第7章 语法制导的语义计算

一个程序是正确的，应包含两方面的含义：

1）语法上是正确的

2）语义上是正确的

紧接在词法分析和语法分析之后，编译程序要做的工作是进行静态语义检查和翻译。

语义分析的概念

简单说，语义分析就是分析语法结构的含义。

语义分析的基本任务：

(1) 类型的确定：确定源程序中标识符所关联数据对象的类型

(2) 类型的检查：对运算及运算分量进行检查。

(3) 确认含义：确认程序中各构造成分组合到一起的含义。

(4) 其它语义检查：如不允许从循环体外转入循环体内。

语法分析—建立语法分析树

语义分析---遍历语法分析树

语法制导翻译---建立与遍历同时完成

语法制导翻译是目前最常用的语义分析技术

属性文法

属性：

现实中的属性常用以描述事物或人的特征、性质、品质等等。如对一个物体，可以有一个颜色属性，对一个人，可以有一个身高的属性。

在编译中，针对语义，可以为文法符号设置属性。

形式上讲，一个属性文法是一个三元组

$$A=(G, V, F)$$

G: 一个上下文无关文法

V: 一个属性的有限集合

F: 关于属性的断言或谓词的有限集合。

每个属性与文法的某个非终结符或终结符相联。

每个断言与文法的产生式相联。

属性文法的例子1

例如：

E→T + T| T or T

T→num| true | false

假设对非终结符T设置了属性t，它表示对应语法成分的类型。则可以有如下的属性文法（用于对表达式的运算做类型检查）：

$E \rightarrow T^1 + T^2$	{ $T^1.t=int$ AND $T^2.t=int$}
$E \rightarrow T^1$ or T^2	{ $T^1.t=bool$ AND $T^2.t=bool$}
T→num	{ T.t :=int }
T→true	{ T.t :=bool }
T→false	{ T.t :=bool }

