构造表达式((a * b) + (c)) 的分析树和语法树：

根据表 4.3 的语法制导定义。

表 4.3 构造表达式语法树的语法制导定义

产生式	语义规则
$E \rightarrow E_1 + T$	$E.nptr = mkNode('+', E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode('*', T_1.nptr, F.nptr)$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow$ **id**	$F.nptr = mkLeaf($ **id**, **id**$.entry)$
$F \rightarrow$ **num**	$F.nptr = mkLeaf($ **num**, **num**$.val)$

4.2 a: 注意分析树和语法树的区别

答 见图 4.2。分析树由短虚线表示，生成的语法树用实线表示，长虚线表示文法符号的属性所指向的语法树结点。分析树上最右边 R 的 s 属性指向所产生的语法树。为使得图形清晰，图 4.2 省略了最右边 R 的 s 属性向它的父结点 R 的 s 属性的复写，及最后向根结点 E 的 $nptr$ 属性的复写。

另外，图中左边文法符号的属性值很多都是指向左下角的语法树结点的指针。这些属性值的复写次序是这样：从 $T.nptr$（下面的 T）到 $R.i$（下面的 R），到 $R.s$（下面的 R），到 $E.nptr$（下面的 E），到 $T.nptr$（上面的 T），最后到 $R.i$（中间的 R）。

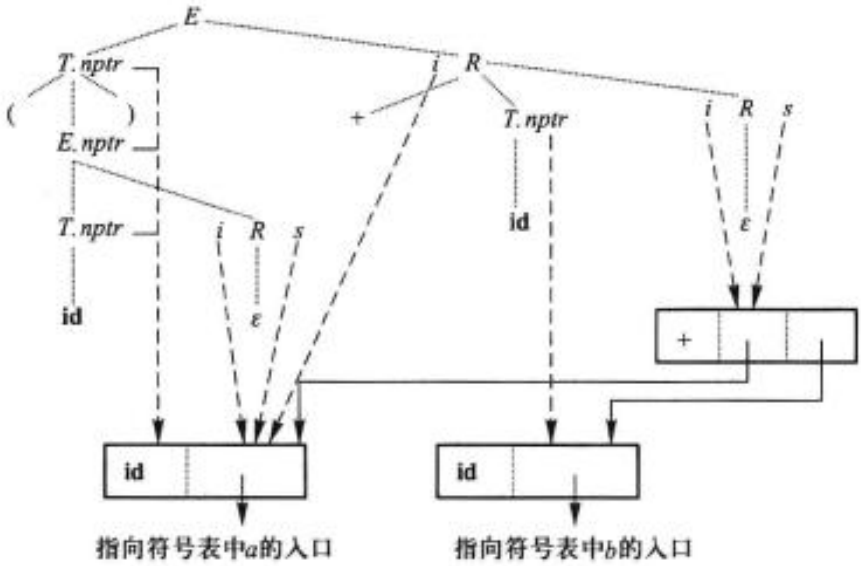


图 4.2 $(a) + b$ 的分析树和语法树

为文法

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

(a) 写一个语法制导定义，它输出括号的对数。

(b) 写一个语法制导定义，它输出括号嵌套的最大深度。

S' —> S	print S.num
S —> (L)	S.num = L.num + 1
S —> a	S.num = 0
L —> L',S	L.num = L'.num + S.num
L —> S	L.num = S.num

为文法

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

(a) 写一个语法制导定义，它输出括号的对数。

(b) 写一个语法制导定义，它输出括号嵌套的最大深度。

S' —> S	print S.depth
S —> (L)	S.depth = L.depth + 1
S —> a	S.depth = 0
L —> L',S	L.depth = max(L'.depth + S.depth)
L —> S	L.depth = S.depth

为下面文法写一个语法制导的定义，它完成一个句子的 **while-do** 最大嵌套层次的计算并输出这个计算结果。

$S \rightarrow E$

$E \rightarrow$ **while** E **do** E | **id** $:= E$ | $E + E$ | **id** | (E)

答　语法制导的定义如下：

$S \rightarrow E$	print($S.loop$)
$E \rightarrow$ **while** E_1 **do** E_2	$E.loop = \max(E_1.loop, E_2.loop) + 1$
$E \rightarrow$ **id** $:= E$	$E.loop = E_1.loop$
$E \rightarrow E_1 + E_2$	$E.loop = \max(E_1.loop, E_2.loop)$
$E \rightarrow$ **id**	$E.loop = 0$
$E \rightarrow (E_1)$	$E.loop = E_1.loop$

