第 6 章 LR 分析

1. LR 分析

(1) LR 分析法

自底向上分析法，即移进-归约的过程

(2) 为什么提出 LR 分析法？

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

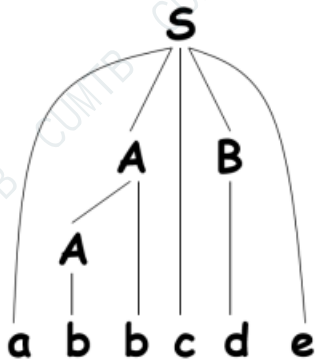
(3) 缺点：构造文法分析器的工作量大

【自底向上分析举例】例 1 文法 G[S]：

① S → aAcBe
② A → b
③ A → Ab
④ B → d

分析句子 abbcde

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



回顾输入串 #abbcde# 的移进-归约分析过程：

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

【讨论】何时移进？何时归约？用哪个产生式归约？

(4) 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。

(5) LR 分析器的种类

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）

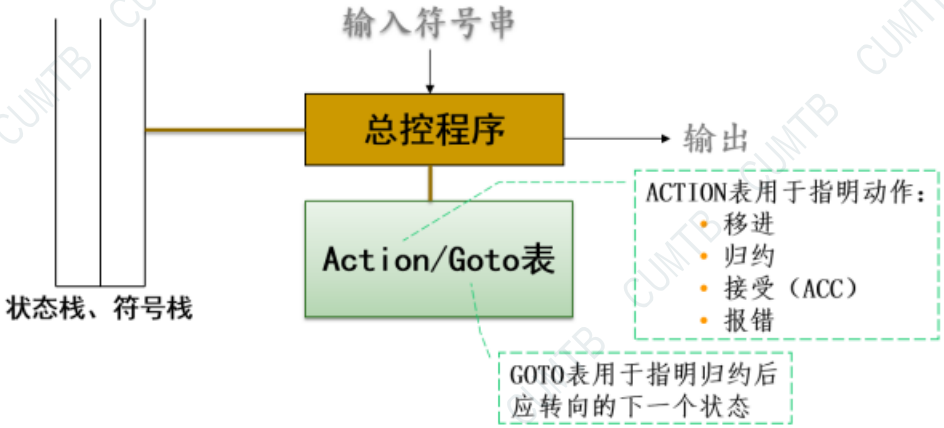
——改进：LALR(1)

(6) LR 分析器的组成

① 总控程序：LR 分析器

② 分析表(或分析函数)：动作表 Action 和 状态转换表 Goto

③ 分析栈：状态栈 和 文法符号栈



(7) LR 分析器的工作过程

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”。

k：所用产生式的右部长度

- **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

2. LR(0)分析

(1)【LR 分析举例】例 1 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。

假定 LR 分析表已经存在：

	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 分析表是分析过程的重要依据，非常重要！

【讨论】

① 对于一个文法，状态集是如何确定的？

② LR 分析表是如何得到的？

(1) 构造 LR(0)分析表的相关概念和方法：

① 可归前缀、活前缀

② 构造识别活前缀的 DFA

③ 利用 DFA 构造 LR 分析表

(2) 可归前缀与活前缀

【举例】

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

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	

- **可归前缀**：每次归约之前符号栈中的内容。如上例中的 ab、aAb、aAcd、aAcBe

- **活前缀**（Viable Prefixes）：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。
  - 【举例】

  可归前缀 ab 的活前缀：　ε , a , ab
  可归前缀 aAb 的活前缀：　ε , a , aA , aAb
  可归前缀 aAcd 的活前缀：　ε , a , aA , aAc , aAcd
  可归前缀 aAcBe 的活前缀：　ε , a , aA , aAc , aAcB , aAcBe

  - 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$\alpha\beta$的前缀，则称 γ 是G的一个**活前缀**。

  - Viable：可实施的，能独立发展的
    capable of being put into practice : workable
    capable of growing and developing <~ seed>

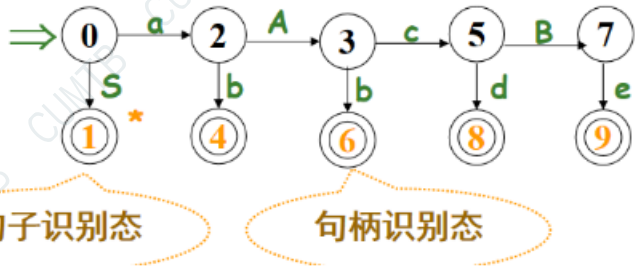
(3) 构造识别活前缀的 DFA

- 识别活前缀的 DFA
  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，同时进行**状态转换**。
  - ✓ 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。

【举例】

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



- 利用 DFA 分析输入串

步骤：

① 从初态出发，（自左向右）扫描源程序，同时进行状态转移

② 到达句柄识别态时，则归约

③ 判断是否归约到文法的句子识别态

是：结束

否：继续①

【举例】例 1 文法 G[S]：

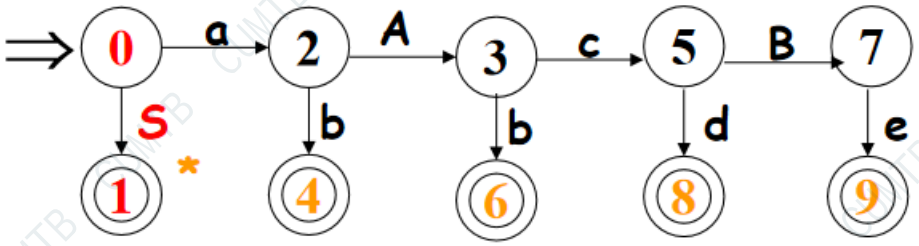
⑤ S → aAcBe

⑥ A → b

⑦ A → Ab

⑧ B → d

对应的识别活前缀的 DFA 如图所示，利用 DFA 分析输入串 #abbcde#。



（动画逐步演示其过程）

- 构造识别活前缀的 DFA 的方法

方法一：形式定义法

方法二：项目法

方法三：项目集规范族法

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

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）：

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

【举例】例 1 文法 G[S]：

① S → aAcBe
② A → b
③ A → Ab
④ B → d

求：LC(S') LC(S) LC(A) LC(B)

不包含句柄在内的所有活前缀的求解：

LC(S′) = {ε}
LC(S) = LC(S′) {ε}
LC(A) = LC(S) {a} ∪ LC(A) {ε}
LC(B) = LC(S) {aAc}
可化简为：
[S′] = ε
[S] = [S′]
[A] = [S]a+[A]
[B] = [S]aAc
求解得：
[S′] = ε
[S] = ε
[A] = a+[A] = a|[A] = aε*=a
[B] = aAc