在上面的属性文法中，与E相联的断言指明：两个T的属性必须相同。

属性文法的例子2

属性文法由Donald E.Knuth提出，他把属性分成两类：继承属性和综合属性。

在编译的许多实际应用中，属性和断言以多种形式出现，也就是说，与每个文法符号相联的可以是各种属性、断言、以及语义规则，或某种程序设计语言的程序段等。

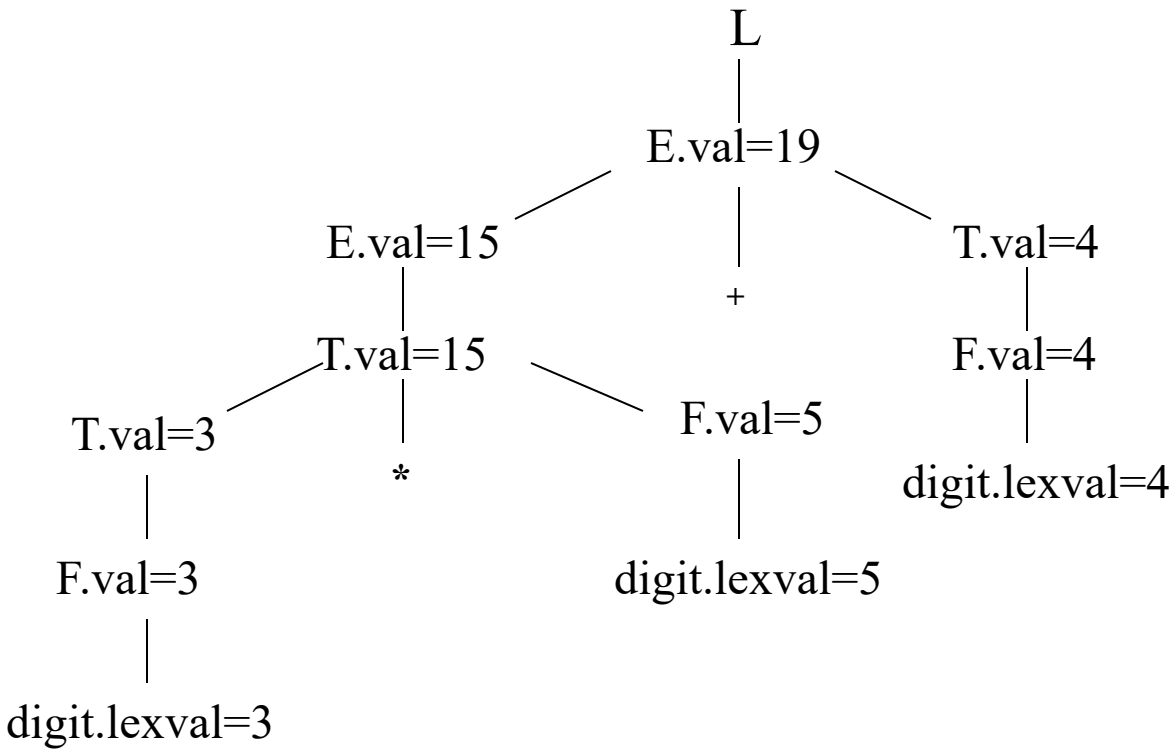
下面是简单算术表达式求值的语义描述：

产生式	语义规则
L→E	{ print(E.val) }
$E \rightarrow E^1 + T$	{ $E.val=E^1.val + T.val$ }
E→T	{ E.val=T.val }
$T \rightarrow T^1 * F$	{ $T.val=T^1.val * F.val$ }
T→F	{ T.val=F.val }
F→(E)	{ F.val=E.val }
F→digit	{ F.val=digit.lexval }

每个非终结符都有一个属性val。对每个产生式来说，它的左部E、T、F的属性的值的计算来自它右部的非终结符，这种属性称作综合属性。单词digit仅有综合属性，它的值由词法分析程序提供。和产生式L→E相联的语义规则是一个过程，打印由E产生的表达式的值。

L的属性可以看作是空的或虚的。

设表达式为3＊5+4，则语义动作打印数值19



3*5+4的带注释的分析树

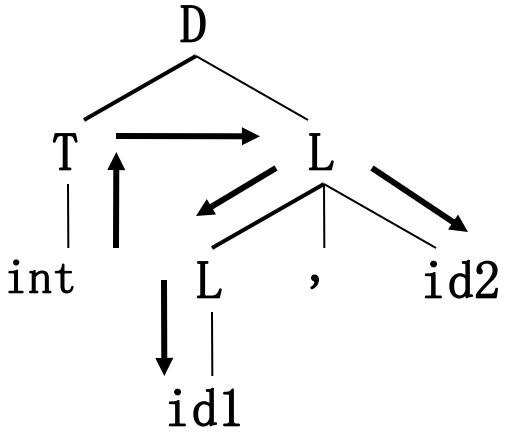
属性文法的例子3

C中的一个说明语句有如下的形式

int a, b, c ;

下面是描述说明语句中各种变量的类型信息的语义规则。

产生式	语义规则
D→TL	L.in := T.type
T→int	T.type := integer
T→real	T.type := real
$L \rightarrow L^1$, id	L^1.in := L.in
	addtype(id.entry, L.in)
L→id	addtype(id.entry, L.in)

