第五章 自底向上语法分析

5.1 算符优先分析

1、已知文法G[S]为：

S→a|∧|(T)

T→T,S|S

(1) 计算G[S]的FIRSTVT 和LASTVT。

(2) 构造G[S]的算符优先关系表并说明G[S]是否为算符优先文法。

(3) 计算G[S]的优先函数。

(4) 给出输入串(a,a)#和(a,(a,a))#的算符优先分析过程。

答：

(1) FIRSTVT(T)是非终结符 T 的最左终结符集合，而 LASTVT(T)是非终结符 T 的最右终结符集合，由定义我们可以求出 G[S]的 FIRSTVT 和 LASTVT，如下所示：

非终结符	FIRSTVT	LASTVT
S	{ a , ^, (}	{ a , ^,) }
T	{ a , ^, (, , }	{ a , ^,) , , }

(2) 算法优先关系表如下所示：

	a	^	(	)	,	#
a				>	>	>
^				>	>	>
(	<	<	<	=	<	
)				>	>	>
,	<	<	<	>	>	
#	<	<	<			=

算法优先关系表中没有多重入口，所以 G[S]是算符优先文法。

(3) 对应的算符优先函数为：

	a	(	)	,	^	#
f	2	1	2	2	2	1
g	3	3	1	1	3	1

(4) 对输入串(a,a)#的算符优先分析过程如下：

栈	当前输入字符	剩余输入字符	操作
#	(	a，a）#	移进
#（	a	，a）#	移进
#（a	,	a）#	规约，S→a
#（N	,	a）#	移进
#（N,	a	）#	移进
#（N，a	）	#	规约，S→a
#（N，N	）	#	规约，T→T，S
#（N	）	#	移进
#（N）	#		规约，S→(T)
#N	#		

对输入串(a,(a,a))#的算符优先分析过程如下：

栈	当前输入字符	剩余输入字符	操作
#	(	a ,(a, a))#	移进
#(	a	,(a , a))#	移进
#(a	,	(a , a))#	规约，S→a
#(N	,	(a , a))#	移进
#(N,	(	a , a))#	移进
#(N,(	a	, a))#	移进
#(N,(a	,	a))#	规约，S→a
#(N,(N	,	a))#	移进
#(N,(N	a	))#	移进
#(N,(N,a	)	)#	规约，S→a
#(N,(N,N	)	)#	规约，T→T，S
#(N,(N	)	)#	移进
#(N,(N)	)	#	规约，S→(T)
#(N,N	)	#	规约，T→T，S
#(N	)	#	移进
#N(N)	#		规约，S→(T)
#N	#		

2、已知文法G[S]为：

S→a|∧|(T)

T→T,S|S

(1) 给出(a,(a,a))和(a,a)的最右推导，和规范归约过程。

(2) 将(1)和题 1 中的(4)进行比较给出算符优先归约和规范归约的区别。

答：

(1)(a,(a,a))的最右推导过程为：

S => (T)
=> (T,S)
=> (T,(T))
=> (T,(T,S))
=> (T,(T,a))
=> (T,(S,a))
=> (T,(a,a))
=> (S,(a,a))
=> (a,(a,a))

(a,a)的最右推导过程为：

S => (T)
=> (T,S)
=> (T,a)
=> (S,a))
=> (a,a)

(a,(a,a))的规范归约过程为：

栈	输入字符	操作
#	(a,(a,a))#	移进
#(	a,(a,a))#	移进
#(a	,(a,a))#	归约，S→a
#(S	,(a,a))#	归约，L→S
#(T	,(a,a))#	移进
#(T,	(a,a))#	移进
#(T,(	a,a))#	移进
#(T,(a	,a))#	归约，S→a
#(T,(S	,a))#	归约，T→S
#(T,(T	,a))#	移进
#(T,(T,	a))#	移进
#(T,(T,a	))#	归约，S→a
#(T,(T,S	))#	归约，T→T,S
#(T,(T	))#	移进
#(T,(T)	)#	归约，S→(T)
#(T,S	)#	移进
#(T,S)	#	归约，T→T,S
#(T)	#	归约，S→(T)
#S	#	成功

