第六章　LR分析方法

第五章我们学过自底向上分析法的关键问题是在分析过程中如何确定句柄。**LR**方法是通过求句柄逐步归约进行语法分析。

LR分析概述

- **LR（k）**分析是根据当前分析栈中的符号串和向右顺序查看输入串的**k(k≥0)**个符号就可以唯一确定分析的动作是移进还是归约以及用哪个产生式归约。
- 从左到右扫描(L)自底向上进行规约(R)
  （是规范规约）

LR分析的优缺点

1)适合文法类足够大,适用于大多数上下文无关文法
2)分析效率高
3)报错及时
4)手工实现工作量大
5)可以自动生成

美国Bell实验室推出的编译程序自动构造工具——YACC：能接受一个用BNF描述的满足LALR（1）上下文无关文法并对其自动构造出LALR（1）分析器。

主要内容

一、LR分析概述（基本构造原理与方法）

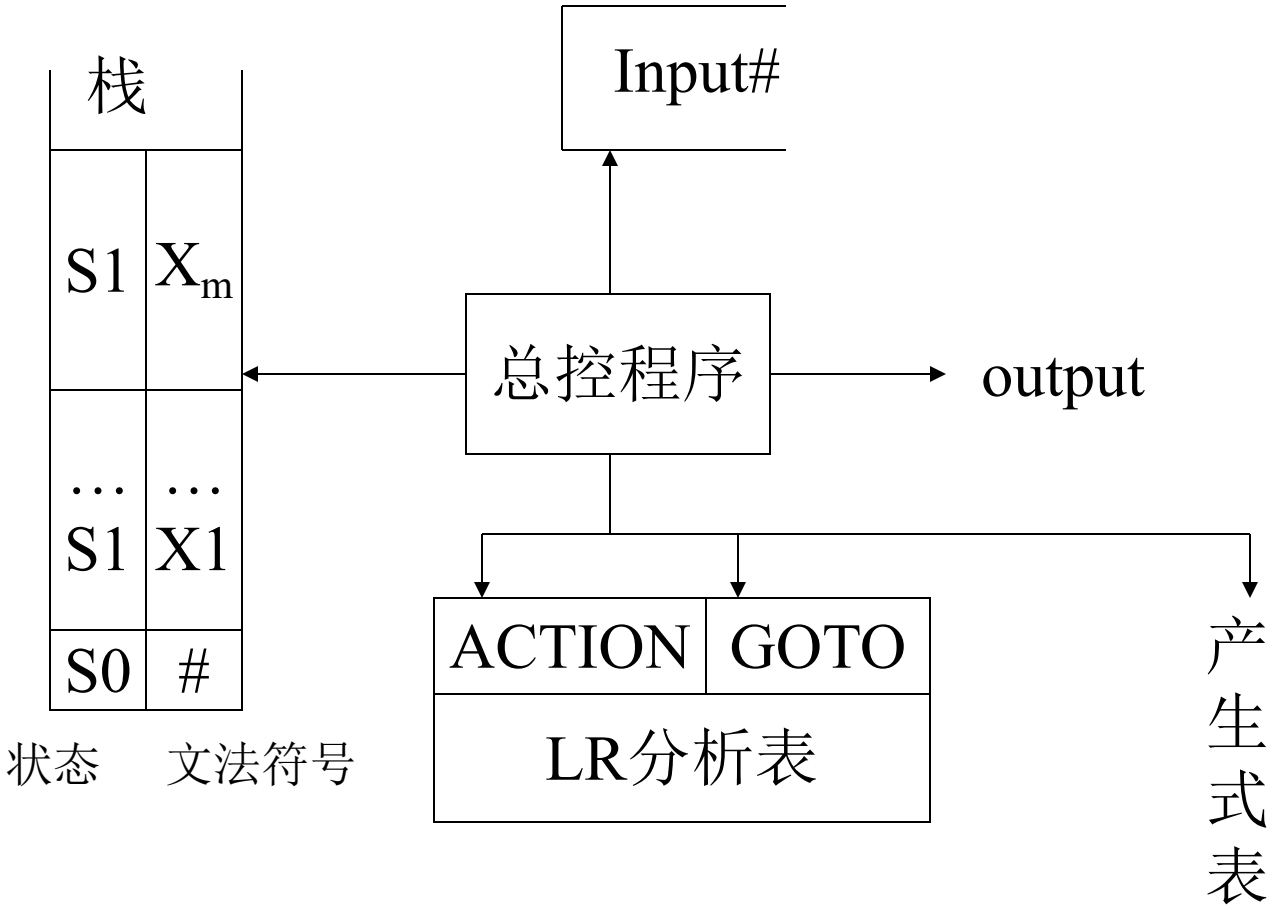
二、LR(0)分析

三、SLR(1)分析

四、LR(1)分析

五、LALR (1)分析

LR分析器模型


栈
Input#
S1
X_m
总控程序
output
…
…
S1
X1
S0
#
状态
文法符号
ACTION
GOTO
LR分析表
产生式表

2、LR分析方法的逻辑结构

逻辑上说，一个LR分析器由3个部分组成：

(1) 总控程序，也可以称为驱动程序。对所有的LR分析器总控程序都是相同的。

(2) 分析表或分析函数，不同的文法分析表将不同，同一个文法采用的LR分析器不同时，分析表也不同，分析表又可分为动作表（ACTION）和状态转换（GOTO）表两个部分，它们都可用二维数组表示。

(3) 分析栈，包括文法符号栈和相应的状态栈，它们均是先进后出栈。

分析器的动作就是由栈顶状态和当前输入符号所决定。

总控程序根据不同的分析表来决定其下一步的处理动作，分析表是根据具体的文法按某种规则构造出来的。**LR方法**是根据具体文法的分析表对输入串进行分析处理。

LR分析过程的思想是在总控程序的控制下，从左到右扫描输入符号串，根据分析栈中的状态和文法及当前输入符号，按分析表完成相应的分析工作。

分析表的组成：

(1) 分析动作表Action

状态 \ 符号	a_1	a_2	…	a_t
S_0	action[S_0 , a_1]	action[S_0 , a_2]	…	action[S_0 , a_t]
S_1	action[S_1 , a_1]	action[S_1 , a_2]	…	action[S_1 , a_t]
…	…	…	…	…
S_n	action[S_n , a_1]	action[S_n , a_2]	…	action[S_n , a_t]

表中action[S_i,a_j]为二维数组，指出当前栈顶为状态S_i，输入符号为a_j是所执行的动作。其动作有四种可能，分别为移进(S)、归约(r)、接受(acc)、出错(error)。

(2) 状态转换表goto

符号 / 状态	X_1	X_2	…	X_t
S_0	goto[S_0 , x_1]	goto[S_0 , x_2]	…	goto[S_0 , x_t]
S_1	goto[S_1 ,x_1]	goto[S_1 , x_2]	…	goto[S_1 , x_t]
…	…	…	…	…
S_n	goto[S_n , x_1]	goto[S_n , x_2]	…	goto[S_n , x_t]

表中goto[S_i,x_j]指出栈顶状态为S_i，文法符号为x_j时应转到的下一状态。

分析表种类的不同决定使用不同的LR分析方法

a) **SLR分析表(简单LR分析表)**

构造简单, 最易实现, 大多数上下文无关文法都可以构造出SLR分析表, 所以具有较高的实用价值。使用SLR分析表进行语法分析的分析器叫SLR分析器

b) **LR分析表(规范LR分析表)**

适用文法类最大, 大多数上下文无关文法都能构造出LR分析表, 但其分析表体积太大. 暂时实用价值不大.

c) LALR分析表(超前LR分析表)

这种表适用的文法类及其实现上难易在上面
两种之间,在实用上很吸引人.
使用LALR分析表进行语法分析的分析器叫LALR分析器。
例：YACC

几点说明

1. 三种分析表对应三类文法

2. 一个SLR文法必定是LALR文法和LR文法

3、LR分析过程：

((1))LR分析步骤：

(a) 将初始状态S_0和句子的左界符#分别进分析栈。

(b) 根据栈顶状态和当前输入符号查动作表，进行如下的工作。

※ 移进：当前输入符号进符号栈，根据状态转换表新的状态进状态栈，继续扫描，从而下一输入符号变成当前输入符号。

※ 归约：(1)按某个产生式进行归约，若产生式的右端长度为n，则符号栈顶和状态顶n个元素同时相应退栈。(2)归约后的文法符号进符号栈，(3)查状态转换表，新的状态进状态栈。

(c) 接受：分析成功，终止分析。

(d) 出错：报告出错信息。

(2) 具体分析过程：

举例说明具体分析过程：

设文法为G[L] （假定已存在LR分析表）

(1) L →E,L

(2) L →E

(3) E →a

(4) E →b

LR分析算法

- 置ip指向输入串w的第一个符号
  - 令S为栈顶状态
  - a是ip指向的符号
  - 重复 begin
  - if ACTION[S,a]=S_j
  - then begin PUSH j,a(进栈)
  - ip 前进(指向下一输入符号)
  - end
  - else if ACTION[S,a]=r_j (第j条产生式为$A\rightarrow\beta$)

