编译原理

第一章　编译程序概述
第二章　PL/0编译程序的实现
第三章　文法和语言
第四章　词法分析
第五章　自顶向下语法分析方法
第六章　自底向上优先分析方法
第七章　LR分析方法
第八章　语法制导翻译和中间代码生成
第九章　符号表
第一〇章　代码优化
第一一章　代码生成

- 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分析法与简单优先分析一样，也是通过求句柄逐步归约进行语法分析。
- 在简单优先分析中，句柄是通过运算符的优先关系而求得，LR方法中句柄是通过求可归前缀而求得。

LR分析

- **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。**
- L：从左到右扫描（left-to-right scanning）
- R：最右推导、规范推导(the rightmost derivation in reverse)
- k：向前看的符号数目(the number of unconsumed “look ahead” input symbols that are used in making parser decisions.)

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。

第七章 LR分析法

一、LR分析概述（基本构造原理与方法）

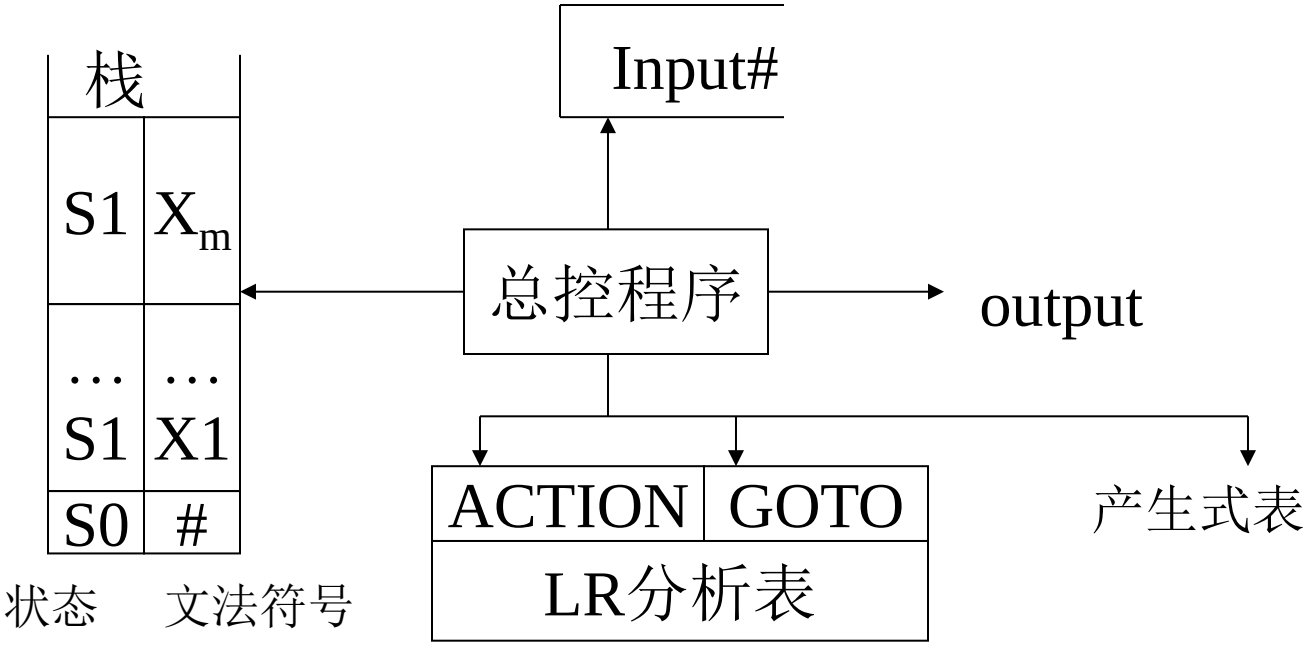
二、LR(0)分析

三、SLR(1)分析

四、LR(1)分析

五、LALR (1)分析

LR分析器模型



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。

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。

分析表的组成：

(1) 分析动作表Action

action[S_i,a_j]指出当前栈顶为状态S_i，输入符号为a_j时所执行的动作，

有四种可能:移进(s)、归约(r)、接受(acc)、出错(error)。

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]

(2) 状态转换表goto

状态 \ 符号	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]

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

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

			ION			
			,	#	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		

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。

(1) L →E,L

(2) L →E

(3) E →a

(4) E →b

输入串为：#a,b,a#

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$\alpha \beta$的前缀，则称是文法G的活前缀
- $\alpha \beta$是含句柄的活前缀，并且句柄是$\alpha \beta$的后端，则称$\alpha \beta$是可归前缀或可规范前缀。
- 在LR分析过程中，实际上是把$\alpha \beta$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$\alpha \beta$（形成可归前缀），即句柄已经形成，则用产生式$A \rightarrow \beta$进行归约。