(a,a)的规范归约过程为：

栈	输入字符	操作
#	(a,a)#	移进
#(	a,a)#	移进
#(a	,a)#	归约，S→a
#(S	,a)#	归约，T→S
#(T	,a)#	移进
#(T,	a)#	移进
#(T,a	)#	归约，S→a
#(T,S	)#	归约，T→T,S
#(T	)#	移进
#(T)	#	归约，S→(T)
#S	#	成功

(2)算符优先文法在归约过程中只考虑终结符之间的优先关系从而确定可归约串，其于非终结符无关，只需要把当前可归约为某一个非终结符，不必知道该终结符是什么，因此去掉了单非终结符的归约。而规范归约的可归约串是句柄，并且必须要准确写出可归约串归约为哪一个终结符。

3、有文法G[S]：

S→V

V→T|ViT

T→F|T+F

F→)V*|(

(1) 给出(+(i(的规范推导。

(2) 指出句型 F+Fi(的短语，句柄，素短语。

(3) G[S]是否为OPG？若是，给出(1)中句子的分析过程。

答：

(1) S => V
=> ViT
=> ViF
=> Vi(
=> Ti(
=> T+F i(
=> T+(i(
=> F+(i(
=> (+(i(

(2)F+Fi(的短语是 F , F+F , (, F+Fi(

F+Fi(的句柄是 F

F+Fi(的素短语是 (

(3)首先求出G[S]的FIRSTVT和LASTVT：

FIRSTVT(S) = {) , (, + ,i }

FIRSTVT(V) = {) , (, + ,i }

FIRSTVT(T) = {) , (, + }

FIRSTVT(F) = {) , (}

LASTVT(S) = { (, * , + ,i }

LASTVT(V) = { (, * , + ,i }

LASTVT(T) = { (, * , +}

LASTVT(F) = { (, * }

接着根据这个求出算符优先关系表

	i	+	)	*	(	#
i	>	<	<		<	
+	>	>	<	>	<	
)	<	<	<	=	<	
*	>	>		>		
(	>	>		>		
#						=

OPG（自底向上算符优先分析法）只考虑算符（终结符）之间的优先关系，分析扫描每个规约式的算符间优先关系，它两两终结符间至多一种优先关系，观察上表发现G[S]满足这个条件，又因为G[S]是OP的，所以G[S]是OPG。

(+(i(的分析过程如下：

栈	当前字符	优先关系	剩余输入串	操作
#	(	# < (	+(i(#	移进
#(	+	(> +	(i(#	归约，F→(
#S	+	# < +	(i(#	移进
#S+	(	+ < (	i(#	移进
#S+	i	(> i	(#	归约，F→(

#S+S	i	+ > i	(#	归约，T→T+F
#S	i	# < i	(#	移进
#Si	(	i < (	#	移进
#Si(	#	(> #		归约，F→(
#SiS	#	i > #		归约，V→ViT
#S	#	# = #		接受

