（若发现问题，请及时告知）

第 6 讲书面作业包括两部分。第一部分为 Lecture06.pdf 中课后作业题目中的 2, 4, 8, 9 以及 11。第二部分为以下题目：

A1　如下是以 $G[S]$ 为基础文法的一个 L 翻译模式：

$S \rightarrow \{\ P.i := 0\ \}\ P\ \{\ \text{print}\ (P.s)\ \}$
$P \rightarrow \wedge\ \{\ P_1.i := P.i\ \}\ P_1\ \{\ P_2.i := P_1.s\ \}\ P_2\ \{\ P.s := P_2.s + 1\ \}$
$P \rightarrow \vee\ \{\ P_1.i := P.i\ \}\ P_1\ \{\ P_2.i := P.i\ \}\ P_2\ \{\ P.s := P_1.s + P_2.s\ \}$
$P \rightarrow \neg\ \{\ P_1.i := P.i\ \}\ P_1\ \{\ P.s := P.i + P_1.s\ \}$
$P \rightarrow \underline{id}\ \{\ P.s := 0\ \}$

试针对该 L 翻译模式构造一个自上而下的递归下降（预测）翻译程序

```
void ParseS ( )                    // 主函数
{
    pi := 0;
    ps := ParseP(pi);
    print (ps);
}

int ParseP ( int i )               // 主函数
{
    switch (lookahead)  {          // lookahead 为下一个输入符号
        case '∧':
            MatchToken ('∧');
            p1i := ____①____;
            p1s := ParseP (p1i);
            p2i := p1s;
            p2s := ParseP (p2i);
            ps := p2s + 1;
            break;
        case '∨':
            MatchToken ('∨');
            p1i := ____②____;
            p1s := ParseP (p1i);
            ____③____;
            p2s := ParseP (p2i);
            ____④____;
            break;
```

```
            case '¬':
                MatchToken ('¬');
                ⑤ ;
                p1s := ParseP (p1i);
                ⑥ ;
                break;
            case id:
                MatchToken(id);
                ps := 0;
                break;
            default:
                printf("syntax error \n")
                exit(0);
        }
        return ps;
}
```

其中使用了与课程中所给的 MatchToken 函数。

该翻译程序中 ①~⑥ 的部分未给出，试填写之。

参考解答：

```
void ParseS ( )                    // 主函数
{
    pi := 0;
    ps := ParseP(pi);
    print (ps);
}

int ParseP ( int i )               // 主函数
{
    switch (lookahead)  {          // lookahead 为下一个输入符号
        case '∧':
            MatchToken ('∧');
            p1i := i;
            p1s := ParseP (p1i);
            p2i := p1s;
            p2s := ParseP (p2i);
            ps := p2s + 1;
            break;
        case '∨':
            MatchToken ('∨');
            p1i := i;
            p1s := ParseP (p1i);
            p2i := i;
```

```
                p2s := ParseP (p2i);
                ps := p1s + p2s;
                break;
            case '¬':
                MatchToken ('¬');
                p1i := i;
                p1s := ParseP (p1i);
                ps := i + p1s;
                break;
            case id:
                MatchToken(id);
                ps := 0;
                break;
             default:
                printf("syntax error \n")
                exit(0);
        }
        return ps;
}
```

其中使用了与课程中所给的 MatchToken 函数。

A2. 给定用来描述某种命题逻辑公式的文法 $G[S]$：

（1）$S \to P$
（2）$P \to P\ P\ \wedge$
（3）$P \to P\ P\ \vee$
（4）$P \to P\ \neg$
（5）$P \to \text{id}$

其中，终结符∧、∨、¬ 分别代表三种逻辑连接词，*id* 是标识符（在计算逻辑公式真值的时候，代表一个命题变元）。

如下是以 $G[S]$ 为基础文法的一个 S 翻译模式：

（1）$S \to P$ { *print* ($P.s$) }
（2）$P \to P_1\ P_2\ \wedge$ { $P.s := f_1\ (P_1.s, P_2.s)$ }
（3）$P \to P_1\ P_2\ \vee$ { $P.s := f_2\ (P_1.s, P_2.s)$ }
（4）$P \to P_1\ \neg$ { $P.s := f_3\ (P_1.s)$ }
（5）$P \to id$ { $P.s := g\ (id)$}