识别活前缀的有限自动机

- P127 识别活前缀的有限自动机(图7.2 – 7.4)
  - 把终结符和非终结符看作有限自动机的输入符号，进行状态转换，到可归前缀，到达识别句柄的终态。
- 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）。

可归前缀：

$S[0]$

$ab[2]$

$aAb[3]$

$aAcd[4]$

$aAcBe[1]$

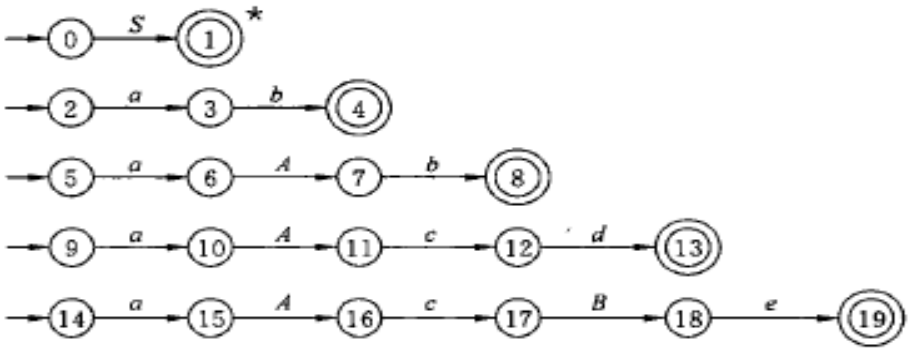


图 7.2　识别活前缀及可归前缀的有限自动机

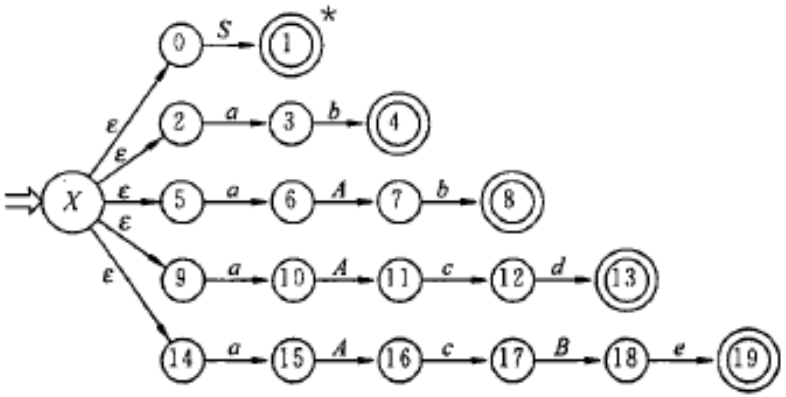


图 7.3　识别活前缀的不确定有限自动机

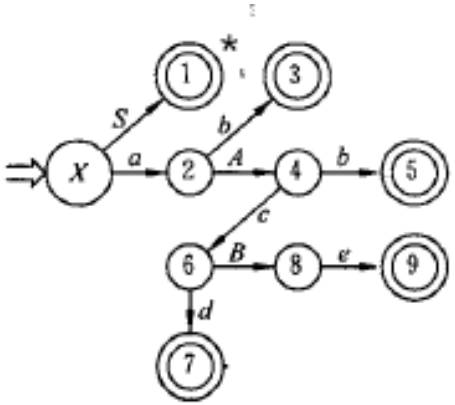
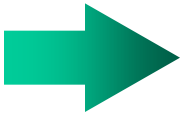


图 7.4　识别活前缀的确定有限自动机

$\#S_0x_1S_1x_2\cdots\cdots x_mS_m$



$S_0S_1\cdots\cdots S_m$

$\# x_1x_2\cdots\cdots x_m$

☆ 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.

