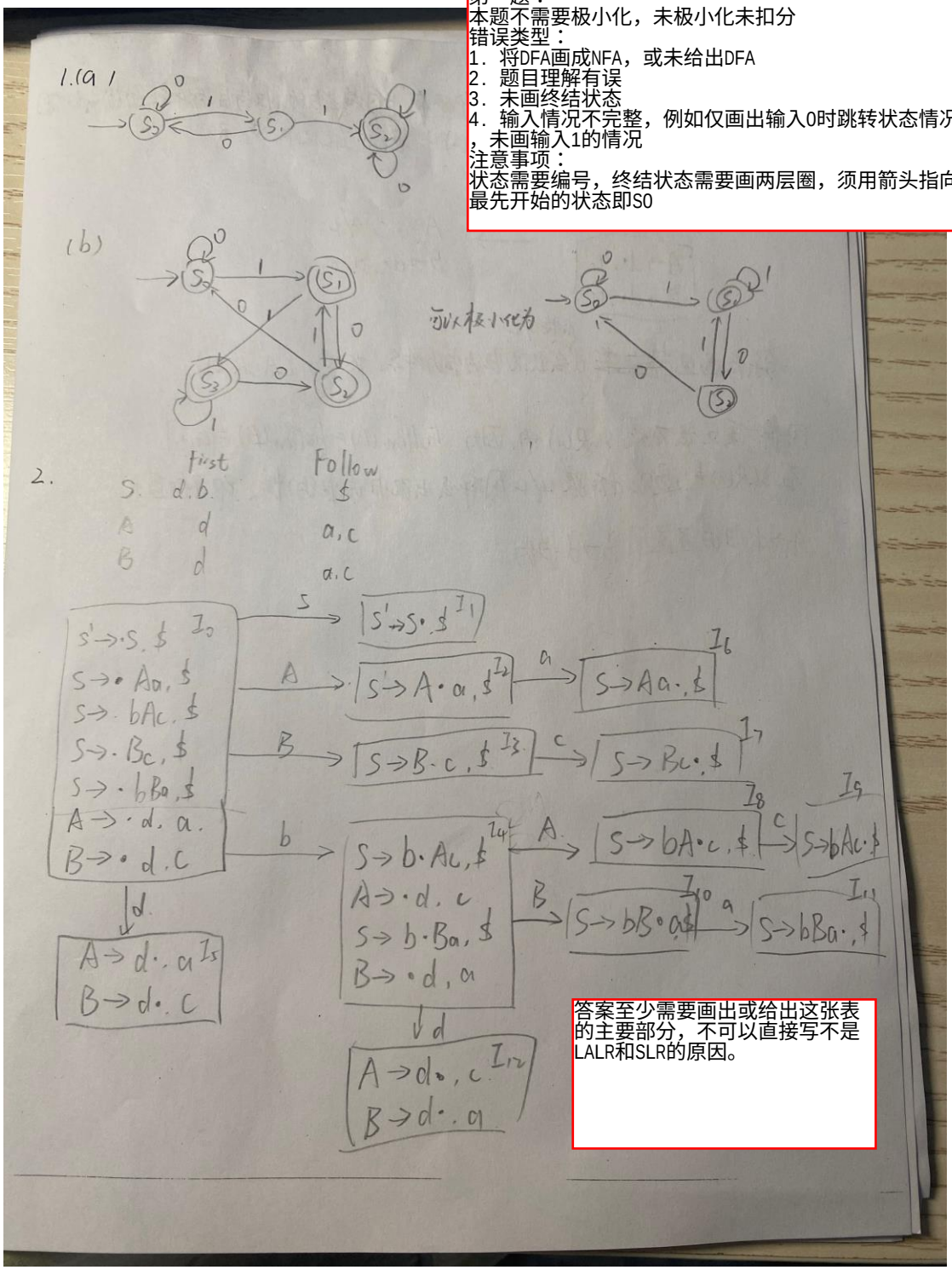
第一题：
本题不需要极小化，未极小化未扣分
错误类型：
1．将DFA画成NFA，或未给出DFA
2．题目理解有误
3．未画终结状态
4．输入情况不完整，例如仅画出输入0时跳转状态情况，未画输入1的情况
注意事项：
状态需要编号，终结状态需要画两层圈，须用箭头指向最先开始的状态即S0



