → i * T.
11
T → (E).
E
T
i
(
+
*
)

返 回

6.3 SLR(1)分析

- LR(0)分析中存在的问题：大多数适用的程序设计语言的文法不能满足LR(0)文法的条件。

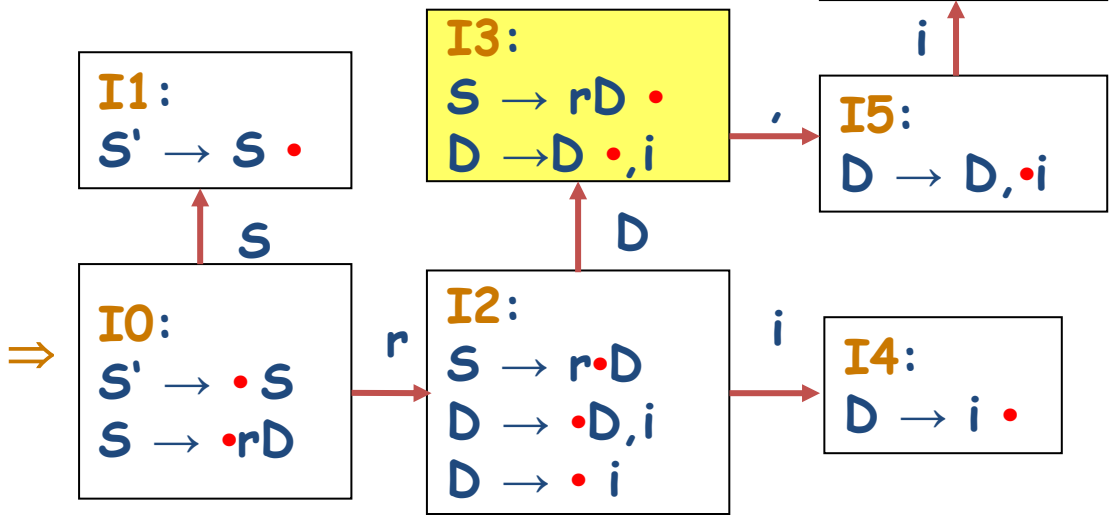
  即：规范族中存在有冲突的项目集（移进-归约冲突，归约-归约冲突）

- 如果解决这种冲突？

  直觉：对于有冲突的状态，向前查看一个符号，以确定采用的动作。

例 文法G：

(0) S' → S
(1) S → rD
(2) D → D,i
(3) D → i



LR(0)分析表：

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3	r_1	r_1, S_5	r_1	r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

I_3:
$S \rightarrow rD \bullet$
$D \rightarrow D \bullet, i$

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3	r_1	r_1, S_5	r_1	r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

解决冲突：

向前查看一个符号，看其是否是S的后跟符号（FOLLOW(S)）

- **是，则归约**
- **否，则移进**

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3		S_5		r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

解决 LR(0)项目集中冲突的方法：

一个LR(0)规范族中含有如下的项目集（状态）I

I = {X →α•bβ , A →γ• , B →δ• }

若有：FOLLOW(A) ∩ FOLLOW(B) = Ø

FOLLOW(A) ∩ {b} = Ø

FOLLOW(B) ∩ {b} = Ø

状态I 面临某输入符号 a

1) 若a=b，则移进

2) 若a∈FOLLOW(A), 则用产生式 A→γ 进行归约

3) 若a∈FOLLOW(B), 则用产生式 B→δ 进行归约

4) 此外，空白报错

SLR()文法

若一个文法的LR(0)分析表中所含有的动作冲突都能用上述方法解决，则称这个文法是SLR(1)文法

“改进的”SLR(1)分析

对所有非终结符都求出其FOLLOW集合，只有归约项目仅对面临输入符号包含在该归约项目左部非终结符的FOLLOW集合中，才采取用该产生式归约的动作。

分析表的构造步骤

a) 若项目$A\rightarrow\alpha\cdot a\beta$ 属于I_k，且转换函数$GO(I_k,a)=I_j$，当a为终结符时，则置ACTION[k,a]为S_j

b) 若$GO(I_k,A)=I_j$，则置GOTO[k,A]=j，其中A为非终结符，j为某一状态号

c) 项目$A\rightarrow\alpha\cdot$ 属于I_k，则对a为任何终结符或‘#’，且满足$a\in FOLLOW(A)$时，置$ACTION[k,a]=r_j$，j为产生式在文法中的编号

d) 若项目$S'\rightarrow S\cdot$ 属于I_k，则置ACTION[k,#] = acc

e) 其它填上“报错标志”