5.2 LR 分析

1、已知文法

A→aAd|aAb| ε

判断该文法是否是 SLR(1)文法，若是构造相应分析表，并对输入串 ab#给出分析过程。

答：设其拓广文法为 G’,增加产生式 S’→ A，由产生式可以得到：

First(S’) = { ε , a } Follow(S’) = { # }

First(A) = { ε ,a } Follow(A) = {d,b,#}

若产生式排序为：

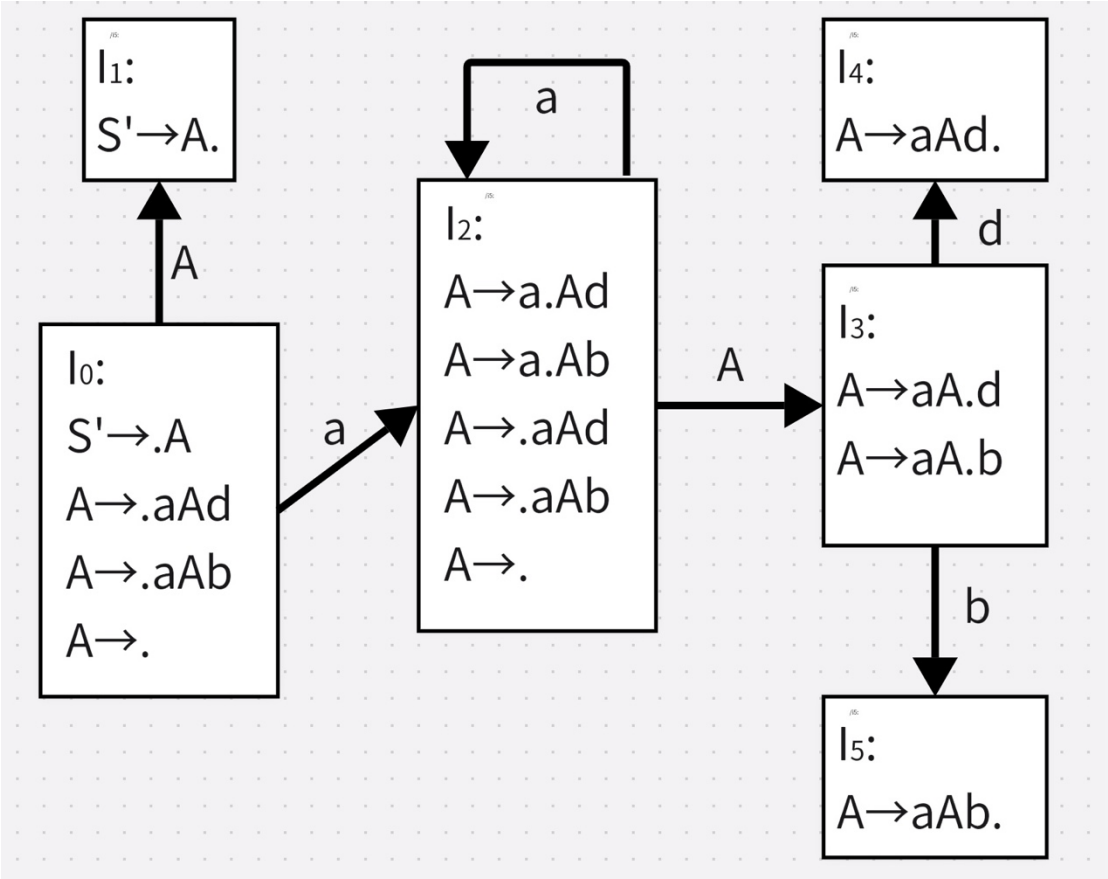
0 S’→ A

1 A → aAd

2 A → aAb

3 A → ε

G’的 LR(0)项目集族及识别活前缀的 DFA 如图所示：



在 I_0 中：

A →.aAd 和 A →.aAb 为移进项目，A →.为归约项目，存在移进-归约冲突，因此所给文法不是 LR(0)文法。

在 I_0、I_2 中：

Follow(A) ∩{a}= {d，b，#} ∩{a}= Ø

所以在 I_0、I_2 中的移进-归约冲突可以由 Follow 集解决，所以 G 是 SLR(1)文法。

构造的 SLR(1)分析表如下：

状态	Action				Goto
	a	d	b	#	A
0	S2	R3	R3	R3	1
1				acc	
2	S2	R3	R3	R3	3
3		S4	S5		
4		R1	R1	R1	
5		R2	R2	R2	

