编译原理与技术

--自底向上的语法分析III

陈俊洁

天津大学智算学部

Outline

- 自底向上的语法分析基本问题
  - 移动－归约分析法
  - 用栈实现移动归约分析
- 算符优先分析法
  - 算符优先分析法定义、优先分析表的确定、优先函数的定义
  - 使用算符优先关系进行分析
  - 算符优先分析中的错误恢复
- LR分析法
  - LR(0)
  - SLR
  - LR(1)
- 语法分析器的自动产生工具Yacc

LR语法分析器概述

- LR(k)语法分析器适用于一大类上下文无关文法的语法分析。K省略时，默认k是1。
  - L指的是从左向右扫描输入字符串
  - R指的是构造最右推导的逆过程
  - k指的是在决定语法分析动作时需要向前看的符号个数
- 构造LR分析表的三种方法
  - 简单LR方法（SLR），最容易实现，功能最弱
  - 规范LR方法，功能最强，代价最高
  - 向前看的LR方法（LALR），其功能和代价介于前两者之间

LR语法分析器的优缺点

- LR分析的优点有以下几点：
  - LR语法分析器能识别几乎所有能用上下文无关文法描述的程序设计语言的结构。
  - LR分析法是已知的最一般的无回溯移动归约语法分析法，而且可以和其他移动归约分析一样被有效地实现。
  - LR分析法分析的文法类是预测分析法能分析的文法类的真超集。
  - 在自左向右扫描输入符号串时，LR语法分析器能及时发现语法错误。
- 这种分析方法的主要缺点是，对典型的程序设计语言文法，手工构造LR语法分析器的工作量太大，需要专门的工具。

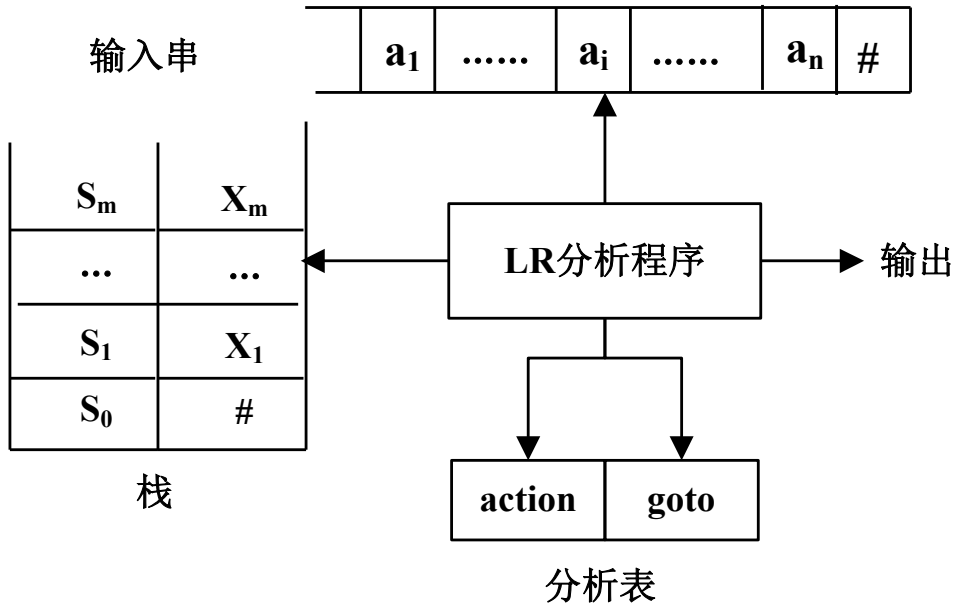
LR语法分析算法

- LR分析的基本思想是，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，即记住“历史”，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，即对未来进行“展望”。

- 当一串貌似句柄的符号串呈现于分析栈的顶端时，我们希望能够根据所记载的“历史”和“展望”以及“现实”的输入符号等三方面的材料，来确定栈顶的符号串是否构成相对某一产生式的句柄。