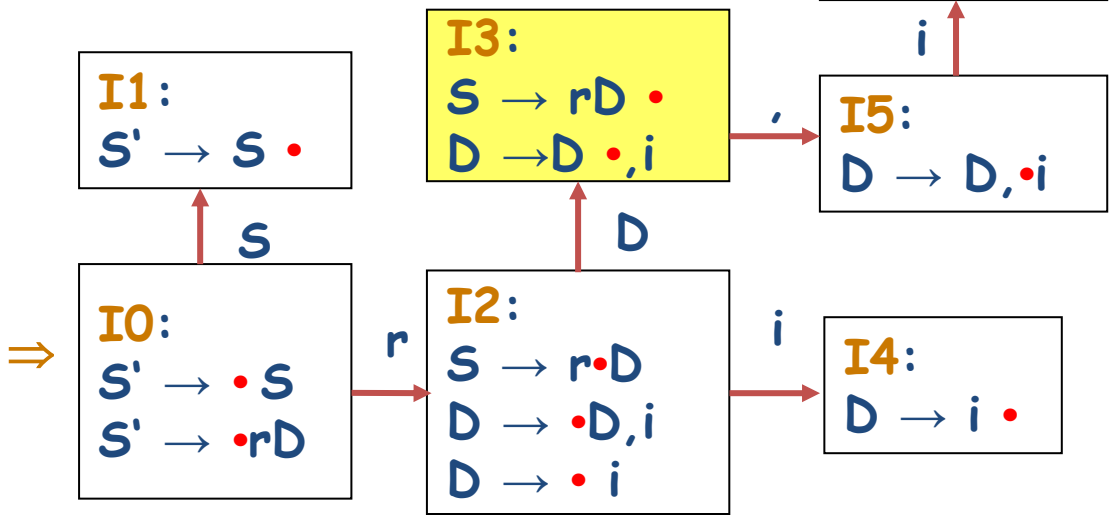
例 文法G：

(0) S' → S

(1) S → rD

(2) D → D,i

(3) D → i



Follow(S)={ # }

Follow(D)={ , # }

对于I3:

Follow(S)∩{,}=Ø

故能使用SLR(1)分析

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3		S_5		r_1		
4		r_3		r_3		
5			S_6			
6		r_2		r_2		

另例 文法G[E]

(1) E → T + E

(2) E → T

(3) T → i * T

(4) T → i

(5) T → (E)

步骤一：拓广文法

(0) S’ → E

(1) E → T + E

(2) E → T

(3) T → i * T

(4) T → i

(5) T → (E)

步骤二：构造 SLR(1)项目集规范族(下页)