对输入串 ab#的分析过程如下表所示：

状态栈	文法符号栈	剩余输入串	动作
0	#	ab#	移进
0 2	#aA	b#	归约，A→ ε
0 2 3	#aA	b#	移进
0 2 3 5	#aAb	#	归约，A→aAb
0 1	#A	#	

2、若有定义二进制数的文法如下：

S→L·L|L

L→LB|B

B→0|1

(1) 试为该文法构造LR 分析表，并说明属哪类LR 分析表。

(2) 给出输入串 101.110 的分析过程。

答：

(1) 扩广文法为 G'，增加产生式 S' → S

由产生式可知：

First(S')={0,1}　　Follow(S')={#}

First(S)={0,1}　　Follow(S)={#}

First(L)={0,1}　　Follow(L)={.,0,1,#}

First(B)={0,1}　　Follow(B)={.,0,1,#}

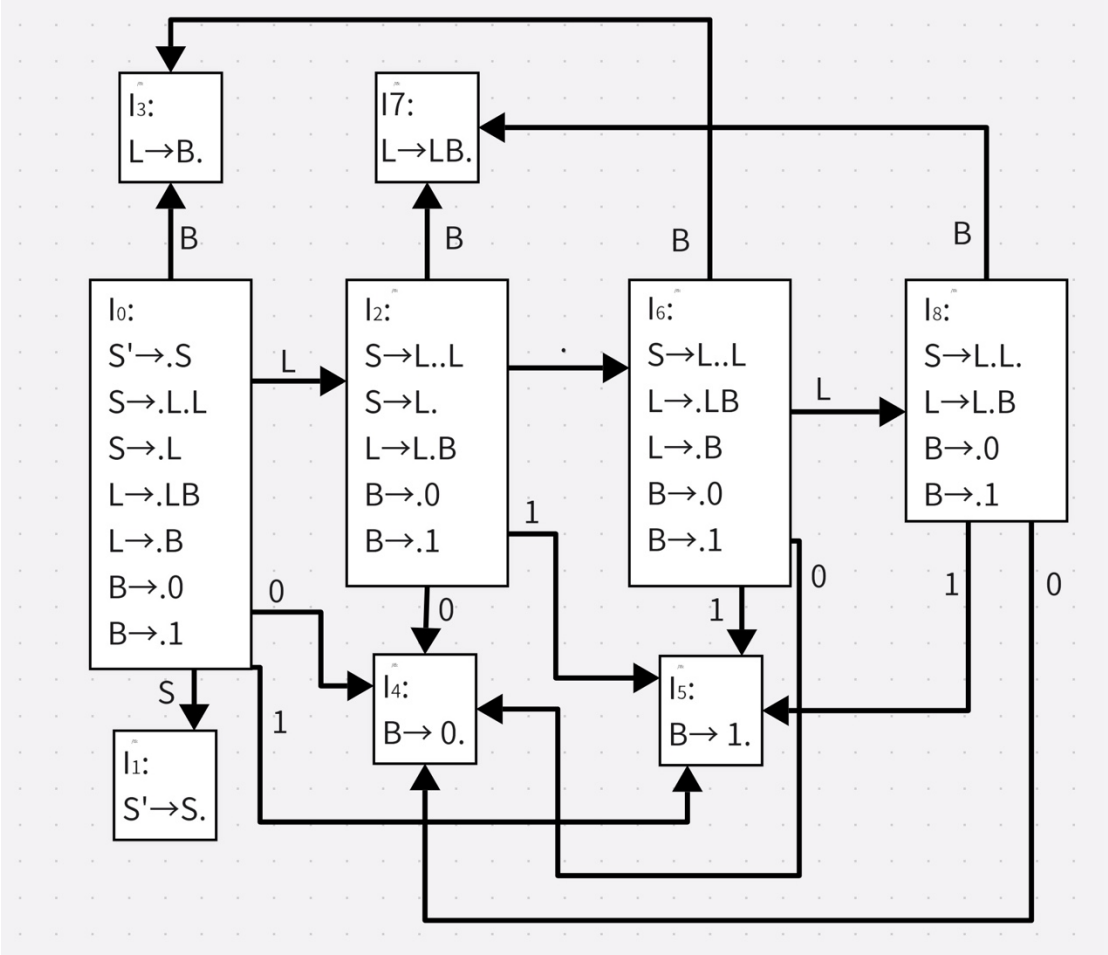
若产生式排序为：

0 S'→S

1 S→L.L

2 S→L

3 L→LB
4 L→B
5 B→0
6 B→1
G’的LR(0)项目集族及识别活前缀的DFA如下图所示：



在I_2中：

B →.0和 B →.1为移进项目，S →L.为归约项目，存在移进-归约冲突，因此所给文法不是LR(0)文法。

在I_2、I_8中：

Follow(s) ∩{0，1}= { #} ∩{0，1}=Ø

所以在I_2、I_8中的移进-归约冲突可以由Follow集解决，所以G是SLR(1)文法。

构造的SLR(1)分析表如下：

状态	Action				Goto		
	.	0	1	#	S	L	B
0		S4	S5		1	2	3
