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

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

活前缀与可归前缀

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]

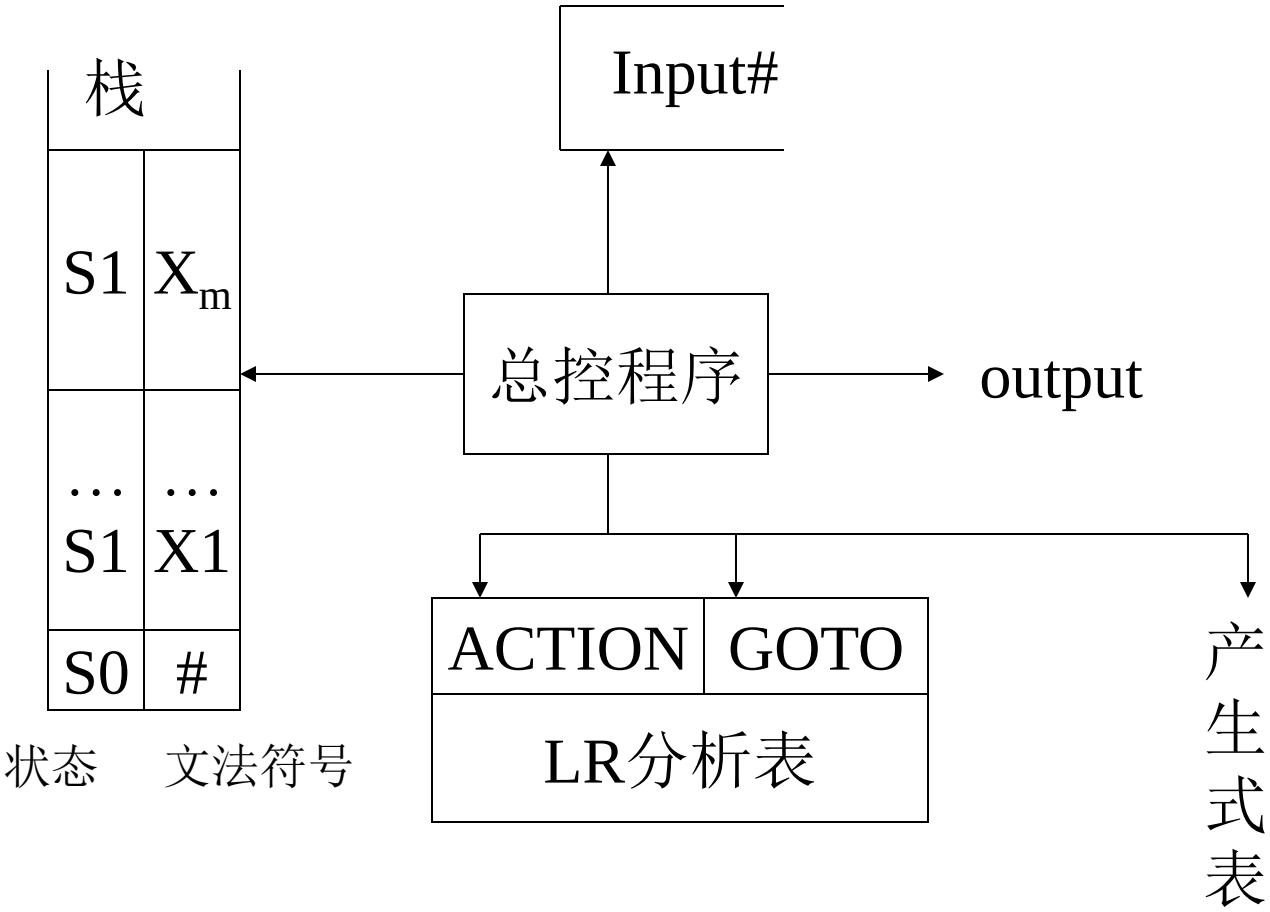
B →d[4]