LR分析算法

- then begin
  - pop |β| 项
  - 令当前栈顶状态为S’
  - push GOTO[S’,A]和A(进栈)
- end
- else if ACTION[s,a]=acc
  - then return (成功）
- else error

end.重复

程，在
绍如何

S_i表示把当前输入符号移进栈，且转第i个状态，即第i个状态进状态栈。

i表示转第i个状态，即第i个状态进状态栈。

r_i表示按第i个产生式进行归约

空白表示分析动作出错。

			ON			
			,	#	L	E
S_0	S_3	S_4			1	2
S_3			r_3	r_3		
S_4			r_4	r_4		
S_5	S_3	S_4			6	2
S_6				r_1		

根据上述分析表，即可对输入串进行分析。如输入串为#a,b,a#具体分析过程如下：

状态栈	符号栈	产生式	输入符号	说明
S_0	#		a,b,a#	S0和#进栈
S_0S_3	#a		,b,a#	a和S3进栈
S_0S_2	#E	E→a	,b,a#	a和S3退栈 E和S2进栈
$S_0S_2S_5$	#E,		b,a#	, 和S5进栈
$S_0S_2S_5S_4$	#E,b		, a#	b和S4进栈
$S_0S_2S_5S_2$	#E,E	E→b	, a#	b和S4退栈 E和S2进栈

状态栈	符号栈	产生式	输入符号	说明
$S_0S_2S_5S_2S_5$	#E,E,		a#	, 和S5进栈
$S_0S_2S_5S_2S_5S_3$	#E,E,a		#	a和S3进栈
$S_0S_2S_5S_2S_5S_2$	#E,E,E	E→a	#	a和S3退栈 E和S2进栈
$S_0S_2S_5S_2S_5S_6$	#E,E,L	L→E	#	E和S2退栈 L和S6进栈
$S_0S_2S_5S_6$	#E,L	L→E,L	#	E,L和$S_2S_5S_6$退 L和S6进栈
S_0S_1	#L	L→E,L	#	E,L和$S_2S_5S_6$退 L和S1进栈
				acc

- 自底向上分析法的关键问题是在分析过程中如何确定句柄。LR方法中的句柄是通过求可归前缀而求得。

活前缀与可归前缀

例：文法G[S]为：

S →aAcBe

A →b

A →Ab

B →d

为产生式加序号变为：

S →aAcBe[1]

A →b[2]

A →Ab[3]

B →d[4]

对于输入串abbcde句子的最右推导（规范推导）如下：

S⇒aAcBe[1] ⇒aAcd[4]e[1] ⇒aAb[3]cd[4]e[1]
⇒ab[2]b[3]cd[4]e[1]

对它的逆过程最左归约(规范归约)为：

ab[2]b[3]cd[4]e[1]

⇐aAb[3]cd[4]e[1]

⇐aAcd[4]e[1]

⇐aAcBe[1]

⇐S

为产生式加序号变为：

S →aAcBe[1]

A →b[2]

A →Ab[3]

B →d[4]

用哪个产生式继续归约仅取决于当前句型的前部。
我们把每次采取归约动作前的符号串部分称为可归前缀。
LR分析的关键就是识别何时到达可归前缀。

在规范句型中形成可归前缀之前包括可归前缀的所有前缀称为活前缀。活前缀为一个或若干规范句型的前缀。在规范归约过程中的任何时刻只要已分析过的部分即在符号栈中的符号串均为规范句型的活前缀，则表明输入串的已被分析过的部分是某规范句型的一个正确部分。

例如：有下面规范句型的前缀：

ε，a，ab

ε，a，aA，aAb

ε，a，aA，aAc，aAcd

ε，a，aA，aAc，aAcB，aAcBe

左部均为活前缀，其中，红色部分为可归前缀

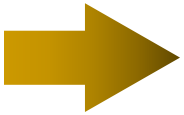
活前缀的定义及在LR分析中的应用

- G=(Vn,Vt,P,S),若有$S' \overset{*}{\underset{R}{\Rightarrow}} \alpha A \omega \Rightarrow \alpha \beta \omega$，$A \rightarrow \beta$（ β 是句柄 ）

  γ 是 α β 的前缀，则称是文法G的活前缀
- **α β 是含句柄的活前缀，并且句柄是 α β 的后端，则称 α β 是可归前缀或可规范前缀。**
- 在LR分析过程中，实际上是把 α β 的前缀（即文法G的活前缀）列出放在符号栈中，一旦在栈中出现 α β（形成可归前缀），即句柄已经形成，则用产生式**A** $\rightarrow$ β 进行归约。

- 对任何一个上下文无关文法，只要能构造出它的识别可归前缀的有限自动机，就可以构造其相应的分析表（状态转换表和动作表）。

$\#S_0x_1S_1x_2\cdots\cdots x_mS_m$



$S_0S_1\cdots\cdots S_m$
$\# x_1x_2\cdots\cdots x_m$

☆ 状态栈：

$S_0,S_1,\ldots,S_m$ 状态

S_0---初始状态　　S_m---栈顶状态

栈顶状态概括了从分析开始到该状态的全部分析历史.

☆ 符号栈：$X_1X_2\cdots\cdots X_m$

为从开始状态(S_0)到当前状态(S_m)所识别的规范句型的活前缀.

☆ 分析表

a. 状态转移表 (GOTO表)

是一个矩阵：
行---分析器的状态
列----文法符号

GOTO表

状态 \ 符号	E	T	F					
S_0								
S_1								
S_2								
:								
S_n								

GOTO表

$\#S_0x_1S_1x_2\cdots\cdots x_{i-1}S_{i-1}\ x_iS_i$

状态 \ 符号	E	T	F						
S_0									
S_1									
S_2									
:									
S_n									

$GOTO[S_{i-1}, x_i]=S_i$

S_{i-1} ---当前状态(栈顶状态)

x_i --- 新的栈顶符号

S_i ------ 新的栈顶状态(状态转移)

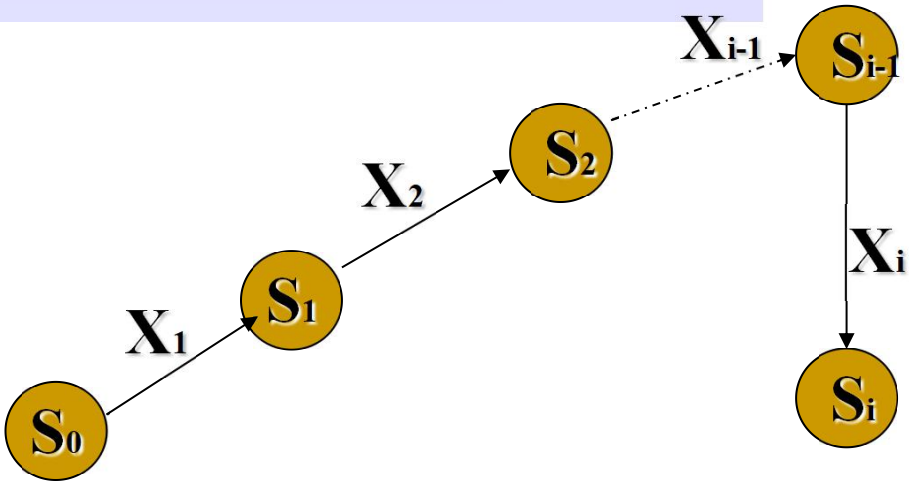
S_i需要满足条件是:

若$X_1X_2\cdots X_{i-1}$是由S_0到S_{i-1}所识别的规范句型的活前缀,则$X_1X_2\cdots X_i$是由S_0到S_i所识别的规范句型的活前缀

通过对有穷自动机的了解,我们可以看出:

状态转移函数GOTO是定义了一个以文法符号集为字母表的有穷自动机，该自动机识别文法所有规范句型的活前缀及可归前缀。

$M=(S, V, GOTO, S_0, Z)$



b. 分析动作表(ACTION表)

识别为活前缀的，则移进；识别为可归前缀的，则归约

ACTION表

输入符号a / 状态s	+	*	i	(	)	#
S_0						
S_1						
S_2						
:						
S_n						

$ACTION[S_i,a]$=动作分析　　$a \in V_t$

分析动作：

1)移进(shift)

$ACTION[S_i,a]=S$

动作：将a推进栈，并设置新的栈顶状态S_j

$S_j=GOTO[S_i,a]$，将指针指向下一个输入符号

2)规约(reduce)

$ACTION[S_i,a]=r_d$

d：文法规则编号　(d) $A\rightarrow\beta$

动作：将符号串β(假定长度为n)连同状态从栈内弹出把A推进栈，并设置新的栈顶状态S_j，$S_j=GOTO[S_{i-n},A]$

3)接受(accept)

$ACTION[S_i,\#]=accept$

4)出错(error)

$ACTION[S_i,a]=error$

☆ 控制程序:(Driver Routine)

1) 根据栈顶状态和现行输入符号，查分析动作表(ACTION表)，执行有分析表所规定的操作；

2) 并根据GOTO表设置新的栈顶状态(即实现状态转移)

（2） LR分析过程

例：文法G[E]