状态转移函数GO

$\#S_0X_1S_1X_2\cdots\cdots X_{i-1}S_{i-1}\,X_iS_i$

$GO[S_{i-1}, X_i]=S_i$

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)

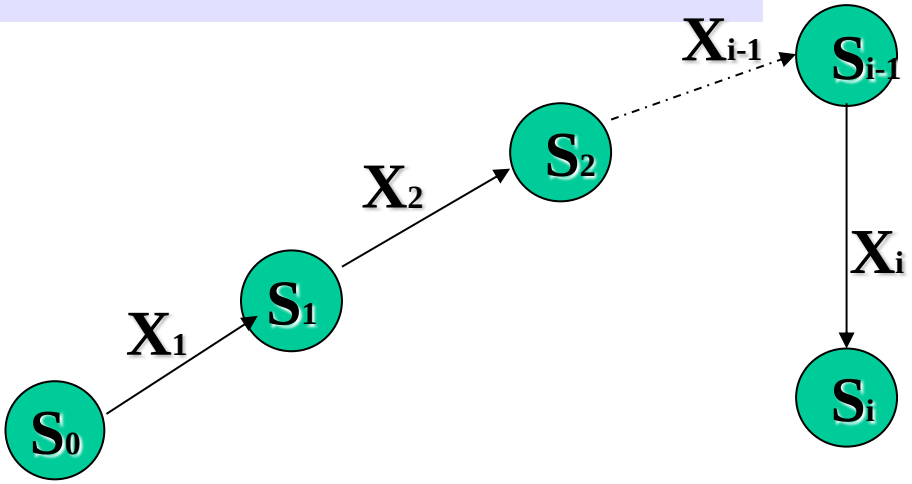
S_i需要满足条件是:

若$X_1X_2\ldots X_{i-1}$是由S_0到S_{i-1}所识别的规范句型的活前缀,则$X_1X_2\ldots X_i$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GO是定义了一个以文法符号集为字母表的有限自动机，该自动机识别文法所有规范句型的活前缀及可归前缀。

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



a. 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　　$a \in V_t$

b. 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								

☆ 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)

分析动作：

1)移进(shift)

$ACTION[S_i,a]=S_j$

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)

$ACTION[S_i,a]=error$

（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		
状态 \ 输入符号	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			

S_i：移入，i为状态
R_i：规约，i为产生式编号

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

分析过程：

i*i+i

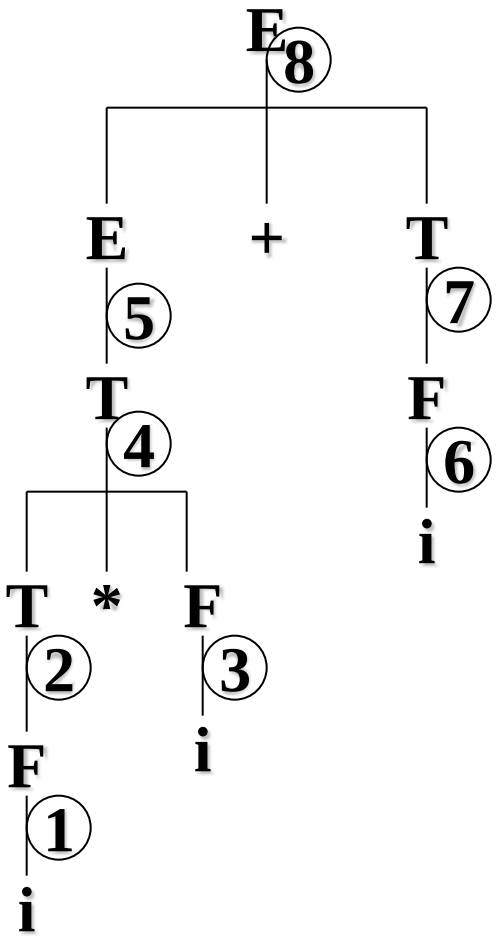
步骤	状态栈	符号	输入串	动作
1	＃0	＃	i*i+i#	初始化
2	＃0i5	＃i	*i+i#	S
3	＃0F3	＃F	*i+i#	R6
4	＃0T2	＃T	*i+i#	R4
5	＃0T2*7	＃T*	i+i#	S
6	＃0T2*7i5	＃T*i	+i#	S
7	＃0T2*7F10	＃T*F	+i#	R6

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

- 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？
  - 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

GO:状态转移函数 $GO[S_i,X]=S_j$

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

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GO

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？

活前缀与产生式的关系：

- **G[S]:**
  若S $\overset{*}{\underset{R}{\Rightarrow}}$ **αAω** $\underset{R}{\Rightarrow}$ **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**

活前缀,与句柄 ,与 LR(0)项目

- 为刻划这种分析过程中，文法G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$A \rightarrow \alpha \ a\beta$，$a \in V_t$，此时把a移进，即a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　P132