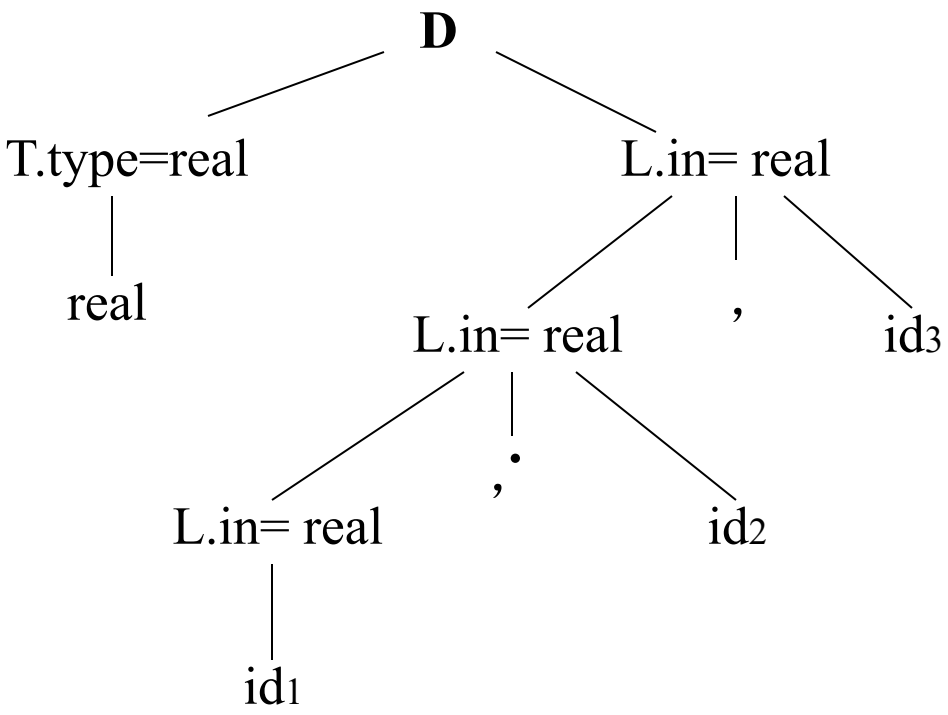


句子 int id1, id2 的语法树。

上面关于说明的属性文法中，非终结符T有综合属性type，其值由说明中的关键字（这里是int或real）决定。L有继承属性in。L.in:=T.type设置L.in为类型type.然后，通过语义动作把继承属性L.in沿着语法树往下传。与L相关的语义动作包含过程addtype,它把每个标识符的类型信息type填入符号表的相应项里。

Real id1,id2,id3

每个L结点都带有继承属性的语法树



综合属性与继承属性

在一个语法制导定义中，$\forall A \rightarrow \alpha \in P$ 都有与之相关联的一套语义规则，规则可表示为：$b = f(c_1, c_2, \ldots, c_k)$