1				Acc			
2	S6	S4	S5	R2			7

3	R4	R4	R4	R4			
4	R5	R5	R5	R5			
5	R6	R6	R6	R6			
6		S4	S5			8	3
7	R3	R3	R3	R3			
8		S4	S5	R1			7

(2) 对输入串 101.110#的分析过程如下表所示：

状态栈	文法符号栈	剩余输入串	操作
0	#	101.110#	移进
0 5	#1	01.110#	归约：B→1
0 2	#B	01.110#	归约：L→B
0 3	#L	01.110#	移进
0 3 4	#L0	1.110#	归约：B→0
0 3 7	#LB	1.110#	归约：L→LB
0 3	#L	1.110#	移进
0 3 5	#L1	.110#	归约：B→1
0 3 7	#LB	.110#	归约：L→LB
0 3	#L	.110#	移进
0 3 6	#L.	110#	移进
0 3 6 5	#L.1	10#	归约：B→1
0 3 6 2	#L.B	10#	归约：L→B
0 3 6 8	#L.L	10#	移进
0 3 6 8 5	#L.L1	01#	归约：B→1
0 3 6 8 7	#L.LB	01#	归约：L→LB
0 3 6 8	#L.L	01#	移进
0 3 6 8 4	#L.L0	1#	归约：B→0
0 3 6 8 7	#L.LB	1#	归约：L→LB
0 3 6 8	#L.L	1#	移进
0 3 6 8 5	#L.L1	#	归约：B→1
0 3 6 8 7	#L.LB	#	归约：L→LB
0 3 6 8	#L.L	#	归约：S→L.L
0 1	#S	#	acc

3、文法G=({U,T,S},{a,b,c,d,e},P,S)