答案至少需要画出或给出这张表的主要部分，不可以直接写不是LALR和SLR的原因。

从上图可以看出，该文法为LR(1)文法，即没有任何移进归约，归约归约冲突，分析表如图，

但该文法不是LALR(1)的，因为LALR(1)中，I_5和I_{12}会和合并，即

$A \to d\cdot, a$
$B \to d\cdot, c$

$A \to d\cdot, c$
$B \to d\cdot, a$

$\longrightarrow$

$A \to d\cdot, a/c$
$B \to d\cdot, a/c$

合并后遇见后接a/c的d会出现归约归约冲突，故不为LALR(1)

同样该文法不是SLR(1)的，因为 $Follow(A) = Follow(B) = \{a, c\}$.

在SLR(1)中，遇见d后跟a/c同样会出现归约归约冲突，即不知道用

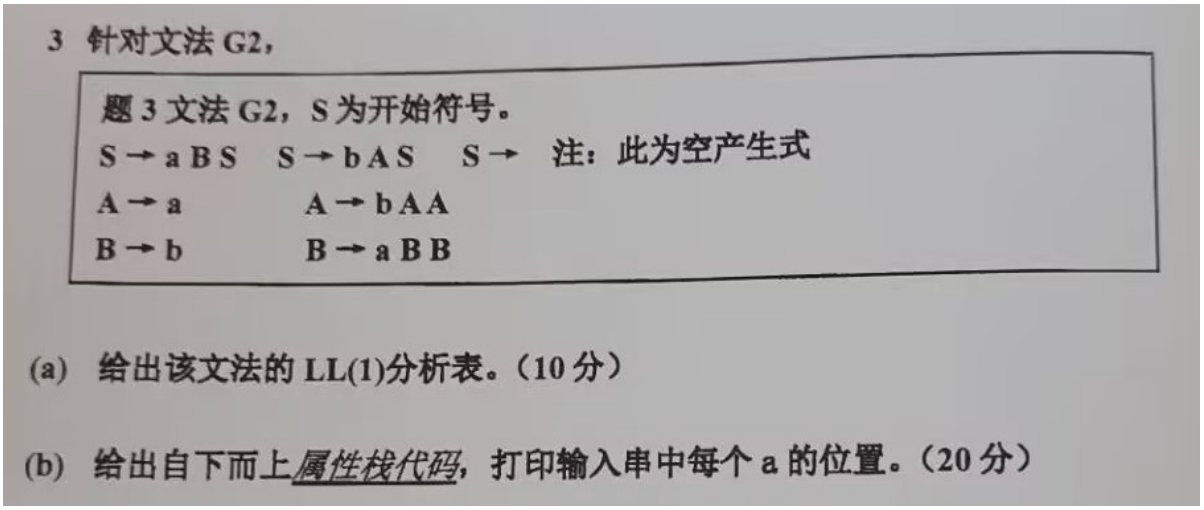
$A \to d$ 归约还是用 $B \to d$ 归约.

建议依次写出不是LALR和SLR的原因。如果合在一起写，这次改卷应该没有扣分，如果扣分来私聊我改下分数。

解析如图.

	a	b	c	d.	$	S	A	B
						1	2	3.
I_0		s4		s5				
I_1					acc			
I_2	s6							
I_3			s7					
I_4							8	10
I_5				s12				
I_6	r5		r6					
I_7					r1			
I_8					r3			
I_9			s9.					
I_{10}					r2			
I_{11}	s11							
I_{12}					r4.			
	r6			r5				

第三题



3 针对文法 G2，

题 3 文法 G2，S 为开始符号。

S → a B S　S → b A S　S →　注：此为空产生式

A → a　　A → b A A

B → b　　B → a B B

(a) 给出该文法的 LL(1)分析表。（10 分）

(b) 给出自下而上*属性栈代码*，打印输入串中每个 a 的位置。（20 分）

(a)