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)

文法的所有可归前缀的求解：

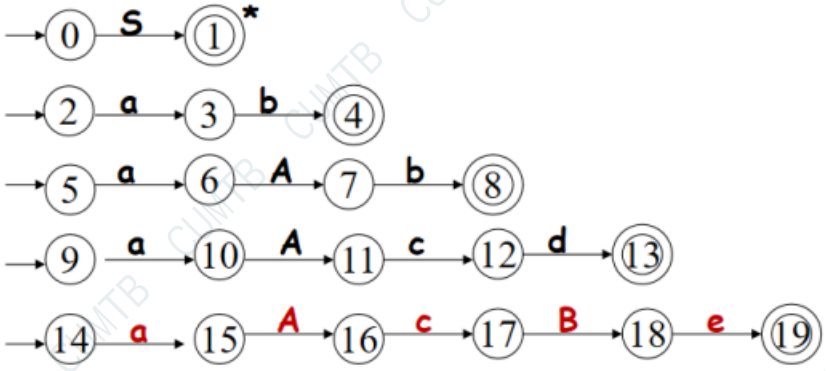
LR(0)C(S′→S) = LC(S′) {S} =S
LR(0)C(S→aAcBe) = LC(S) {aAcBe} =aAcBe
LR(0)C(A→b) = LC(A) {b} =ab
LR(0)C(A→Ab) = LC(A) {Ab} =aAb
LR(0)C(B→d) = LC(B) {d} =aAcd

根据形式定义法，求得的可归前缀如下：

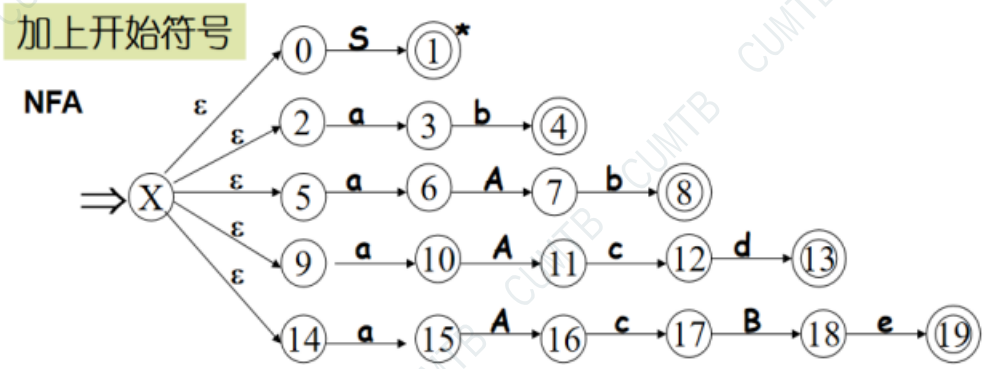
S
ab
aAb
aAcd
aAcBe

如何根据可归前缀构造识别文法活前缀的有限自动机？

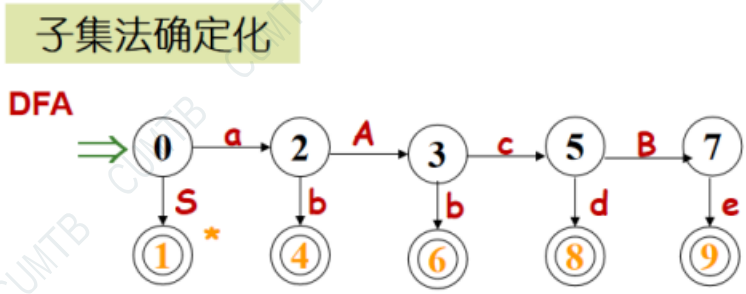
① 构造识别其活前缀及可归前缀的有限自动机：



② 加上开始符号



③ 子集法确定化



总结：形式定义法构造识别活前缀的 DFA 的步骤

① 根据形式定义求文法的可归前缀

② 为每个可归前缀构造FA

③ 增加一个初态，整合所有的FA为一个NFA

④ 利用子集法，构造等价的DFA

【讨论】形式定义法理论严格，但实现复杂。是否存在一种比较实用的方法？

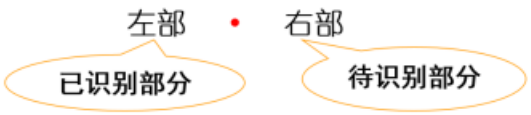
(5) 项目法

① 求出文法的所有项目

- 定义：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。

【举例】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 •



特例： A→ ε 只有一个项目 A→•

- 构造文法的所有产生式的所有项目

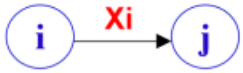
说明：每个项目都为 NFA 的一个状态

项目类型：

- ✓ 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 按照一定规则构造 NFA

- 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 A→α•
  - ✓ 句子识别态：接受项目 S'→S•
- 确定状态之间的转换关系
  - 若 项目 i 为 $X \rightarrow X_1X_2...X_{i-1} \bullet X_i...X_n$
    项目 j 为 $X \rightarrow X_1X_2...X_{i-1}X_i \bullet X_{i+1}...X_n$



  - 若 项目i为$X \rightarrow \gamma \bullet A\delta$
    项目k为$A \rightarrow \bullet\beta$

③ NFA 到 DFA 等价变换

子集法

【举例】如下拓广后的文法G，使用项目法构造识别活前缀的DFA。

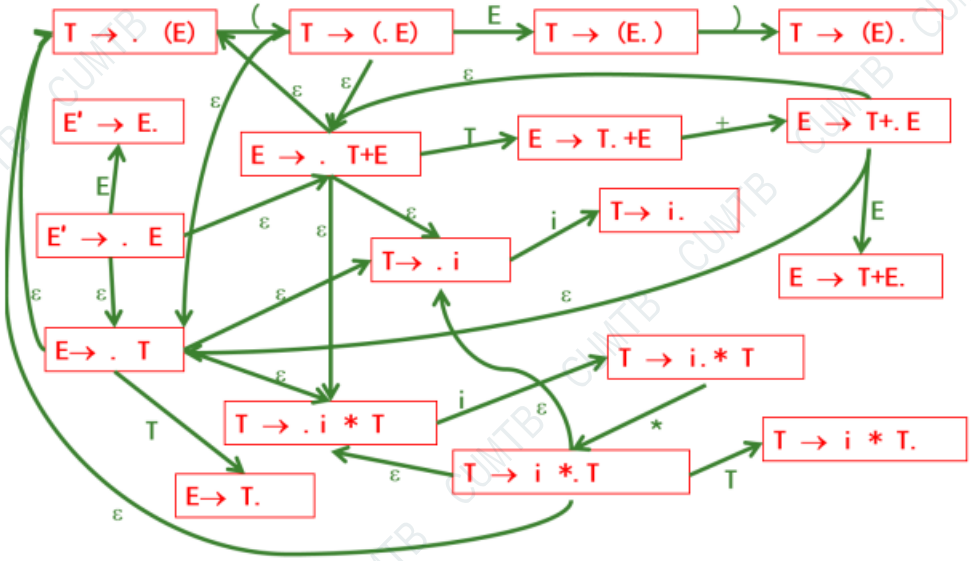
G： E’ →E
E→T + E
E→T
T→i * T
T→i
T→ (E)