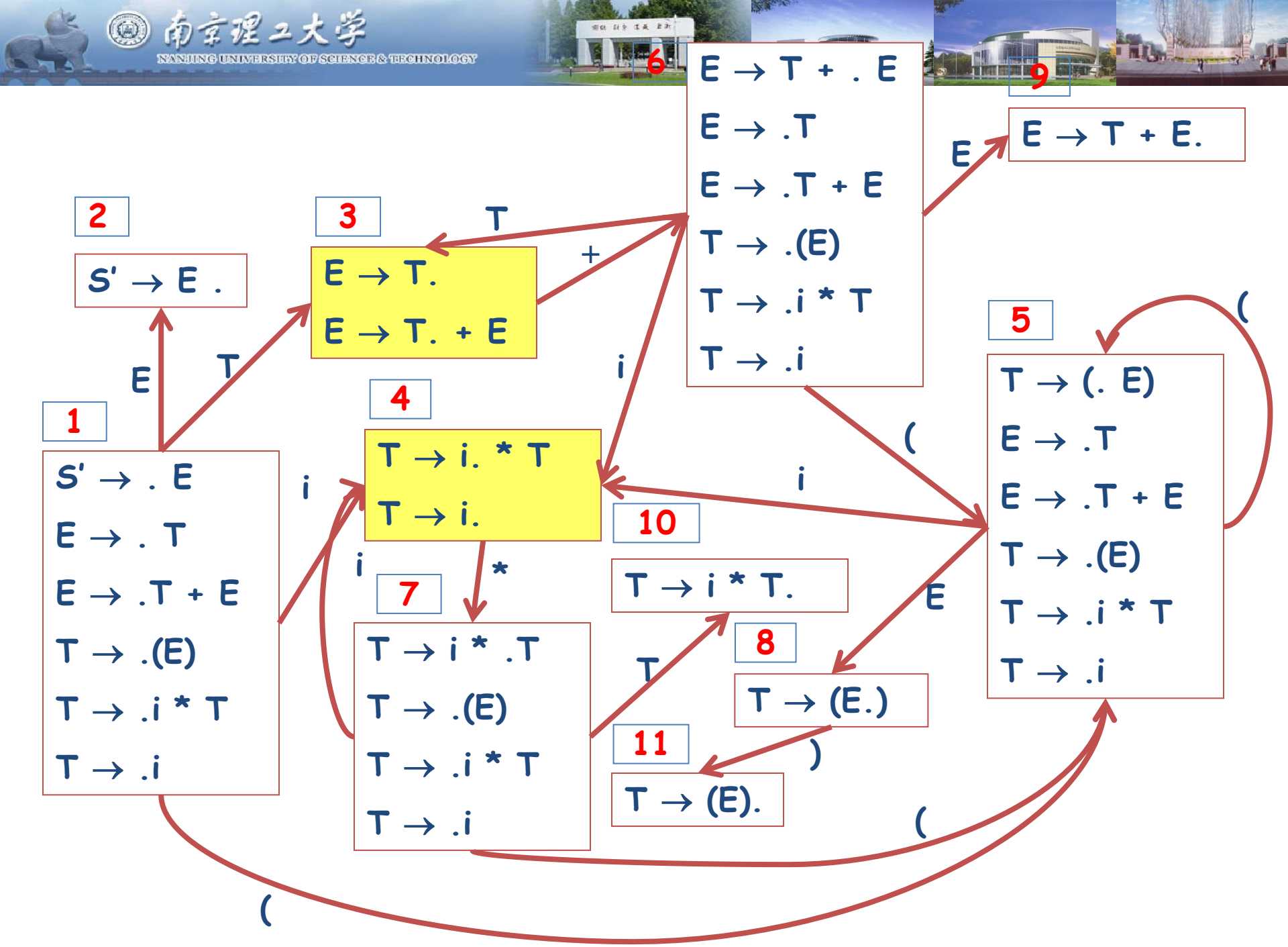
步骤三：求各VN的 FOLLOW集

步骤四：判别是否为 SLR(1)文法

步骤五：构造 DFA

步骤六：构造 LR(0)分析表

步骤七：分析句子


1
S' → . E
E → . T
E → .T + E
T → .(E)
T → .i * T
T → .i
2
S' → E .
3
E → T.
E → T. + E
4
T → i. * T
T → i.
5
T → (. E)
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
6
E → T + . E
E → .T
E → .T + E
T → .(E)
T → .i * T
T → .i
7
T → i * .T
T → .(E)
T → .i * T
T → .i
8
T → (E.)
9
E → T + E.
10
T → i * T.
11
T → (E).
E
T
i
+
*
(
)

仍有许多文法构造的LR(0)项目集规范族存在的冲突不能用SLR(1)方法解决!

引入：LR(1)分析法

6.4 LR(1)分析

主要思想：

- 若项目集 A→α·Bβ 属于I时，则 B→·γ 也属于I
- 把 FIRST(β) 作为用产生式归约的搜索符（称为向前搜索符），作为用产生式 B→γ 归约时查看的符号集合（用以代替SLR(1)分析中的FOLLOW集），并把此搜索符号的集合也放在相应项目的后面，这种处理方法即为LR(1)方法

LR(1)项目集规范族的构造

初始项目 S'→·S ,# ，求闭包后再用转换函数逐步求出整个文法的 LR(1)项目集族。

1）构造 LR(1)项目集的闭包函数

a) I的项目都在CLOSURE(I)中

b) 若 A→α·Bβ ,a 属于CLOSURE(I)，

则 B→· γ ,b 也属于CLOSURE(I), 其中 b∈FIRST(βa)

c) 重复b)直到CLOSURE(I)不再扩大

2)转换函数的构造

GOTO (I , X) = CLOSURE (J)