算法 3.2　构造预测分析表。

输入　文法 G。

输出　分析表 M。

方法　对文法的每个产生式 $A \to \alpha$，执行(1)和(2)。

(1) 对 $FIRST(\alpha)$的每个终结符 a，把 $A \to \alpha$ 加入 $M[A, a]$。

(2) 如果 ε 在 $FIRST(\alpha)$中，对 $FOLLOW(A)$的每个终结符 b(包括 \$)，把 $A \to \alpha$ 加入 $M[A, b]$(包括 $M[A, \$]$)。

M 剩下的条目没有定义，都是出错条目，通常用空白表示。 □

《编译原理》 陈意云、张昱 第3版 P60

$First(aBS) = \{a\}$　$First(bAS) = \{b\}$　$First(\epsilon) = \{\epsilon\}$

$First(a) = \{a\}$　$First(bAA) = \{b\}$

$First(b) = \{b\}$　$First(aBB) = \{a\}$

$Follow(S) = \{\$\}$

非终结符	输入符号		
	a	b	$
S	S→aBS	S→bAS	S→ϵ
A	A→a	A→bAA	
B	B→aBB	B→b	

(b)

先写出翻译方案如下（翻译方案有很多种写法，这是其中一种）：

其中before是继承属性，表示文法符号前的字符数；out是综合属性，表示这个文法符号推出的字符总数。

$$
\begin{aligned}
S' \rightarrow & \{S.before = 0\} S \\
S \rightarrow & b \ \{A.before = S.before + 1\} \\
& A \ \{S_1.before = A.before + A.out\} \\
& S_1 \ \{S.out = 1 + A.out + S_1.out\} \\
S \rightarrow & a \ \{print(S.before + 1); \\
& B.before = S.before + 1\} \\
& B \ \{S_1.before = B.before + B.out\} \\
& S_1 \{S.out = 1 + B.out + S_1.out\} \\
S \rightarrow & \epsilon \ \{S.out = 0\} \\
A \rightarrow & a \ \{print(A.before + 1); \\
& A.out = 1\} \\
A \rightarrow & b \ \{A_1.before = A.before + 1\} \\
& A_1 \ \{A_2.before = A_1.before + A_1.out\} \\
& A_2 \ \{A.out = 1 + A_1.out + A_2.out\} \\
B \rightarrow & b \ \{B.out = 1\} \\
B \rightarrow & a \ \{print(S.before + 1); \\
& B_1.before = B.before + 1\} \\
& B_1 \ \{B_2.before = B_1.before + B_1.out\} \\
& B_2 \ \{B.out = 1 + B_1.out + B_2.out\}
\end{aligned}
$$

引入标记非终结符M, N, O，模拟继承属性的计算：

翻译方案	属性栈代码
$S' \rightarrow M \ \{S.before = M.s\}$	
$\quad S$	
$M \rightarrow \epsilon \ \{M.s = 0\}$	$Stack[ntop] = 0$
$S \rightarrow b \ \{N.i = S.before\}$	
$\quad N \ \{A.before = N.s\}$	
$\quad A \ \{O.i = A.before + A.out\}$	
$\quad O \ \{S_1.before = O.s\}$	
$\quad S_1 \ \{S.out = 1 + A.out + S_1.out\}$	$Stack[ntop] = 1 + Stack[top-2] + Stack[top]$
$N \rightarrow \epsilon \ \{N.s = N.i + 1\}$	$Stack[ntop] = Stack[top-1] + 1$
$O \rightarrow \epsilon \ \{O.s = O.i\}$	$Stack[ntop] = Stack[top-1] + Stack[top]$
$S \rightarrow a \ \{print(S.before + 1);$	$print(Stack[top-1] + 1)$
$\quad N.i = S.before\}$	
$\quad N \ \{B.before = N.s\}$	
$\quad B \ \{O.i = B.before + B.out\}$	
$\quad O \ \{S_1.before = O.s\}$	
$\quad S_1 \{S.out = 1 + B.out + S_1.out\}$	$Stack[ntop] = 1 + Stack[top-2] + Stack[top]$
$S \rightarrow \epsilon \ \{S.out = 0\}$	$Stack[ntop] = 0$
$A \rightarrow a \ \{print(A.before + 1);$	$print(Stack[top-1] + 1)$
$\quad A.out = 1\}$	$Stack[ntop] = 1$
$A \rightarrow b \ \{N.i = A.before\}$	
$\quad N \ \{A_1.before = N.s\}$	
$\quad A_1 \ \{O.i = A_1.before + A_1.out\}$	
$\quad O \ \{A_2.before = O.s\}$	
$\quad A_2 \ \{A.out = 1 + A_1.out + A_2.out\}$	$Stack[ntop] = 1 + Stack[top-2] + Stack[top]$

