编译原理参考答案

蓝小斌 陈嘉豪

第七章 语法制导的语义计算

1. 下面的文法$G[S']$描述由布尔常量$false, true$,联结词∧(合取)、∨(析取)、¬(否定)构成的不含括号的二值布尔表达式的集合：

$S' \to S$

$S \to S \vee T | T$

$T \to T \wedge F | F$

$F \to \neg F | false | true$

试设计一个基于$G[S']$的属性文法，它可以计算出每个二值布尔表达式的取值。如对于句子$\neg true \vee \neg false \wedge true$，输出是 true。

答：

$S' \to S$	$\{print\ S.value\}$
$S \to S_1 \vee T$	$\{ if(S_1.value == true\ or\ T.value == true)$
	$then\ S.value = true$
	$then\ S.value = false\}$
$S \to T$	$\{S.value = T.value\}$
$T \to T_1 \wedge F$	$\{if(T_1.value == true\ and\ F.value == true)$
	$then\ T.value = true$
	$then\ T.value = false$
$T \to F$	$T.value = F.value\}$
$T \to \neg F$	$\{if(F.value == true)$
	$then\ T.value = false$
	$else\ T.value = true\}$
$F \to false$	$\{F.value = false\}$
$F \to true$	$\{F.value = true\}$

2. 给定文法$G[S]$：

$S \to (L) | a$

$L \to L, S | S$

如下是相应于G[S]的一个属性文法(或翻译模式)：

$S \to (L)$	$\{S.num := L.num + 1;\}$
$S \to a$	$\{S.num := 0;\}$
$L \to L_1, S$	$\{L.num := L_1.num + S.num;\}$
$L \to S$	$\{L.num := S.num;\}$

图 7.19 分别是输入串$(a,(a))$的语法分析树和对应的带标注语法树，但后者的属性值没有标出，试将其标出(即填写图 7.19 右图中符号=右边的值)。

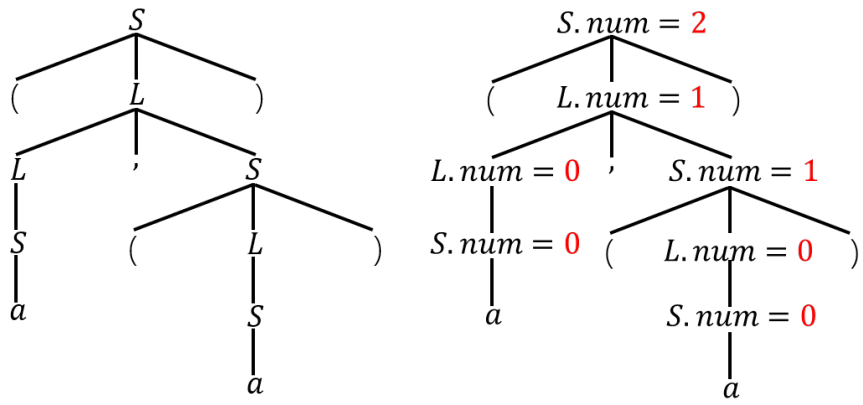


图 7.19 题 2 的语法分析树和带标注语法树

答：如上图所示

4. 以下是简单表达式(只含加、减运算)计算的一个属性文法$G(E)$:

$E \to TR$ $\{R.in := T.val; E.val := R.val\}$

$R \to +TR_1$ $\{R_1.in := R.in + T.val; R.val = R_1.val\}$

$L \to -TR_1$ $\{R_1.in := R.in - T.val; R.val := R_1.val\}$

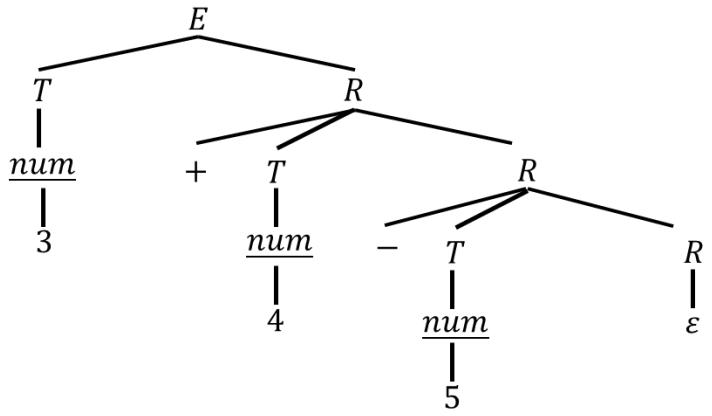
$R \to \varepsilon$ $\{R.val := R.in;\}$