其中，*print* 为打印函数，f_1，f_2，f_3，g 为其他语义函数。

(a) 如果在 LR 分析过程中根据这一翻译模式进行自下而上语义计算，试写出在按每个产生式归约时语义处理的一个代码片断（设语义栈由向量 *val* 表示，归约

前栈顶位置为 *top*，终结符不对应语义值，而每个非终结符的综合属性都只对应一个语义值，本题中可用 *val*[*i*].*s* 表示；不用考虑对 *top* 的维护）。

(b) 指定语义函数

$f_1(x, y)$ = **if** (x = 1 **and** y = 1) **then** 1

else if (x = 0 **and** y = 0) **then** 1

else 0

$f_2(x, y)$ = **if** (x = 0 **and** y = 1) **then** 1

else if (x = 1 **and** y = 0) **then** 1

else 0

$f_3(x)$ = **if** x = 0 **then** 1 **else** 0

$g(a) = 1$　　$g(b) = 1$　　$g(c) = 0$

那么对于合法输入串*a b* ¬ ∧ *a b c* ∧ ∨ *a* ¬ *c b* ∧ ∨ ∨ ∨ ，翻译模式的语义计算结果是什么？（即 *print* 的打印结果，你无需给出计算过程）

参考解答：

(a)

（1）$S \to P$　　{ *print* (*val*[*top*].*s*) }　　0.5分

（2）$P \to P_1\ P_2\ \wedge$　　{ *val*[*top*-2].*s* := f_1 (*val*[*top*-2].*s*, *val*[*top*-1].*s*) }　1分

（3）$P \to P_1\ P_2\ \vee$　　{ *val*[*top*-2].*s* := f_2 (*val*[*top*-2].s, *val*[*top*-1].*s*) }　1分

（4）$P \to P_1\ \neg$　　{ *val*[*top*-1].*s* := f_3 (*val*[*top*-1].*s*) }　1分

（5）$P \to \underline{id}$　　{ *val*[*top*].*s* := *g* (*val*[*top*]) }　0.5分

或 { *val*[*top*].*s* := *g* (*id*) }

(b) 1

以下是 Lecture06 文档中的题目

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 18 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（即填写右下图中符号“=”右边的值）。

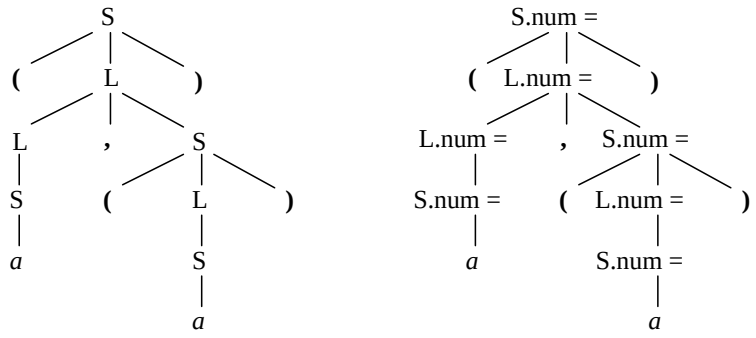
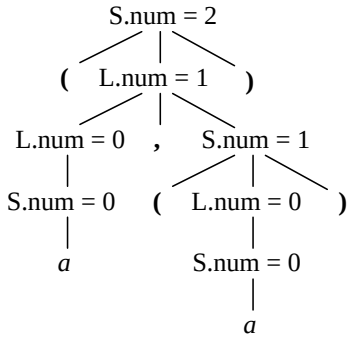


图 18　题 2 的语法分析树和带标注语法树

参考解答：



4. 题 2 中所给的 $G[S]$ 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 19 是 $G[S]$ 的一个 LR 分析表，图 20 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，试补齐之。

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 19　题 4 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）				
15）				
16）	接受			