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

文法：

$S' \rightarrow E$

$E \rightarrow aA \mid bB$

$A \rightarrow cA \mid d$

$B \rightarrow cB \mid d$

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

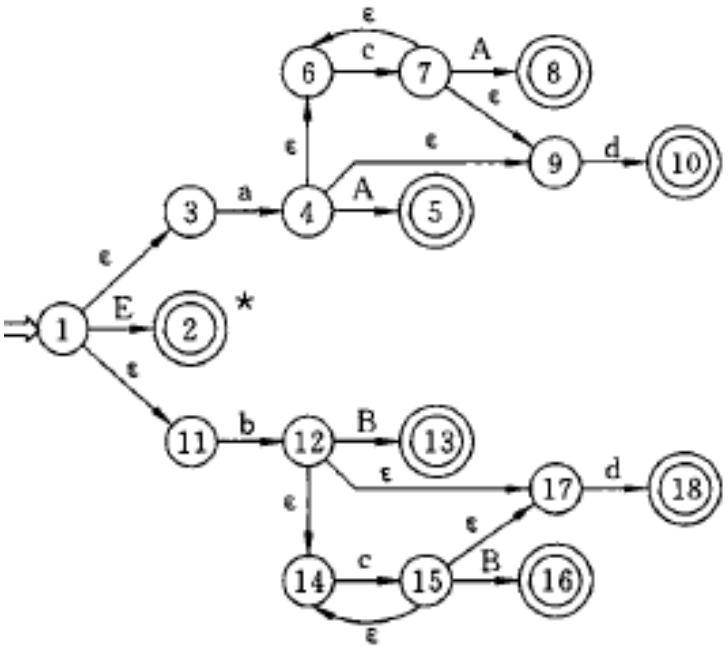


图 7.7　识别活前缀的 NFA

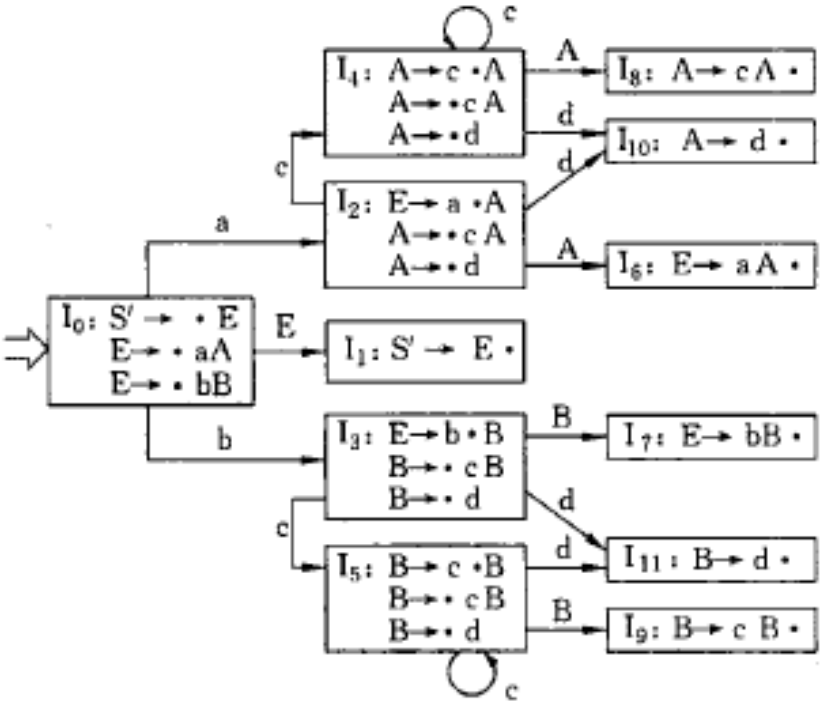


图 7.8　识别活前缀的有限自动机 DFA

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(S_i)的计算：

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

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GO的定义:

GO(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,GO(I,X)=closure(J)的直观意义是:

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 Closure({E'→.E})=I_0

E→.E+T

E→.T

T→.T*F

T→.F

F→.(E)

F→.i

I_1: E'→E. GOTO(I_0,E)=closure({E'→E.