其中P 为：

S→UTa|Tb

T→S|Sc|d

U→US|e

(1) 判断G 是LR(0)，SLR(1)，LALR(1)还是LR(1)，说明理由。

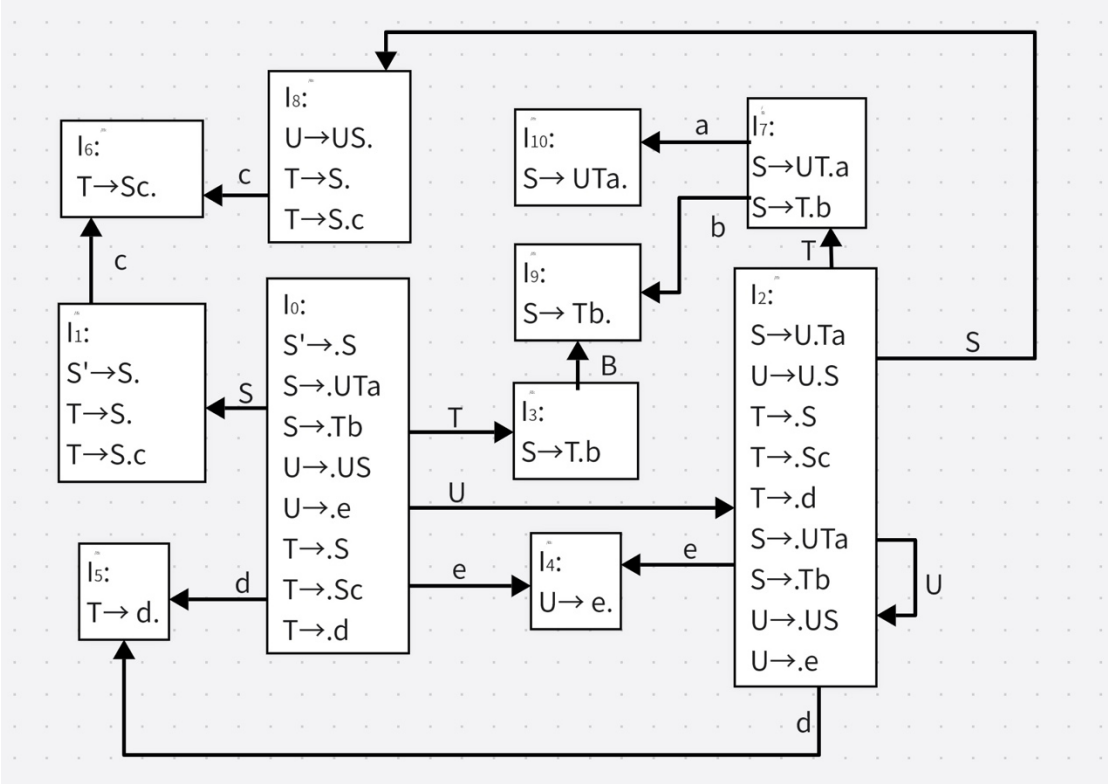
(2) 构造相应的分析表。

答：

(3) 扩广文法为 G’,增加产生式 S’ → S

由产生式可知：

First(S’)={d,e}　　Follow(S’)={#}
First(S)={d,e}　　Follow(S)={a,b,c,d,e,#}
First(U)={e}　　Follow(U)={d,e}
First(T)={d,e}　　Follow(T)={a,b}
若产生式排序为：
0 S’→S
1 S→Uta
2 S→Tb
3 T→S
4 T→Sc
5 T→d
6 U→US
7 U→e
G’的LR(0)项目集族及识别活前缀的DFA如下图所示：



在I_1中：

S’→S.为接受项目，T →S.为归约项目，T →S.c为移进项目，存在接受-归约和移进-归约冲突，因此所给文法不是LR(0)文法。

在I_1中：

Follow(S’) ∩ Follow(T)= { #} ∩{a , b}=Ø
Follow(T) ∩{ c}= { a , b} ∩{c}=Ø

在I_8中：

Follow(U) ∩ Follow(T) ∩{ c}={d,e}∩{ a , b} ∩{c}=Ø

所以在 I_1 中的接受-归约和移进-归约冲突与 I_8 中的移进-归约和归约-归约冲突可以由 Follow 集解决，所以 G 是 SLR(1)文法。

(2) 构造的 SLR(1)分析表如下：

状态	Action						Goto		
	a	b	c	d	e	#	S	U	T
0				S5	S4		1	2	3
1	R3	R3	S6			Acc			
2				S5	S4		8	2	7
3		S9							
4				R7	R7				
5	R5	R5							
6	R4	R4							
7	S10	S9							
8	R3	R3	S6	R6	R6				
9	R2	R2	R2	R2	R2	R2			
10	R1	R1	R1	R1	R1	R1			