非终结符的综合属性 *loop* 用来记住其 **while-do** 的最大嵌套层次。

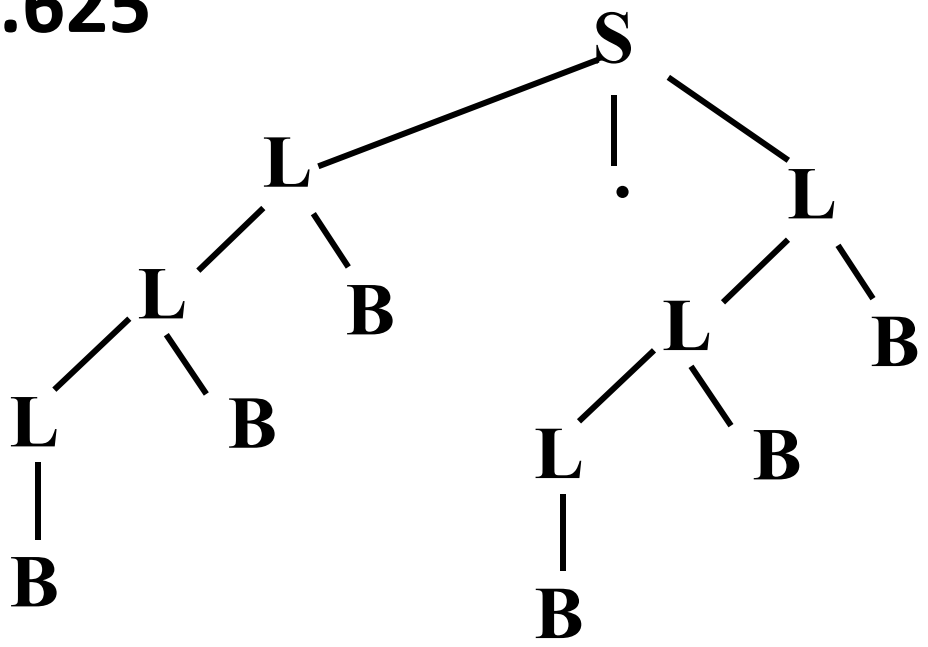
Slide 9 例题三

- 为下面文法写一个语法制导的定义，借助继承属性，给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625

$S \rightarrow L \,.\, L \mid L$

$L \rightarrow L\ B \mid B$

$B \rightarrow 0 \mid 1$



Slide 9 例题三

- S.val 综合属性
- L.val 综合属性
- L.isLeft 继承属性 判断是否在小数点的左边，值为布尔，true-> 整数部分；false ->小数部分
- L.len 综合属性，小数点右边第几位

Slide 9 例题三

产生式	语法规则	
1)	S -> L_1.L_2	S.val = L_1.val + L_2.val/L_2.f
2)	S -> L	S.val = L.val
3)	L -> L_1B	L.val = L_1.val*2 + B.val L.f = L_1.f * 2
4)	L -> B	L.val = B.val L.f = 2
5)	B -> 0	B.val = 0
6)	B -> 1	B.val = 1

下列文法产生由+作用于整常数或实常数的表达式。两个整数相加时，结果是整型，否则是实型。

$E \rightarrow E + T \mid T$

$T \rightarrow$ **num.num** | **num**

扩充(a)的语法制导定义，既决定类型，又把表达式翻译成后缀表示。使用一元算符 **inttoreal** 把整数变成等价的实数，使得后缀式中+的两个对象有同样的类型。

$E \rightarrow E_1 + T$	if (E_1. *type* == *real* && T. *type* == *int*) | E. *type* = *real*; print (T. *lexeme*) ; print ('**inttoreal**') | else if (E_1. *type* == *int* && T. *type* == *real*) | E. *type* = *real*; print ('**inttoreal**') ; print (T. *lexeme*) | else | E. *type* = E_1. *type*; print (T. *lexeme*) | print ('+')
$E \rightarrow T$	E. *type* = T. *type*; print (T. *lexeme*)
$T \rightarrow$ **num**. **num**	T. *type* = *real*; T. *lexeme* = **num**. **num**. *lexeme*
$T \rightarrow$ **num**	T. *type* = *int*; T. *lexeme* = **num**. *lexeme*

由下列文法产生的表达式包括赋值表达式。

$S \rightarrow E$

$E \rightarrow E = E \mid E + E \mid (E) \mid \mathbf{id}$

表达式的语义和 C 语言的一样，即 $b=c$ 是把 c 的值赋给 b 的赋值表达式，而且 $a=(b=c)$ 把 c 的值赋给 b，然后再赋给 a。构造一个语法制导定义，它检查赋值表达式的左部是否为左值。

答 用综合属性 *val* 表示 E 是否为左值表达式，其取值可以是 *lvalue* 和 *rvalue*，分别表示左值表达式和右值表达式。那么所求语法制导定义如下：