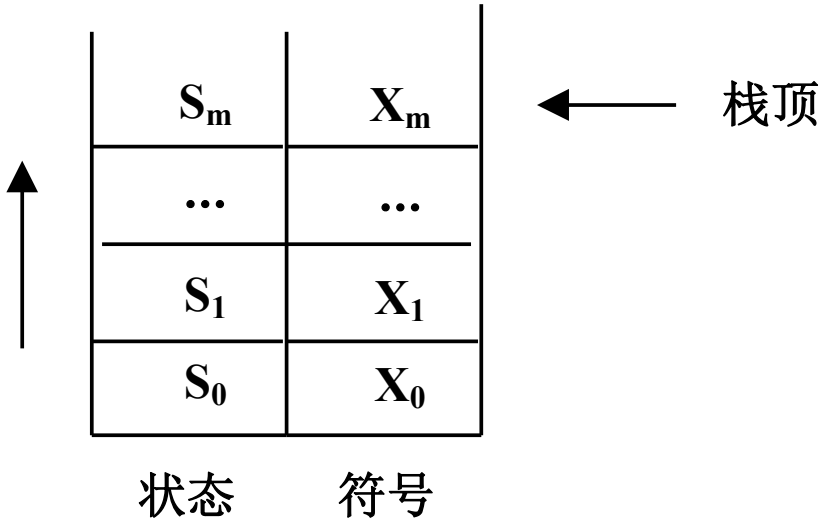
LR语法分析器的结构

- 一个LR分析器实质上是一个带先进后出存储器（栈）的确定有限状态自动机(DFA)。



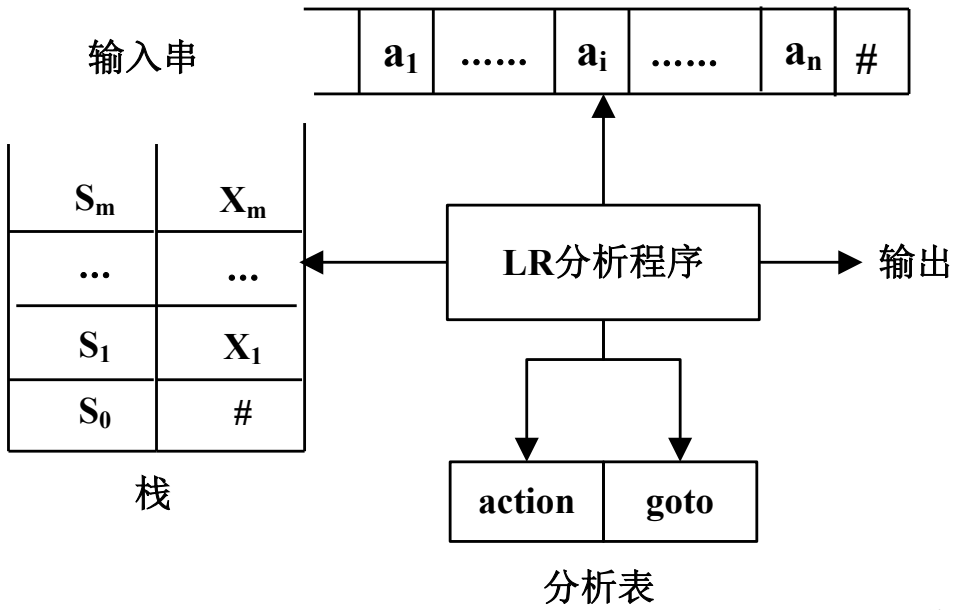
LR语法分析器的结构——栈

- 我们将把“历史”和“展望”材料综合的抽象成某些“状态”，存放在栈里。
- 栈里的每个状态概括了从分析开始直到某一归约阶段的全部“历史”和“展望”资料。



LR语法分析器的结构——分析表

- ACTION[s ,a]规定了当前状态s面临输入符号a时应采取什么动作。
- GOTO[s, X]规定了状态s面对文法符号X（终结符或非终结符）时下一个状态是什么。



动作表： **action[S_m,a_i]**表示下一个输入符号为**a_i**, 栈顶状态为**S_m**时,分析算法应执行的动作;若**action[s_m,a_i]=s_i**,表示移进, 然后栈顶状态为**s_i**; 若**action[s_m,a_i]= r_j**表示使用产生式**(j)**归约; 若**action[s_m,a_i]= acc**表示接受输入串 ; 若**action[s_m,a_i]= err**表示语法错误.

状态转换表： **GOTO[S_i,X]**表示归约出非终 结符号**X**之后,当前栈顶状态为**Si**时,分析栈 应转换到的下一个状态, 即栈顶的新状态.

LR语法分析器的结构——ACTION表

- 每一项ACTION[s,a]所规定的动作是下述四种可能之一。
  - 移进：把(s,a)的下一个状态s’ =GOTO[s,a]和输入符号a推进栈，下一个输入符号变成现行输入符号
  - 归约：指用某一个产生式A→β进行归约。假若β的长度为r，归约的动作是去除栈顶的r个项，使状态S_{m-r}变成栈顶状态，然后把（S_{m-r}, A）的下一个状态s’ =GOTO[S_{m-r}, A]和文法符号A推进栈。
  - 接受：宣布分析成功，停止分析器的工作。
  - 报错：发现源程序含有错误，调用出错处理程序。

注意：若a为终结符, 则GOTO[S,a]的值已列在action[S,a]的sj之中(状态j). 因此 GOTO表仅对所有非终结符A列出GOTO[S, A]的值

LR分析器的工作原理

一个*LR*分析器的工作过程就是一步一步的变换三元式直至到“接受”或“报错”为止.

分析栈中的串和等待输入的符号串构成如下形式的三元组:

$(S_0S_1S_2\ldots S_m,\ \#X_1X_2\ldots X_m,\ a_ia_{i+1}\ \ldots a_n\#)$

1. 其初态为: $(S_0,\ \#,\ a_1a_2\ \ldots a_ia_{i+1}\ldots a_n\#)$
2. 假定当前分析栈的栈顶为状态S_m,下一个输入符号为a_i,分析器的下一个动作
   - 如果**action**$[S_m, a_i]$=**移进**且**S=GOTO**$[S_m,a_i]$,则分析器执行移进, 三元组变成 $(S_0S_1S_2\ldots S_mS, \#X_1X_2\ldots X_ma_i,\ a_{i+1}\ldots a_n\#)$即分析器将输入符号$a_i$和状态S移进栈, a_{i+1}变成下一个符号
   - 如果**action**$[S_m, a_i]$=**归约**$A\rightarrow\beta$,则分析器执行归约, 三元组变成 $(S_0S_1S_2\ldots S_{m-r}S, \#X_1X_2\ldots X_{m-r}A, a_ia_{i+1}\ \ldots a_n\#)$, 此处S= **GOTO**$[S_{m-r}, A]$, r为$\beta$的长度且$\beta=X_{m-r+1}X_{m-r+2}\ldots X_m$.
   - 若**action**$[S_m,a_i]$= **acc**,则接收输入符号串,语法分析完成.
   - 若**action**$[S_m,a_i]$= **err**,则发现语法错误,调用错误恢复子程序进行处理.