$B \rightarrow b \ \{B.out = 1\}$ $\qquad Stack[ntop] = 1$

$B \rightarrow a \ \{print(S.before + 1);$ $\qquad print(Stack[top - 1] + 1)$

$\quad N.i = B.before\}$

$\quad N \ \{B_1.before = N.s\}$

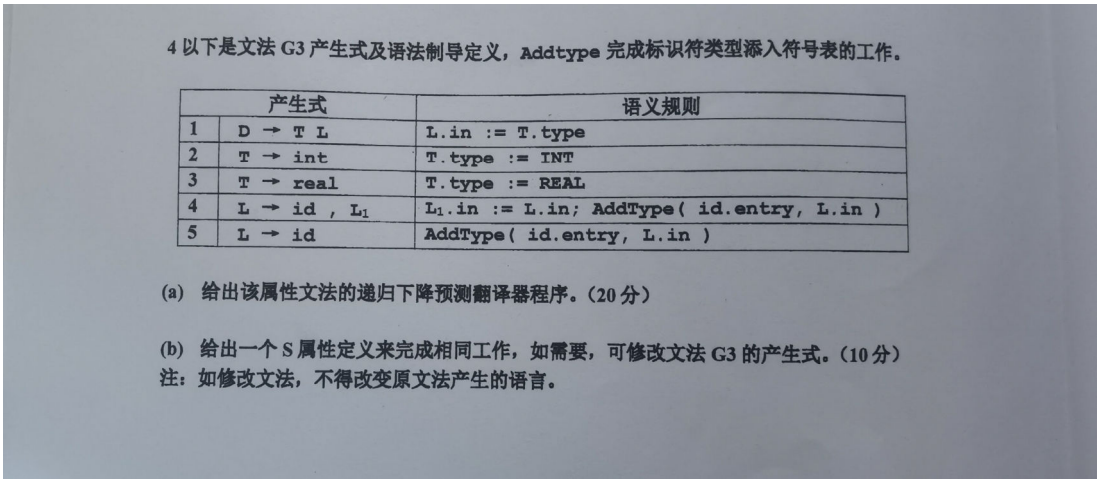
$\quad B_1 \ \{O.i = B_1.before + B_1.out\}$

$\quad O \ \{B_2.before = O.s\}$

$\quad B_2 \ \{B.out = 1 + B_1.out + B_2.out\}$ $\qquad Stack[ntop] = 1 + Stack[top - 2] + Stack[top]$

> 参考资料：https://blog.csdn.net/weixin_56462041/article/details/129015814

编译原理小测第四题参考答案



4 以下是文法 G3 产生式及语法制导定义，Addtype 完成标识符类型添入符号表的工作。

	产生式	语义规则
1	D → T L	L.in := T.type
2	T → int	T.type := INT
3	T → real	T.type := REAL
4	L → id , L_1	L_1.in := L.in; AddType(id.entry, L.in)
5	L → id	AddType(id.entry, L.in)

(a) 给出该属性文法的递归下降预测翻译器程序。(20 分)