其中：f是一个函数，且满足下面两种情况之一：

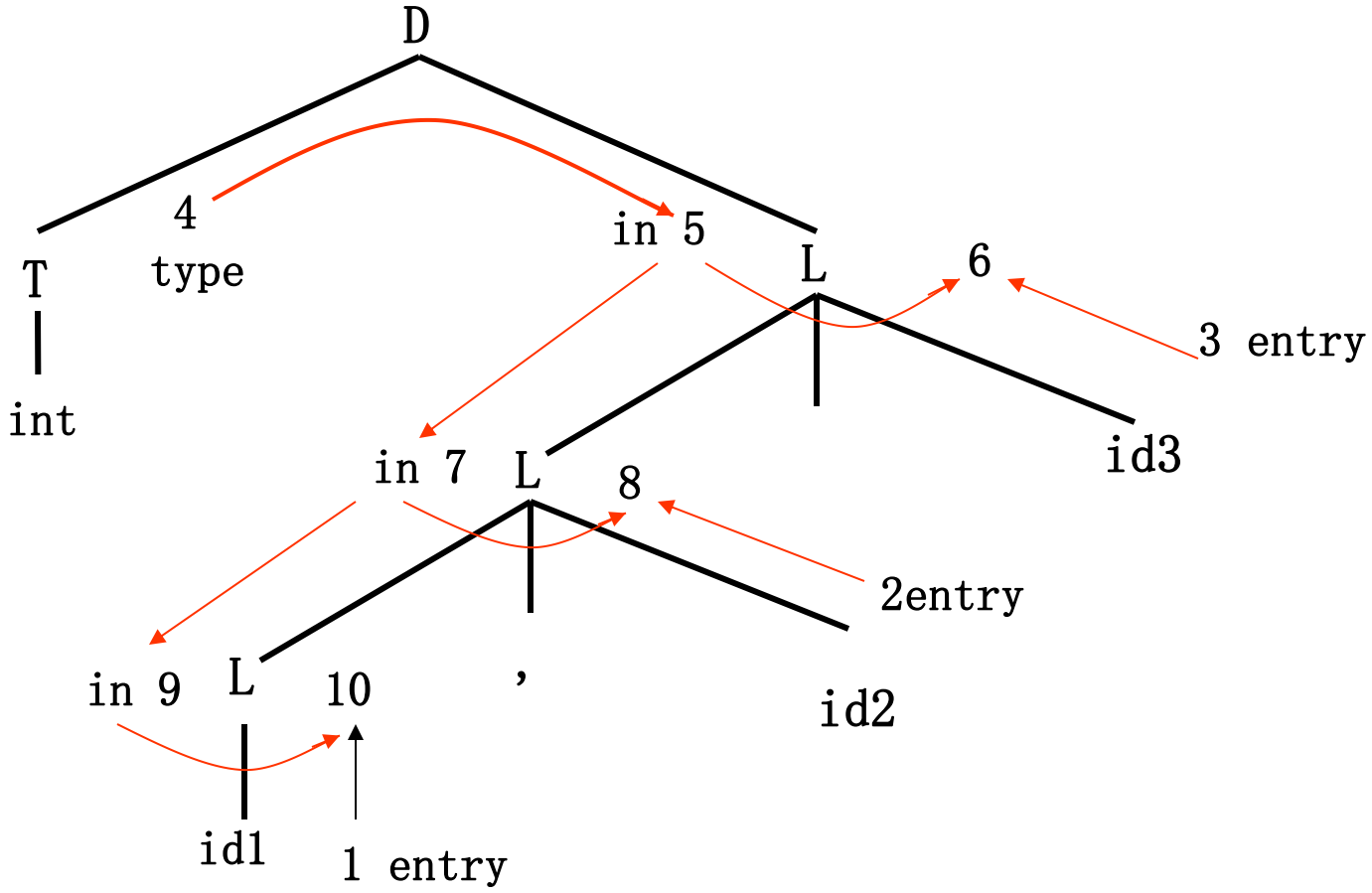
1. b是A属性，c1,c2,…,ck是α中的文法符号的属性，或者A的其它属性，则称b是A的综合属性；

