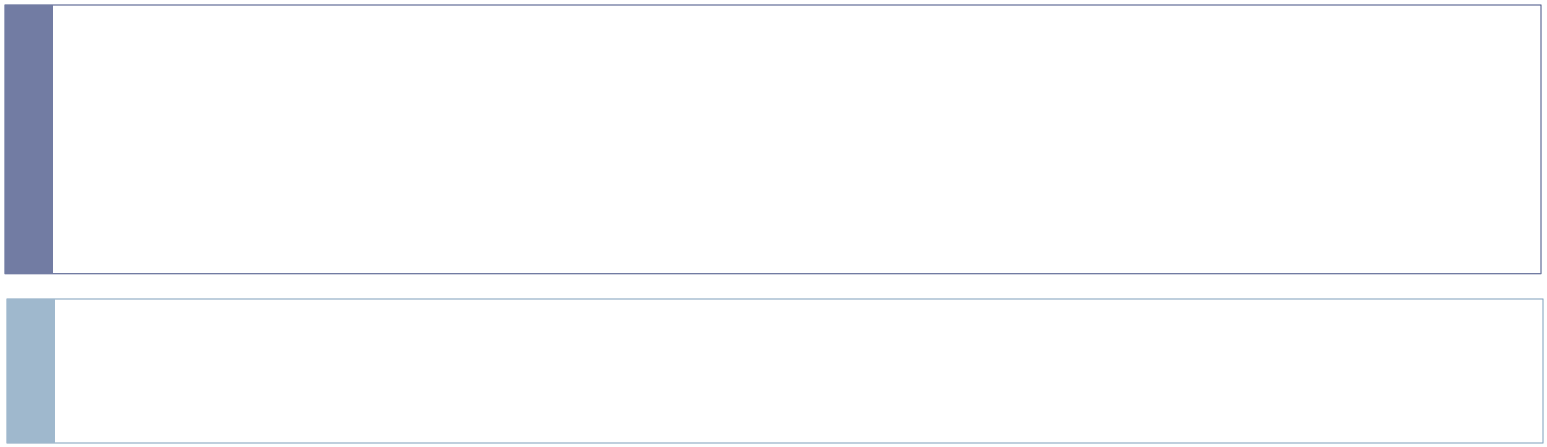
Chapter 5 语法分析-自下而上分析



Outlines

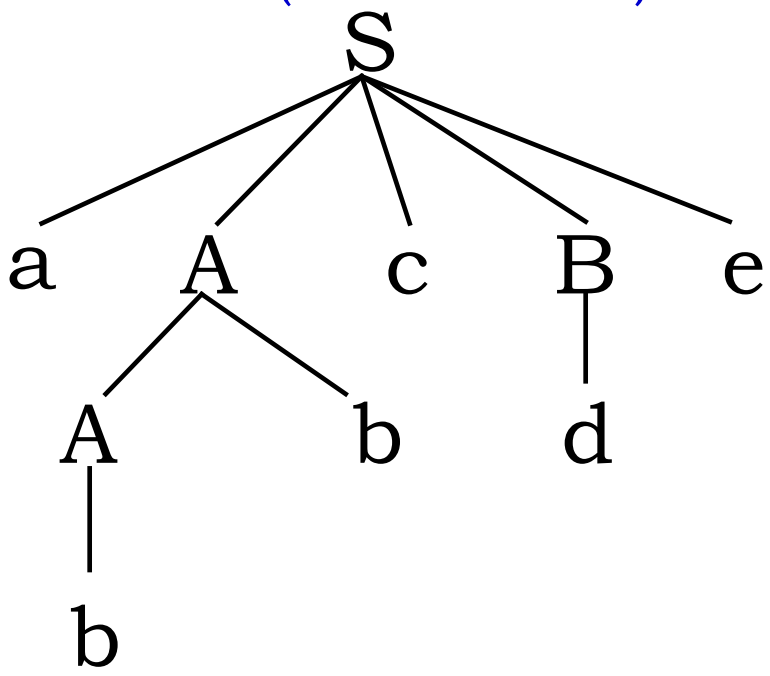
- 自下而上分析基本问题
  - 归约
  - 规范归约
  - 符号栈的使用
  - 语法树的表示
- 算符优先分析法
- LR分析法