(b) 给出一个 S 属性定义来完成相同工作，如需要，可修改文法 G3 的产生式。(10 分)
注：如修改文法，不得改变原文法产生的语言。

(a)、

```
node_ptr D() {
    node_ptr t_ptr, l_ptr, s;
    // D -> T L
    if (lookahead == "int" || lookahead == "real") {
        t_ptr = T();                // 调用 T 以确定类型
        L_in = T_type;              // 语义动作：将 T.type 赋给 L.in
        l_ptr = L(L_in);            // 调用 L 处理标识符列表
        s = l_ptr;                  // 语义动作：传递结果
    } else {
        throw runtime_error("Syntax Error in D");
    }
    return s;
}

// 解析 T -> int | real
node_ptr T() {
    node_ptr s;
    if (lookahead == "int") {
        match("int");
        T_type = "INT";             // 语义动作
        s = mkleaf("int");          // 构造叶节点
    } else if (lookahead == "real") {
        match("real");
        T_type = "REAL";            // 语义动作
        s = mkleaf("real");         // 构造叶节点
    } else {
        throw runtime_error("Syntax Error in T");
    }
    return s;
}
```

```
// 解析 L -> id, L1 | id
node_ptr L(string L_in) {
    node_ptr id_ptr, l1_ptr, s;
    if (lookahead == "id") {
        string id_entry = current_id; // 保存标识符
        id_ptr = mkleaf("id", id_entry);  // 构造叶节点
        match("id");
        if (lookahead == ",") {          // 对应 L -> id, L1
            match(",");
            AddType(id_entry, L_in);      // 语义动作
            l1_ptr = L(L_in);            // 递归处理 L1
            s = mknode(",", id_ptr, l1_ptr); // 语义动作：构造节点
        } else {                         // 对应 L -> id
            AddType(id_entry, L_in);      // 语义动作
            s = id_ptr;                  // 语义动作：返回 id 节点
        }
    } else {
        throw runtime_error("Syntax Error in L");
    }
    return s;
}
```

(b)

5.6 重写表 5-8 中语法制导定义的基础文法，使得类型信息仅用综合属性来传递。

表 5-8 利用继承属性 *L.in* 传递类型信息的语法制导定义

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type =$ integer
$T \rightarrow$ real	$T.type =$ real
$L \rightarrow L_1$, id	$L_1.in = L.in$ $addtype(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addtype(\text{id}.entry, L.in)$

解答：

表 5-8 所示是利用继承属性 *L.in* 传递类型信息的语法制导定义，其基础文法产生的句子形如 C 语言中简单变量的声明语句，如 int a，b，c。

为了将类型关键字 int 或 real 定义的类型信息和后面声明的变量名联系在一起，还要避免使用继承属性，可将文法改写为

$D \rightarrow L$

$L \rightarrow L_1$, id

$L \rightarrow T$ id

$T \rightarrow$ int

$T \rightarrow$ real

这样，重写后的语法制导定义如表 5-9 所示。

表 5-9 仅用综合属性来传递类型信息的语法制导定义

产 生 式	语 义 规 则
$D \rightarrow L$	
$L \rightarrow L_1$, id	$L.type = L_1.type$ addtype(id.*entry*, $L_1.type$)
$L \rightarrow T$ id	$L.type = T.type$ addtype(id.*entry*, $T.type$)
$T \rightarrow$ int	$T.type =$ integer
$T \rightarrow$ real	$T.type =$ real

4.18 C 语言和 Pascal 语言的变量声明的文法分别列在下面左右两边：

$D \rightarrow TL$	$D \rightarrow L : T$
$T \rightarrow$ **int**	$L \rightarrow L$, **id**
$T \rightarrow$ **real**	$L \rightarrow$ **id**
$L \rightarrow L$, **id**	$T \rightarrow$ **integer**
$L \rightarrow$ **id**	$T \rightarrow$ **real**

分别为它们构造一个翻译方案，把标识符的类型填入符号表中。

答 C 语言的翻译方案如下，其中 **id** 的 *entry* 属性是符号表条目的指针，过程 *addType* 是将类型信息填入符号表。

$D \rightarrow T \{L.in = T.type;\} L$

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

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

$L \rightarrow \{L_1.in = L.in;\} L_1$, **id** $\{addType$ (**id**.*entry*, $L.in)$;$\}$