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。

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。

LR分析器模型


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

☆ 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)

$ACTION[S_i,a]=error$

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			

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

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？

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

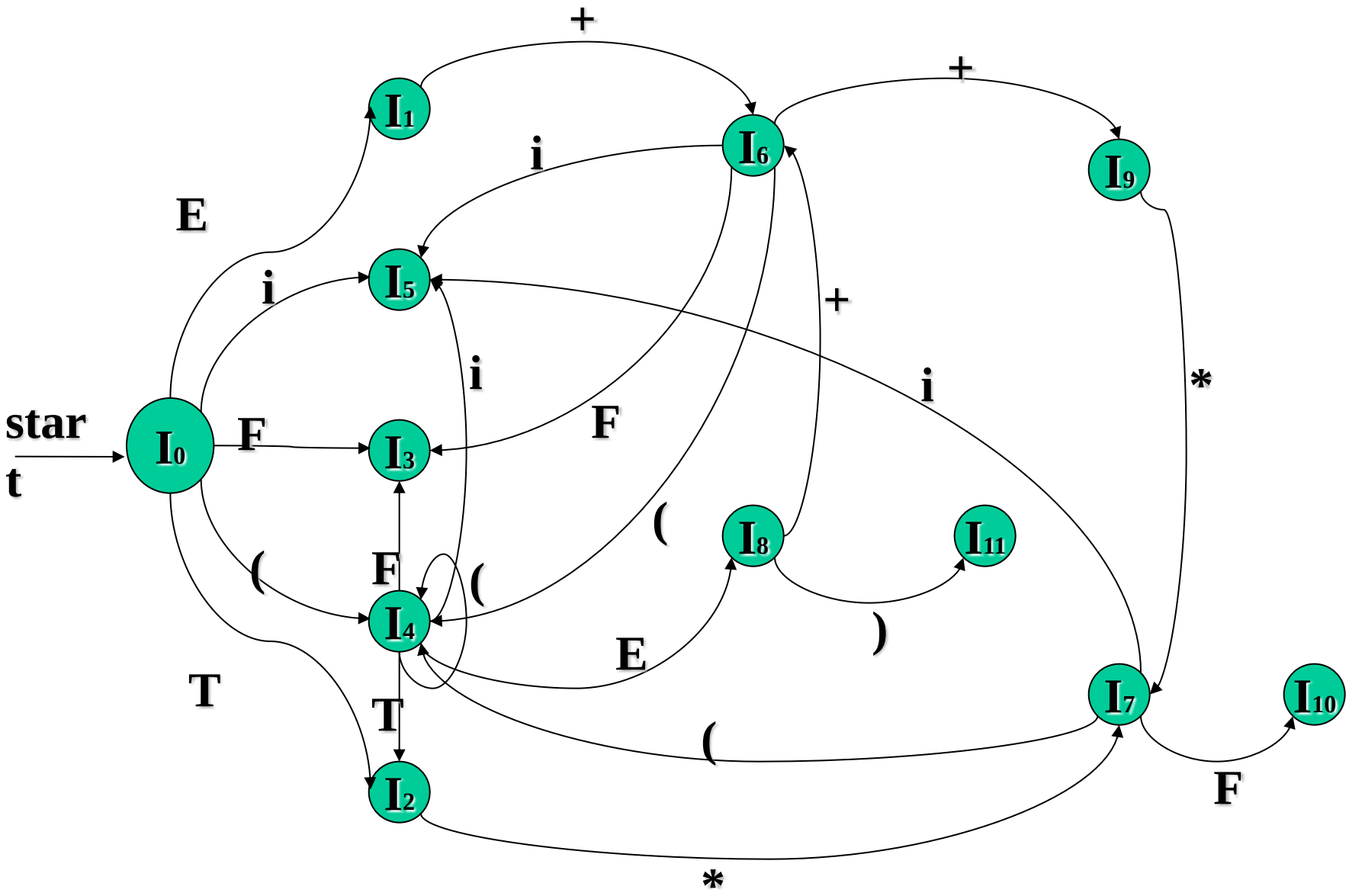
F→.(E)

F→.i

I_1: E'→E. GOTO(I_0,E)=closure({E'→E.