解：

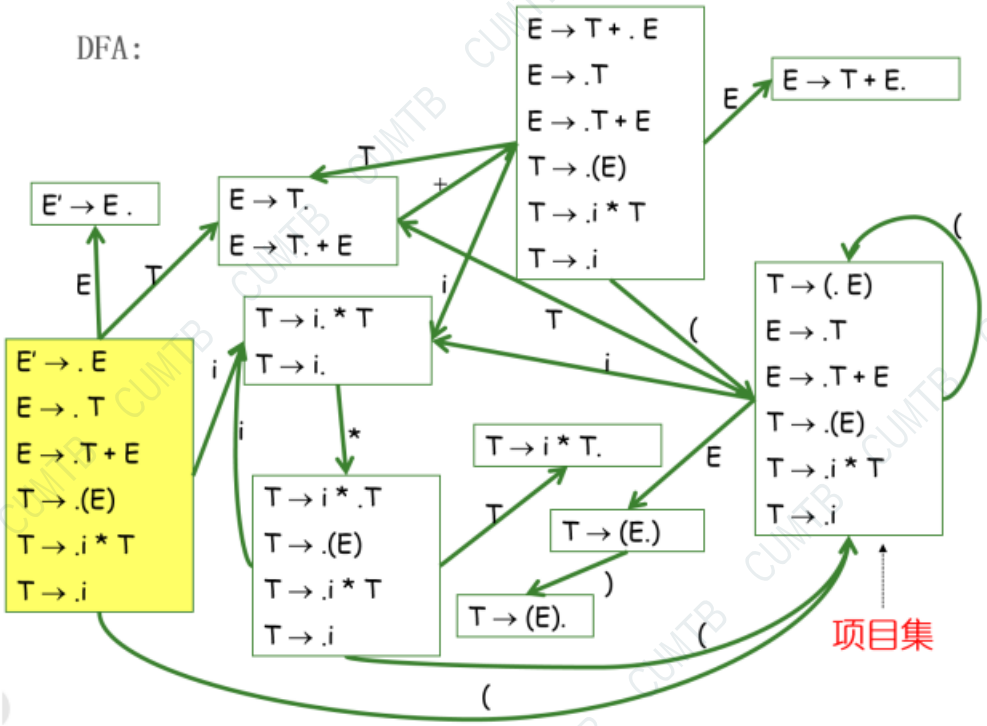
① 文法的所有项目 ② 确定初态、句柄/句子识别态

[1] E’ → • E　　[1]（初态）
[2] E’ → E •　　[2]（句子识别态）
[3] E → • T + E
[4] E → T • + E
[5] E → T + • E
[6] E → T + E •　　[6]（句柄识别态）
[7] E → • T
[8] E → T •　　[8]（句柄识别态）
[9] T → • i * T
[10] T → i • * T
[11] T → i * • T
[12] T → i * T •　　[12]（句柄识别态）
[13] T → • i
[14] T → i •　　[14]（句柄识别态）
[15] T → •(E)
[16] T → (• E)
[17] T → (E •)
[18] T → (E) •　　[18]（句柄识别态）

③ 确定状态之间的转换关系



④ NFA 到 DFA 的等价变换（子集法）



(6) 项目集规范族法

- 项目集规范族：识别文法活前缀的 DFA 项目集（状态）的全体。

- 项目法缺点：NFA 确定化为 DFA 的工作量较大。

【讨论】是否能直接构造出“LR(0)项目集规范族”和“识别活前缀的 DFA”？

步骤：

① 拓广文法　　S'→S

② 通过核的闭包和转换函数，求出 LR(0)项目集规范族

- **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) —— 构成项目集**
  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

- **转换函数：GO(I，X)= CLOSURE(J)**
  I 为包含某一项目集的状态，X 为文法符号
  J = { 任何形如 A→αX•β 的项目，A→α•Xβ 属于I }

■ **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

【举例】文法 G[S]：

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。

解：

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' → •S
S → •aAcBe

$I_1[I_0 -S→ I_1]$
S' → S•

$I_2[I_0 -a→ I_2]$
S → a•AcBe
A → •b
A → •Ab

$I_3[I_2 -A→ I_3]$
S → aA•cBe
A → A•b

$I_4[I_2 -b→ I_4]$
A → b•

$I_5[I_3 -c→ I_5]$
S → aAc•Be
B → •d

$I_6[I_3 -b→ I_6]$
A → Ab•

$I_7[I_5 -B→ I_7]$
S → aAcB•e

$I_8[I_5 -d→ I_8]$
B → d•

$I_9[I_7 -e→ I_9]$
S → aAcBe•

③ 判别是否为 LR(0)文法

LR(0)文法：没有“**移进-归约冲突**”项目集，且没有“**归约-归约冲突**”项目集。

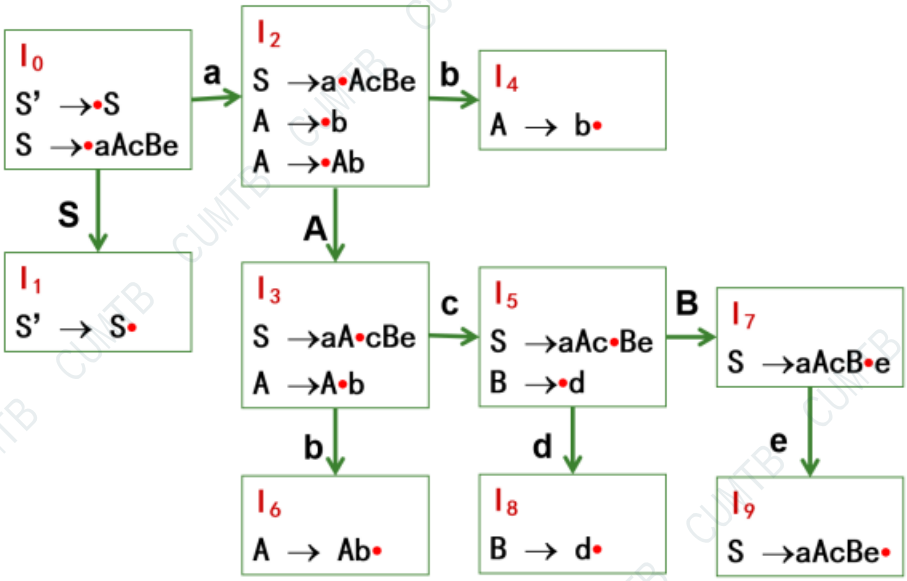
✓ 移进项目：$A→ \alpha •a \beta \quad (a∈V_T)$
✓ 归约项目：$A→ \alpha •$