LR语法分析器分析过程举例

文法G：

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

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			

*i*i+i的分析过程*

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#
6	0275	#T*i	+i#
7	02710	#T*F	+i#
8	02	#T	+i#
9	01	#E	+i#
10	016	#E+	i#
11	0165	#E+i	#
12	0163	#E+F	#
13	0169	#E+T	#
14	01	#E	#

s_j: 把下一状态j和现行输入符号a移进栈
r_j: 按第j个产生式进行规约

acc: 接受
空白格：出错标志，报错

对上述例子的分析

- LR语法分析器不需要扫描整个栈就可以知道什么时候句柄出现在栈顶。栈顶的状态符号包含了所需要的一切信息。
- 请注意一个非常重要的事实：如果仅由栈的内容和现实的输入符号就可以识别一个句柄，那么就可以用一个有限自动机自底向上扫描栈的内容和检查现行输入符号来确定呈现于栈顶的句柄是什么（当栈顶呈现一个句柄时）。
- 实际上，LR分析表的GOTO函数就是这样一个有限自动机，只是它不需要每次移动都读栈。

LR文法

- 一个文法，如果能为其构造一张分析表,使得它的每个入口均是唯一确定的，则这个文法为LR文法.
- 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。
  - 我们关注k<=1的情形

LL文法要求每个非终结符的所有候选首字符均不相同，因为预测程序认为，一旦看到首符之后就看准了该用哪一个产生式进行推导。

LR分析程序只有在看到整个右部所推导的东西之后才认为是看准了归约方向。因此，LR方法比LL方法更加一般化。

LR(0)项目集族和LR(0)分析表的构造

- 前缀：字的前缀是指该字的任意首部。
  - 字abc的前缀有ε、a、ab、abc。
- 活前缀：指规范句型的一个前缀，这种前缀不含句柄之后任何符号。之所以称为活前缀，是因为在右边添加一些符号之后，就可以使它称为一个规范句型。
- 在LR分析工作过程中的任何时候，栈里的文法符号（自栈底而上）$X_0 X_1 \cdots X_m$应该构成活前缀，把输入串的剩余部分配上之后即应成为规范句型。
  - 只要输入串的已扫描部分保持可归约成一个活前缀，那就意味着所扫描过的部分没有错误。

回顾例子

考虑文法：

(1) S→aAcBe

(2) A→b

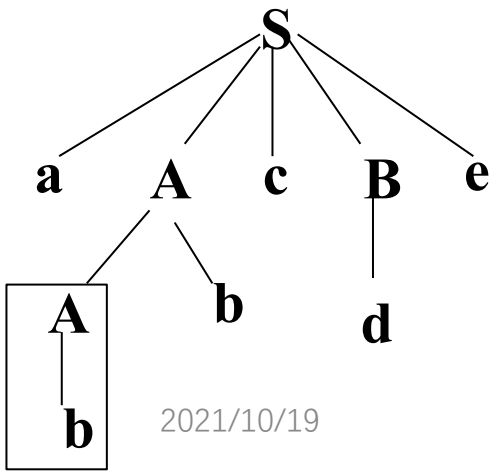
(3) A→Ab

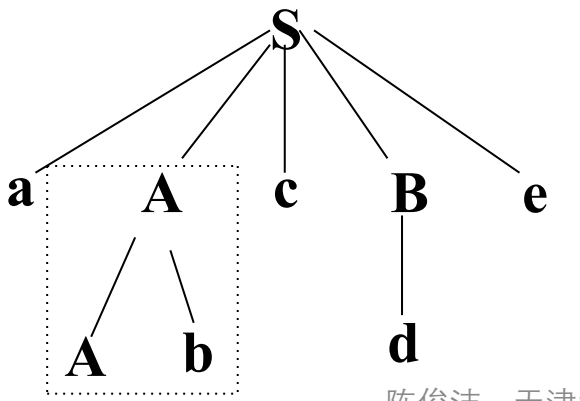
(4) B→d

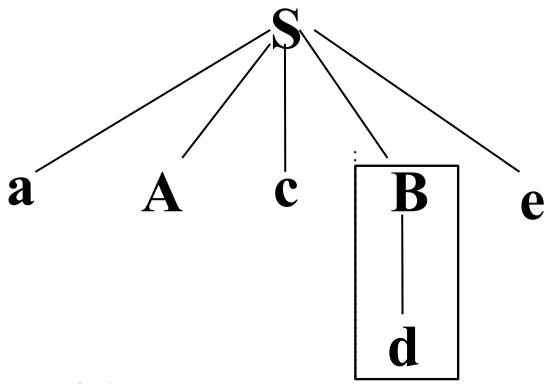
将输入串**abbcde**归约到**S.**

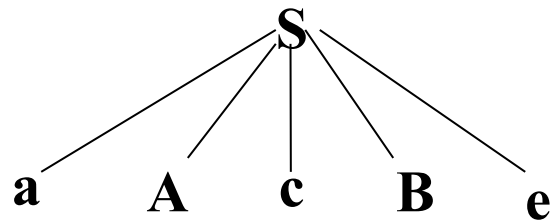
对该文法的句子**abbcde**逐步寻找句柄, 并用相应产生式的左部符号去替换, 得到如下归约过程: (画底线的部分是句柄).