2. $c_1,c_2,\ldots,c_k$是A或α中的任何文法符号的属性,则称b是α中的符号的一个继承属性。

在两种情况下，都说属性b依赖于属性c_1，c_2，…，c_k。

语法制导翻译概论

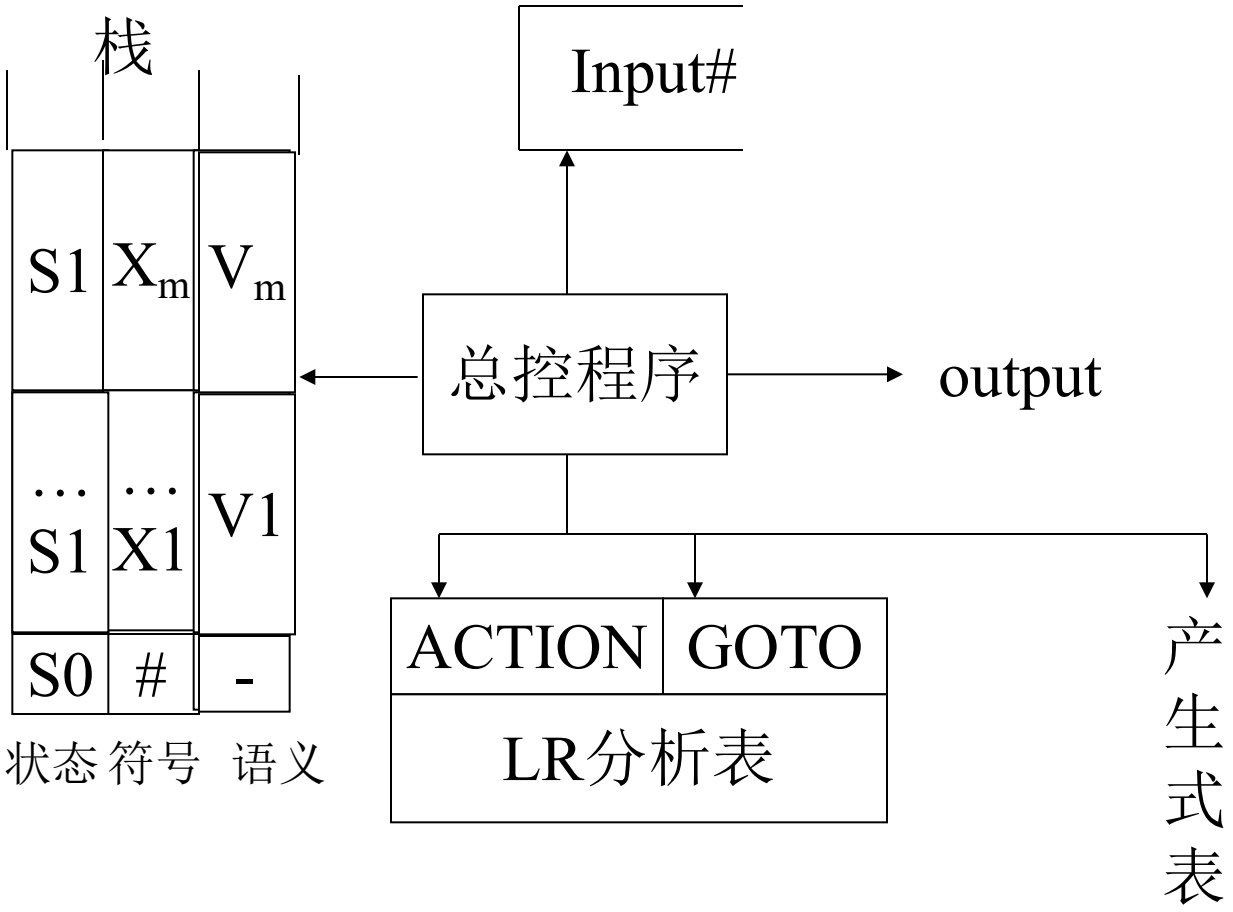
语法制导的一般概念：对单词符号串进行分析，构造语法分析树，然后根据需要构造属性依赖图，遍历语法树并在语法树的各结点处按语义规则进行计算。



产生式	语义规则
0）L→E	print（E.val）
1）E→E¹＋T	E.val：＝E¹.val＋T.val
2）E→T	E.val：＝T.val
3）T→T¹*F	T.val：＝T¹.val×F.val
4）T→F	T.val：＝F.val
5）F→（E）	F.val：＝E.val
6）F→digit	F.val:＝digit.lexval

LR分析器可以改造为一个翻译器**增加语义栈**，
在对输入串进行语法分析的同时对属性进行计算。

ＬＲ分析器模型


栈
Input#
S1
X_m
V_m
总控程序
output
…
S1
…
X1
V1
S0
#
-
状态 符号 语义
ACTION
GOTO
LR分析表
产生式表

