第五章 作业

5.1
5.4
5.5（1）
5.8
5.10

参考答案

5.1

结点带有属性值的分析树称为注释分析树。相应地，计算各结点属性值的一系列相互关联的活动称为给分析树加注释。

根据教材表 5-1 中的基础文法，可以构造出表达式(4*7+1)*2 的分析树如图 1 所示。

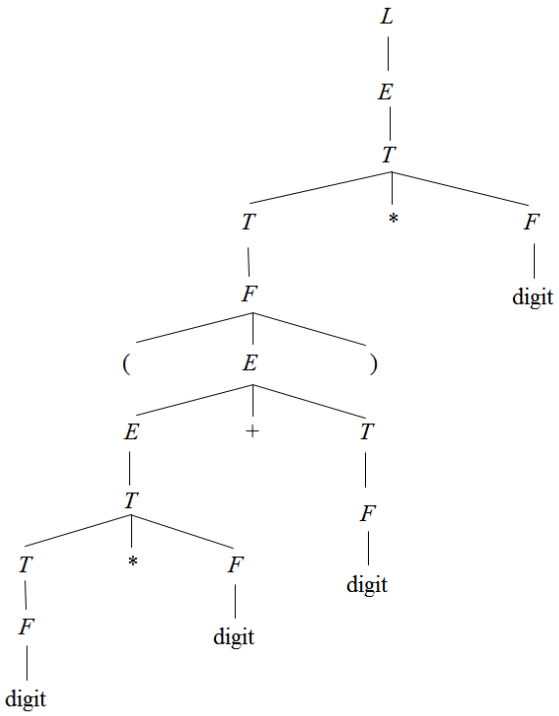


图 1 表达式(4*7+1)*2 的分析树

根据表 5-1 中的语法制导定义，符号 *E*、*T* 和 *F* 有综合属性 val，并给出了他们的计算规则，符号 *L* 有虚拟综合属性，即打印输出 *E* 的属性值 E.val，终结符号 digit 有综合属性 lexval，其值即词法分析程序识别出的数字的值。根据图 1 所示分析树，利用教材中表 5-1 中的语法制导定义，为分析树加注释，即自底向上计算出每个结点符号的属性值，得到图 2 所示的注释分析树。

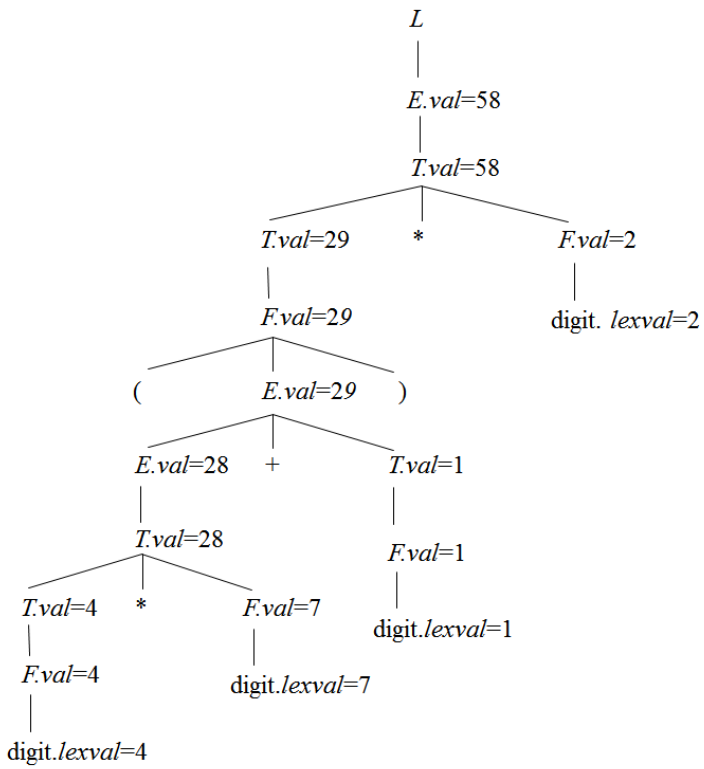
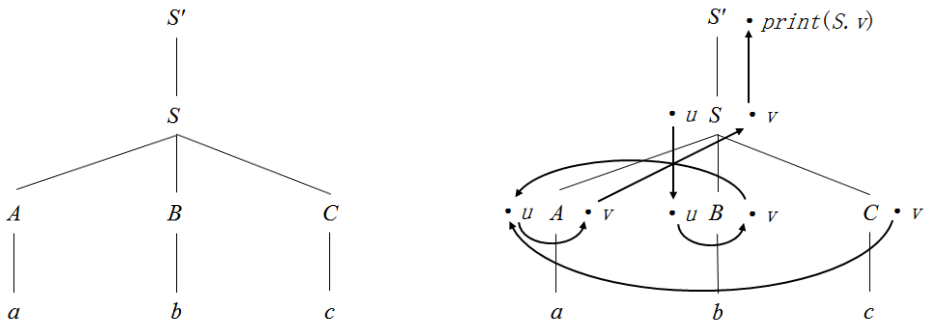