Bottom-up

- 自下而上分析
  - 从输入字符的角度而言
    - 从输入开始
    - 逐步进行“归约”
    - 直至归约到文法的开始符号
  - 从语法树的角度而言
    - 从语法树的末端开始
    - 步步向上“归约”
    - 直到根结

例1 abbcde

aAbcde (A → b)
aAcde (A → Ab)
aAcBe (B → d)
S (S → aAcBe)



Shift-reduce

- 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⇒β,则称β是句型αβδ相对于规则A→β的**直接短语**。一个句型的最左直接短语称为该句型的**句柄**。

Shift-reduce

- 在**移进—归约分析模式**中，符号栈的使用有以下四种操作形式。
  - ① 移进(shift)：把当前输入中的下一个终结符移进栈；
  - ② 归约(reduce)：句柄在栈顶已形成，用适当产生式左部代替句柄；
  - ③ 接受(accept)：宣告分析成功；
  - ④ 报错(error)：发现语法错误，调用错误恢复例程。

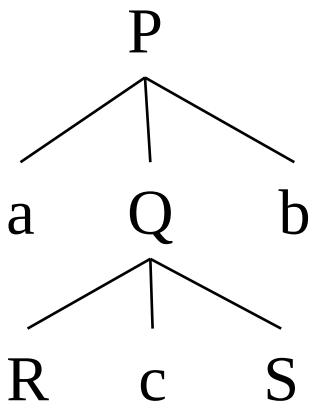
Operator precedence analysis

假定G不含ε-产生式，ab为终结符

$a \doteq b \Leftrightarrow P \to \cdots ab \cdots$ 或 $P \to \cdots aQb \cdots$

$a \lessdot b \Leftrightarrow P \to \cdots aR \cdots$ 且 $(R \overset{+}{\Rightarrow} b \cdots$ 或 $R \overset{+}{\Rightarrow} Qb \cdots)$

$a \gtrdot b \Leftrightarrow P \to \cdots Rb \cdots$ 且 $(R \overset{+}{\Rightarrow} \cdots a$ 或 $R \overset{+}{\Rightarrow} \cdots aQ)$