(1)E::=E+T

(2)E::=T

(3)T ::=T*F

(4)T::=F

(5)F::=(E)

(6)F::=i

该文法是SLR文法，故可以构造出SLR分析表(ACTION表和GOTO表)

ACTION表　　GOTO表

文法符号 / 状态	i	+	*	(	)	#	E	T	F
0(S_0)	S5			S4			1	2	3
1(S_1)		S6				ACCEPT			
2(S_2)		R2	S7		R2	R2			
3(S_3)		R4	R4		R4	R4			
4(S_4)	S5			S4			8	2	3
5(S_5)		R6	R6		R6	R6			
6(S_6)	S5			S4				9	3
7(S_7)	S5			S4					10
8(S_8)		S6			S11				
9(S_9)		R1	S7		R1	R1			
10(S_{10})		R3	R3		R3	R3			
11(S_{11})		R5	R5		R5	R5			

文法G[E]：(1)E::=E+T　(2)E::=T　(3)T ::=T*F
(4)T::=F　(5)F::=(E)　(6)F::=i

	ACTION 表						GOTO 表		
输入符号 状态	i	+	*	(	)	#	E	T	F
0	S5			S4			1	2	3
1		S6				ACCEPT			
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			

文法G[E]
(1)E::=E+T
(2)E::=T
(3)T ::=T*F
(4)T::=F
(5)F::=(E)
(6)F::=i

S_i：移入，i为状态
R_i：规约，i为产生式编号

分析过程： i*i+i 12*3+5

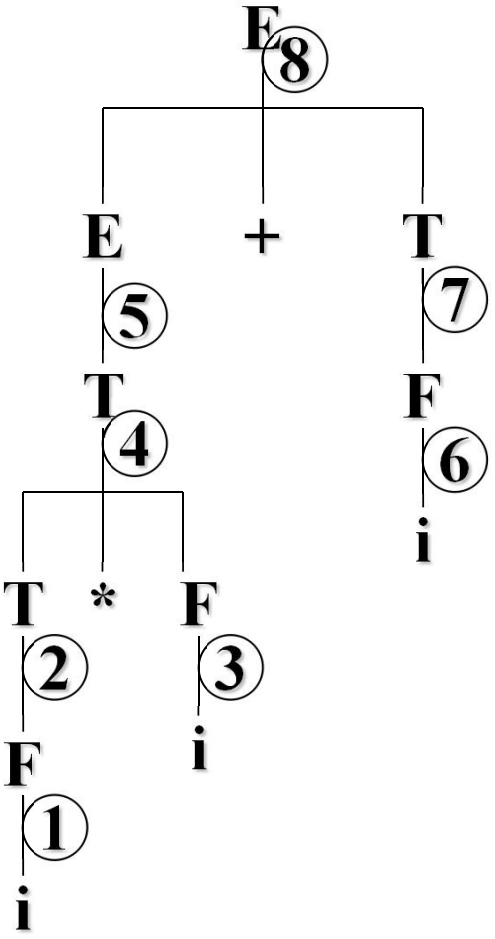
步骤	状态栈	符号	输入串	动作
1	#0	#	i*i+i#	初始化
2	#0i5	#i	*i+i#	S
3	#0F3	#F	*i+i#	R6
4	#0T2	#T	*i+i#	R4
5	#0T2*7	#T*	i+i#	S
6	#0T2*7i5	#T*i	+i#	S
7	#0T2*7F10	#T*F	+i#	R6

8	#0T2	#T	+i#	R3
9	#0E1	#E	+i#	R2
10	#0E1+6	#E+	i#	S
11	#0E1+6i5	#E+i	#	S
12	#0E1+6F3	#E+F	#	R6
13	#0E1+6T9	#E+T	#	R4
14	#0E1	#E	#	R1
15		#E		Accept

由分析过程可以看到:

(1) 每次规约总是规约**当前句型的句柄**，是规范规约，可以看到语法树(算符优先是规约最左素短语)

(2) 分析的每一步栈内符号串均是规范句型的活前缀，与输入串的剩余部分构成规范句型.



LR方法概述（回顾）

- LR分析法如何分析句子？
- 移进归约（从左到右扫描(L)自底向上进行规约(R)）
- 移进归约的关键问题是什么？
- 判断符号栈顶的符号串是否构成句柄。
- LR分析法如何识别句柄？
- LR分析法在分析过程中并不是直接分析符号栈中的符号是否形成句柄，而是通过识别可归前缀来识别句柄。
- 具体地，LR分析法的分析过程可以看作识别活前缀和可归前缀的过程，只要符号栈中的符号串构成活前缀，就表示已分析过的部分是正确的，继续移进；直到符号栈中的符号串构成可归前缀，则表示当前栈顶符号串已形成句柄，则进行归约。

- 如何识别活前缀和可归前缀？
- 通过有限自动机来识别。
- 如何构造识别活前缀和可归前缀的有限自动机？
- 根据文法规则
- 状态集合：列出所有活前缀的识别状态
- 符号表：所有的终结符和非终结符
- 状态转换函数： $f(K_i, a)= K_j$ 某一个活前缀的识别状态，在输入符号表中的一个符号之后，转向另一个活前缀的识别状态。
- 终态集：所有可归前缀的识别状态

LR(0)分析

- 构造LR分析器的关键是构造其分析表
- 构造LR分析表的方法是:
- （1）根据文法构造识别规范句型活前缀的有穷自动机DFA
- （2）由DFA构造LR分析表

1. 构造DFA

(1) DFA是一个五元式 M=(S, V, GOTO, S_0, Z)

S：有穷状态集

在此具体情况下，S=LR(0)项目集规范族LR(0)。

项目集规范族：其元素是由项目所构成的集合.

V：文法字汇表 $V_n \cup V_t$

S_0：初始状态

GOTO:状态转移函数　GOTO[S_i,X]=S_j

S_i,S_j∈S　S_i,S_j为项目集合　X∈$V_n \cup V_t$

表示当前状态S_i面临文法符号为X时，应将状态转移到S_j

Z：终态集合

∴构造DFA方法：

1） 确定S集合，即LR（0）项目集规范族，同时确定S_0

2） 确定状态转移函数GOTO

LR(0)分析

- 确定状态集合
- 每一个状态对应文法中某一个产生式规则的某一个项目
- 每一个产生式规则的每一个项目代表一个活前缀的识别状态。
- 产生式规则的项目？

活前缀与句柄的关系：

- G[S]:
- 若$S \overset{*}{\underset{R}{\Rightarrow}} \alpha A \omega \underset{R}{\Rightarrow} \alpha \beta \omega$ r是α β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

活前缀,与句柄 ,与 LR(0)项目

- 为刻划这种分析过程中的文法G的每一个产生式的右部符号已有多大一部分被识别（出现在栈顶）的情况，分别用标有圆点的产生式来指示位置。

  A→β．刻划产生式A→β的 右部β已出现在栈顶

  A→β_1．β_2 刻划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

  A→．β 刻划没有句柄的任何符号在栈顶，此时期望A→β的右部所推出的符号串

- 对于A→ε的LR(0)项目只有A→．

项目：文法G的每个产生式(规则)的右部添加一个圆点就构成一个项目。

例:产生式:A→XYZ

项　目: A→.XYZ

A→X.YZ

A→XY.Z

A→XYZ.

产生式:A→ε

项　目: A→.

项目的直观意义:
指明在分析过程中的某一时刻已经规约的部分和等待规约部分。

其中，ε、X、XY、XYZ为活前缀，XYZ是可归前缀。

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 ·**

项目类型：

项目类型有归约项目、移进项目、待约项目和接受项目。

① 归约项目：后继符号为空的项目称为归约项目。

如：$A \rightarrow \alpha \cdot$　此时已把α分析结束，α已在栈顶，从而可按相应的产生式进行归约。

② 移进项目：后继符号为终结符的项目称为移进项目。如 $A \rightarrow \alpha \cdot X\beta$，$X \in V_T$

此时把X移进，即X进符号栈。

③ 待约项目：后继符号为非终结符的项目，称为待约项目。

此时期待着从余留的输入符号中进行归约而得到X。

④ 接受项目：当归约项目为S’ → S · 时则表明已分析成功，即输入串为该文法的句子，相应状态为接受状态。

构造识别活前缀的NFA　P131

NFA的确定化——将NFA的一些状态组成集合构成DFA的状态

状态集合的闭包

LR(0)项目集规范族的构造

状态内容是项目集组成的，它分为基本(BASIC)部分和闭包(CLOSURE)部分。

※求BASIC和CLOSURE的方法如下：

设S_i是S_k关于符号X的后继状态，则有

BASIC(Si)的计算：

$$BASIC(S_i)=\{A\rightarrow\alpha X\cdot\beta \mid A\rightarrow\alpha\cdot X\beta\in S_k\}$$