图 20　题 4 的分析和求值过程

参考解答：

14）	0246	– – 1 –	#(L)	#
15）	01	– 2	#S	#

8.　给定文法 *G*[*S*]：

$$S \rightarrow M\,A\,b\,B$$

$A \to A\ a \mid \varepsilon$

$B \to B\ a \mid B\ b \mid \varepsilon$

$M \to \varepsilon$

在文法 $G[S]$ 基础上设计如下翻译模式：

$S \to M$ { $A.in\_num := M.num$ }

$A\ b$ { $B.in\_num := A.num$ }

B { *if* $B.num=0$ *then* $S.accepted := true$ else $S.accepted := false$ }

$A \to$ { $A_1.in\_num := A.in\_num$ } $A_1\ a$ { $A.num := A_1.num - 1$ }

$A \to \varepsilon$ { $A.num := A.in\_num$ }

$B \to$ { $B_1.in\_num := B.in\_num$ } $B_1\ a$ { $B.num := B_1.num - 1$ }

$B \to$ { $B_1.in\_num := B.in\_num$ } $B_1\ b$ { $B.num := B_1.num$ }

$B \to \varepsilon$ { $B.num := B.in\_num$ }

$M \to \varepsilon$ { $M.num :=$ 100 }

不难看出，嵌在产生式中间的语义动作集中仅含复写规则，并且在自底向上的语法分析过程中，文法符号的所有继承属性均可以通过归约前已出现在分析栈中的综合属性唯一确定地进行访问。试写出在按每个产生式归约时语义计算的一个代码片断（设语义栈由向量 *v* 表示，归约前栈顶位置为 *top*，终结符不对应语义值，而每个非终结符的综合属性都只对应一个语义值，可用 *v*[i].*num* 或 *v*[i]. *accepted* 访问，不用考虑对 *top* 的维护）。

参考解答：

结果之一：

$S \to MAbB$ { *if* v[top].*num=0 then* v[top-3]. *accepted := true)*
else v[top-3]. *accepted := false)* }

$A \to A_1\ a$ { v[top-1] . *num* := v[top-1] . *num* - 1}

$A \to \varepsilon$ { v[top+1] . *num* := v[top].*num* }

$B \to B_1\ a$ { v[top-1] . *num* := v[top-1] . *num* - 1}

$B \to B_1\ b$ { v[top-1] . *num* := v[top-1] . *num* }

$B \to \varepsilon$ { v[top+1] . *num* := v[top-1] . *num* }

$M \to \varepsilon$ { v[top+1] . *num* := *100* }

9. 变换如下翻译模式，使嵌在产生式中间的语义动作集中仅含复写规则，并使得在自底向上的语法分析过程中，文法符号的所有继承属性均可以通过归约前已出现在分析栈中的确定的综合属性进行访问：

$D \to D_1$; T { $L.type := T.type$; $L.offset := D_1.width$; $L.width := T.width$ } L
{$D.width := D_1.width + L.num \times T.width$ }

$D \to T$ { $L.type := T.type$; $L.offset :=$ 0 ; $L.width := T.width$ } L
{$D.width := L.num \times T.width$ }

$T \to$ integer { $T.type := int$; $T.width :=$ 4 }

$T \to$ real { $T.type := real$; $T.width :=$ 8 }

$L \to$ { $L_1.type := L.type$; $L_1.offset := L.offset$; $L_1.width := L.width$; } L_1 , id

{ *enter* (id.*name*, *L. type*, *L. offset* + L_1.*num* × *L. width*) ; *L.num* := L_1.*num* +1 }

$L \rightarrow$ id { *enter* (id.*name*, *L. type*, *L. offset*) ; *L.num* := 1}

参考解答：

$D \rightarrow D_1$; *T* { *L.type* := *T.type*; *L.offset* := D_1.*width* ; *L.width* := *T.width* } *L*
{*D.width* := D_1.*width* + *L.num* × *T.width* }

$D \rightarrow M\ N\ T$ { *L.type* := *T.type*; *L.offset* := *M.s* ; *L.width* := *T.width* } *L*
{*D.width* := *L.num* × *T.width* }

$T \rightarrow$ integer { *T.type* := *int* ; *T.width* := 4 }

$T \rightarrow$ real { *T.type* := *real* ; *T.width* := 8 }

$L \rightarrow$ { L_1. *type* := *L. type* ; L_1. *offset* := *L. offset* ; L_1. *width* := *L. width* ; } L_1 , id
{ *enter* (id.*name*, *L. type*, *L. offset* + L_1.*num* × *L. width*) ; *L.num* := L_1.*num* +1 }