$a \doteq b$

$a \lessdot c$

$c \gtrdot b$

LR

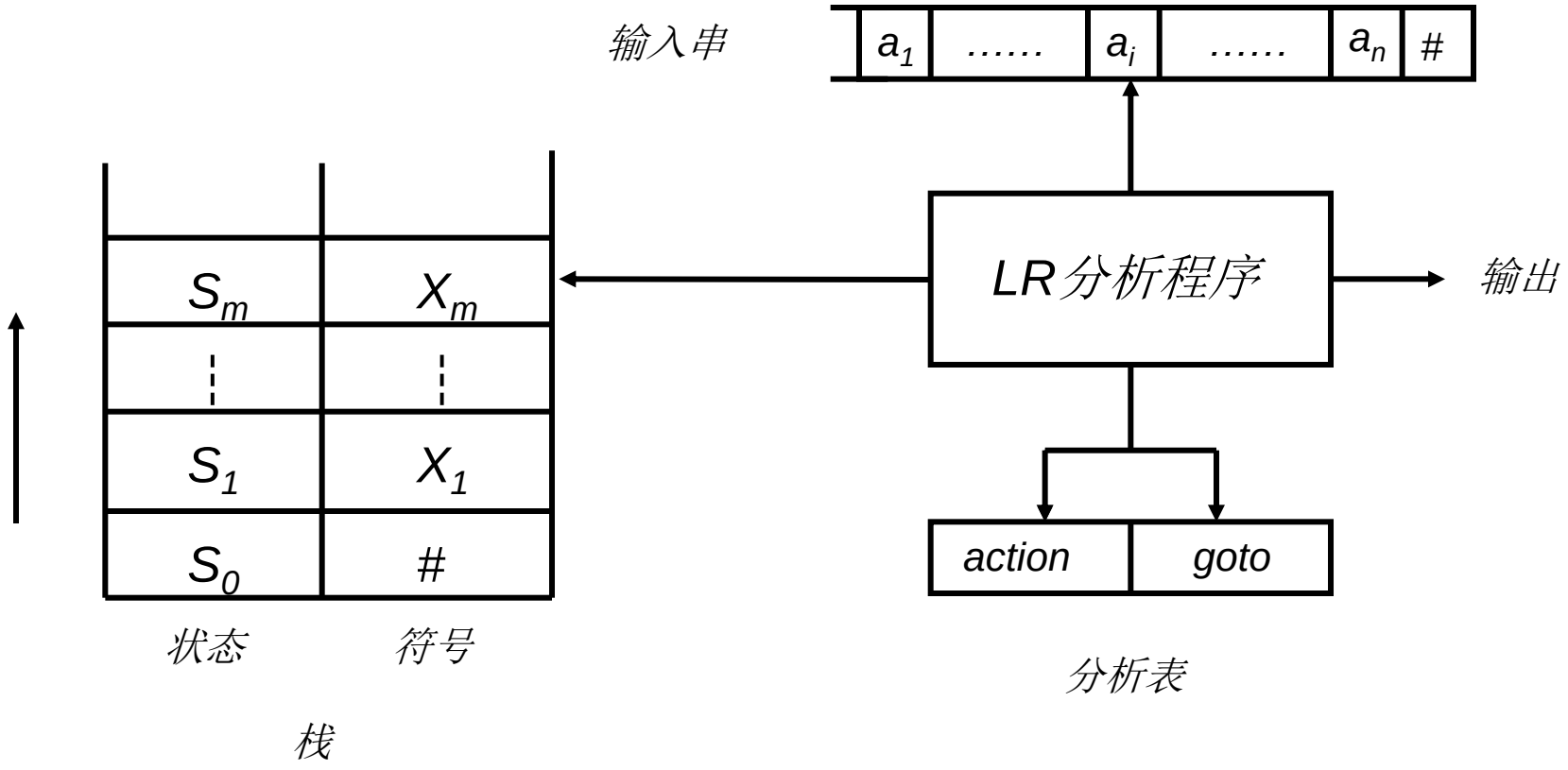
- 基本思想：在规范归约过程中，
  - 一方面记住“历史”
    - 记住已移进和归约出的整个符号串
  - 另一方面“展望”未来
    - 根据所用的产生式推测未来可能碰到的输入符号
- 当一串貌似句柄的符号串呈现于分析栈的顶端时
  - 根据 “历史”、“展望”和“现实”的输入符号等三方面信息
  - 来确定栈顶的符号串是否构成相对某一产生式的句柄

LR

- 一个LR分析器实质上是一个带先进后出存储器（栈）的确定状态有限状态自动机
  - 将“历史”与“展望”信息抽象成为某些状态
  - 先进后出存储器（分析栈）用于存放状态
  - 栈里的每个状态都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
  - 任何时候，栈顶的状态都代表整个历史和已推测的展望
- LR分析器的每一步工作都是由栈顶和现行输入符号所唯一决定的

LR

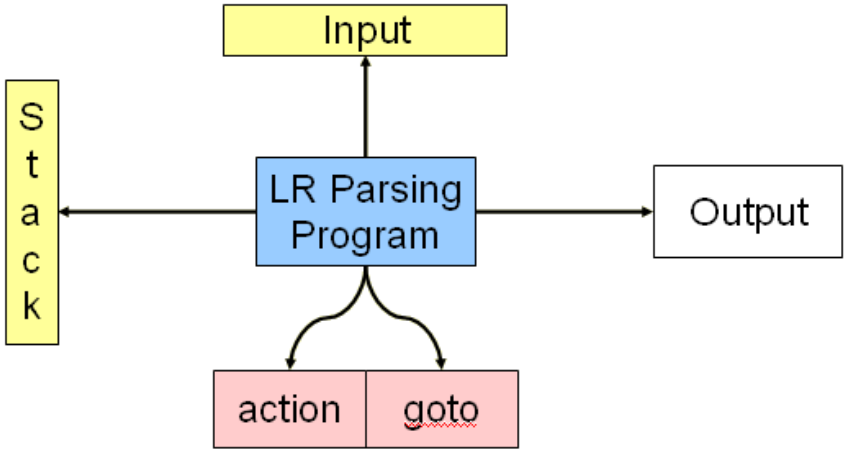
- LR分析程序的实质：分析栈 + DFA



LR table

- 分析表由ACTION表和GOTO表两部分组成。
  - ACTION(s，a)：表示当状态s面临输入a时的动作
  - GOTO(s，x)：表示面对文法符号x的下一状态