CLOSURE(S_i)的计算：

① $BASIC(S_i)\subset CLOSURE(S_i)$

② 若$A \to \alpha \cdot Y\beta \in CLOSURE(S_i)$, 且$Y \in V_n$则

$Y \to \cdot r \in CLOSURE(S_i)$,r为符号串

③重复②直到$CLOSURE(S_i)$不再增加为止。

例如：文法G[S]：　$S \to A$

$A \to aAb$

$A \to c$

设开始状态为S_0，则S_0的状态内容为：

$BASIC(S_0)=\{S \to \cdot A\}$

$CLOSURE(S_0)=\{S \to \cdot A , A \to \cdot aAb, A \to \cdot c\}$

A. 项目集闭包closure的定义和计算:

令I是文法G'的任一项目集合，定义closure(I)为项目集合I的闭包，可用一个过程来定义并计算closure(I)。

Procedure closure(I);

begin

将属于I的项目加入closure(I);

repeat

for closure(I)中的每个项目$A\rightarrow\alpha.B\beta(B\in V_n)$ **do**

将$B\rightarrow.r(r\in V^*)$加入closure(I)

until closure(I)不再增大

end

例:G'[E'] 令I={E'→.E}

closure(I)={E'→.E, E→.E+T, E→.T, T→.T*F, T→.F,
F→.(E), F→.i }

B 状态转移函数GOTO的定义:

GOTO(I,X)=closure(J)

I：项目集合

X：文法符号，X∈V

J：项目集合

J={任何形如A→αX.β的项目| A→α.Xβ∈I}

closure(J):项目集J的闭包仍是项目集合

所以,GOTO(I,X)=closure(J)的直观意义是:

它规定了识别文法规范句型活前缀的DFA从状态I(项目集)出发,经过X弧所应该到达的状态(项目集合)

LR(0)项目集规范族的构造算法:

P133步骤（1）、（2）、（3）

```
G'→LR(0)
Procedure   ITEMSETS(G');
      begin
         LR(0):={closure({E'→.E})};
         repeat
            for  LR(0)中的每个项目集I和G'的每个符号X  do
               if  GOTO(I , X)非空,且不属于LR(0)
                  then   把GOTO(I , X)放入LR(0)中
            until   LR(0)不再增大
end
```

例.求G'[E']的LR(0)

V={E, T, F, I, +, *, (,)}

(0) E'→E (4) T→F

((1)) E→E+T (5) F→(E)

((2)) E→T (6) F→i

((3)) T→T*F

G'[E']共有20个项目

LR(0)={I_0,I_1,I_2,…I_{11}} 有12个项目组成:

I_0: E'→.E

E→.E+T

E→.T

T→.T*F

T→.F

F→.(E)

F→.i

Closure({B'→.B})=I_0

I_1: E'→E.

E→E.+T

GOTO(I_0,E)=closure({E'→E.

E→E.+T})

=I_1

I_2: $E \rightarrow T.$ $GOTO(I_0,T)=closure(\{E \rightarrow T.\ \ T \rightarrow T.*F\})= I_2$

$T \rightarrow T.*F$

I_3: $T \rightarrow F.$ $GOTO(I_0,F)=closure(\{T \rightarrow F.\})= I_3$

I_4: $F \rightarrow (.E)$ $GOTO(I_0,()=closure(\{F \rightarrow (.E)\})= I_4$

$E \rightarrow .E+T$

$E \rightarrow .T$

$T \rightarrow .T*F$

$T \rightarrow .F$

$F \rightarrow .(E)$

$F \rightarrow .i$

I_5: $F \rightarrow i.$ $GOTO(I_0,i)=closure(\{F \rightarrow i.\})= I_5$

$GOTO(I_0,*)=\varphi$

$GOTO(I_0,+)=\varphi$

$GOTO(I_0,))=\varphi$

I_0: $E' \rightarrow .E$

$E \rightarrow .E+T$

$E \rightarrow .T$

$T \rightarrow .T*F$

$T \rightarrow .F$

$F \rightarrow .(E)$

$F \rightarrow .i$

I_1: E'→E.
E→E.+T

I_2: E→T.
T→T.*F

I_6:
- E→E+.T
- T→.T*F
- T→.F
- F→.(E)
- F→.i

GOTO(I_1,+)=closure({E→E+.T})= I_6
GOTO(I_1,其他符号)为空

I_7:
- T→T*.F
- F→.(E)
- F→.i

GOTO(I_2,*)=closure({T→T*.F})= I_7
GOTO(I_2,其他符号)为空
GOTO(I_3,其他符号)为空

I_4: $F\rightarrow(.E)$ $E\rightarrow.E+T$
$E\rightarrow.T$ $T\rightarrow.T*F$
$T\rightarrow.F$ $F\rightarrow.(E)$ $F\rightarrow.i$

I_8: $F\rightarrow(E.)$, $E\rightarrow E.+T$

$GOTO(I_4,E)=closure(\{F\rightarrow(E.),E\rightarrow E.+T\})= I_8$

$GOTO(I_4,T)= I_2 \in LR(0)$

$GOTO(I_4,F)= I_3 \in LR(0)$

$GOTO(I_4,()= I_4 \in LR(0)$

$GOTO(I_4,i)= I_5 \in LR(0)$

$GOTO(I_4,+)=\varphi$

$GOTO(I_4,*)=\varphi$

$GOTO(I_4,))=\varphi$ $GOTO(I_5,$其他符号$)=\varphi$

I_9: $E\rightarrow E+T.$, $E\rightarrow T.*F$

$GOTO(I_6,T)=closure(\{E\rightarrow E+T.,T\rightarrow T.*F\})= I_9$

$GOTO(I_6,F)= I_3$

$GOTO(I_6,()= I_4$

$GOTO(I_6,i)= I_5$

I_{10}:T→T*F.　GOTO(I_7,T)=closure({T→T*F .})= I_{10}

GOTO(I_7,()= I_4

GOTO(I_7,i)= I_5

I_{11}:F→(E).　GOTO(I_8,))=closure({F→(E) .})= I_{11}

GOTO(I_7,()= I_4

GOTO(I_7,i)= I_5

求完所有I_i的后继

GOTO(I_8,+)= I_6

GOTO(I_9,*)= I_7

GOTO(I_{10},所有符号)=φ,　GOTO(I_{11},所有符号)=φ

构造DFA

$M=(S,\ V,\ GOTO,\ S_0,\ Z)$

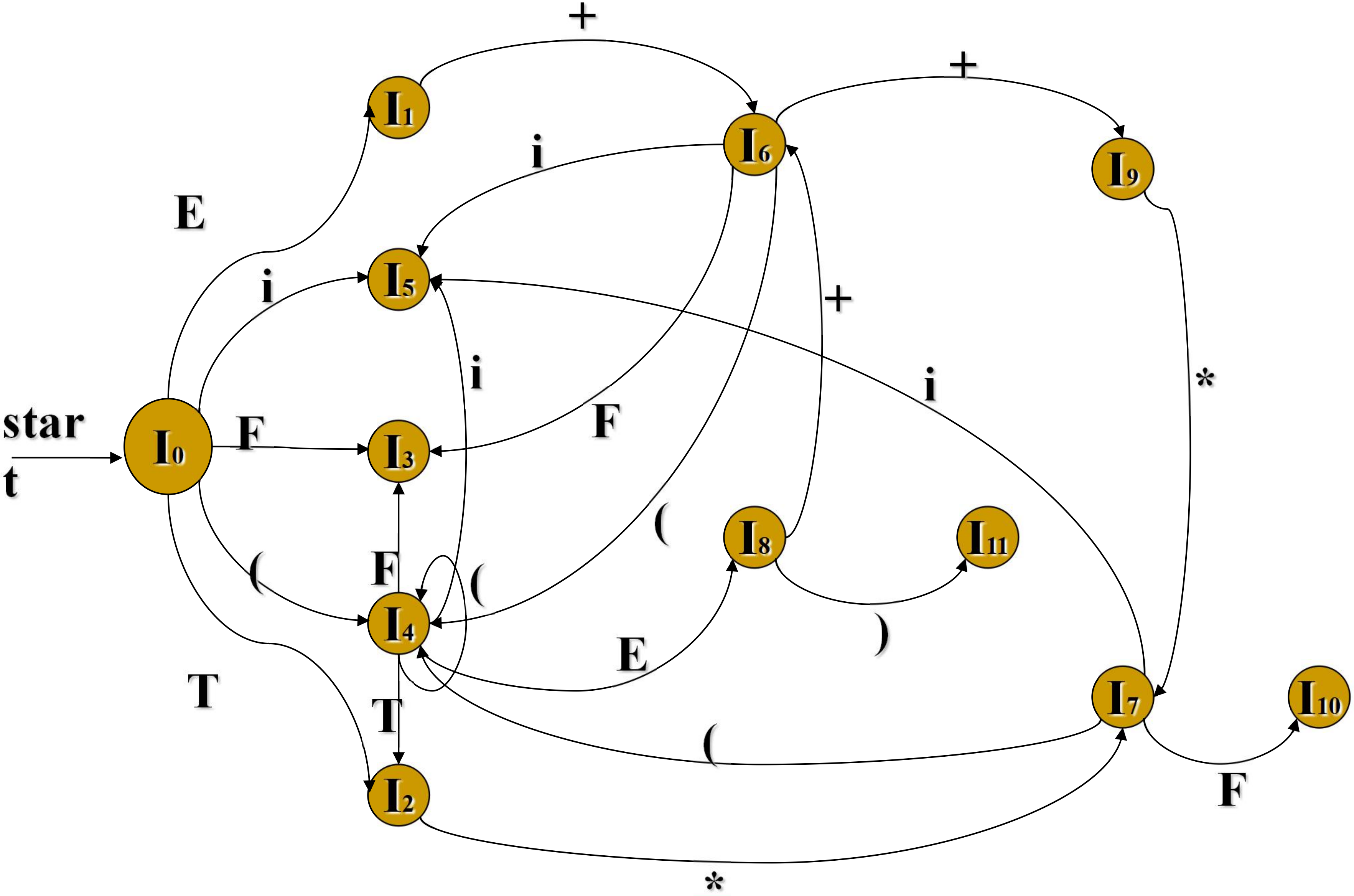
$S=\{I_0,\ I_1,\ I_2,\ \ldots,\ I_{11}\}=LR(0)$