句型	归约过程		
a<u>b</u>bcde	**(2)**	**A**	**b**
a<u>Ab</u>cde	**(3)**	**A**	**Ab**
aAc<u>d</u>e	**(4)**	**B**	**d**
<u>aAcBe</u>	**(1)**	**S**	**aAcBe**
S			









LR(0)项目的定义

- 对于一个文法G，我们首先要构造一个NFA，它能识别G的所有活前缀。这个NFA的每一个状态是下面定义的一个“项目”。
- 文法G的每一个产生式的右部添加一个圆点称为G的一个LR(0)项目（简称项目）。
  例如产生式A→XYZ对应有四个项目：

  A→•XYZ

  A→X•YZ

  A→XY•Z

  A→XYZ•

  产生式A→ε只对应一个项目A→•。

LR(0)项目的说明

- 直观上说，一个项目指明了在分析过程的某时刻我们看到产生式多大部分。
  - 例如， A→•XYZ这个项目意味着，我们希望能从后面输入串中看到可以从XYZ推出的符号串。
  - A→X•YZ这个项目意味着，我们已经从输入串中看到能从X推出的符号串，我们希望能进一步看到可以从YZ推出的符号串。
- 将每个项目看成NFA的一个状态，我们就可以构造这样一个NFA，用来识别文法所有的活前缀。

LR(0)项目间的转换规则

- 如果状态i和j来自同一个产生式，而且状态j的圆点只落后于状态i的原点一个位置，
  - 如状态i为$X \rightarrow X_1 \cdots X_{i-1} \bullet X_i \cdots X_n$，
  - 状态j为$X \rightarrow X_1 \cdots X_i \bullet X_{i+1} \cdots X_n$，
  - 那么就从状态i画一条标志为X_i的弧到状态j。
- 假若状态i的圆点之后的那个符号为非终结符，
  - 如i为$X \rightarrow \alpha \bullet A\beta$，A为非终结符，
  - 就从状态i画ε弧到所有$A \rightarrow \bullet\gamma$状态。
- NFA的接受状态就是那些圆点出现在最右边的项目。

LR(0)项目分类

- 归约项目
  - 凡圆点在最右的项目，如A→α•称为一个“归约项目”
- 接受项目
  - 对文法的开始符号S’的归约项目，如S’→α•称为“接受”项目。
- 移进项目
  - 形如A→α•aβ的项目，其中a为终结符，称为“移进”项目。
- 待约项目
  - 形如A→α•Bβ的项目，其中B为非终结符，称为“待约”项目。

识别LR(0)活前缀的NFA例子I

文法G：

S’ → E
E → aA | bB
A → cA | d
B → cB | d

文法的项目有：

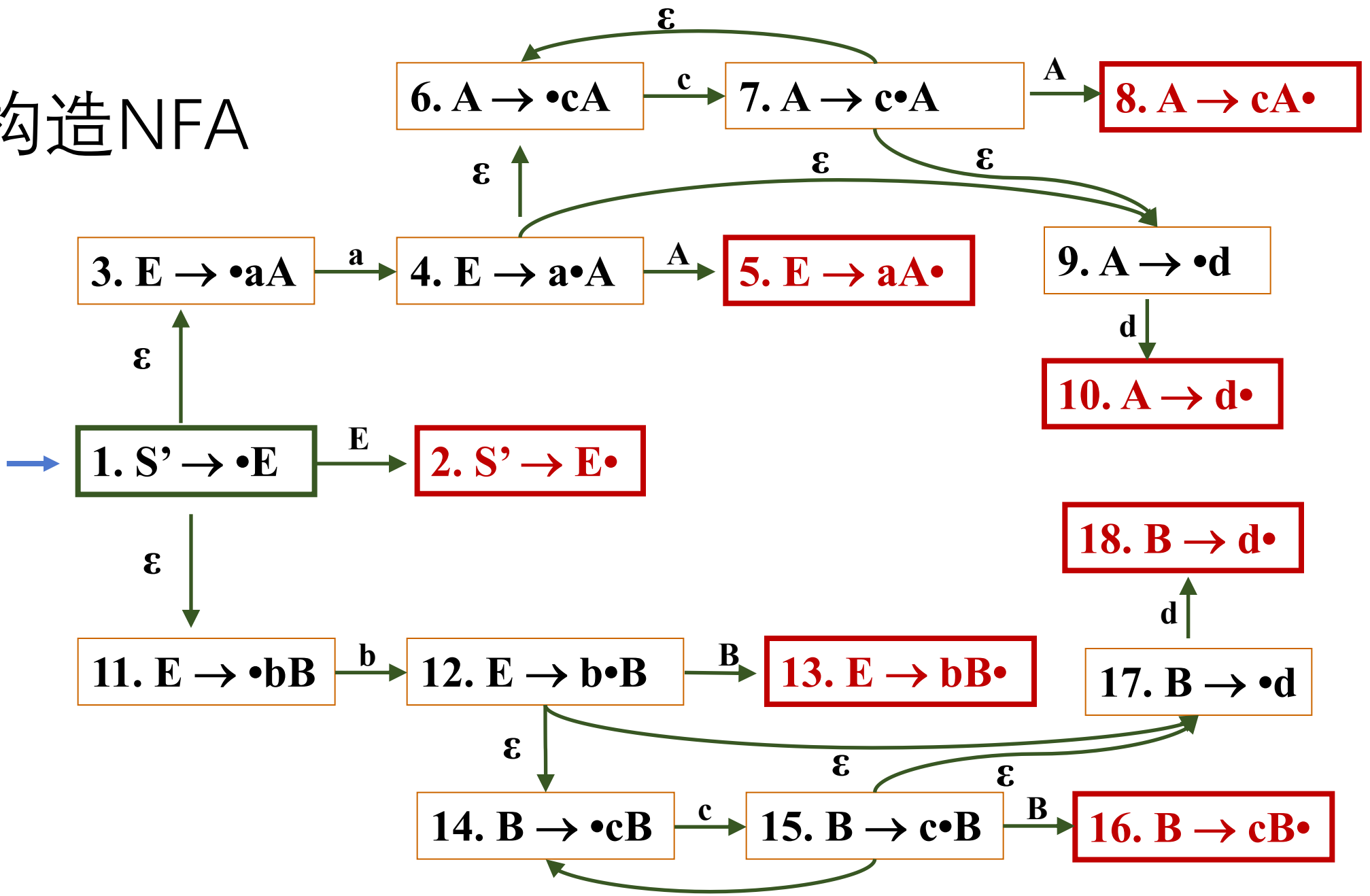
1. S’ → •E
2. S’ → E•
3. E → •aA
4. E → a•A
5. E → aA•
6. A → •cA
7. A → c•A
8. A → cA•
9. A → •d
10. A → d•
11. E → •bB
12. E → b•B
13. E → bB•
14. B → •cB
15. B → c•B
16. B → cB•
17. B → •d
18. B → d•