$T \to \underline{num}$ $\{T.val := lexval(num)\}$

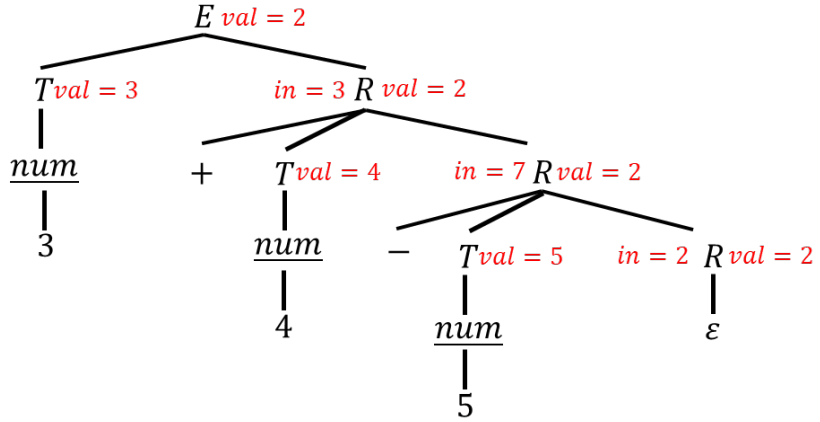
其中，$lexval(num)$表示从词法分析程序得到的常数值。

试给出表达式$3+4-5$的语法分析树和相应的带标注语法分析树。

答：语法分析树如下



带标注的语法树如下：



5. 题 2 中所给的$G[E]$的属性文法是一个S-属性文法，故可以在自底向上分析过程中增加语义栈来计算属性值，图 7.20 是$G[S]$的一个LR分析表，图 7.21 描述了输入串$(a,(a))$的分析和求值过程(语义栈中的值对应$S.num$或$L.num$)，其中，第 14、15 行没有给出，试着补全。

状态	ACTION					GOTO	
	a	,	(	)	#	S	L
0	S_3		S_2			1	
1					acc		
2	S_3		S_2			5	4
3		r_2		r_2	r_2		
4		S_7		S_6			
5		r_4		r_4			
6		r_1		r_1	r_1		
7	S_3		S_2			8	
8		r_3		r_3			

图 7.20 题 5 的LR分析表

步骤	状态栈	语义栈	符号栈	余留符号串
1	0	_	#	$(a,(a))$#
2	02	_ _	#(	$a,(a))$#
3	023	_ _ _	#(a	$,(a))$#
4	025	_ _ 0	#(S	$,(a))$#
5	024	_ _ 0	#(L	$,(a))$#
6	0247	_ _ 0 _	#(L,	$(a))$#
7	02472	_ _ 0 _ _	#(L,(	$a)$#
8	024723	_ _ 0 _ _ _	#(L,(a	$))$#
9	024725	_ _ 0 _ _ 0	#(L,(S	$))$#
10	024724	_ _ 0 _ _ 0	#(L,(L	$))$#
11	0247246	_ _ 0 _ _ 0 _	#(L,(L)	$)$#
12	02478	_ _ 0 _ 1	#(L,S	$)$#
13	024	_ _ 1	#(L	$)$#
14	0246	_ _ 1 _	#(L)	#
15	01	_ 2	#S	#
16	接受			

图 7.21 题 5 的分析和求值过程

答：如上图所示

7.设题 4 中属性文法的基础文法为$G[E]$。

(1) 说明$G[E]$是$LL(1)$文法。

(2) 如下是以$G[E]$作为基础文法设计的翻译模式：

$E \to T \ \{R.in := T.val\} \ R \ \{E.val := R.val\}$

$R \to +T \ \{R_1.in := R.in + T.val\} \ R_1 \ \{R.val := R_1.val\}$

$R \to -T \ \{R_1.in := R.in - T.val\} \ R_1 \ \{R.val := R_1.val\}$

$R \to \varepsilon \quad \{R.val := R.in;\}$

$T \to \underline{num} \quad \{T.val := lexval(num)\}$

试争对该翻译模式构造相应的递归下降(预测)翻译程序(如题 6,可直接使用例 7.9 中的MatchToken函数)。

答:

每个产生式的$SELECT$集合如下:

$SELECT(E \to T) = \{num\}$

$SELECT(R \to +T) = \{+\}$

$SELECT(R \to -T) = \{-\}$

$SELECT(R \to \varepsilon) = \{\#\}$

$SELECT(T \to num) = \{num\}$

相同左部产生式的$SELECT$交集为

$SELECT(R \to +T) \cap SELECT(R \to -T) \cap SELECT(R \to \varepsilon) = \{+\} \cap \{-\} \cap \{\#\} = \emptyset$

所以该文法为$LL(1)$文法。

对应的递归下降翻译程序为

文法$G[E]$对应的递归下降翻译程序

```
1.  int ParseE()
2.  {
3.      Tv:=ParseT();           //变量 Tv 对应属性 T.value
4.      Ri:=Tv;                 //变量 Ri 对应属性 R.in
5.      Rv:=ParseR(Ri);
6.      Ev:=Rv;
7.       return Ev;
8.  }
9.  int ParseR(int f)           //形参 f 对应属性 R.in
10. {
11.     if(lookahead == '+')    //lookahead 是当前扫描的输入符号
12.     {
13.         MatchToken('+');
14.         Tv:=ParseT();
15.         R1i:=f+Tv;          //R1i 对应属性 R1.in
16.         R1v:=ParseR(R1i);
17.         Rv:=R1v;
18.     }
19.     else if(lookahead == '-')
20.     {
21.         MatchToken('-');
22.         Tv:=ParseT();
23.         R1i=f-Tv;
24.         R1v:=ParseR(R1i);
25.         Rv:=R1v;
26.     }
27.     else if(lookahead == '#')
28.     {
```

```
29.         Rv = f;
30.     }
31.     else
32.     {
33.         printf("Syntax error");
34.         exit(0);
35.     }
36.    return Rv;
37. }
38. int ParseT()
39. {
40.     if(lookahead == '(lexvalnum)')
41.     {
42.         MatchToken('(lexvalnum)');
43.         Tv:=lexval(num);
44.     }
45.     else
46.     {
47.         printf("Syntax error.");
48.         exit(0);
49.     }
50.     return Tv;
51. }
52. void MatchToken(int expected)
53. {
54.     if(lookahead!= expected)
55.     {
56.         printf("syntax error\n");
57.         exit(0);
58.     }
59.     else
60.     {
61.         lookahead=getToken();
62.     }
63. }
```