  - ACTION[s，a]表中的动作种类：
  - ① 移进
  - ② 归约
  - ③ 接受
  - ④ 报错



Example

例9 文法G

(1) E→E + T　(2)E→T

(3) T→T*F　(4)T→F

(5) F→(E)　(6)F→i

分析表中记号的含义

sj:　把下一状态 j 和现行输入符号 a 移进栈;

rj:　按第 j 个产生式进行归约，栈中移出数个元素，数量等于产生式右部长度;

acc:　接受;

空白格：出错标志，报错

Example

状态	ACTION(动作)						GOTO(转换)		
	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			

1)E→E + T

2)E→T

3)T→T*F

4)T→F

5)F→(E)

6)F→i

Example

利用上述分析表，假定输入串为 i * i + i，描述LR分析器的工作过程。

	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	#

LR

- 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，则我们把这个文法称为LR文法。

- LR（k）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

LR(0)

- LR(0)分析表的构造步骤
  - 确定G的LR(0)项目
  - 以LR(0)项目为状态,构造一个能识别文法G的所有活前缀的NFA
  - 利用子集法,将NFA确定化,成为以项目集合为状态的DFA根据
  - 上述DFA可直接构造出LR分析表

LR(0) Collection

- LR(0)项目（简称项目）
- 文法G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。

- 如：A→XY对应三个项目：

  A→·XY　　A→X·Y　　A→XY·

  而：　A→ε的项目 A→·

LR(0) Collection

- 项目的意义：指明在分析过程的某时刻，我们看到产生式多大一部分

- 字的前缀：指该字的任意首部如：abc前缀：ε，a，ab，abc

- 活前缀：规范句型的一个前缀，该前缀是不含句柄之后的任何符号。

LR(0) collection

- G[S]:
- 若S $\overset{*}{\underset{rm}{\Rightarrow}}$ αAω $\underset{rm}{\Rightarrow}$αβω　r是αβ的前缀，则
- 称r是G的一个活前缀
- 1.活前缀已含有句柄的全部符号，表明产生式A→β的 右部β已出现在栈顶
- 2.活前缀只含句柄的一部分符号表明A→$\beta_1\beta_2$的右部子串β_1已出现在栈顶，期待从输入串中看到β_2推出的符号
- 3. 活前缀不含有句柄的任何符号，此时期望A→β的右部所推出的符号串

NFA

- NFA的状态：是一个LR(0)项目
- 构造方法：
- a.文法开始符号的形如S’→•S的项目为NFA的唯一初态；
- b.若状态i和j出自同一个产生式，而且j的圆点只落后于I的圆点一个位置，就从i画一条标志为Xi的弧到j。
- （即，$i: X \to X_1 \cdots X_{i-1} \cdot X_i \cdots X_n$
- $j: X \to X_1 \cdots X_{i-1} X_i \cdot \cdots X_n$）
- c.若状态i的圆点之后的符号为非终结符，如I为
- X→α·Aβ，其中A属于V_N，就从状态i画ε弧到所有 A→·γ状态。