【举例】判定上例是否为 LR(0)文法。

该文法没有“移进-归约冲突”项目集，且没有“归约-归约冲突”项目集，故是 LR(0)文法。

④ 由转换函数建立状态之间的连接关系，直接得到识别活前缀的 DFA

【举例】构造上例的识别活前缀 DFA



(7) 利用 DFA 构造 LR 分析表

① LR(0)分析表的重要性：是总控程序的分析动作依据。

② 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非终结符时应转向的下一个状态）

③ LR(0)分析表的构造算法

设 C={ I0, I1, …, In }

1) 若移进项目 A→α•aβ属于Ik，且转换函数f(Ik,a)=Ij

则 置Action[k,a] = Sj

2) 若f(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 终结符 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。

【举例】拓广后的文法 G[S']：

S' → S

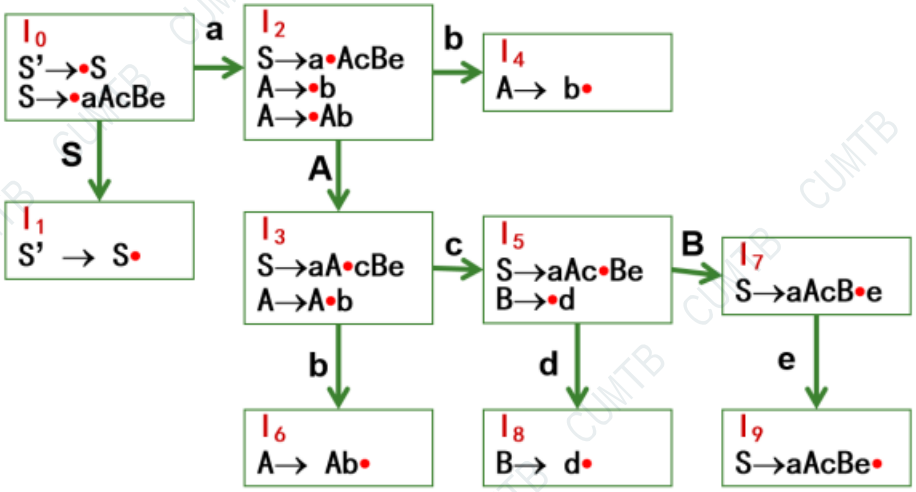
① S → aAcBe

② A → b

③ A → Ab

④ B → d

对应的识别活前缀 DFA 如下，请构造 LR(0)分析表。



解：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			

✧ 利用 LR(0)分析表对句子进行分析，前面已讲过：

LR分析句子

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。

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			

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	

(8) 总结：项目集规范族法进行 LR(0)分析的步骤

步骤一：拓广文法

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

(9) 【讨论】为什么 LR(0)需要判断冲突？

【举例】文法 G[E']：E' → E

① E → T + E

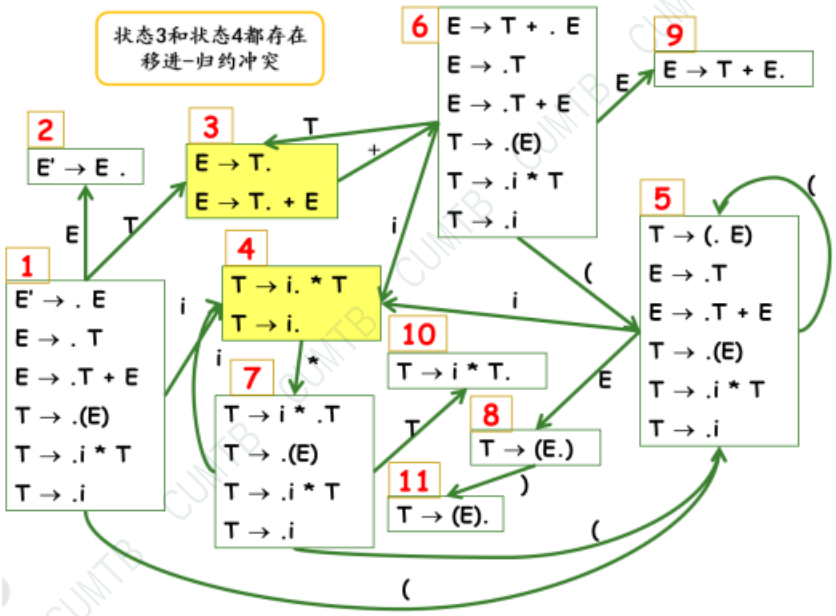
② E → T

③ T → i * T

④ T → i

⑤ T → (E)

对应的识别活前缀 DFA：



则

状态	Action						Goto		
	+	*	(	)	i	#	S	A	B
1									
2									
3	r/S	r	r	r	r	r			
4	r	r/S	r	r	r	r			
5									
6									
7									
8									
9									
10									

状态 3、5 处存在动作选择冲突

【讨论】如何解决 LR(0)的这种冲突？

对于有冲突的状态，向前查看一个符号，以确定采用的动作。

—— SLR(1)分析法

3. SLR(1)分析

(1) 目的：试图解决 LR(0)分析中的“移进-归约冲突”和“归约-归约冲突”。

【举例】

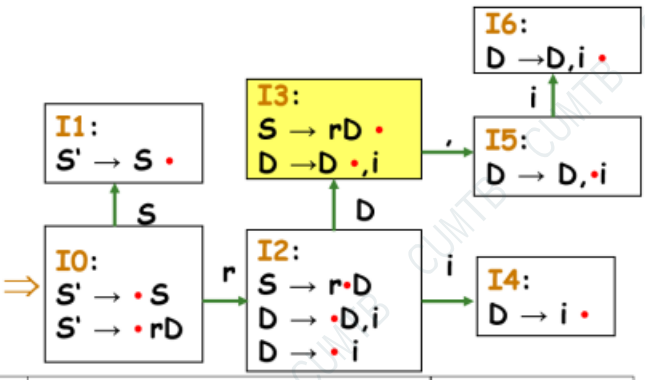
例 文法G：

(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		

(2) 解决冲突的主要思想：

向前查看一个符号，看其是否是归约项目左部的后跟符号，即是否属于 FOLLOW(S)