$L \rightarrow$ id { *enter* (id.*name*, *L. type*, *L. offset*) ; *L.num* := 1}

$M \rightarrow \varepsilon$ { *M.s* := *0* }

$N \rightarrow \varepsilon$ { }

11．如下翻译模式，其基础文法是 *G*[*S*]：

S → *A b* {*B.in_num* := *A* .*num* + 100}
B { *if B.num=0 then S.accepted* := *true*
else S.accepted := *false* }

S → *A b b* {*B.in_num* := *A* .*num* +50 }
B { *if B.num=0 then S.accepted* := *true*
else S.accepted := *false* }

$A \rightarrow A_1\ a$ {*A. num* := A_1.*num* + 1}

$A \rightarrow \varepsilon$ {*A. num* := 0}

$B \rightarrow$ {B_1.*in_num* := *B.in_num*} $B_1\ a$ {*B. num* := B_1.*num* - 1}

$B \rightarrow \varepsilon$ {*B. num* := *B.in_num* }

（*a*）变换该翻译模式，使嵌在产生式中间的语义动作集中仅含复写规则，并使得在自底向上的语义计算过程中，文法符号的所有继承属性均可以通过归约前已出现在分析栈中的确定的综合属性进行访问。

（*b*）如果在 LR 分析过程中根据（*a*）所得到的新翻译模式进行自底向上的语义计算，试写出在按每个产生式归约时语义计算的一个代码片断（假设语义栈由向量 *v* 表示，归约前栈顶位置为 *top*，终结符不对应语义值，而每个非终结符的综合属性 *x* 都只对应一个语义值，可用 *v*[i].*x* 访问，不用考虑对 *top* 的维护）。

参考解答：

（*a*）结果之一：

S → *A b* {*M.i* := *A* .*num*} *M* {*B.in_num* := *M.s* }
B { *if B.num=0 then S.accepted* := *true*
else S.accepted := *false* }

$S \rightarrow A\ b\ b\ \{\ N.i := A.num\ \}\ N\ \{B.in\_num := N.s\ \}$
$B\ \ \{\ if\ B.num=0\ \ then\ \ S.accepted := true$
$else\ \ S.accepted := false\ \}$
$A \rightarrow A_1\ a\ \ \{A.\ num := A_1.num + 1\}$
$A \rightarrow \varepsilon\ \ \ \{A.\ num := 0\}$
$B \rightarrow \{B_1.in\_num := B.in\_num\}\ \ B_1\ a\ \ \{B.\ num := B_1.num - 1\}$
$B \rightarrow \varepsilon\ \ \{B.\ num := B.in\_num\ \}$
$M \rightarrow \varepsilon\ \ \ \{\ M.s := M.i + 100\ \}$
$N \rightarrow \varepsilon\ \ \ \{\ N.s := N.i + 50\}$

（*b*）结果之一:

$S \rightarrow A\ b\ M\ B\ \ \{\ if\ \ v[top].num =0\ \ then\ \ v[top-3].accepted := true$
$else\ \ v[top-3].accepted := false\ \}$
$S \rightarrow A\ b\ b\ N\ B\ \ \{\ if\ \ v[top].num =0\ \ then\ \ v[top-4].accepted := true$
$else\ \ v[top-4].accepted := false\ \}$
$A \rightarrow A_1\ a\ \ \{\ v[top-1].num := v[top-1].num + 1\}$
$A \rightarrow \varepsilon\ \ \ \{\ v[top+1].num := 0\}$
$B \rightarrow B_1\ a\ \ \{\ v[top-1].num := v[top-1].num - 1\}$
$B \rightarrow \varepsilon\ \ \{\ v[top+1].num := v[top].s\ \}$
$M \rightarrow \varepsilon\ \ \ \{\ v[top+1].s := v[top-1].num + 100\ \}$
$N \rightarrow \varepsilon\ \ \ \{\ v[top+1].s := v[top-2].num + 50\}$