Example

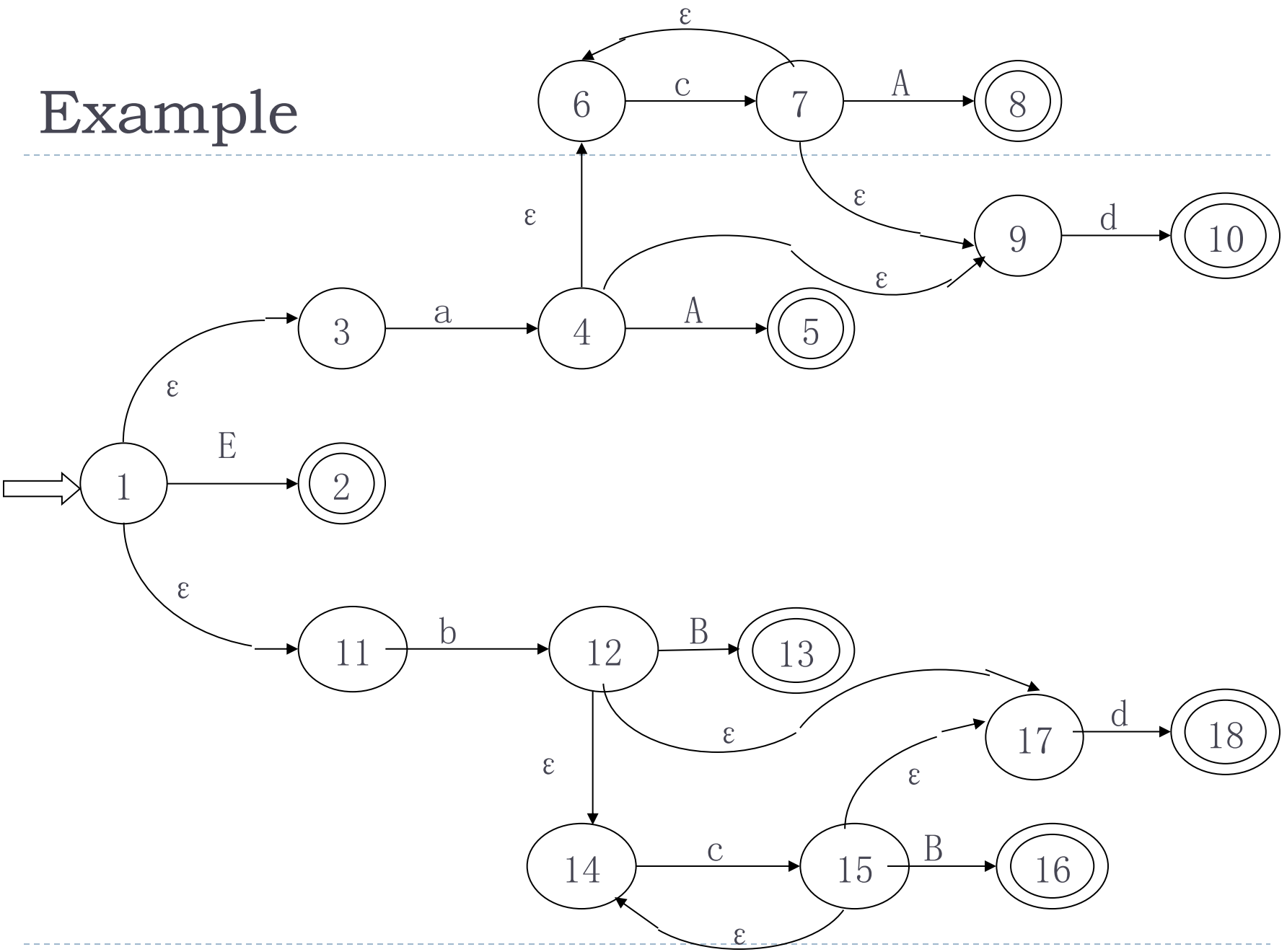
- 例10 文法 G
- **S′ —> E**
- **E —> aA|bB**
- **A —> cA|d**
- **B —> cB|d**

- 文法G的所有LR(0)项目

Example

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 ·

Example



NFA->DFA

- 使用子集方法，将NFA确定化，使之成为一个以项目集合为状态的DFA。

- 相关定义：
  - LR(0)项目集规范族：构成识别一个文法活前缀的DFA的项目集（状态）的全体。
  - 归约项目：$A \rightarrow \alpha \cdot$
  - 接受项目：$S' \rightarrow \alpha \cdot$　　(S'—文法的开始符号)
  - 移进项目：$A \rightarrow \alpha \cdot a\beta$　　(a—终结符)
  - 待约项目：$A \rightarrow \alpha \cdot B\beta$　　(B—非终结符)

DFA

- 利用CLOSURE方法构造LR(0)项目集规范族
- 拓广文法 (Augmented grammar)
- CLOSURE(I)算法(其中I为G的任一项目集)
  - 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。

NFA->DFA

- 步骤一:令NFA的初态为I,求其CLOSURE(I),得到初态项目集。即:

  求CLOSURE（{S’→•S}）

- 步骤二:对所得项目集I和文法G的每个文法符号X(包括V_T和V_N) 计算GO（I，X） =CLOSURE（J），得到新的项目集。
  - 其中J={任何形如A → αX• β的项目 | A → α • X β属于I}
- 步骤三:重复步骤二，直至没有新的项目集出现。