识别LR(0)活前缀的NFA例子II

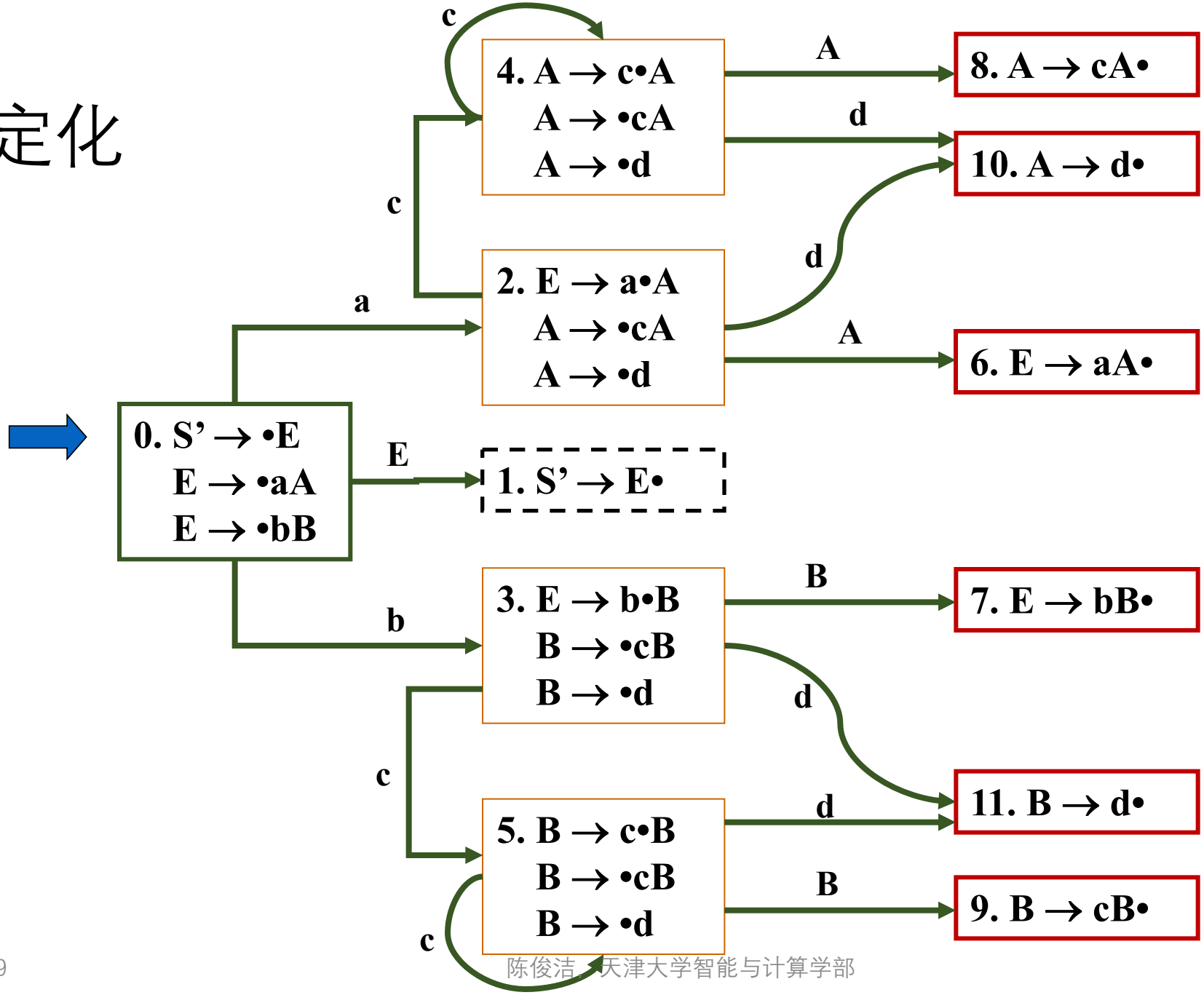
1. S’ → •E
2. S’ → E•
3. E → •aA
4. E → a•A
5. E → aA•
6. A → •cA
7. A → c•A
8. A → cA•
9. A → •d
10. A → d•
11. E → •bB
12. E → b•B
13. E → bB•
14. B → •cB
15. B → c•B
16. B → cB•
17. B → •d
18. B → d•

- 初态
  - S’ → •E [1]
- 句柄识别态
  - E → aA• [5]
  - A → cA• [8]
  - A → d• [10]
  - E → bB• [13]
  - B → cB• [16]
  - B → d• [18]
- 句子识别态
  - S’ → E• [2]

构造NFA


ε
6. A → •cA
c
7. A → c•A
A
8. A → cA•
ε
ε
ε
3. E → •aA
a
4. E → a•A
A
5. E → aA•
9. A → •d
d
10. A → d•
ε
1. S' → •E
E
2. S' → E•
ε
18. B → d•
d
11. E → •bB
b
12. E → b•B
B
13. E → bB•
17. B → •d
ε
ε
ε
14. B → •cB
c
15. B → c•B
B
16. B → cB•
ε

确定化


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

LR(0)项目集规范族的构造

- 用ε-CLOSURE(闭包)的办法来构造一个文法G的LR(0)项目集规范族。
- 准备工作：
  - 假定文法G是一个以S为开始符号的文法，我们构造一个文法G’，它包含整个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”状态。

