第 8 讲书面作业包括两部分。第一部分为 Lecture08.pdf 中课后作业题目中的 4，5，8 以及 9。第二部分为以下题目：

A1　给定文法 $G[S]$：

$$S \to (L) \mid a \mid b$$
$$L \to L + S \mid S + S \mid S$$

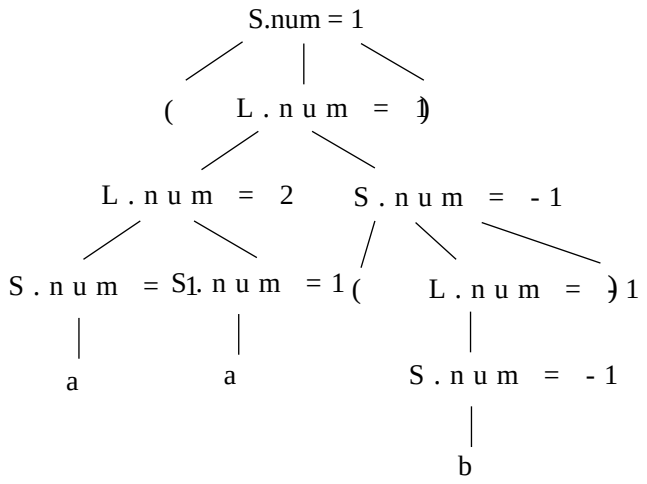
如下是相应于 $G[S]$ 的一个属性文法（或翻译模式）：

$S \to (L)$　　{ $S.num := L.num$; }
$S \to a$　　{ $S.num := 1$; }
$S \to b$　　{ $S.num := -1$; }
$L \to L_1 + S$　　{ L.num := L_1.num + S.num; }
$L \to S_1 + S_2$　　{ L.num := S_1.num + S_2.num; }
$L \to S$　　{ L.num := S.num; }

针对输入串 $(a+a+(b))$，请写出其带标注语法树。

（朱俸民）

参考解答：



A2　回顾上次作业定义的类型表达式 T 的文法 $G[T]$：

$$T \to T * T \mid T \texttt{ -> } T \mid \texttt{(} T \texttt{)} \mid \texttt{Int} \mid \texttt{Bool}$$

一个类型表达式 T 可以是基本类型（这里仅考虑整数类型 `Int` 和布尔类型 `Bool`，皆为终结符），乘积类型（$T * T$），或者函数类型（T `->` T）。与普通的算术表达式类似，我们通过括号`()`来指定最高的优先级。此外，运算符`*`的优先级高于`->`。运算符`*`具有左结合性，而`->`具有右结合性。

文法 $G[T]$ 可以改写为 S-翻译模式：

$T \rightarrow U$ { *T.type* := *U.type* }

$U \rightarrow V$ `->` U_1 { *U.type* := TArrow(*V.type*, U_1.*type*) }

$U \rightarrow V$ { *U.type* := *V.type* }

$V \rightarrow V_1$ `*` W { *V.type* := TProd(V_1.*type*, *W.type*) }

$V \rightarrow W$ { *V.type* := *W.type* }

$W \rightarrow$ `(` T `)` { *W.type* := *T.type* }

$W \rightarrow$ `Int` { *W.type* := TInt }

$W \rightarrow$ `Bool` { *W.type* := TBool }

其中，我们用如下表所示的类型表达式构造子（在 Java 中可以理解为类型表达式的四个子类对应的构造函数）来存储一个类型表达式的语法节点到综合属性 *type* 中。

构造子	返回值
TProd(*x*, *y*)	元祖类型表达式 *x* * *y*
TArrow(*x*, *y*)	函数类型表达式 *x* -> *y*
TInt	整数类型表达式
TBool	布尔类型表达式

（1）以下是在 LR 分析过程中，根据上述 *S*-翻译模式进行自底向上语义计算时的一个代码片断，它体现了语义栈上的操作（设语义栈由向量 *v* 表示，归约前栈顶位置为 *top*，不用考虑对 *top* 的维护）。请补全(a) - (c)处的代码片段。

$T \rightarrow U$ { *v*[*top*].*type* := *v*[*top*].*type* }

$U \rightarrow V$ `->` U_1 { (a) }

$U \rightarrow V$ { *v*[*top*].*type* := *v*[*top*].*type* }

$V \rightarrow V_1$ `*` W { (b) }

$V \rightarrow W$ { *v*[*top*].*type* := *v*[*top*].*type* }

$W \rightarrow$ `(` T `)` { (c) }

$W \rightarrow$ `Int` { *v*[*top*].*type* := TInt }

$W \rightarrow$ `Bool` { *v*[*top*].*type* := TBool }

（2）针对此 *S*-翻译模式，经消除基础文法左递归后可以得到的 *L*-翻译模式：

$T \rightarrow U$ { *T.type* := *U.type* }

…

$W \rightarrow$ `(` T `)` { *W.type* := *T.type* }

$W \rightarrow$ `Int` { *W.type* := TInt }

$W \rightarrow$ `Bool` { *W.type* := TBool }

请补全有关乘积类型和函数类型的产生式及其语义动作。

（3）变换（2）中的 *L*-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。在（2）的基础上，请仅写出需要发生变化的和新增的产生式及语义动作。

（4）如果在 LR 分析过程中根据（3）中的 *L*-翻译模式进行自底向上的语义计算，可以设计如下代码片断，体现语义栈上的操作（设语义栈由向量 *v* 表示，归约前栈顶位置为 *top*，不用考虑对 *top* 的维护）：