$V=\{+,\ *,\ \ i,\ (,\ ),\ E,\ T,\ F\}$

$GOTO(I_m,\ X)=I_m$

$S_0=I_0$

$Z=S-\{I_0\}=\{I_1,\ I_2,\ \ldots,\ I_{11}\}$



star
t
I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
E
i
F
(
T
+
+
+
i
i
F
i
(
(
F
E
)
T
(
*
*
F

- 除I0以外,其余状态都是活前缀的识别状态,从I0到每一状态的每条路径都识别和接受一个规范句型的活前缀。

项目集的相容性：

〖定义〗满足下列两个条件的项目集称为相容的项目集。

※ 无移进项目和归约项目并存。

※ 无多个归约项目并存。

例如：若有项目集｛$A \rightarrow \alpha \cdot \beta$，$B \rightarrow \alpha \cdot$｝此时栈顶为α，根据项目集无法确定是移进还是归约。项目集是不相容的。

对一个文法的LR(0)项目集规范族不存在移进归约或归约归约冲突时，称该文法为LR(0)文法。

LR(0)分析表的构造

- LR（0）分析表由两部分组成：
- 动作表表示当前状态下面临输入符号应做的动作是移进、归约、接受或出错；
- 状态转换表表示在当前状态下面临文法符号时应转向的下一个状态。

(2) LR(0)分析表的构造

构造原则：

设有文法G[S]，则LR(0)分析表的构造规则为：

① 对于$A \rightarrow \alpha \cdot X\beta \in S_i$，$GOTO(S_i,X)=S_j$

若$X \in V_t$，则置$action[S_i,X]=S_j$

若$X \in V_n$，则置$goto[S_i,X]=j$

② 对于$A \rightarrow \alpha \cdot \in S_i$，若$A \rightarrow \alpha$是文法的第j个产生式，则对所有的$x \in V_t$，均置$action[S_i,x]=r_j$

③ 若$S \rightarrow \alpha \cdot \in S_i$，则置$action[S_i,\#]=acc$

④ 其他均置出错。

LR(0)分析表的构造

- 假定C={I_0, I_1,……，I_n}，令每个项目集I_k的下标k 为分析器的一个状态，因此，G` 的LR(0)分析表含有状态0，1，……，n。令那个含有项目S`→．S的I_k的下标k为初态。ACTION和GOTO可按如下方法构造：
- 若项目A→α．aβ 属于I_k且GO (I_k, a)= I_j, a为终结符，则置ACTION[k, a]为“把状态j和符号a移进栈”，简记为“sj”;
- 若项目A→α．属于I_k, 那么，对任何终结符a, 置ACTION[k, a]为“用产生式A→α 进行规约”，简记为“rj”;其中，假定A→α 为文法G`的第j个产生式;
- 若GO (I_k, A)= I_j, A为非终结符，则置GOTO(k, A)=j;
- 若项目S`→S．属于I_k, 则置ACTION[k, #]为“接受”，简记为“acc”;
- 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”

例子：文法G为：

(0) S’ →E

(1) E →aA

(2) E →bB

(3) A →cA

(4) A →d

(5) B →cB

(6) B →d

该文法的状态描述序列见下表：

状态	项目集	后继符号	后继状态
S_0	{S’ → ·E	E	S_1
	E→ ·aA	a	S_2
	E→ ·bB	b	S_3
S_1	{S’ →E · }	#S’ →E	S_{12}
S_2	{E→a ·A	A	S_6
	A → · cA	c	S_4
	A→ · d}	d	S_{10}
S_3	{E→b·B	B	S_7
	B→ · cB	c	S_5
	B→ · d}	d	S_{11}

状态	项目集	后继符号	后继状态
S_4	{A →c · A A→ · cA A→ · d}	A c d	S_8 S_4 S_{10}
S_5	{B→c · B B→ · cB B→ · d}	B c d	S_9 S_5 S_{11}
S_6	{E→aA · }	#E→aA	S_{12}
S_7	{E→bB · }	#E→bB	S_{12}
S_8	{A →cA · }	#A →cA	S_{12}

状态	项目集	后继符号	后继状态
S_9	{B→cB · }	#B→cB	S_{12}
S_{10}	{A→d · }	#A→d	S_{12}
S_{11}	{B→d · }	#B→d	S_{12}
S_{12}	{ }		

根据状态描述序列和分析表的构造规则得到的LR(0)分析表如下：

状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
S_0	S_2	S_3				1		
S_1					acc			
S_2			S_4	S_{10}			6	
S_3			S_5	S_{11}				7
S_4			S_4	S_{10}			8	
S_5			S_5	S_{11}				9
S_6	r_1	r_1	r_1	r_1	r_1			
S_7	r_2	r_2	r_2	r_2	r_2			
S_8	r_3	r_3	r_3	r_3	r_3			
S_9	r_5	r_5	r_5	r_5	r_5			
S_{10}	r_4	r_4	r_4	r_4	r_4			
S_{11}	r_6	r_6	r_6	r_6	r_6			

对于输入串#bccd#的分析过程如下：

状态栈	符号栈	产生式	输入符	action	goto	说明
S_0	#		bccd#	S_3		b和S_3进栈
S_0S_3	#b		ccd#	S_5		c和S_5进栈
$S_0S_3S_5$	#bc		cd#	S_5		c和S_5进栈
$S_0S_3S_5S_5$	#bcc		d#	S_{11}		d和S_{11}进栈
$S_0S_3S_5S_5S_{11}$	#bccd	$B \rightarrow d$	#	r6	9	d和S_{11}退栈 B和S_9进栈
$S_0S_3S_5S_5S_9$	#bccB	$B \rightarrow cB$	#	r5	9	…
$S_0S_3S_5S_9$	#bcB	$B \rightarrow cB$	#	r5	7	…
$S_0S_3S_7$	#bB	$E \rightarrow bB$	#	r2	1	…
S_0S_1	#E		#	acc		接受

三、SLR(1)分析表的构造

- 1、问题的提出：
- 只有当一个文法G是LR(0)文法，即G的每一个状态项目集相容（P134）时，才能构造出LR(0)分析表；
- 由于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，因此本节将介绍对于LR(0)规范族中冲突的项目集（状态）用向前查看一个符号的办法进得处理，以解决冲突。
- 因为只对有冲突的状态才向前查看一个符号，以确定做那种动作，因而称这种分析方法为简单的LR(1)分析法，用SLR(1)表示。

文法G为：

(0) $S' \rightarrow S$

((1)) $S \rightarrow rD$

((2)) $D \rightarrow D,i$

((3)) $D \rightarrow i$

状态描述序列见下表：

状态	项目集	后继符号	后继状态
S_0	$S' \rightarrow \cdot S$ $S \rightarrow \cdot rD$	S r	S_1 S_2
S_1	$S' \rightarrow S \cdot$	#$S' \rightarrow S$	S_7

S_2	$S \rightarrow r \cdot D$ $D \rightarrow \cdot D,i$ $D \rightarrow \cdot i$	D D i	S_3 S_3 S_4
S_3	$S \rightarrow rD \cdot$ $D \rightarrow D \cdot ,i$	#$S \rightarrow rD$,	S_7 S_5
S_4	$D \rightarrow i \cdot$	#$D \rightarrow i$	S_7
S_5	$D \rightarrow D, \cdot i$	i	S_6
S_6	$D \rightarrow D,i \cdot$	#$D \rightarrow D,i$	S_7
S_7	{ }		

分析每个状态包含的项目集，不难发现在状态S_3中含项目：

$S \rightarrow rD \cdot$　　为归约项目

$D \rightarrow D \cdot , i$　　为移进项目

也就是按$S \rightarrow rD \cdot$项目的动作认为用$S \rightarrow rD$产生式进行归约的句柄已形成，不管当前的输入符号是什么，都应把rD归约成S。但是按$D \rightarrow D \cdot ,i$项目当面临输入符为’,’号时，应将‘,’号移入符号栈，状态转向S_5。显然该文法不是LR(0)文法，S_3项目集不相容。也可在构造它的LR(0)分析表时发现这个问题，如下表所示。