- 是，则归约
- 否，则移进

【举例】上例中的 LR(0) 分析表可改写成：

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		

(3) LR(0)项目集解决冲突的方法

一个LR(0)规范族中含有如下的项目集（状态）I

$I = \{ X \rightarrow \alpha \cdot b\beta \ , \quad A \rightarrow \gamma \cdot \ , \quad B \rightarrow \delta \cdot \ \}$

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 归约

3）若a∈FOLLOW(B)，则用产生式 B→δ 归约

4）此外，空白报错

(4) SLR(1)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。

(5) “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。

(6) SLR(1)分析表的构造步骤

a) 若项目A→α•aβ 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

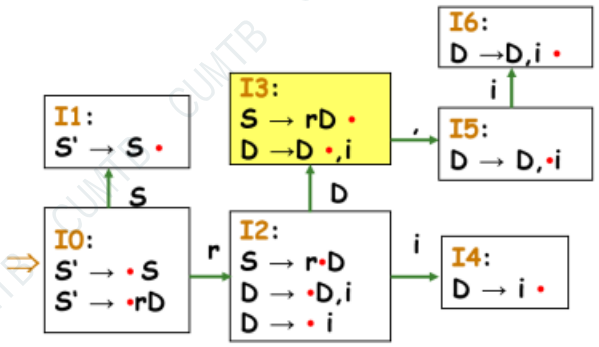
c) 项目A→α• 属于I_k，则对a为任何终结符或‘#’，且满足a∈FOLLOW(A)时，置ACTION[k,a] = r_j，j为产生式在文法中的编号

d) 若项目S‘→S• 属于I_k，则置ACTION[k,#] = acc

e) 其它填上“报错标志”

【举例】

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		

(7) 总结 SLR(1)分析的步骤

步骤一：拓广文法
步骤二：构造 SLR(1)项目集规范族
步骤三：求各非终结符的 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 SLR(1)分析表
步骤七：分析句子

(8) SLR(1)存在的问题：仍有许多项目集规范族的冲突不能用 SLR(1)方法解决!
解决方法：引入LR(1)分析法

4. LR(1)分析

(1) 主要思想:

- 若项目集 A→α·Bβ 属于状态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

(2) 对比

SLR(1)用于产生式归约的符号　B→ γ·　，Follow(B)

LR(1)用于产生式归约的符号　B→ γ·　，First(β)
A→α·Bβ

(3) 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

① I的项目都在CLOSURE(I)中

② 若 A→α•Bβ ,a 属于CLOSURE(I),

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

③ 重复 3 直到CLOSURE(I)不再扩大

2) 构造转换函数　　GOTO（I，X）= CLOSURE（J）

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}

【举例】

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 •, #

(4) 【讨论】 ①上例中可有存在“移进-归约冲突”或“归约-归约冲突”的项目集？
②LR(1)是如何解决冲突的?

状态	Action						Goto		
	a	b	c	d	e	#	S	A	B
1									
2									
3									
4									
5			S_9	r					
6									
7			r	S_{11}					
8									
9									
10									

(5) 构造识别活前缀的 DFA

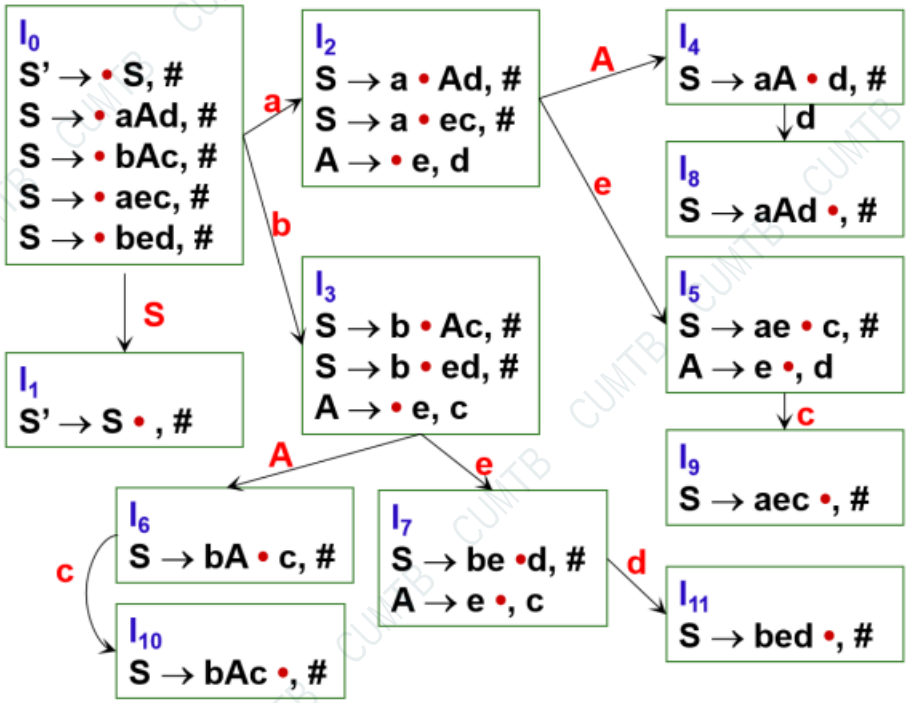
① DFA 的弧：转换函数（标红部分）

② 初态：I_0

③ 句柄识别态：归约项目所在的状态

④ 句子识别态：S’ → S •, #

【举例】根据上例中的 LR(1) 项目集规范族，构造识别活前缀的 DFA



其中，初态为 I_0 ，句柄识别态为 I_5、I_7、I_8、I_9、I_{10}、I_{11}，句子识别态为 I_1

(6) LR(1)分析表的构造

1) 若项目 $A\rightarrow\alpha\cdot a\beta, b$ 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 $A\rightarrow\alpha\cdot, a$ 属于 I_k ，置 $ACTION[k, a] = r_j$

j为产生式编号

4) 若项目 S ‘→S•, # 属于Ik ，则置 $ACTION[k, \#]=acc$

5) 其它空白，表示“报错”

