（若发现问题，请及时告知）

A1，A2 是选自 Lecture04 文档中的题目（叙述上略微有修订）

A3，A4，A6 是选自 Lecture06 文档中的题目（叙述上略微有修订）

（若发现问题，请及时告知）

A1. 以下是某简单语言的一段代码。语言中不包含数据类型的声明，所有变量的类型默认为整型（假设占用一个存储单元）。语句块的括号为‘begin’和‘end’组合；赋值号为‘:=’，不等号为‘<>’。每一个过程声明对应一个静态作用域（假定采用多遍扫描机制，在静态语义检查之前每个作用域中的所有表项均已生成，即可以向前引用同一作用域中后面出现的符号）。该语言支持嵌套的过程声明，但只能定义无参过程，且没有返回值。

```
（1） var a0, b0, a2;
（2） procedure fun1 ;
（3）         var a1, b1;
（4）         procedure fun2 ;
（5）                 var a2;
（6）                 begin
（7）                         a2 := a1 + b1;
（8）                         if(a0 <> b0) then call fun3;
  .                           ……    /*不含任何声明语句*/
  .                  end;
  .          begin
                      a1 := a0 - b0;
                      b1 := a0 + b0;
（x）                 If   a1 < b1   then   call fun2 ;
  .                  ……     /*不含任何声明语句*/
  .          end ;
  .  procedure fun3 ;
  .          var a3;
  .          begin
                 a3 := a0*b0 ;
（y）            if(a2 <> a3) call fun1 ;
.                ……    /*不含任何声明语句*/
```

```
.           end ;
.   begin
.           a0 := 1;
.           b0 := 2;
.           a2 := a0/b0 ;
.           call fun3;
.           ......    /*不含任何声明语句*/
.   end .
```

若实现该语言时符号表的组织采用多符号表结构，即每个静态作用域均对应一个符号表。试指出：在分析至语句（x）时，当前开作用域有几个？分别包含哪些符号？在分析至语句（y）时，所访问的a2是在哪行语句声明的？

参考解答：

在分析至语句（x）时，当前开作用域有2个；分别包含符号{ a0,b0,a2,fun1, fun3}和{ a1,b1,fun2}。在分析至语句（y）时，所访问的a2是在第（1）行语句声明的。

A2. 如下是某语言(PL/0)的一段代码，若该语言编译器的符号表如4.1节所述的那样，采用一个全局的单符号表栈结构。对于下列程序片断，当编译器在处理到第一个 call p 语句（第 7 行）以及第二个 call p 语句（第 t 行，即过程 q 的第 4 行）时，试分别列出每个开作用域中的符号。

```
（1）  var a,b;
（2）  procedure   p ;
（3）          var s;
（4）          procedure   r ;
（5）                  var v;
（6）                  begin
（7）                          call p;
  .                               …...
  .                         end;
  .             begin
  .                     If   a < b   then   call r ;
  .                     …...
  .             end ;
  .       procedure   q ;
  .             var x,y;
  .             begin
（t）                     call p ;
  .                     …...
  .             end ;
  .       begin
```

```
.          a := 1;
.          b := 2;
.          call q;
.          …...
.     end .
```

参考解答：

在处理到第一个 call p 语句（第 7 行）时，每个开作用域中的符号：

主过程作用域中的 a, b, p

过程 p 作用域中的 s, r

过程 p 内过程 r 作用域中的 v

在处理到第二个 call p 语句（第 t 行，即过程 q 的第 4 行）时，每个开作用域中的符号：

主过程作用域中的 a, b, p, q

过程 q 作用域中的 x, y

A3. 给定文法 $G[S]$：

$$S \to （L） \mid a$$
$$L \to L\,,\, S \mid S$$

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; }

以下两个图分别是输入串（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）。

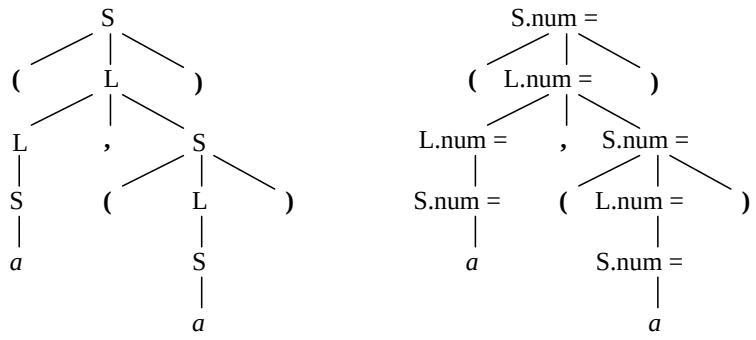
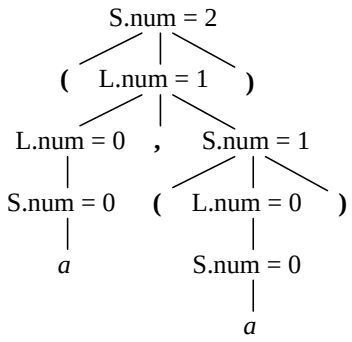


图 18　题 2 的语法分析树和带标注语法树

参考解答：



A4. 题 A3 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。如下是 *G*[*S*] 的一个 *LR* 分析表：

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			

另外，下图描述了输入串（*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。

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			

参考解答：

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

A5. 如下是以某个 $G[S]$ 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。

A6. 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：

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

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

$T \rightarrow \underline{\text{integer}} \quad \{ \; T.type := int \; ; \; T.width := 4 \; \}$

$T \rightarrow \underline{\text{real}} \quad \{ \; T.type := real \; ; \; T.width := 8 \; \}$

$L \rightarrow$ { 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}

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* }
$N \rightarrow \varepsilon$ { }

A7. 给定用来描述某种命题逻辑公式的文法 *G*[*S*]：

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