E→E.+T E→E.+T})

=I_1


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

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｛$A \to \alpha\ \beta$，$B \to \alpha\ \cdot$｝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

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

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

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（表7.3）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行处理，以解决冲突。
- 只对有冲突的状态才向前查看一个符号，以确定做哪种动作，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’ → S

(1) S → rD

(2) D →D,i

(3) D →i

输入串： ri,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’ → ·S S → ·rD	S r	S_1 S_2
S_1	S’ → S ·	#S’ → 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 → rD ,	S_7 S_5
S_4	$D \rightarrow i \cdot$	#D→i	S_7
S_5	$D \rightarrow D, \cdot i$	i	S_6
S_6	$D \rightarrow D,i \cdot$	#D →D,i	S_7
S_7	{ }		

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

- $S \rightarrow rD \cdot$项目：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。
- $D \rightarrow D \cdot, i$ 项目：面临输入符为’,’号时，应将‘,’号移入符号栈，状态转向S_5
- 该文法不是LR(0)文法，S_3项目集不相容。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→α ·bβ, A → γ ·, B→δ· }

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→ γ ·， B → δ ·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	S_9
	T→ · T*F	T	S_9
	T → ·F	F	S_3
	F→·（E）	(	S_4
	F→· i }	i	S_5
S_7	{T→T* · F	F	S_{10}
	F→ ·（E）	(	S_4
	F→ · i }	i	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S_1：{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→F·，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→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$A \to \alpha \cdot X\beta$，$goto[S_i, X] \in S_j$，若$X \in V_t$，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$A \to \alpha \cdot \in S_i$，若$A \to \alpha$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$S \to \alpha \cdot \in S_i$，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`$\underset{R}{\Longrightarrow}S\underset{R}{\Longrightarrow}aAd\underset{R}{\Longrightarrow}$aed
- S`$\underset{R}{\Longrightarrow}S\underset{R}{\Longrightarrow}$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`$\underset{R}{\Longrightarrow}S\underset{R}{\Longrightarrow}bAc\underset{R}{\Longrightarrow}$bec
- S`$\underset{R}{\Longrightarrow}S\underset{R}{\Longrightarrow}$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$GOTO(I_k,a)= I_j$，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$GOTO(I_k,A)= I_j$，则置GOTO表[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