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

LR(0)项目集规范族的构造

- 假定I是文法G’ 的任一项目集，定义和构造I的闭包CLOSURE(I)的办法是：
  - I的任何项目都属于CLOSURE(I)；
  - 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)；
  - 重复执行上述两步骤直至CLOSURE(I)不再增大为止。

- 函数GO(I, X)是一个状态转换函数。
  - 第一个变元I是一个项目集，
  - 第二个变元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→αX•β的项目 | A→α•Xβ属于I}

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

构造LR(0)项目集规范族例子

- 初始状态I_0的项目集规范族：
  I_0={ S’ → •E，E → •aA，E → •bB}

- I_1、I_2、I_3分别是GO(I_0,E)，GO(I_0,a)和GO(I_0,b)
  - I_1=CLOSURE(S’ →E•)={S’ →E•}
  - I_2=CLOSURE(E→a•A)={E→a•A, A→•cA, A→•d}
  - I_3=CLOSURE(E→b•B)={E→b•B, B→•cB, B→•d}

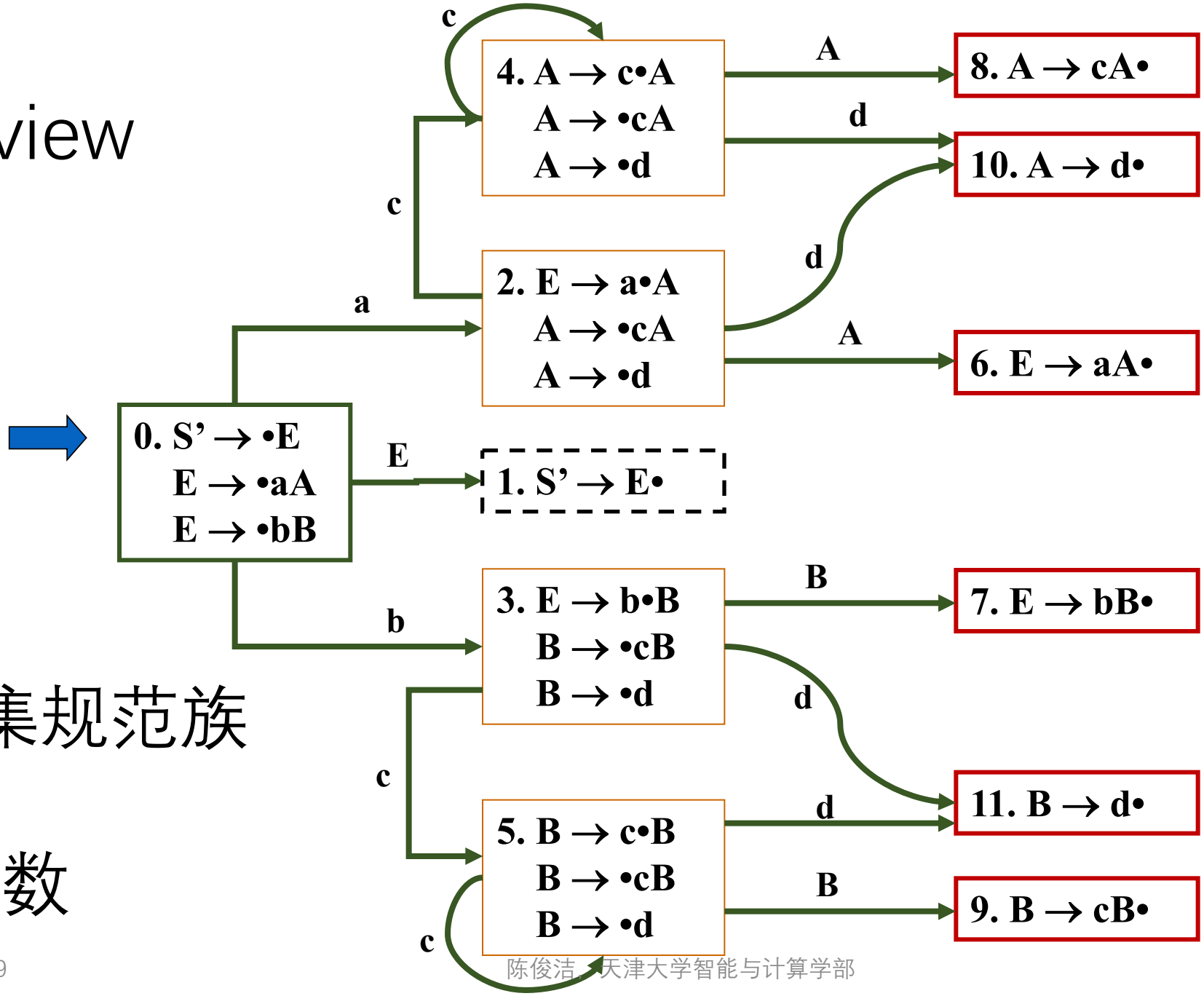
文法G：
(0) S’ → E
(1) E → aA
(2) E → bB
(3) A → cA
(4) A → d
(5) B → cB
(6) B → d

LR(0)项目集规范族的构造算法

- 通过函数CLOSURE和GO构造一个文法G的拓广的文法G ‘的LR(0)项目集规范族.

```
Procedure  itemsets(G’ ):
  begin
     C:={ closure( {S’ →·S})};
     repeat
        for C 中的每个项目集I和G’ 的每个符号X  do
           if  GO(I,X) 非空且不属于C then
                    GO(I,X)放入C族中
      until C不再增大
   end
```