【举例】根据上例 DFA 构造 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		

(7) 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。

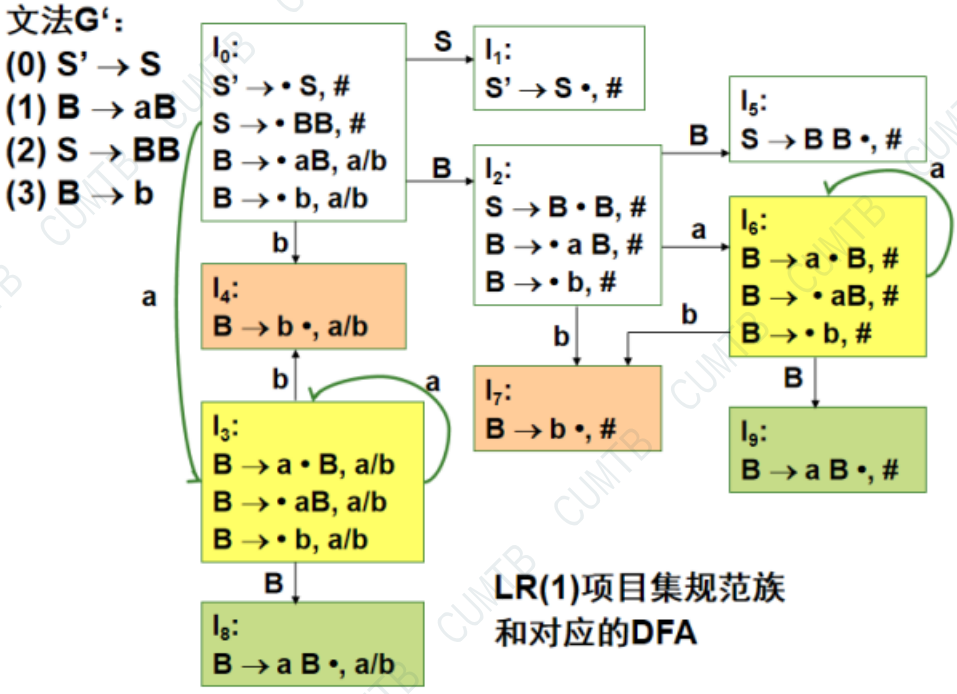
(8) LR(1)分析法存在问题：LR(1)项目集的构造对某些项目集的分裂可能使状态数目剧烈的增长。

(9) 问题的解决：采用LALR(1)分析法合并同心集。

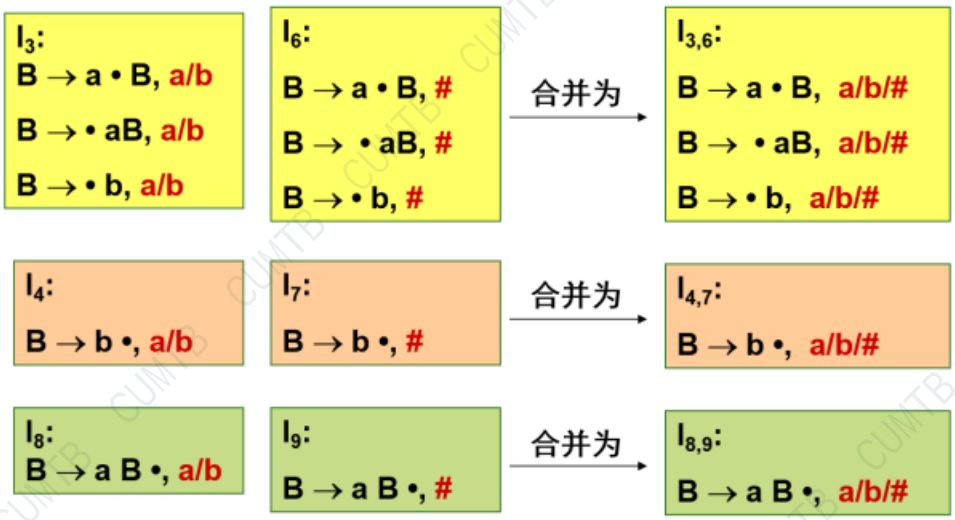
5. LALR(1)分析

【举例】

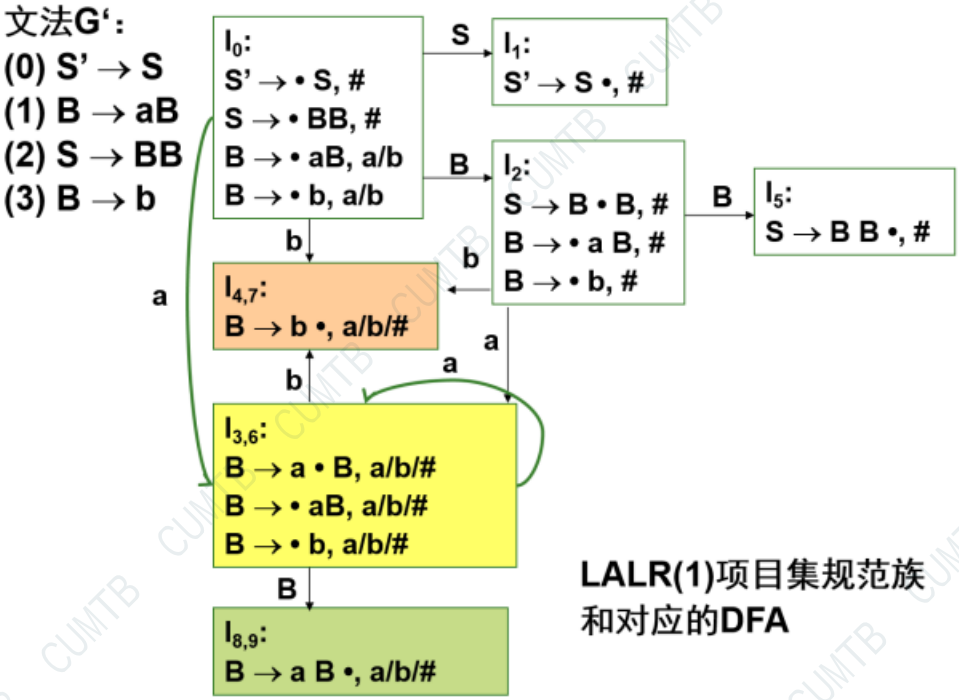
(1) LR(1)项目集规范族和对应的DFA



(2) 合并同心集



(3) LALR(1)项目集规范族和对应的DFA



(4) 合并同心集的几点说明

- 同心集合并后心仍相同，只是向前搜索符集合为各同心集向前搜索符的和集
- 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
- 合并同心集后可能会推迟发现错误的时间，但错误出现的位置仍是准确的

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	$S_{3,6}$	$S_{4,7}$			5
3,6	$S_{3,6}$	$S_{4,7}$			8,9
4,7	r_3	r_3	r_3		
5			r_2		
8,9	r_1	r_1	r_1		