  经过以上步骤构造出的项目集的全体即为LR(0)项目集规范族。

Example

- 例11 文法 G
- **S′ —> E**
- **E —> aA|bB**
- **A —> cA|d**
- **B —> cB|d**

- 将其NFA确定化 ,并构造该文法的LR(0)分析表。

Example

- **假定：**

0、S′—> E　　1、E —> aA

2、E —> bB　　3、A —> cA

4、A —> d　　5、B —> cB

6、B —> d

Example

CLOSURE ({ S'->•E }) = { S'->•E, E->•aA, E->•bB }

此即为DFA的状态0

令I = { S'->•E, E->•aA, E->•bB }

GO(I, a) = CLOSURE({ E->a•A })

= { E->a•A, A->•cA, A->•d }

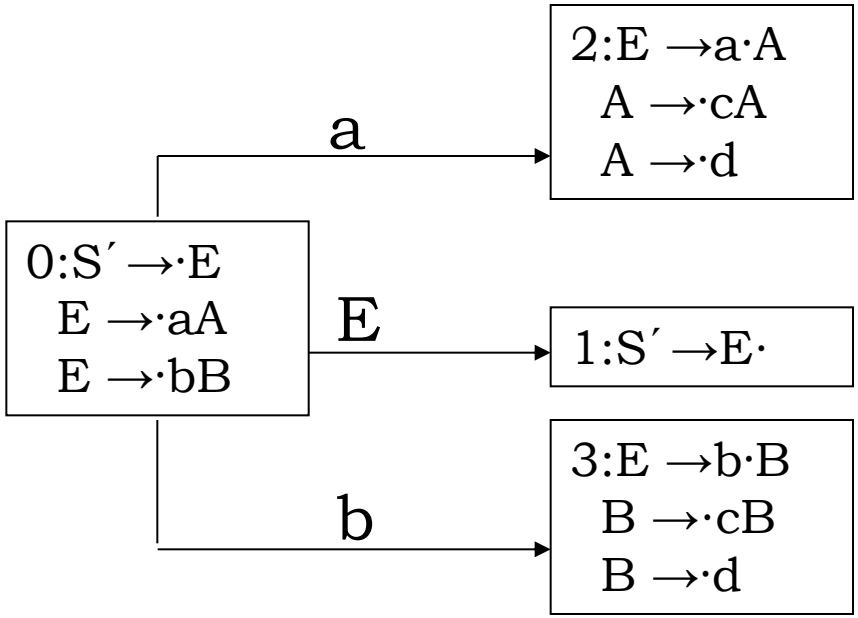
GO(I, b) = CLOSURE({ E->b•B })