图 2 表达式(4*7+1)*2 的分析树

5.4

(1) 符串 *abc* 的分析树及相应的依赖图分别如图 3(a)和图 3(b)所示。



(a) 符串 *abc* 的分析树　　(b) 符串 *abc* 的依赖图

图 3 符串 *abc* 的分析树及相应的依赖图

(2) 根据上述依赖图，对有向图中的结点进行拓扑排序，可以得到如下的一个有效的语义规则执行顺序：

S.u=5

C.v=2

B.u=*S.u*

B.v=*B.u*

A.u=*B.v*+*C.v*

A.v=3**A.u*

S.v=*A.v*

print(*S.v*)

(3) 根据上面的计算次序执行每一条语义动作，则翻译完成时输出结果 21。

(4) 根据修改后的语法制导定义，可以构造出字符串 *abc* 的分析树和依赖图，如图 4 所示。

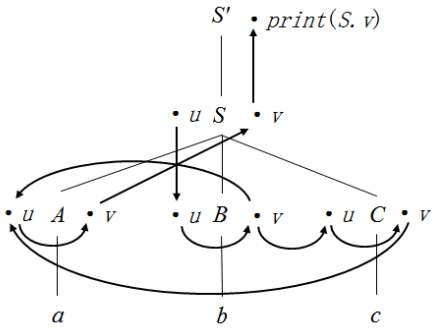


图 4 修改后，符串 *abc* 的分析树及相应的依赖图

根据该依赖图，拓扑排序的结果是得到如下的语义规则计算顺序：

S.u=5

B.u=*S.u*

B.v=*B.u*

C.u=*B.v*

C.v=*C.u*-2

A.u=*B.v*+*C.v*

A.v=3**A.u*

S.v=*A.v*

print(*S.v*)

翻译完成时输出的结果值是 24。

5.5（1）

该语法制导定义的翻译目标是确定输入表达式的类型，表达式的类型由参与运算的子表达式的类型确定。若参与运算的数是整型数，则其类型为整型，否则为实型；若两个子表达式的类型都是整型，则结果表达式也是整型，否则是实型。

为此，设计综合属性 *E.type*、*T.type*，用来记录由 *E* 或 *T* 产生的表达式的类型。

确定每个子表达式类型的语法制导定义如表 1 所示。

表 1 确定每个子表达式类型的语法制导定义

产生式	语义规则
$E \rightarrow E_1+T$	if (E_1.*type*==integer)&&(*T.type*==integer) *E.type*=integer; else *E.type*=real
$E \rightarrow T$	*E.type*=*T.type*
$T \rightarrow$num.num	*T.type*=real
$T \rightarrow$num	*T.type*=integer