6. 二义性文法在 LR 分析中的应用【自学】

使用 **优先关系** 和 **结合性** 来解决冲突。

【课堂练习】 1

1. 文法 $A \rightarrow aAd \mid aAb \mid \varepsilon$

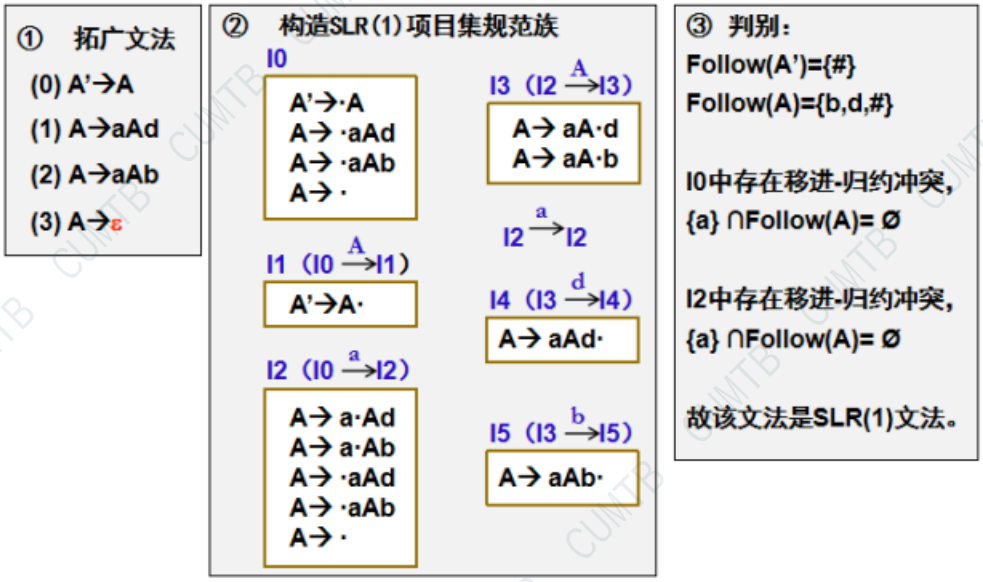
(1) 判断该文法是否为SLR(1)文法；

(2) 若是，请构造相应的分析表；

(3) 并对输入串#ab#给出分析过程。

解：

(1) 判断该文法是否为SLR(1)文法



(2) 构造相应的分析表

状态	Action				Goto
	a	b	d	#	A
0	S2	r3	r3	r3	1
1				acc	
2	S2	r3	r3	r3	3
3		S5	S4		
4		r1	r1	r1	
5		r2	r2	r2	

(3) 对输入串#ab#给出分析过程

	状态栈	符号栈	待输入串	动作
1	0	#	ab#	S2 移进
2	02	#a	b#	r3 （A → ε ）
3	023	#aA	b#	S5 移进
4	0235	#aAb	#	r2 （A →aAb）
5	01	#A	#	acc

【课后作业】 15

15. 文法　　S → a | ^ | (T)

　　T → T,S | S

(1) 构造 SLR(1)和 LR(1)分析表。

(2) 给出输入串 #(a,a# 的分析过程。

(3) 说明(1)中两种分析表发现错误的时刻和输入串的出错位置有何区别。

解：

(1) 构造分析表

- **SLR(1)**

① 拓广文法

(0) S'→S

(1) S→a

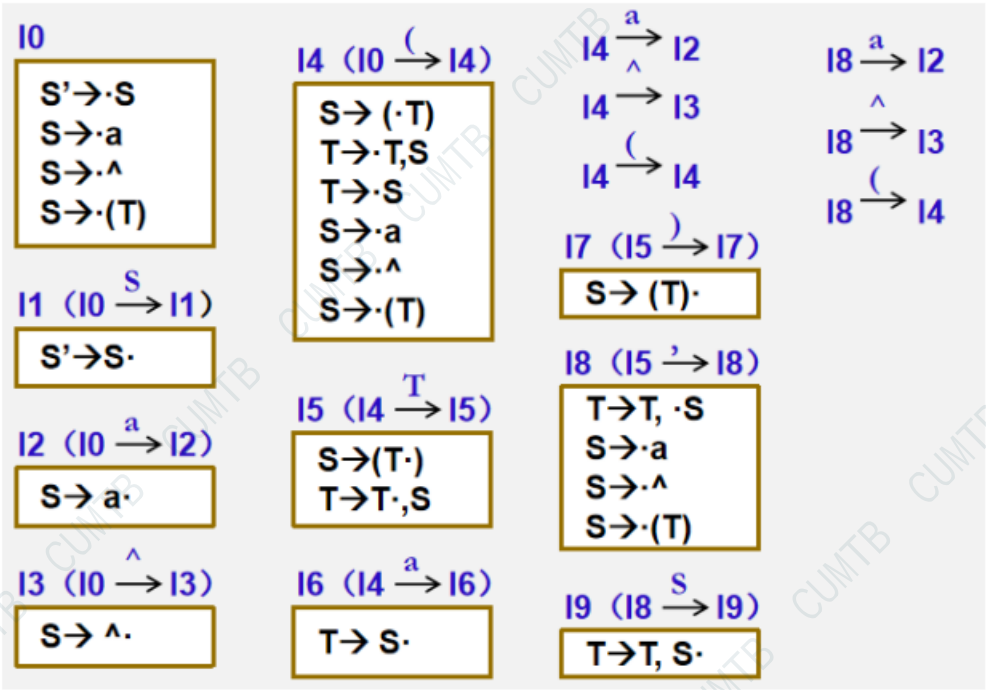
(2) S→^

(3) S→(T)

(4) T→T,S

(5) T→S

② 构造 LR(0)项目集规范族：



③ 判别

无移进-归约冲突，无归约-归约冲突，该文法是 LR(0)文法，也是 SLR(1)文法。

Follow(S')={#}

Follow(S)=Follow(S')∪Follow(T)={#　)　，}

Follow(T)={)　，}

④ 构造 SLR(1)分析表

状态	Action						Goto	
	a	^	(	)	,	#	S	T
0	S2	S3	S4				1	
1						acc		
2				r1	r1	r1		
3				r2	r2	r2		
4	S2	S3	S4				6	5
5				S7	S8			
6				r5	r5			
7				r3	r3	r3		
8	S2	S3	S4				9	
9				r4	r4			