E→E.+T E→E.+T})

=I_1

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

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

I_4: $F \rightarrow (.E)$ GOTO(I_0,()=closure({F→(.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→i.})= I_5

GOTO(I_0,*)=φ

GOTO(I_0,+)=φ

GOTO(I_0,))=φ

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

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

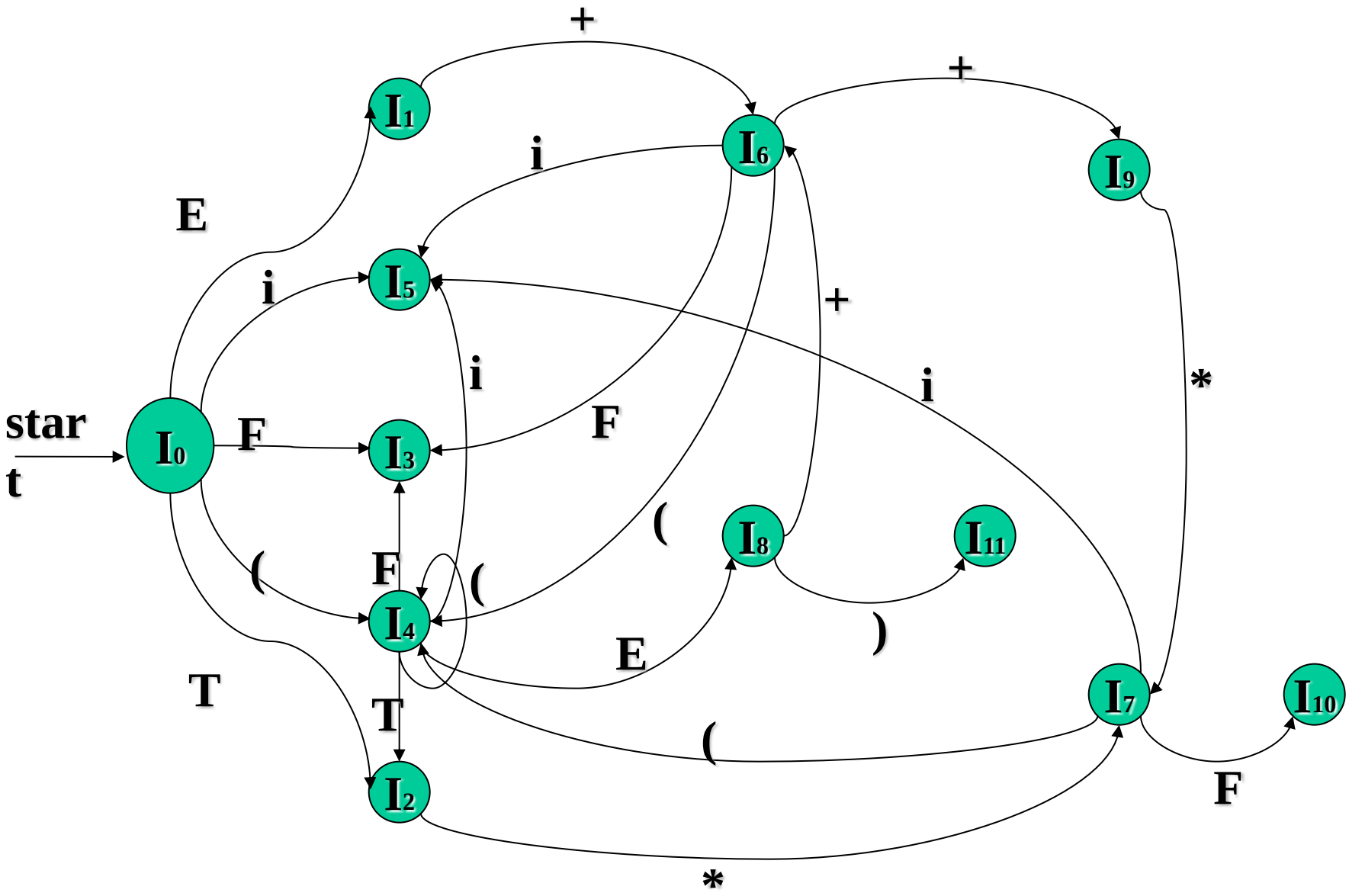
$S=\{I_0,\ I_1,\ I_2,\ \ldots,\ I_{11}\}=LR(0)$

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

$GOTO(I_m,\ X)=I_n$

$S_0=I_0$

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


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

- 除I_0以外,其余状态都是活前缀的识别状态,从I_0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。 **P134**

※ 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 →α β，B →α ·**｝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$A \to \alpha \cdot X\beta \in S_i$，$GO(S_i,X)=S_j$

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

④ 其他均置出错。

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→ ·aA E→ ·bB	E a b	S_1 S_2 S_3
S_1	{S’ →E ·}	#S’ →E	S_{12}
S_2	{E→a ·A A → ·cA A→ ·d}	A c d	S_6 S_4 S_{10}
S_3	{E→b ·B B→ ·cB B→ ·d}	B c d	S_7 S_5 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_7	{E→bB ·}	#E→bB	
S_8	{A →cA ·}	#A →cA	

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

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

作业

- P166　2、3(1,3,4)、9