其中：I 为 LR(1)的项目集，X为一文法符号，

J={任何形如 A→αX·β ,a 的项目，其中 A→α·Xβ,a ∈I}

文法G'：

(0) S' → S
(1) S → aAd
(2) S → bAc
(3) S → aec
(4) S → bed
(5) A → e

I_0
S' → • S, #
S → • aAd, #
S → • bAc, #
S → • aec, #
S → • bed, #

I_1 [I_0 --S→ I_1]
S' → S • , #

I_2: [I_0 --a→ I_2]
S → a • Ad, #
S → a • ec, #
A → • e, d

I_3 [I_0 --b→ I_3]
S → b • Ac, #
S → b • ed, #
A → • e, c

I_4 [I_2--A→ I_4]
S → aA • d, #

I_5 [I_2--e→ I_5]
S → ae • c, #
A → e •, d

I_6 [I_3 --A→ I_6]
S → bA • c, #

I_7 [I_3 --e→ I_7]
S → be •d, #
A → e •, c

I_8 [I_4 --d→ I_8]
S → aAd •, #

I_9 [I_5 --c→ I_9]
S → aec •, #

I_{10} [I_6--c→I_{10}]
S → bAc •, #

I_{11} [I_7--d→I_{11}]
S → bed •, #

I_0
$S' \rightarrow \bullet S, \#$
$S \rightarrow \bullet aAd, \#$
$S \rightarrow \bullet bAc, \#$
$S \rightarrow \bullet aec, \#$
$S \rightarrow \bullet bed, \#$

S

I_1
$S' \rightarrow S \bullet, \#$

a

I_2
$S \rightarrow a \bullet Ad, \#$
$S \rightarrow a \bullet ec, \#$
$A \rightarrow \bullet e, d$

A

I_4
$S \rightarrow aA \bullet d, \#$

d

I_8
$S \rightarrow aAd \bullet, \#$

e

I_5
$S \rightarrow ae \bullet c, \#$
$A \rightarrow e \bullet, d$

c

I_9
$S \rightarrow aec \bullet, \#$

b

I_3
$S \rightarrow b \bullet Ac, \#$
$S \rightarrow b \bullet ed, \#$
$A \rightarrow \bullet e, c$

A

I_6
$S \rightarrow bA \bullet c, \#$

c

I_{10}
$S \rightarrow bAc \bullet, \#$

e

I_7
$S \rightarrow be \bullet d, \#$
$A \rightarrow e \bullet, c$

d

I_{11}
$S \rightarrow bed \bullet, \#$

LR(1)分析表的构造

1) 若项目 A→α·aβ,b 属于I_k，且 GO(I_k,a)= I_j ，当a为终结符时，则置 ACTION[k,a]为S_j
2) 若GO(I_k,A)=I_j，则置 GOTO[k,A]=j，
   其中A为非终结符，j为某一状态号
3) 若项目 A→α·,a 属于I_k，则对a为任何终结符或‘#’，置ACTION[k,a] = r_j，j为产生式编号
4) 若项目 S'→S·,# 属于I_k，则置ACTION[k,#]=acc
5) 其它空白，表示“报错”

LR(1)分析表

状态	ACTION						GOTO	
	a	b	c	d	e	#	S	A
0	S2	S3					1	
1						acc		
2					S5			4
3					S7			6
4				S8				
5			S9	r5				
6			S10					
7			r5	S11				
8						r1		
9						r3		
10						r2		
11						r4		