5.8

(1) 由文法可知，类型信息由 T 的产生式确定，设计综合属性 *T.type*，用于记录声明语句中类型关键字确定的类型信息，变量名表由 L 的产生式确定，为了获取名字的类型信息，设计继承属性 *L.in*，用于将类型信息传递给声明语句中的每个名字。为了将类型信息和名字联系在一起，设计过程 *addtype*(id.*entry*, *type*)，其功能是实现将类型信息 *type* 写入符号表中由 id.*entry* 指向的记录中。

确定声明语句中变量类型的语法制导定义如表 2 所示。

表 2　确定变量类型的语法制导定义

产生式	语义规则
$D\rightarrow LT$	*L.in*=*T.type*
$T\rightarrow$integer	*T.type*=integer
$T\rightarrow$real	*T.type*=real
$L\rightarrow L_1$,id	L_1.in=*L*.in *addtype*(id.*entry*, *L.in*)
$L\rightarrow$id	*addtype*(id.*entry*, *L.in*)

(2) 该定义不是 L 属性定义。

因为，L 属性定义要求，与每个产生式 $A\rightarrow X_1X_2\ldots X_n$ 相应的每条语义规则计算的属性或者是左部符号 A 的综合属性，或者是右部某符号 X_j（$1\leq j\leq n$）的继承属性，并且该继承属性只能依赖于 A 的继承属性和/或者产生式中 X_j 左边的符号 X_1、X_2、…、X_{j-1} 的属性。

对于该语法制导定义，由于在产生式 $D\rightarrow LT$ 的语义规则中，继承属性 *L.in* 依赖于它右边符号 T 的属性，故不满足 L 属性定义对继承属性的约束。

5.10

(1) 该语法制导定义中定义了综合属性 *B.ht* 和继承属性 *B.ps*。*B.ps* 要么取值常数，要么依赖于产生式左部符号 B 的继承属性，满足 L 属性定义对语义规则的约束，所以该语法制导定义是 L 属性定义。

(2) 为该语法制导定义设计的翻译方案如下。

$S\rightarrow$ { *B.ps*=10 }
　　B { *S.ht*=*B.ht*}
$B\rightarrow$ { B_1.*ps*=*B.ps*}
　　B_1 { B_2.ps=*B*.ps}
　　B_2 { *B*.ht=max(B_1.ht, B_2.ht) }
$B\rightarrow$ { B_1.*ps*=*B.ps*}
　　B_1 sub { B_2.ps=shrink(*B*.ps)}
　　B_2 { *B*.ht=disp(B_1.ht, B_2.ht) }
$B\rightarrow$text {*B.ht*=text.h×*B.ps*}

(3) 对于上述翻译方案，若将插入在产生式中间的语义动作移走，就可以用 *LR* 方法进行翻译。为此，需要引入标记非终结符号 L、M 和 N，以及相应的产生式 $L\rightarrow\varepsilon$、$M\rightarrow\varepsilon$和 $N\rightarrow\varepsilon$。通过 L 实现继承属性的初始化（即 *B.ps*=10）、通过 M 和 N 实现 B 的产生式中右部的第二个 B（即 B_2）的继承属性的计算。需要为标记非终结符号 L 设计综合属性 s，为标记非终结符号 M 和 N 设计继承属性 i 和综合属性 s。

相应的语法制导定义如表 3 所示

表 3 可以利用 LR 技术进行翻译的语法制导定义

产生式	语义规则
$S\rightarrow LB$	$B.ps=L.s$ $S.ht=B.ht$
$L\rightarrow\varepsilon$	$L.s$=10
$B\rightarrow B_1MB_2$	$B_1.ps=B.ps$ $M.i=B.ps$ $B_2.ps=M.s$ $B.ht$=max($B_1.ht$, $B_2.ht$)
$B\rightarrow B_1$subNB_2	$B_1.ps=B.ps$ $N.i=B.ps$ $B_2.ps=N.s$ $B.ht$=disp($B_1.ht$, $B_2.ht$)
$B\rightarrow$text	$B.ht$=text.$h\times B.ps$
$M\rightarrow\varepsilon$	$M.s=M.i$
$N\rightarrow\varepsilon$	$N.s$=shrink($N.i$)