表 7.10 LR(1)分析表

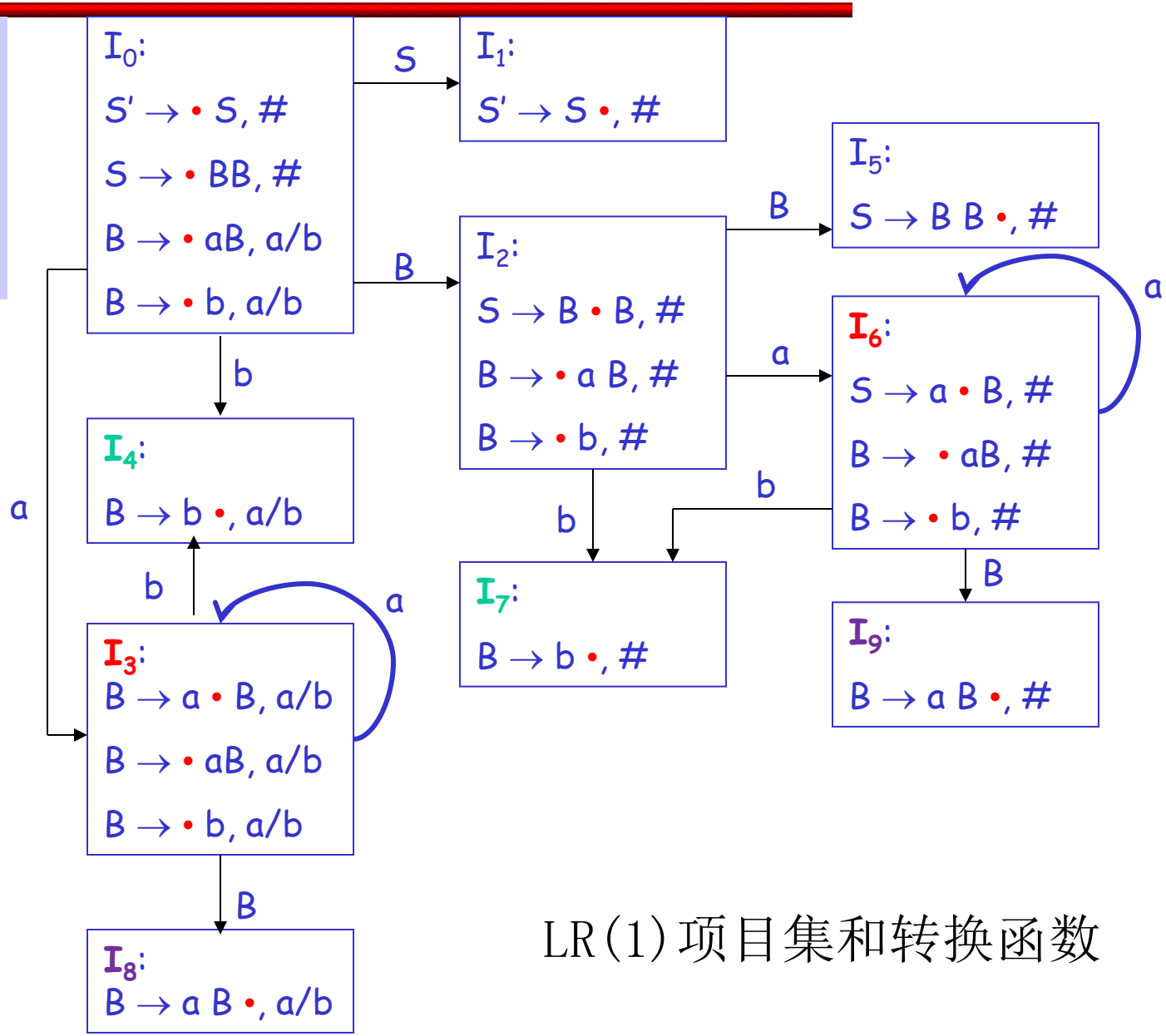
状态	ACTION						GOTO	
	a	*b*	*c*	*d*	*e*	#	S	A
0	S_2	S_3					1	
1						*acc*		
2					S_5			4
3					S_7			6
4				S_8				
5			S_9	r_5				
6			S_{10}					
7			r_5	S_{11}				
8						r_1		
9						r_3		
10						r_2		
11						r_4		

LALR(1)分析

文法*G'*：

(0) S' → S
(1) B → aB
(2) S → BB
(3) B → 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:
$S \rightarrow a \cdot B, \#$
$B \rightarrow \cdot aB, \#$
$B \rightarrow \cdot b, \#$

合并为 →

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

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。

合并同心集的几点说明

- 同心集合并后心仍相同，只是超前搜索符集合为各同心集超前搜索符的合集，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
  - 举例说明 P149

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_1		
8,9	r_2	r_2	r_2		

(0) $S' \rightarrow S$　　(2) $B \rightarrow aB$

(1) $S \rightarrow BB$　　(3) $B \rightarrow b$

表 7.13　对输入串 $ab\#$ 用 LR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	$ab\#$	S_3	
2	03	$\#a$	$b\#$	S_4	
3	034	$\#ab$	#	出错	