状态	ACTION				GOTO	
	r	,	i	#	S	D
S_0	S_2				1	
S_1				acc		
S_2			S_4			3
S_3	r_1	r_1, S_5	r_1	r_1		
S_4	r_3	r_3	r_3	r_3		
S_5			S_6			
S_6	r_2	r_2	r_2	r_2		

- 如何解决这种移进-归约冲突？
- LR（0）在归约时不向前看输入符号；
- 在LR（0）基础上，如果出现不相容的项目（存在移进-归约冲突或归约-归约冲突）则LR（k）方法通过向前看k个输入符号来解决冲突（利用上下文信息来消除当前的歧义）
- SLR（1）：（1）只在项目出现冲突时S，（2）才向前看1个输入符号LR（1）；

SLR(1)分析表的构造方法思想

- 在出现移进-归约冲突或归约-归约冲突时，通过观察归约成的非终结符的后跟符号集合，来区分移进-归约动作或不同的归约动作。
- 上例冲突的解决
- 归约还是移进？
- 如果归约，那么要构成一个合法的句子，该非终结符后都可以跟哪些终结符？
- 而输入符号串中的当前符号是什么？
- 是否匹配？
- 匹配：则归约；不匹配：则不归约。

后跟符号集合的定义：

设$G=(V_T, V_N, P, S)$是上下文无关文法，$A \in V_N$，S是开始符号，

$Follow(A)=\{a \mid S \overset{*}{\Rightarrow} uA\beta$且$a \in V_T$，$a \in First(\beta)$，$u \in V_T^*$，$\beta \in V^+\}$。针对非终结符

若$S \overset{*}{\Rightarrow} uA\beta$，且$\beta \overset{*}{\Rightarrow} \varepsilon$，则$\# \in Follow(A)$

（#表示输入串的结束符，或句子括号）

可写成为：

$Follow(A)=\{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, \ a \in V_T\}$

若$S \overset{*}{\Rightarrow} \ldots A$，则$\# \in Follow(A)$。

Follow(A)是所有句型中出现在紧接A之后的终结符或“#”。

2、SLR(1)分析表的构造方法思想

在构造SLR(1)分析表时，根据不同的向前看符号，将S_i中的各项目所对应的动作加以区分，从而即可使冲突动作得到解决。

假定一个LR(0)规范族中含有如下的项目集（状态）S_i

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

也就是在该项目集中含有移进-归约冲突和归约-归约冲突。其中α,β, γ ,δ为文法符号串，b为终结符。方法如下：

对于归约项目$A\rightarrow\gamma\cdot$，$B\rightarrow\delta\cdot$ 分别求Follow(A)和Follow(B)，

Follow(A)是所有句型中出现在紧接A之后的终结符或“#”。

如果满足如下条件：

FOLLOW(A)∩FOLLOW(B)＝φ
FOLLOW(A)∩{b}＝φ
FOLLOW(B)∩{b}＝φ

那么，当在状态S_i时面临某输入符号为a时，则构造分析表时用以下方法即可解决冲突动作。

(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，报错。

- 如果对于一个文法的LR（0）项目集规范族所含有的动作冲突都能用以上方法来解决，则称该文法为SLR（1）文法。

3、SLR(1)分析表的构造

例如文法：

(0) S’ →E

(1) E→E+T

(2) E→T

(3) T→T*F

(4) T→F

(5) F→(E)

(6) F→i

状态描述序列如下：

状态	项目集	后继符号	后继状态
S_0	{ S'→ · E	E	S_1
	E→ · E+T	E	S_1
	E→ · T	T	S_2
	T → · T *F	T	S_2
	T → · F	F	S_3
	F→ · （E）	(	S_4
	F→ · i }	i	S_5
S_1	{ S'→E ·	#S'→E	S_{12}
	E→E · +T }	+	S_6

状态	项目集	后继符号	后继状态
S_2	{E→T· T→T·*F }	#E→T· *	S_{12} S_7
S_3	{T→F·}	#T→F	S_{12}
S_4	{F→（·E） E→·E+T E→·T T→·T*F T→·F F→·（E） F→·i }	E E T T F (i	S_8 S_8 S_2 S_2 S_3 S_4 S_5

状态	项目集	后继符号	后继状态
S_5	{F→ i · }	#F→ i	S_{12}
S_6	{E→E＋·T T→ · T*F T → · F F→·（E） F→· i }	T T F (i	S_9 S_9 S_3 S_4 S_5
S_7	{T→T* · F F→ ·（E） F→ · i }	F (i	S_{10} S_4 S_5

状态	项目集	后继符号	后继状态
S_8	{F→(E ·) E→E · +T }	) +	S_{11} S_6
S_9	{E→E+T · T→T · *F }	#E→E+T *	S_{12} S_7
S_{10}	{T→T*F · }	#T→T*F	S_{12}
S_{11}	{F→(E) · }	#F→(E)	S_{12}
S_{12}	{ }		

由上图可见，S_1、S_2和S_9的项目集均不相容，其有移进项目和归约项目并存，构造LR(0)分析表如下：

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
S_0	S_5			S_4			1	2	3
S_1		S_6				acc			
S_2	r_2	r_2	r_2 S_7	r_2	r_2	r_2			
S_3	r_4	r_4	r_4	r_4	r_4	r_4			
S_4	S_5			S_4			8	2	3
S_5	r_6	r_6	r_6	r_6	r_6	r_6			
S_6	S_5			S_4				9	3
S_7	S_5			S_4					10
S_8		S_6			S_{11}				
S_9	r_1	r_1	r_1 S_7	r_1	r_1	r_1			
S_{10}	r_3	r_3	r_3	r_3	r_3	r_3			
S_{11}	r_5	r_5	r_5	r_5	r_5	r_5			

从上表也可见在S_1, S_2, S_9中存在移进-归约冲突。这个表达式不是LR(0)文法，也就不能构造LR(0)分析表，现在分别考查这三个项目（状态）中的冲突是否能用SLR(1)方法解决。

对于S1：{S'→E·，E→E·＋T}

由于Follow(S')={＃}，而S'→·E是唯一的接受项目，所以当且仅当遇到句子的结束符“＃”号时才被接受。又因{＃}∩{＋}=ϕ，因此S_1中的冲突可解决。

对于S_2：S_2={E→T·，T→T·*F}

计算Follow(E) = {#, +,)}

所以Follow(E)∩{*}=ϕ

因此面临输入符为‘+’，‘)’或‘#’号时，则用产生式E→T进行归约。

当面临输入符为‘*’号时，则移进, 其它情况则报错。

对于S_9：S_9={ E →E+T ·, T→T · *F }

计算Follow(E) = { #, +,) } , 所以Follow(E)∩{*}=ϕ

因此面临输入符为‘+’，‘)’或‘#’号时，则用产生式E→E+T进行归约。

当面临输入符为‘*’号时，则移进。 其它情况则报错。

由以上考查，该文法在S_1，S_2，S_9三个项目集(状态)中存在的移进-归约冲突都可以用SLR(1)方法解决，因此该文法是SLR(1)文法。我们可构造其相应的SLR(1)分析表。

SLR(1)分析表的构造与LR(0)分析表的构造类似，仅在含有冲突的项目集中分别进行处理。

进一步分析我们可以发现如下事实：例如在状态S_3中，只有一个归约项目$T \rightarrow F \cdot$ ，按照SLR(1)方法，在该项目中没有冲突，所以保持原来LR(0)的处理方法，不论当前面临的输入符号是什么都将用产生式$T \rightarrow F$进行归约。

但是很显然T的后跟符没有‘（’符号，如果当前面临输入符是‘（’，也进行归约显然是错误的。因此我们对所有归约项目都采取SLR(1)的思想，即对所有非终结符都求出其Follow集合，这样凡是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。

对于这样构造的SLR(1)分析表我们称它为改进的SLR(1)分析表。

改进的SLR(1)分析表的构造方法如下：

((1))对于$A\rightarrow\alpha\cdot X\beta$，$GO[S_i, X]\in S_j$，若$X\in V_i$，则置$actoin[S_i, X]=S_j$

若$X\in V_n$，则置$goto[S_i, X]=j$

(2) 对于归约项目$A\rightarrow\alpha\cdot\in S_i$，若$A\rightarrow\alpha$为文法的第j个产生式，则对任何输入符号a，若$a\in Follow(A)$，则置$action[S_i, a]=r_j$

(3) 若$S\rightarrow\alpha\cdot\in S_i$，则置$action[S_i, \#]=acc$

(4) 其它情况均置出错。

改进的SLR(1)分析表如下：

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
S_0	S_5			S_4			1	2	3
S_1		S_6				acc			
S_2		r_2	S_7		r_2	r_2			
S_3		r_4	r_4		r_4	r_4			
S_4	S_5			S_4			8	2	3
S_5		r_6	r_6		r_6	r_6			
S_6	S_5			S_4				9	3
S_7	S_5			S_4					10
S_8		S_6			S_{11}				
S_9		r_1	S_7		r_1	r_1			
S_{10}		r_3	r_3		r_3	r_3			
S_{11}		r_5	r_5		r_5	r_5			