$T \rightarrow U$	{ *v*[*top*].*type* := *v*[*top*].*type* }
...	
$W \rightarrow$ (T)	{ *v*[*top*-2].*type* := *v*[*top*-1].*type* }
$W \rightarrow$ `Int`	{ *v*[*top*].*type* := TInt }
$W \rightarrow$ `Bool`	{ *v*[*top*].*type* := TBool }

请补全其他的产生式及其语义动作。

（朱俸民）

参考解答：

（1）

(a) *v*[*top*-2].*type* := TArrow(*v*[*top*-2].*type*, *v*[*top*].*type*)

(b) *v*[*top*-2].*type* := TProd(*v*[*top*-2].*type*, *v*[*top*].*type*)

(c) *v*[*top*-2].*type* := *v*[*top*-1].*type*

（2）

$U \rightarrow V$ { *Q*.*i* := *V*.*type* } Q { *U*.*type* := *Q*.*s* }

$Q \rightarrow$ -> U { *Q*.*s* := TArrow(*Q*.*i*, *U*.*type*) }

$Q \rightarrow \varepsilon$ { *Q*.*s* := *Q*.*i* }

$V \rightarrow W$ { *P*.*i* := *W*.*type* } P { *V*.*type* := *P*.*s* }

$P \rightarrow$ * W { P_1.*i* := TProd(*P*.*i*, *W*.*type*) } P_1 { *P*.*s* := P_1.*s* }

$P \rightarrow \varepsilon$ { *P*.*s* := *P*.*i* }

（3）

$P \rightarrow$ * W { *M*.*i* := *P*.*i* ; *M*.*j* := *W*.*type* } M { P_1.*i* := *M*.*s* } P_1 { *P*.*s* := P_1.*s* }

$M \rightarrow \varepsilon$ { *M*.*s* := TProd(*M*.*i*, *M*.*j*) }

（4）

$U \rightarrow V\ Q$	{ *v*[*top*-1].*type* := *v*[*top*].*s* }
$Q \rightarrow$ -> U	{ *v*[*top*-1].*s* := TArrow(*v*[*top*-2].*type*, *v*[*top*].*type*) }
$Q \rightarrow \varepsilon$	{ *v*[*top*+1].*s* := *v*[*top*].*type* }
$V \rightarrow W\ P$	{ *v*[*top*-1].*type* := *v*[*top*].*s* }
$P \rightarrow$ * $W\ M\ P_1$	{ *v*[*top*-3].*s* := *v*[*top*].*s* }
$P \rightarrow \varepsilon$	{ *v*[*top*+1].*s* := *v*[*top*].*type* }
$M \rightarrow \varepsilon$	{ *v*[*top*+1].*s* := TProd(*v*[*top*-2].*type*, *v*[*top*].*type*) }

以下是 Lecture08 文档中的题目

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	#

5. 给定 LL(1)文法 *G*[*S*]：

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

如下是以 *G*[*S*] 作为基础文法设计的翻译模式：

$S \to A\ b$ {*B.in_num* := *A* .*num*}　*B*　{ *if B.num=0 then*　*print(“Accepted!”)*
else　*print(“Refused!”)*　}

$A \to aA_1$　{*A. num* := A_1.*num* + 1}

$A \to \varepsilon$　　{*A. num* := 0}

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

$B \to b$　{B_1.*in_num* := *B.in_num*}　B_1　{*B. num* := B_1.*num* }

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

试针对该翻译模式构造相应的递归下降（预测）翻译程序（参考例 9）。（可直接使用 2.3 中的 MatchToken 函数）

参考解答：

```
void ParseS( )                    // 主函数
{
    A_num := ParseA();            //变量 A_num对应属性A.num
    MatchToken('b');
    B_in_num := A_num;            //变量 B_in_num 对应属性B.in_num
    B_num := ParseB(B_in_num);          //变量 B_num对应属性B.num
    if B_num =0   then   print("Accepted!")
                  else   print("Refused!");
}

int ParseA( )
{
    switch (lookahead)  {             // lookahead为下一个输入符号
        case ′ a′ :
            MatchToken('a');
            A1_num := ParseA();
            A_num := A1_num+1;
```

```
                break;
            case ′ b′ :
                A_num := 0;
                break;
            default:
                printf("syntax error \n")
                exit(0);
        }
        return A_num;
    }

    int ParseB( int in_num )
    {
        switch (lookahead)  {
            case ′ a′ :
                MatchToken(‘a’);
                B1_in_num := in_num;          //变量 B1_in_num 对应属性B1.in_num
                B1_num := ParseB(B1_in_num);
                B_num := B1_num-1;
                break;
            case ′ b′ :
                MatchToken(‘b’);
                B1_in_num := in_num;
                B1_num := ParseB(B1_in_num);
                B_num := B1_num;
                break;
            case ′ #′ :
                B_num := in_num;
                break;
            default:
                printf("syntax error \n")
                exit(0);
        }
        return B_num;
    }
```

8. 给定文法 $G[S]$：

$$S \to M\ A\ b\ B$$
$$A \to A\ a \mid \varepsilon$$
$$B \to B\ a \mid B\ b \mid \varepsilon$$
$$M \to \varepsilon$$

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

参考解答：

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

{*D.width* := D_1.*width* + *L.num* × *T.width* }

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

T → integer { *T.type* := *int* ; *T.width* := 4 }

T → real { *T.type* := *real* ; *T.width* := 8 }

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

M → ε { *M.s := 0* }

N → ε { }