- **LR(1)**

① 拓广文法

(0) S′→S

(1) S→a

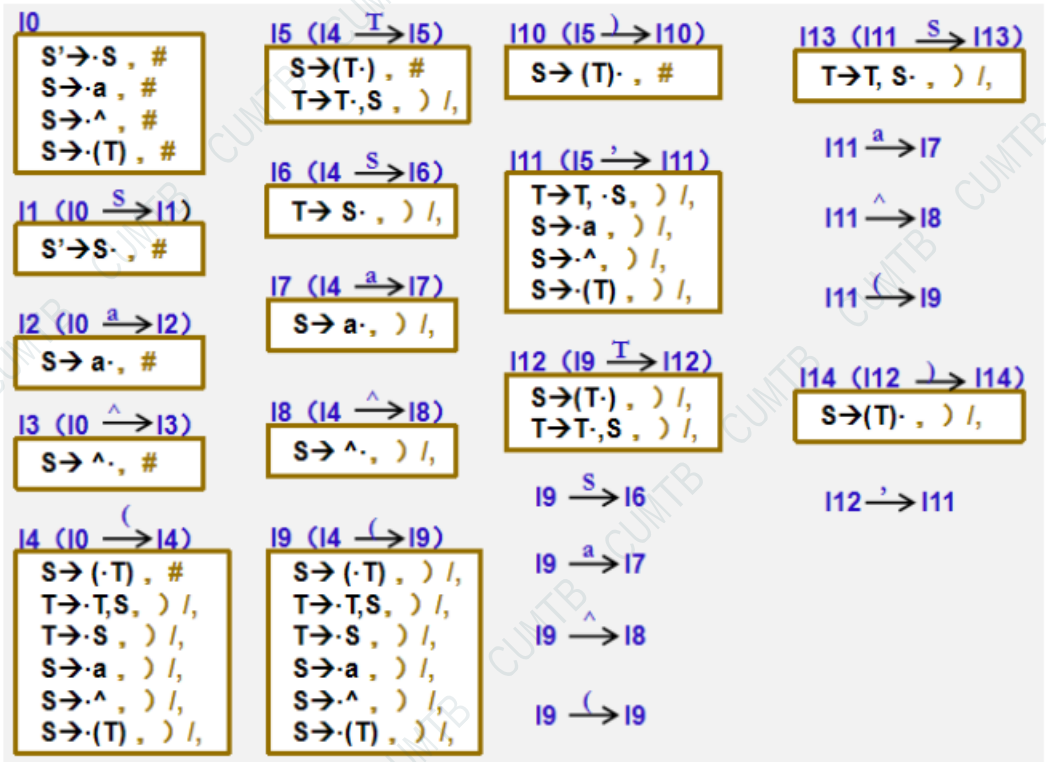
(2) S→^

(3) S→(T)

(4) T→T,S

(5) T→S

② LR(1)项目集规范族:



③ 判别

无冲突，该文法是 LR(1)文法。

④ 构造分析表

状态	Action						Goto	
	a	^	(	)	,	#	S	T
0	S2	S3	S4				1	
1						acc		
2						r1		
3						r2		
4	S7	S8	S9				6	5
5				S10	S11			
6				r5	r5			
7				r1	r1			
8				r2	r2			
9	S7	S8	S9				6	12
10						r3		
11	S7	S8	S9				13	
12				S14	S11			
13				r4	r4			
14				r3	r3			

(2) 给出输入串 #(a,a# 的分析过程。

■ SLR(1)分析

	状态栈	符号栈	待输入串	动作
1	0	#	(a,a#	S4 移进
2	04	#(	a,a#	S2 移进
3	042	#(a	,a#	r1（S→a）
4	046	#(S	,a#	r5 (T→S)
5	045	#(T	,a#	S8 移进
6	0458	#(T,	a#	S2 移进
7	04582	#(T,a	#	r1（S→a）
8	04589	#(T,S	#	报错

■ LR(1)分析

	状态栈	符号栈	待输入串	动作
1	0	#	(a,a#	S4 移进
2	04	#(	a,a#	S7 移进
3	047	#(a	,a#	r1（S→a）
4	046	#(S	,a#	r5 (T→S)
5	045	#(T	,a#	S11 移进
6	045(11)	#(T,	a#	S7 移进
7	045(11)7	#(T,a	#	报错

(3) 说明(1)中两种分析表发现错误的时刻和输入串的出错位置有何区别。

SLR(1)在第 8 步发现错误，LR(1)在第 7 步发现错误

SLR(1)报错的位置是第 5 个单词符号#，LR(1)报错的位置是第 4 个单词符号 a

说明 LR(1)比 SLR(1)能更加及时和准确地指出语法错误。

【本章小结】

- LR(0)
  1. 拓广文法： S’→S
  2. 构造LR(0)项目集规范族
  3. 判断： 不存在“移进-归约冲突”和“归约-归约冲突”
  4. 构造识别活前缀的DFA
     (1) 构造状态结点和弧
     (2) 确定“句子识别态”和“句柄识别态”
  5. 构造LR(0)分析表
     (1) 根据DFA标记为非终结符的弧，构造GOTO表
     (2) 根据DFA标记为终结符的弧，构造ACTION表的Si
     (3) 根据DFA的句柄识别态，构造ACTION表的ri
     (4) 根据DFA的句子识别态，构造ACTION表的acc
  6. 分析句子

     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”。
     k：所用产生式的右部长度
     - 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

- SLR(1)
  1. 拓广文法： S’→S
  2. 构造LR(0)项目集规范族
  3. 判断：
     Follow(归约项目的左部)∩{移进项目的待移进符号}= Ø
     Follow(归约项目*i*的左部)∩Follow(归约项目*j*的左部) = Ø
  4. 构造识别活前缀的DFA
     (1) 构造状态结点和弧
     (2) 确定“句子识别态”和“句柄识别态”
  5. 构造LR(0)分析表
     (1) 根据DFA标记为非终结符的弧，构造GOTO表
     (2) 根据DFA标记为终结符的弧，构造ACTION表的Si
     (3) 根据DFA的句柄识别态，只对Follow(归约项目左部)的符号，构造ACTION表的ri
     (4) 根据DFA的句子识别态，构造ACTION表的acc
  6. 分析句子 同LR(0)

- LR(1)
  1. 拓广文法： S’→S
  2. 构造LR(1)项目集规范族 （有向前搜索符）
  3. 判断：
     {归约项目的向前搜索符}∩{移进项目的待移进符号}= Ø
     {归约项目*i*的向前搜索符}∩{归约项目*j*的向前搜索符} = Ø
  4. 构造识别活前缀的DFA
     (1) 构造状态结点和弧
     (2) 确定“句子识别态”和“句柄识别态”
  5. 构造LR(0)分析表
     (1) 根据DFA标记为非终结符的弧，构造GOTO表
     (2) 根据DFA标记为终结符的弧，构造ACTION表的Si
     (3) 根据DFA的句柄识别态，只对归约项目的向前搜索符，构造ACTION表的ri
     (4) 根据DFA的句子识别态，构造ACTION表的acc
  6. 分析句子 同LR(0)

- LALR(1)
  1. 拓广文法 同LR(1)
  2. 构造LR(1)项目集规范族 同LR(1)
  3. 判断 同LR(1)
  4. 合并同心集
  5. 构造识别活前缀的DFA 同LR(1)
  6. 构造LR(0)分析表 同LR(1)
  7. 分析句子 同LR(0)