四、LR(1)分析表的构造

1、问题的提出

在SLR(1)方法中，对于某状态S_i，其项目集若不相容时，可根据SLR(1)分析表的构造规则来解决冲突分析动作，但如果不相容的项目集中的向前看集合及其有关集合相交时，就不可能通过SLR(1)分析表构造规则来构造SLR(1)分析表。这时就用LR(1)分析。

SLR

- 例：文法G:
- （0）S`→S　　(1) S→aAd　　(2) S→bAc
- (3) S→aec　　(4) S→bed　　(5) A→e

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:
S' → S •

I_2:
S → a •Ad
S → a • ec
A → • e

I_3:
S → b • Ac
S → b • ed
A → • e

I_4:
S → aA • d

I_5:
S → ae • c
A → e •

I_6:
S → bA • c

I_7:
S → be • d
A → e •

I_8:
S → aAd •

I_9:
S → ae c •

I_{10}:
S → bAc •

I_{11}:
S → bed •

查看I_5，I_7中的冲突，
体会LR(1)如何解决

- I_5:　S →ae．c
- 　　A →e.
- **S`**==>S==>aAd==>aed
- 　S`==>S==>aec
- 活前缀ae遇到c应移进；遇到d应归约
- 　I_7:　S →be．d
- 　　A →e.
- 　S`==>S==>bAc==>bec
- 　S`==>S==>bed

- 活前缀be遇到d应移进；遇到c应归约

- 并不是Follow（A）的每个元素在含A的所有句型中都会在A的后面出现

LR（1）方法

- 在每个项目中增加向前搜索符
- 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)项目集族的构造：针对初始项目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)， B→γ是文法的产生式，β∈V*，b∈FIRST(βa),则B→·γ,b也属于CLOSURE(I)

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| A→α·Xβ,a属于I}

一个文法符号串的first集合计算方法：

如果文法符号串$\alpha \in V^*$, $\alpha = X_1X_2 \ldots X_n$,

1、当$X_1 \overset{*}{\Rightarrow} \varepsilon$，则$first(\alpha) = first(X_1)$

2、当对任何j（$1 \le j \le i-1$，$2 \le i \le n$），$\varepsilon \in first(X_j)$

则$first(\alpha) = (first(X_1) - \{\varepsilon\}) \cup (first(X_2) - \{\varepsilon\}) \cup \ldots \cup (first(X_{i-1}) - \{\varepsilon\}) \cup first(X_i)$

3、当$first(X_j)$都含有ε时$(1 \le j \le n)$，则

$first(\alpha) = first(X_1) \cup first(X_2) \cup \ldots \cup first(X_j) \cup \{\varepsilon\}$

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:
S' → S •, #

I_2:
S → a •Ad, #
S → a • ec, #
A → • e, d

I_3:
S → b • Ac, #
S → b • ed, #
A → • e, c

I_4:
S → aA • d, #

I_5:
S → ae • c, #
A → e •, d

I_6:
S → bA • c, #

I_7:
S → be • d, #
A → e •, c

I_8:
S → aAd •, #

I_9:
S → ae c •, #

I_{10}:
S → bAc •, #

I_{11}:
S → bed •, #

查看I_5, I_7中的冲突，
体会LR(1)如何解决

LR(1)分析表的构造

- LR(1)分析表的构造与LR(0)分析表的构造在形式上基本相同，不同之处在于：归约项目的归约动作取决于该归约项目的向前搜索符号集。

1) 若项目[$A\rightarrow\alpha\cdot a\beta,b$]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$S_j$

2) 若项目[$A\rightarrow\alpha\cdot,a$]属于$I_k$，则对a为任何终结符或‘#’，置$ACTION[k,a]=r_j$，j为产生式在文法G'中的编号

3) 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

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

5) 其它填上“报错标志”

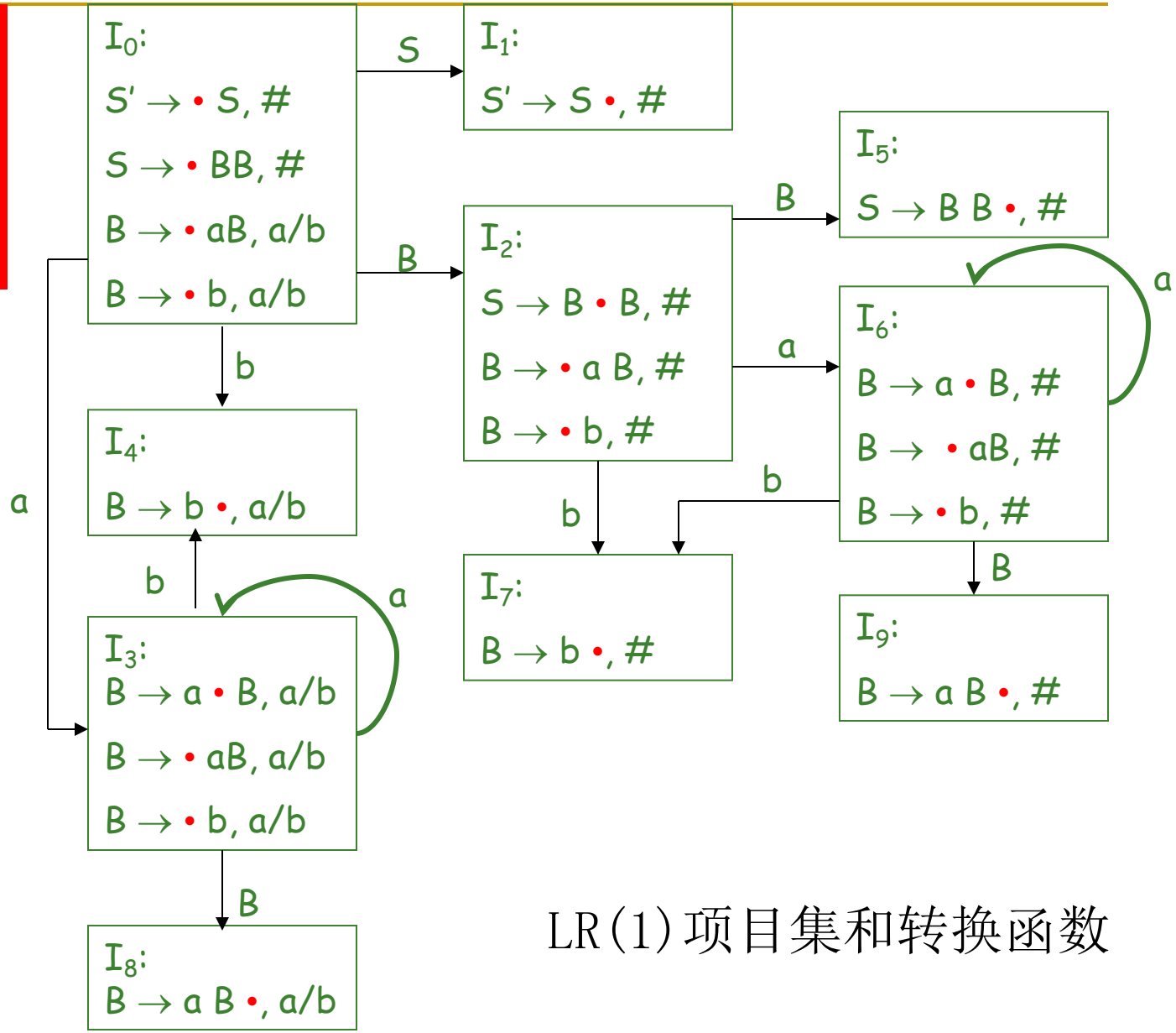
例：P146 表7.10

LALR(1)分析

文法*G'*：

(0) $S' \rightarrow S$
(1) $B \rightarrow aB$
(2) $S \rightarrow BB$
(3) $B \rightarrow b$

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



LR(1)项目集和转换函数

如果两个LR(1)项目集去掉搜索符之后是相同的，
则称这两个项目集具有相同的心。

分析可发现I_3和I_6， I_4和I_7， I_8和I_9分别为同心集

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

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

合并为 →

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

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

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

合并为 →

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

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

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

合并为 →

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

LALR(1)分析

- 对LR(1)项目集规范族合并同心集，若合并同心集后不产生新的冲突，则为LALR(1)项目集。

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		

G’[S’]: (0) S’→S

(1) S → L=R

(2) S → R

(3) L → *R

(4) L → i

(5) R → L

判断该文法是否是LR(0)、SLR(1)、LR(1)、LALR(1)文法。

- 构造LR（0）项目集规范族
- If 所有的项目集都是相容的，则为LR（0）文法；
- Else if 冲突项目可以通过考察非终结符的后跟符号集来解决，则为SLR（1）文法；
- Else 构造LR（1）项目集规范族
- If 任何项目集中都不存在动作冲突，则为LR（1）文法；
- 对LR（1）项目集规范族进行同心集的合并，如合并之后仍不存在冲突，则为LALR（1）文法。