6.5 LALR(1)分析

- LR(1)分析法的本质：

  对某些存在冲突的项目集分裂，避免发生冲突。

- LR(1)分析法存在问题：

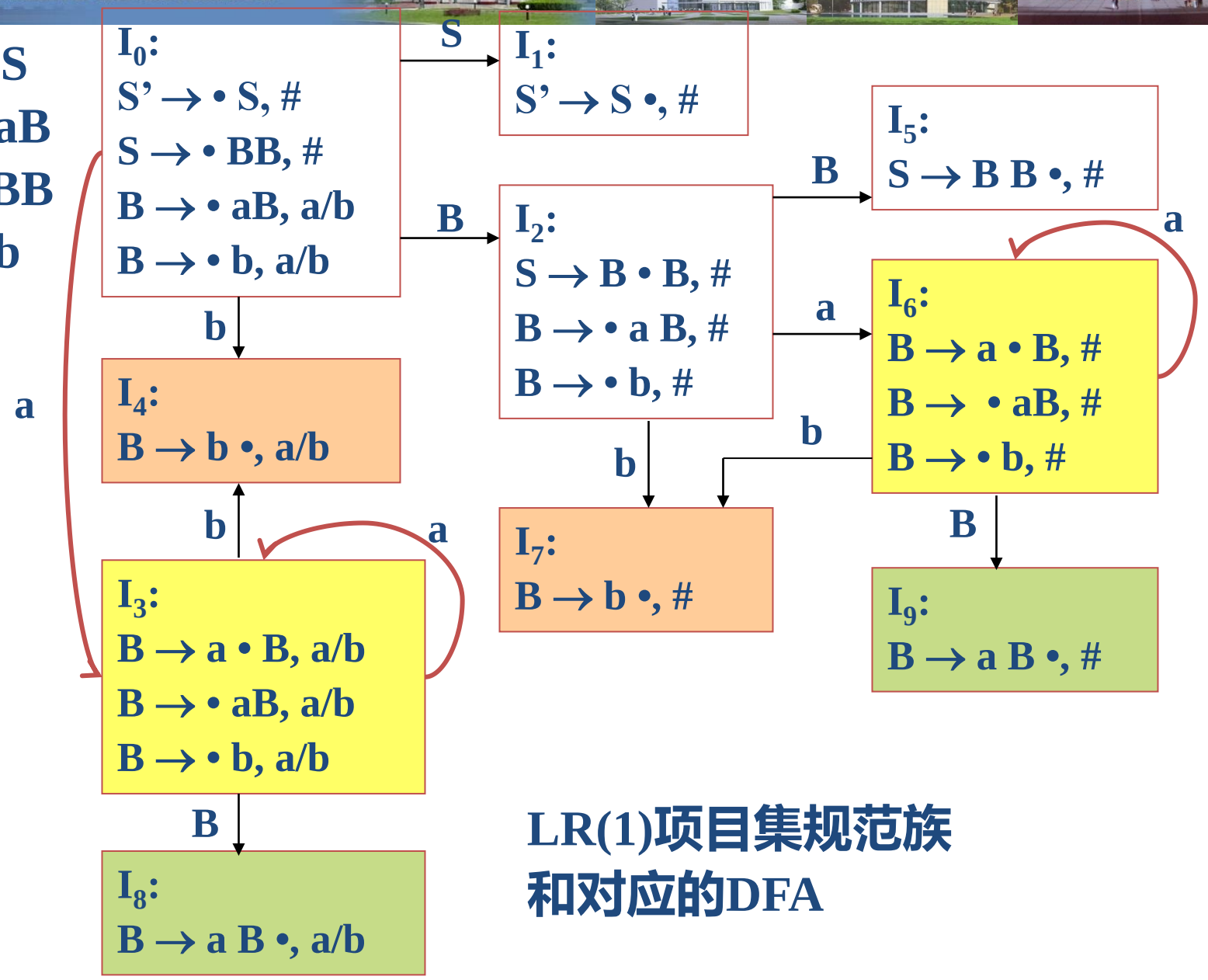
  LR(1)项目集的构造对某些项目集的分裂可能使状态数目剧烈的增长。

- 问题的解决：

  采用LALR(1)分析法合并同心集。

文法G：

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



LR(1)项目集规范族
和对应的DFA

合并同心集

I_3:
$B \rightarrow a \bullet B$, a/b
$B \rightarrow \bullet aB$, a/b
$B \rightarrow \bullet b$, a/b

I_6:
$B \rightarrow a \bullet B$, #
$B \rightarrow \bullet aB$, #
$B \rightarrow \bullet b$, #

合并为

$I_{3,6}$:
$B \rightarrow a \bullet B$, a/b/#
$B \rightarrow \bullet aB$, a/b/#
$B \rightarrow \bullet b$, a/b/#

I_4:
$B \rightarrow b \bullet$, a/b

I_7:
$B \rightarrow b \bullet$, #

合并为

$I_{4,7}$:
$B \rightarrow b \bullet$, a/b/#

I_8:
$B \rightarrow a\ B \bullet$, a/b

I_9:
$B \rightarrow a\ B \bullet$, #

合并为

$I_{8,9}$:
$B \rightarrow a\ B \bullet$, a/b/#

合并同心集的几点说明

- 同心集合并后心仍相同，只是超前搜索符集合为各同心集超前搜索符的和集
- 合并同心集后转换函数自动合并
- **LR(1)**文法合并同心集后也只可能出现归约-归约冲突，而没有移进-归约冲突

文法G'：

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

LALR(1)项目集规范族和对应的DFA

状态	ACTION			GOTO	
	a	b	#	S	B
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

合并同心集后

状态	ACTION			GOTO	
	a	b	#	S	B
0	$S_{3,6}$	$S_{4,7}$		1	2
1			acc		
2	$S_{3,6}$	S_7			5
3,6	$S_{3,6}$	S_4			8,9
4,7	r_3	r_3	r_3		
5			r_1		
8,9	r_2	r_2	r_2		

6.6 二义性文法在LR分析中的应用

- 对于某些二义文法，可以人为地给出优先性和结合性的规定，从而可以构造出比相应非二义性文法更优越的LR分析器

总结

- **LR(0)**
- **SLR(1):** 生成的 **LR(0)**项目集如有冲突，则根据非终结符的**FOLLOW**集决定移进或归约
- **LR(1)、LR(k):** 项目集由心与向前搜索符组成，搜索符长度为**1**或**k**
- **LALR(1):** 对**LR(1)**项目集规范族合并同心集

谢谢各位同学！