表 7.14　对输入串 $ab\#$ 用 LALR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	$ab\#$	$S_{3,6}$	
2	0(3,6)	$\#a$	$b\#$	$S_{4,7}$	
3	0(3,6)(4,7)	$\#ab$	#	r_3	(8,9)
4	0(3,6)(8,9)	$\#aB$	#	r_2	2
5	02	$\#B$	#	出错	

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。

该文法的 LR(0)项目集规范族为：

I_0: $S' \rightarrow \cdot S$
$S \rightarrow \cdot L=R$
$S \rightarrow \cdot R$
$L \rightarrow \cdot *R$
$L \rightarrow \cdot i$
$R \rightarrow \cdot L$

I_1: $S' \rightarrow S \cdot$

I_2: $S \rightarrow L \cdot =R$
$R \rightarrow L \cdot$

I_3: $S \rightarrow R \cdot$

I_4: $L \rightarrow * \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot *R$
$L \rightarrow \cdot i$

I_5: $L \rightarrow i \cdot$

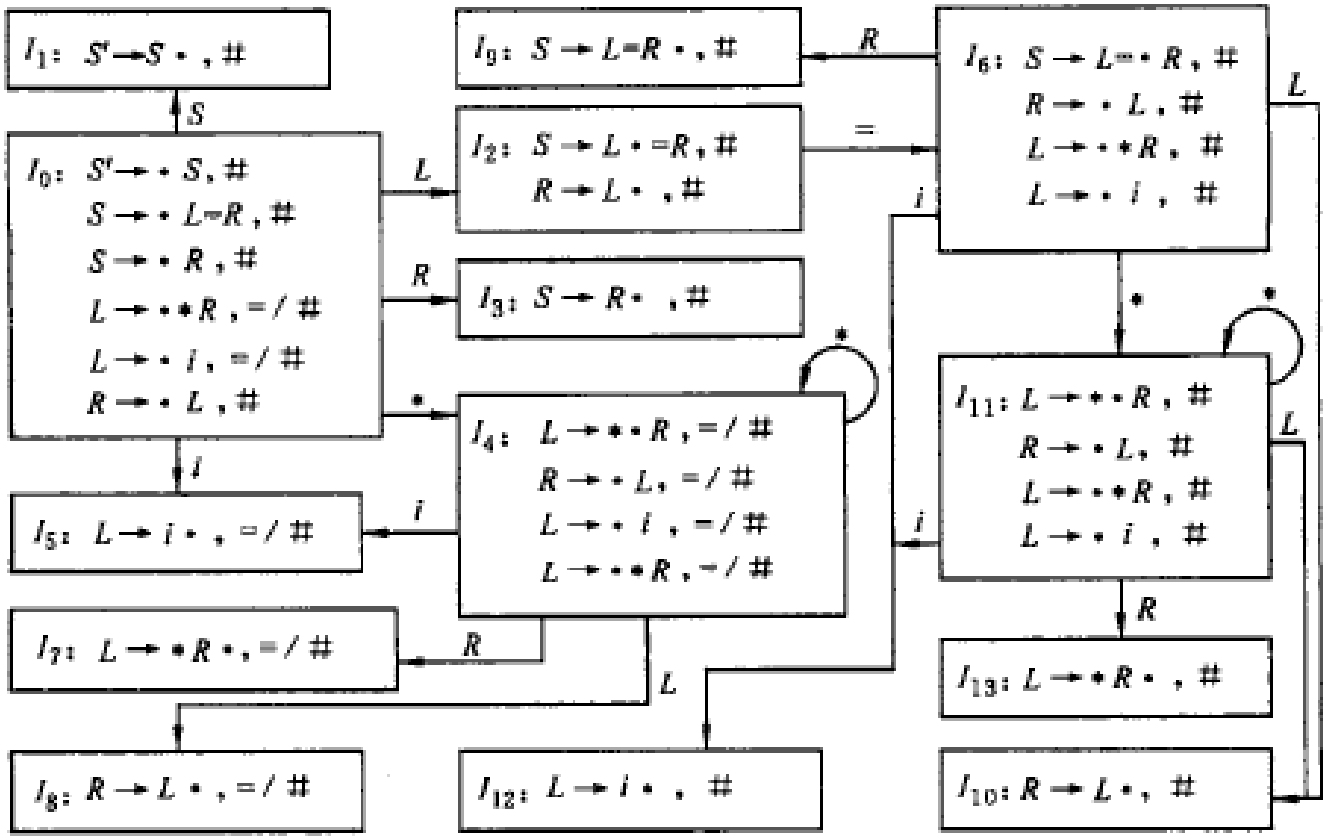
I_6: $S \rightarrow L= \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot *R$
$L \rightarrow \cdot i$