$S \rightarrow E$	
$E \rightarrow E_1 = E_2$	if (E_1.*val* == *rvalue*) print ("error"); E.*val* = *rvalue*
$E \rightarrow E_1 + E_2$	E.*val* = *rvalue*
$E \rightarrow (E_1)$	E.*val* = E_1.*val*
$E \rightarrow$ **id**	E.*val* = *lvalue*

分析 这里仍然用综合属性来解决这个问题。在赋值表达式 $E_1 = E_2$ 中，E_1 必须是左值表达式。E_1 是否为左值表达式，从 E_1 本身的结构是可以判断出来的，因此只用综合属性肯定可以求解。

另外，如果 E 是左值表达式，那么 (E) 仍然是左值表达式，这是 C 语言的规定。

程序的文法如下：

$P \rightarrow D$

$D \rightarrow D ; D \mid \mathbf{id} : T \mid \mathbf{proc}\ \mathbf{id} ; D ; S$

(b) 用继承属性 l 表示 D 中声明的变量 **id** 的嵌套深度。

$P \rightarrow \{D.l = 1;\} D$

$D \rightarrow \{D_1.l = D.l;\} D_1 ; \{D_2.l = D.l;\} D_2$

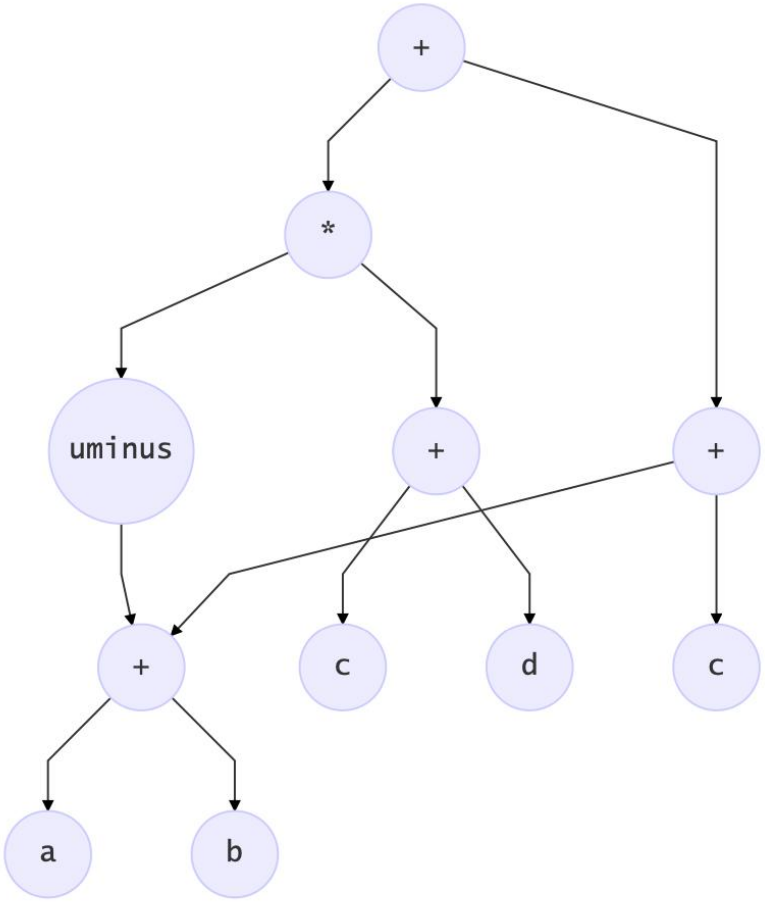
$D \rightarrow \mathbf{id} : T \{\text{print}(\mathbf{id}.name, D.l);\}$

$D \rightarrow \mathbf{proc}\ \mathbf{id} ; \{D_1.l = D.l + 1;\} D_1 ; S$

7.1 把算术表达式-(a+b)*(c+d)+(a+b+c)翻译成：

(b) 有向无环图；

(d) 三地址代码。




```
t1 = a + b
t2 = - t1
t3 = c + d
t4 = t2 * t3
t5 = t1 + c
t6 = t4 + t5
```

❑请结合上述中间代码语法制导定义，为下面的高级语言代码生成符号标号的中间代码

❖假设所有的变量都已经提前定义，信息存在符号表内，可通过lookup查询

```
if(a > 5 and b < 3) then
  i = 0;
  while(i <= 100)
    i = i + 1;
else
  x = x + 5;
```

❖符号标号的中间代码请参考slide 20页

```
        t1 = lookup(a)
        t2 = loopup(b)
        if t1 > 5 goto L.next
        goto L.false
L.next:
        if t2 < 3 goto L.true
        goto L.false
L.true:
        i = 0
while.begin:
        if i <= 100 goto while.true
        goto L.end
while.true:
        t3 = i + 1
        i = t3
        goto while.begin
L.false:
        t4 = lookup(x)
        x = t4 + 5
L.end:
```