= { E->b•B, B->•cB, B->•d }

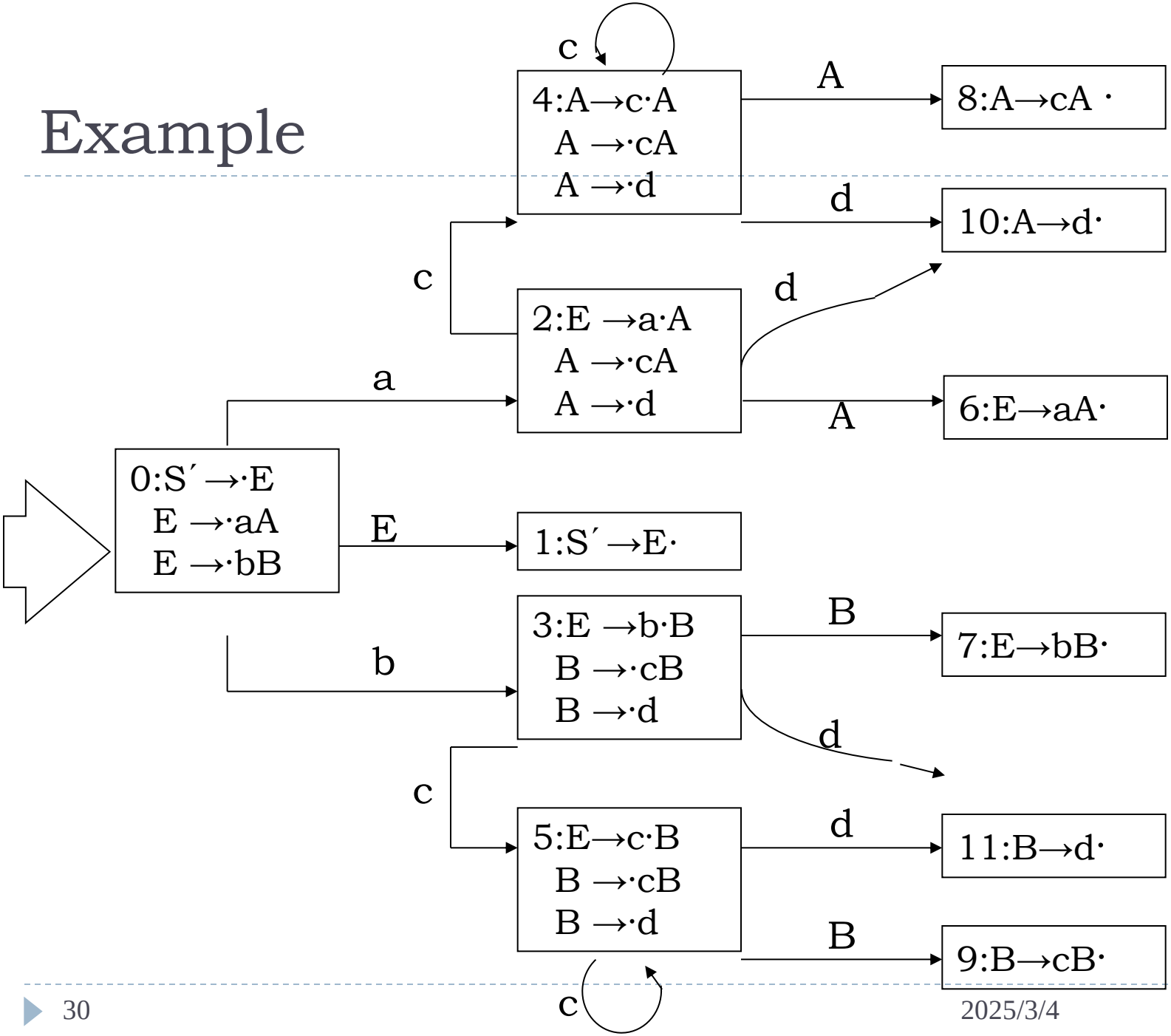
GO(I, E) = CLOSURE({ S'->E• })

= { S'->E• }

Example



Example


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:E→c·B
B →·cB
B →·d
6:E→aA·
7:E→bB·
8:A→cA ·
9:B→cB·
10:A→d·
11:B→d·
E
a
b
c
A
B
d

Action & GOTO tables

- 相关定义：
- LR(0)文法：不存在以下两种冲突的文法

  移进 - 归约冲突

  归约 - 归约冲突
- LR(0)表：由LR(0)文法得到的分析表
-
- LR(0)分析器：使用LR(0)表的分析器

Table construction

a、若项目A→α·aβ属于I_k且GO(I_k,a) = I_j，a为终结符，且置ACTION[k，a]为“把（j，a）移进栈”，简记为“sj”；

b、若项目A→α·属于I_k，那么，对任何输入符号a（或者结束符＃）置ACTION[k，a]为“用产生式A→α进行归约”，简记为“rj”；

其中，假定A→α为文法G’的第j个产生式；

c、若项目S’→S·属于I_k，则置ACTION[k，＃]为“接受”，简记为“acc”；

d、若GO（I_k，A）＝I_j，A为非终结符，则置GOTO[k，A]＝j；

e、分析表中凡不能使用规则1至4填入信息的空白格均置上“出错标志”。

Example

状态	ACTION(动作)					GOTO(转换)		
	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			

- 例：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	#

SLR分析表的构造

- LR(0)文法太简单，没有实用价值.
- 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。

- 假定LR(0)规范族的一个项目集$I=\{A_1\rightarrow\alpha\cdot a_1\beta_1, A_2\rightarrow\alpha\cdot a_2\beta_2, \ldots, A_m\rightarrow\alpha\cdot a_m\beta_m, B_1\rightarrow\alpha\cdot, B_2\rightarrow\alpha\cdot, \ldots, B_n\rightarrow\alpha\cdot\}$如果集合$\{a_1, \ldots, a_m\}$, $FOLLOW(B_1), \ldots, FOLLOW(B_n)$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$a\in FOLLOW(B_i)$，i=1,2,...,n，则用产生式$B_i\rightarrow\alpha$进行归约；
  3. 此外，报错。
- 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:

- 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.
  - 拓广方法：增加$S' \rightarrow S$，新起始符号
- 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′→· 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.