Review



项目集规范族

+

GO函数

文法G：

(0) S’ → E

(1) E → aA

(2) E → bB

(3) A → cA

(4) A → d

(5) B → cB

(6) B → d

有效项目

- 有效项目
  - 我们说项目A→β_1•β_2对活前缀$\alpha\beta_1$是有效的，其条件是存在规范推导 $S' \overset{*}{\Rightarrow} \alpha A\omega \Rightarrow \alpha\beta_1\beta_2\omega$。
  - 事实上,活前缀γ有效项目集,是从上述的DFA的初态出发,经读出γ后而得到的那个项目集状态
  - 在任何时候,分析栈中的活前缀$X_1X_2\cdots X_m$的有效项目集正是栈顶状态S_m所代表的那个集合.
- 例子
  - ***DFA*** 的一个活前缀***bc,***使***DFA*** 到达 ***5***，***5***有三个项目 下面说明项目集对bc是有效的. 考虑下面 三个推导:
    - (1) S' =>E=>bB=>bcB
    - (2) S' =>E=>bcB =>bccB
    - (3) S' =>E =>bB =>bcB =>bcd

5. B → c•B
B → •cB
B → •d

推导（1） （2） （3） 分别证明了B → c•B B → •cB B → •d 的有效性

LR(0)项目集规范族的讨论

- LR(0)项目集可能出现的冲突
  - 同一项目可能对好几个活前缀都是有效的（当一个项目出现在好几个不同的集合中便是这种情况）。
  - 若归约项目$A\rightarrow\beta_1\bullet$对活前缀$\alpha\beta_1$是有效的，则它告诉我们应该把符号串$\beta_1$**归约**为A。即把活前缀$\alpha\beta_1$变成$\alpha A$。
  - 若移进项目$A\rightarrow\beta_1\bullet\beta_2$对活前缀$\alpha\beta_1$是有效的，则它告诉我们，句柄尚未形成，因此，下一步动作应该是**移进**。
  - 但是，可能存在这样的情形，对同一活前缀，存在若干项目对它都是有效的，而且告诉我们应该做的事情各不相同，互相冲突。这种冲突通过向前多看几个输入符号，或许能够解决，但有些冲突却是无法解决的。

LR(0)文法

- 假如一个文法G的拓广文法G’ 的活前缀识别自动机的每个状态（项目集）不存在下述情况：
  - 既含移进项目又含归约项目；
  - 含多个归约项目。

  则称G是一个LR(0)文法。换言之，LR(0)文法规范族的每个项目集不包含任何冲突项目。

LR(0)分析表的构造

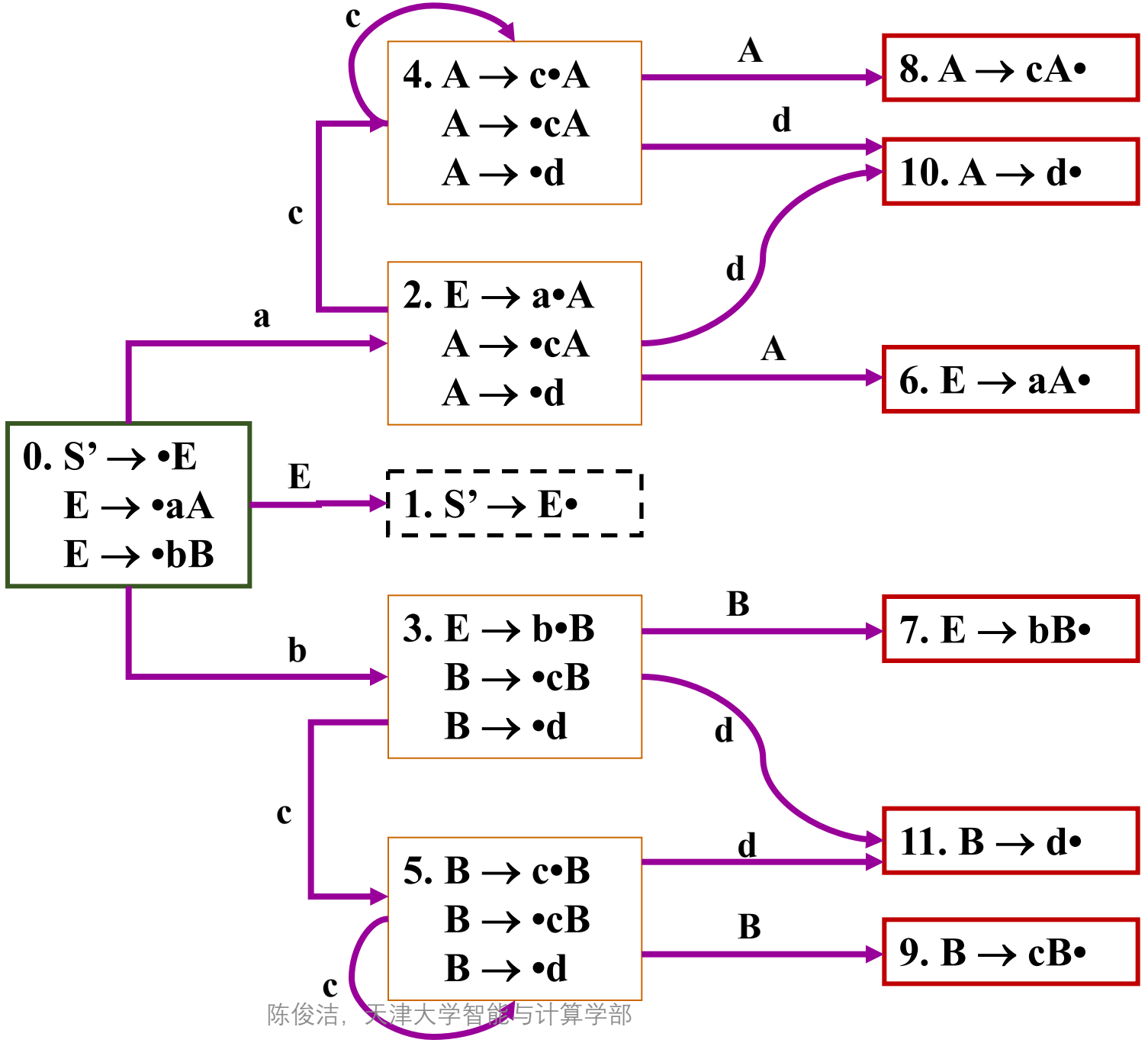
- 假定项目集规范族C={$I_0,I_1,\cdots,I_n$}。令每一个项目集I_k的下标k作为分析器的状态。分析表的ACTION子表和GOTO子表可按如下方法构造
  - (0) 令那个包含项目S’ →•S的集合I_k的下标k为分析器的初态。
  - (1) 若项目A→α•aβ属于I_k且GO(I_k , a)= I_j, a为终结符，置ACTION[k,a]为“把(j,a)移进栈”，简记为“s_j”。
  - (2) 若项目A→α•属于I_k，对**任何**终结符a(或结束符#)，置ACTION[k,a]为“用产生式A→α进行归约”，简记为“r_j”（假定产生式A→α是文法G’的第j个产生式）。
  - (3) 若项目S’ →S•属于I_k，则置ACTION[k,#]为“接受”，简记为“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填上“报错标志”。