与表 3 中语法制导定义相应的翻译方案如下。

$S\rightarrow L$ { $B.ps=L.s$ }
　B { $S.ht=B.ht$ }
$B\rightarrow$　{ $B_1.ps=B.ps$ }
　B_1 { $M.i=B.ps$ }
　M { $B_2.ps=M.s$ }
　B_2 { $B.ht$=max($B_1.ht$, $B_2.ht$) }
$B\rightarrow$　$B_1.ps=B.ps$ }
　B_1 *sub* { $N.i=B.ps$ }
　N { $B_2.ps=N.s$) }
　B_2 { $B.ht$=disp($B_1.ht$, $B_2.ht$) }
$B\rightarrow$text {$B.ht$=text.$h\times B.ps$}
$L\rightarrow\varepsilon${ $L.s$=10 }
$M\rightarrow\varepsilon${ $M.s=M.i$ }
$N\rightarrow\varepsilon${ $N.s$=shrink($N.i$) }

(4) 由于上述翻译方案中的所有继承属性都由复制规则赋值，所以，语法制导定义的实现都是通过跟踪它们在 *val* 栈中的位置来获得属性值的。设变量 *top* 和 *ntop* 分别是归约前和归约后的栈顶指针。假设当前句柄的长度是 r，则 $ntop=top-r+1$，则实现上述翻译方案的代码如表 4 所示。

表 4 实现翻译方案 5.7 的代码

产生式	语义规则
$S\rightarrow LB$	val[ntop]=val[top]
$L\rightarrow\varepsilon$	val[ntop]=10
$B\rightarrow B_1MB_2$	val[ntop]=max(val[top-2], val[top])
$B\rightarrow B_1$subNB_2	val[ntop]=disp(val[top-3], val[top])
$B\rightarrow$text	val[ntop]=val[top]×val[top-1]
$M\rightarrow\varepsilon$	val[ntop]=val[top-1]
$N\rightarrow\varepsilon$	val[ntop]=shrink(val[top-2])

(5) 假定有输入符号串 Esub2.s，并且假定：①词法分析识别出 E、sub、2、.、和 s 的属性 *h* 的值均为 1；②shrink(x)=0.3*x，disp(x, y)=x+y；③为说明方便，用文法符号代替与之对应的状态。则其分析过程如表 5 所示。

表 5　对输入符号串 Esub2.s 的分析过程

步骤	栈	输入	分析动作
(1)	state: Val:	E sub 2 . s $	归约　*L*→ε val[ntop]=10
(2)	state: *L* Val:10	E sub 2 . s $	移进
(3)	state: *L*　E Val:10　1	sub 2 . s $	归约　*B*→text val[ntop]=val[top]×val[top-1]
(4)	state: *L*　*B* Val:10　10	sub 2 . s $	移进
(5)	state: *L*　*B*　sub Val:10　10　-	2 . s $	归约　*N*→ε val[ntop]=shrink(val[top-2])
(6)	state: *L*　*B*　sub　*N* Val:10　10　-　3	2 . s $	移进
(7)	state: *L*　*B*　sub　*N*　2 Val:10　10　-　3　1	. s $	归约　*B*→text val[ntop]=val[top]×val[top-1]
(8)	state: *L*　*B*　sub　*N*　*B* Val:10　10　-　3　3	. s $	归约　*B*→*B* sub*NB* val[ntop]=disp(val[top-3], val[top])
(9)	state: *L*　*B* Val:10　13	. s $	归约　*M*→ε val[ntop]=val[top-1]
(10)	state: *L*　*B*　*M* Val:10　13　10	. s $	移进
(11)	state: *L*　*B*　*M*　. Val:10　13　10　1	s $	归约　*B*→text val[ntop]=val[top]×val[top-1]
(12)	state: *L*　*B*　*M*　*B* Val:10　13　10　10	s $	归约　*B*→*B MB* val[ntop]=max(val[top-2], val[top])
(13)	state: *L*　*B* Val:10　13	s $	归约　*M*→ε val[ntop]=val[top-1]
(14)	state: *L*　*B*　*M* Val:10　13　10	s $	移进
(15)	state: *L*　*B*　*M*　s Val:10　13　10　1	$	归约　*B*→text val[ntop]=val[top]×val[top-1]
(16)	state: *L*　*B*　*M*　*B* Val:10　13　10　10	$	归约　*B*→*B MB* val[ntop]=max(val[top-2], val[top])
(17)	state: *L*　*B* Val:10　13	$	归约　*S*→*LB* val[ntop]=val[top]
(18)	state: *S* Val:13	$	接受