$L \rightarrow$ **id** $\{addType$ (**id**.*entry*, $L.in)$;$\}$

对于 Pascal 语言，先将文法改写如下：

$D \rightarrow$ **id** L

$L \rightarrow$, **id** $L \mid : T$

$T \rightarrow$ **integer** | **real**

下面给出改写文法的翻译方案：

$D \rightarrow$ **id** $L \{addType$ (**id**.*entry*, $L.type)$;$\}$

$L \rightarrow$, **id** $L_1 \{L.type = L_1.Type; addType$ (**id**.*entry*, $L_1.type)$;$\}$

$L \rightarrow : T \{L.type = T.type;\}$

$T \rightarrow$ **integer** $\{T.type = integer;\}$

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

分析 先说明为什么要重新设计 Pascal 语言的变量声明的文法。Pascal 语言变量声明的形式和 C 语言对应形式的重要区别是，Pascal 语言的类型信息出现在变量的后面。从产生式 $D \rightarrow L : T$ 看，类型信息要从右边流向左边。因此针对该文法是写不出 L 属性定义的，因为非终结符 L 的属性计算需要它右边的符号 T 的属性。

如何克服困难？从实现该翻译方案的手工编程所用的数据结构可以得到一点启发。由于不知道 L 会推出多少个变量标识符，因此很可能把读到的变量标识符都压到一个栈中。然后当有了类型信息后，再退栈并把类型信息填入符号表。那么，如何让翻译方案也体现这种先压栈后退栈的操作顺序呢？自下而上的分析是移进-归约方式，如果让读到标识符时都做移进动作，读到类型时才开始做归约动作，那就能把类型信息填入符号表。因此将文法重新设计，一直到句子全读完时才做归约。从图 4.3 的 **id**, **id**, **id** : **real** 的分析树可以看出这一点。

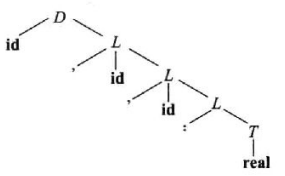


图 4.3 **id**, **id**, **id** : **real** 的分析树

可以这样小结，如果在句子中，信息是从右向左流动的，那么文法应该设计成从右向左归约（如图 4.3 所示），此时属性计算很可能只用综合属性就能完成，就像上面改造后的 Pascal 语言变量声明的翻译方案那样。

S 属性定义容易写，也容易理解，而且像 Yacc 这样的生成器就能胜任处理 *S* 属性定义的工作。那么，上面 C 语言变量声明的翻译方案能否也通过改造文法而写出只用综合属性的翻译方案？回答是肯定的。先将文法改造如下：

$D \rightarrow D$, **id**

$D \rightarrow T$ **id**

$T \rightarrow$ **int**

$T \rightarrow$ **real**

变量声明 **int id**, **id**, **id** 的分析树见图 4.4。然后加上语义动作而构成的翻译方案如下：

$D \rightarrow D_1$, **id**　　{ *addType* (**id**. *entry*, D_1. *type*); *D*. *type* = D_1. *type*; }

$D \rightarrow T$ **id**　　{ *addType* (**id**. *entry*, *T*. *type*); *D*. *type* = *T*. *type*; }

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

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

上面的小结需要进行补充。如果在句子中，信息是从左向右流动的，那么文法应该设计成从左向右归约（如图 4.4 所示），此时属性计算一般都可以只用综合属性完成，上面改造后的 C 语言变量声明的翻译方案就是这样。

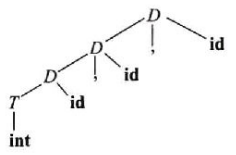


图 4.4　**int id**, **id**, **id** 的分析树

把这两个例子结合起来得到的一点启示是，在设计语言时，要考虑让归约方向和句子中信息流动的方向尽量保持一致，这时语法制导定义比较容易设计。

也许有人会说，改造后的文法可读性较差。这没有关系，语言实现时用的文法并非一定要和用户使用的文法一样。