几种文法的比较

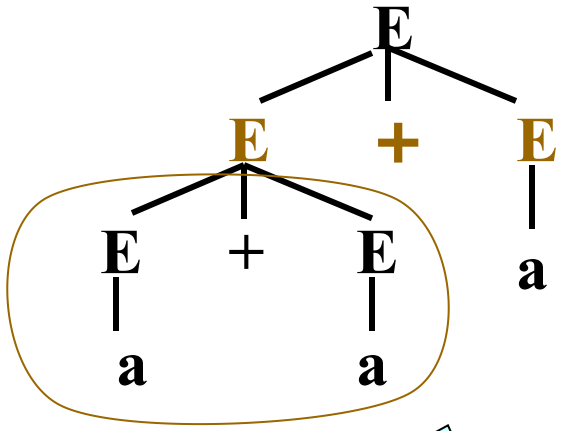
- LR(0)
- SLR(1): 生成的LR(0)项目集如有冲突，则根据非终结符的FOLLOW集决定
- LR(1)、LR(k): 项由 核心与向前搜索符组成，搜索符长度为1或k
- LALR(1): 对LR(1)项目集规范族合并同心集
- **由弱到强：LR（0）、SLR（1）、LALR（1）、LR（1）**
- **LR（1）中的向前搜索符号集合是与该项目相关的非终结符号的Follow集的子集；**
- **LALR项目的搜索符一般是与该项目相关的非终结符号的Follow集的子集，这正是LALR分析法比SLR分析法强的原因。**

7.6 二义性文法的应用

任何二义性文法都不是LR(K)的。例 G(E):

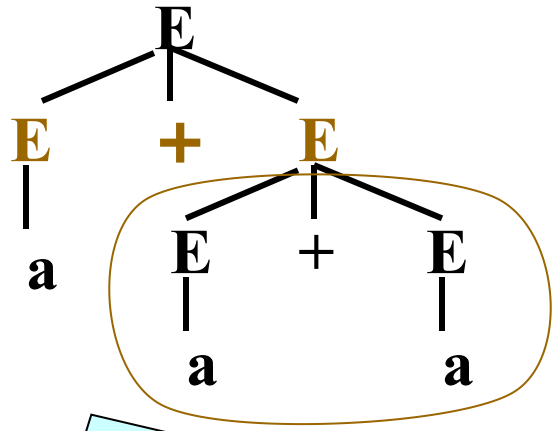
1) E→E+E
2) E→E*E
3) E→a

1、对 a+a+a 可以有下面的两棵不同的语法树



a+a+a → (a+a)+a

$$a+a+a \overset{r}{\Rightarrow} E+a+a \overset{r}{\Rightarrow} E+E+a \overset{r}{\Rightarrow} E+a$$



a+a+a → a+(a+a)

$$a+a+a \overset{r}{\Rightarrow} a+a+a \overset{r}{\Rightarrow} E+E+a \overset{r}{\Rightarrow} E+E+E \overset{r}{\Rightarrow} E+E$$

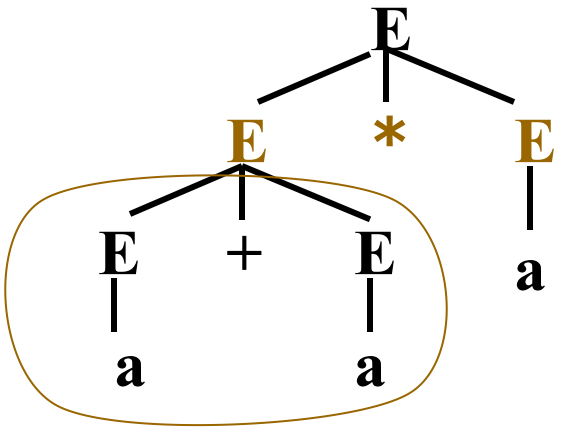
任何二义性文法都不是LR(K)的。例 G(E)：

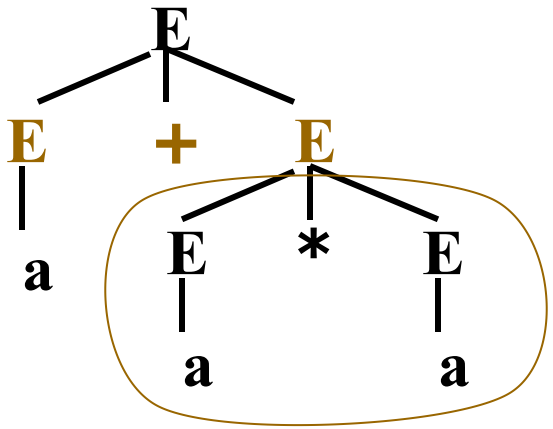
1）E→E+E

2）E→E*E

3）E→a

1、对 a+a*a 也可以有下面的两棵不同的语法树





a+a*a → (a+a)*a

$a+a*a \overset{r}{\Rightarrow} E+a*a \overset{r}{\Rightarrow} E+E*a \overset{r}{\Rightarrow} E*a$ …

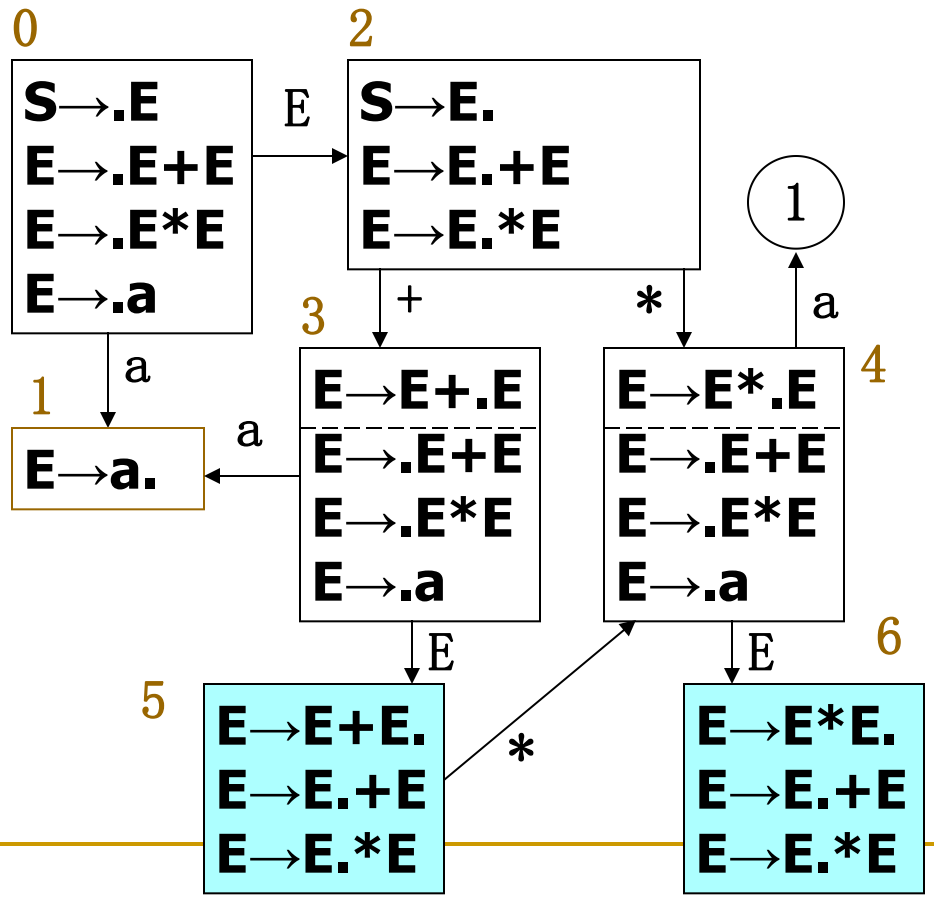
a+a*a → a+(a*a)

$a+a*a \overset{r}{\Rightarrow} a+a*a \overset{r}{\Rightarrow} E+E*a \overset{r}{\Rightarrow} E+E*E \overset{r}{\Rightarrow} E+E$ …

二义性文法的应用

例 G(S):

0) S→E
1) E→E+E
2) E→E*E
3）E→a



Action表 | Goto表

	a	+	*	#	S	E
0	S1					2
1	r3	r3	r3	r3		
2		S3	S4	acc		
3	S1					5
4	S1					6
5		r1	S4	r1		
6		r2	r2	r2		

例：分析输入串 a+a*a#

0) S→E
1) E→E+E
2) E→E*E
3）E→a

输入串 a+a*a# 的分析过程

状态	分析栈	输入流	动作
0	#	a+a*a#	S1
01	#a	+a*a#	r3
02	#E	+a*a#	S3
023	#E+	a*a#	S1
0231	#E+a	*a#	r3
0235	#E+E	*a#	S4
02354	#E+E*	a#	S1
023541	#E+E*a	#	r3
023546	#E+E*E	#	r2
0235	#E+E	#	r1
02	#E	#	acc

	Action表				Goto表	
	a	+	*	#	S	E
0	S1					2
1	r3	r3	r3	r3		
2		S3	S4	acc		
3	S1					5
4	S1					6
5		r1	S4	r1		
6		r2	r2	r2		

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

对输入串abbcde#的LR分析过程

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

S_i:移进，将状态i和**输入符**进栈

r_i:归约，用第i个产生式归约，同时状态栈与符号栈退出相应个符号，并把GOTO表相应状态和第i个产生式的左部非终结符入栈。

	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			