下面对表 5 所示分析过程中的归约动作进行说明。

步骤(1)所示状态面临归约动作，即用产生式 $L\rightarrow\varepsilon$进行归约，归约时执行语义动作 $L.s$=10 的实现代码 val[ntop]=10，执行的结果是实现了继承属性值的入栈，见步骤(1)所示状态。

步骤(9)和(13)所示的状态面临用产生式 $M\rightarrow\varepsilon$进行归约，归约时执行语义动作 $M.s=M.i$ 的实现代码 val[ntop]=val[top-1]，执行的结果是实现了产生式 $B\rightarrow B_1MB_2$ 中 B_2 的继承属性值的入栈，见步骤(10)和(14)所示状态。

步骤(5)所示的状态面临用产生式 $N\rightarrow\varepsilon$进行归约，归约时执行语义动作 $N.s$=shrink($N.i$)的实现代码 val[ntop]=shrink(val[top-2])，执行的结果是实现了产生式 $B\rightarrow B_1\text{sub}NB_2$ 中 B_2 的继承属性值的入栈，见步骤(6)所示状态。

步骤(3)、(7)、(11)和(15)所示状态都面临归约动作，此时要把栈顶的 text 归约为 B，归约时，执行语义动作 $B.ht$=text.$h\times B.ps$ 的实现代码 val[ntop]=val[top]×val[top-1]，可以看出，当需要继承属性 $B.ps$ 的值时，可以在当前句柄的下面取得（即 val[top-1]）。

步骤(8)所示状态面临用产生式 $B\rightarrow B_1\text{sub}NB_2$ 进行归约，此时要把栈顶的 $B_1\text{sub}NB_2$ 归约为 B，归约时，执行计算左部符号 B 的综合属性的语义动作 $B.ht$=disp($B_1.ht$, $B_2.ht$)的实现代码 val[ntop]=disp(val[top-3], val[top])，结果见步骤(9)所致状态。

步骤(12)和(16)所示状态面临用产生式 $B\rightarrow B_1MB_2$ 进行归约，此时要把栈顶的 B_1MB_2 归约为 B，归约时，执行计算左部符号 B 的综合属性的语义动作 $B.ht$=max($B_1.ht$, $B_2.ht$)的实现代码 val[ntop]=max(val[top-2], val[top])，结果见步骤(13)和(17)所致状态。

步骤(17)所示状态面临用产生式 $S\rightarrow LB$ 进行归约，归约时，执行语义动作 $S.ht=B.ht$ 的实现代码 val[ntop]=val[top]，结果见步骤(18)所致状态。

从上述分析可以看出，当归约时，需要访问继承属性 $B.ps$ 的值时，可以从栈中直接取得，即在 val 栈中对应于 L、M 和 N 的单元中。当用 B 的产生式进行归约，即把栈顶的句柄归约为 B 时，B 的继承属性 $B.ps$ 的值在栈中的位置总是恰好在该被归约句柄的下面，表 5-12 中步骤(3)、(7)、(8)、(11)、(12)、(15)和(16)所示是面临将栈顶句柄归约为 B 的状态，可以看出，当前句柄的下面恰好就是 L、M 或者 N。