I_7: $L \rightarrow *R \cdot$

I_8: $R \rightarrow L \cdot$

I_9: $S \rightarrow L=R \cdot$

SLR(1): Follow(R)={#, =}
不能用SLR(1)解决



I_4 和 I_{11} 为：
{ $L \rightarrow * \cdot R$, =/#
$R \rightarrow \cdot L$, =/#
$L \rightarrow \cdot i$, =/#
$L \rightarrow \cdot * R$, =/# }

I_5 和 I_{12} 为：
{ $L \rightarrow i \cdot$, =/# }

I_7 和 I_{13} 为：
{ $L \rightarrow * R \cdot$, =/# }

I_8 和 I_{10} 为：
{ $R \rightarrow L \cdot$, =/# }

图 7.13　LR(1)项目集及转换函数

同心集合并后，无规约-规约冲突，则为LR(1)和LALR(1)文法

(0) $S' \rightarrow S$

(1) $S \rightarrow aAd$

(2) $S \rightarrow bBd$

(3) $S \rightarrow aBe$

(4) $S \rightarrow bAe$

(5) $A \rightarrow c$

(6) $B \rightarrow c$

我们可以直接构造它的 LR(1)项目集如下：

I_0： $S' \rightarrow \cdot S$ ，#
$S \rightarrow \cdot aAd$ ，#
$S \rightarrow \cdot bBd$ ，#
$S \rightarrow \cdot aBe$ ，#
$S \rightarrow \cdot bAe$ ，#

I_1： $S' \rightarrow S \cdot$ ，#

I_2： $S \rightarrow a \cdot Ad$ ，#
$S \rightarrow a \cdot Be$ ，#
$A \rightarrow \cdot c$ ，d
$B \rightarrow \cdot c$ ，e

I_3： $S \rightarrow b \cdot Bd$ ，#
$S \rightarrow b \cdot Ae$ ，#
$B \rightarrow \cdot c$ ，d
$A \rightarrow \cdot c$ ，e

I_4： $S \rightarrow aA \cdot d$ ，#

I_5： $S \rightarrow aB \cdot e$ ，#

I_6： $A \rightarrow c \cdot$ ，d
$B \rightarrow c \cdot$ ，e

I_7： $S \rightarrow bB \cdot d$ ，#

I_8： $S \rightarrow bA \cdot e$ ，#

I_9： $A \rightarrow c \cdot$ ，e
$B \rightarrow c \cdot$ ，d

I_{10}： $S \rightarrow aAd \cdot$ ，#

I_{11}： $S \rightarrow aBe \cdot$ ，#

I_{13}： $S \rightarrow bBd \cdot$ ，#

I_{14}： $S \rightarrow bAe \cdot$ ，#

同心集合并：

I_6：$A \rightarrow c \cdot$ ，d
$B \rightarrow c \cdot$ ，e

I_9：$A \rightarrow c \cdot$ ，e
$B \rightarrow c \cdot$ ，d

若合并后则变为：

$I_{6,9}$：$A \rightarrow c \cdot$ ，d/e
$B \rightarrow c \cdot$ ，e/d

- 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。**

作业

- P166　2、3(1,3,4)、9