按上述方法构造的分析表的每个入口都是唯一的. 称此分析表是一个***LR(0)***表. 使用***LR(0)***表的分析器叫做一个***LR(0)***分析器

Review

文法G：

(0) S’ → E
(1) E → aA
(2) E → bB
(3) A → cA
(4) A → d
(5) B → cB
(6) B → d



LR(0)分析表

状态	action					goto		
	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*				*9*
6	*r1*	*r1*	*r1*	*r1*	*r1*			
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*			

构造LR(0)分析表练习

- 设右如下文法：G={V_T, V_N, S’ , P}, 其中：
  V_T={var ： ， id real} （空格是分隔符，逗号是终结符）

  V_N={S’ S I T} （空格是分隔符）

P如下：

(1) S’ →S
(2) S→var I:T
(3) I→I,id
(4) I→id
(5) T→real

构造该文法的LR(0)分析表

(1) 列出G的所有LR(0)项目

S’ →. S　　S’ →S.

S→. var I:T　S→var. I:T　S→var I. :T　S→var I: . T　S→varI: T.

I→. I,id　　I→I. ,id　　I→I, . id　　I→I, id.

I→. id　　I→id.

T→. real　　T→real.

(2) 构造G4的项目集规范族及识别活前缀的DFA

(3) 构造G的LR(0)分析表

	Action						GOTO		
	var	:	,	id	real	#	S	I	T
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									

(2) 构造G的项目集规范族

I0={ S' →. S , S→. var I:T }
I1={ S' →S. }
I2={ S→var. I:T, I→. I,id , I→. id }
I3={ S→var I. :T, I→I. ,id }
I4={ I→id. }
I5={ S→var I: . T, T→. real }
I6={ I→I, . id }
I7={ S→var I: T. }
I8={ T→real. }
I9={ I→I, id. }

GO(I0, S)=I1
GO(I0, var)=I2
GO(I2, I)=I3
GO(I2, id)=I4
GO(I3, :)=I5
GO(I3, ,)=I6
GO(I5, T)=I7
GO(I5, real)=I8
GO(I6, id)=I9

(1) S' →S
(2) S→var I:T
(3) I→I,id
(4) I→id
(5) T→real

(1) 列出G的所有LR(0)项目

S’ →. S　　S’ →S.

S→. var I:T　S→var. I:T　S→var I. :T　S→var I: . T　S→varI: T.

I→. I,id　　I→I. ,id　　I→I, . id　　I→I, id.

I→. id　　I→id.

T→. real　　T→real.

(2) 构造G的项目集规范族

I0={ S’ →. S , S→. var I:T }　　GO(I0, S)=I1

I1={ S’ →S. }　　GO(I0, var)=I2

I2={ S→var. I:T, I→. I,id , I→. id }　　GO(I2, I)=I3

I3={ S→var I. :T, I→I. ,id }　　GO(I2, id)=I4

I4={ I→id. }　　GO(I3, :)=I5

I5={ S→var I: . T, T→. real }　　GO(I3, ,)=I6

I6={ I→I, . id }　　GO(I5, T)=I7

I7={ S→var I: T. }　　GO(I5, real)=I8

I8={ T→real. }　　GO(I6, id)=I9

I9={ I→I, id. }

(3) 构造G的LR(0)分析表

	Action						GOTO		
	var	:	,	id	real	#	S	I	T
0	S2						1		
1						Acc			
2				s4				3	
3		S5	S6						
4	R4	R4	R4	R4	R4	R4			
5					S8				7
6				S9					
7	R2	R2	R2	R2	R2	R2			
8	R5	R5	R5	R5	R5	R5			
9	R3	R3	R3	R3	R3	R3			

写出 var a, b : real 的分析过程

	Action						GOTO		
	var	:	,	id	real	#	S	I	T
0	S2						1		
1						Acc			
2				s4				3	
3		S5	S6						
4	R4	R4	R4	R4	R4	R4			
5					S8				7
6				S9					
7	R2	R2	R2	R2	R2	R2			
8	R5	R5	R5	R5	R5	R5			
9	R3	R3	R3	R3	R3	R3			

步骤	状态	符号	输入串
1	0	#	var a, b : real#
2	02	#var	a, b : real#