2 + 3 * 5 的分析和计值过程

步骤	动作	状态栈	语义栈(值栈)	符号栈	余留输入串
1）		0	－	#	2＋3* 5 #
2）		0 5	－－	# 2	＋3 * 5 #
3）	r 6	0 3	－2	# F	＋3 * 5 #
4）	r 4	0 2	－2	# T	＋3 * 5 #
5）	r 2	0 1	－2	# E	＋3 * 5 #
6）		0 1 6	－2－	# E＋	3 * 5 #
7）		0 1 6 5	－2－－	# E＋3 玳	* 5 #
8）	r 6	0 1 6 3	－2－3	# E＋F 玳	* 5 #
9）	r 4	0 1 6 9	－2－3	# E＋T 玳	* 5 #
10）		0 1 6 9 7	－2－3－	# E＋T*	玳 5 #
11）		0 1 6 9 7 5	－2－3－－	# E＋T* 5	#
12）	r 6	0 1 6 9 7（1 0）	－2－3－5	# E＋T* F	#
13）	r 3	0 1 6 9	－2－（1 5）	# E＋T	#
14）	r 1	0 1 拟	－（1 7）	# E	#
15）	接受				

BOTTOM—UP

语义处理是作类型检查，对二目运算符的运算对象进行类型匹配审查。

(LR分析）：增加语义栈 归约时进行语义动作.

例5 G[E]:

(1) E →T+T

(2) E →T or T

(3) T →n

(4) T →b

```
E→T1 + T2
    { if  T1.type = int  and  T2.type= int
      then  E.type :=int
      else  error}
 E →T1 or T2
   { if  T1.type = bool and T2.type= bool
      then   E.type :=bool
      else  error}
T → n   { T.type :=  int}
T → b   { T.type := bool}
```

w =n + n#

4	n	--
o	#	--

2	T	int
o	#	--

5	+	--
2	T	int
o	#	--

4	n	--
5	+	--
2	T	int
o	#	--

6	T	int
5	+	--
2	T	int
o	#	--

1	E	int
0	#	--

G[E]:
(1) E →T+T
(2) E →T or T
(3) T →n
(4) T →b

G[E]:的 LR(0)分析表

状态	action					GOTO	
	+	o	n	b	#	E	T
0			S4	S3		1	2
1					acc		
2	s5	s7					
3	r4	r4	r4	r4	r4		
4	r3	r3	r3	r3	r3		
5			s4	s3			6
6	r1	r1	r1	r1	r1		
7			s4	s3			8
8	r2	r2	r2	r2	r2		

w =n +b

4	n	--
o	#	--

2	T	int
o	#	--

5	+	--
2	T	int
o	#	--

3	b	--
5	+	---
2	T	int
o	#	--

6	T	bool
5	+	--
2	T	int
o	#	--

1	E	error
o	#	--

G[E]:

(1) E →T+T

(2) E →T or T

(3) T →n

(4) T →b

G[E]:的 LR(0)分析表

	action					GOTO	
状态	+	o	n	b	#	E	T
0			S4	S3		1	2
1					acc		
2	s5	s7					
3	r4	r4	r4	r4	r4		
4	r3	r3	r3	r3	r3		
5			s4	s3			6
6	r1	r1	r1	r1	r1		
7			s4	s3			8
8	r2	r2	r2	r2	r2		

语义规则

语义规则也叫语义子程序或语义动作

语义规则通常有两种表现形式：

语法制导定义和翻译模式

语法制导定义

语法制导定义是关于语言翻译高层次规格说明，它隐藏了具体实现细节，使用户不用显式地说明翻译发生的顺序

例：下面是将中缀表达式转化为后缀表达式的文法和相应的语法制导定义

产生式	语法制导定义
L→ E	print(E.val)
E →E1+E2	E.val=E1.val||E2.val||'+'
E →digit	E.val=Digit.lexval

语法制导定义 只考虑“做什么”，用抽象的属性表示文法符号所代表的语义

翻译模式

翻译模式也叫翻译方案

一个翻译模式是一个上下文无关文法，其中被称为语义动作的程序段被嵌入到产生式的右部。

一个翻译模式类似于语法制导定义，只是语义规则的计算顺序是显式给出的。

这是一种语法分析和语义动作交错的表示法，他表达在按深度优先遍历分析树的过程中何时执行语义动作。

翻译模式给出了使用语义规则进行计算的顺序。可看成是分析过程中翻译的注释。

例2：

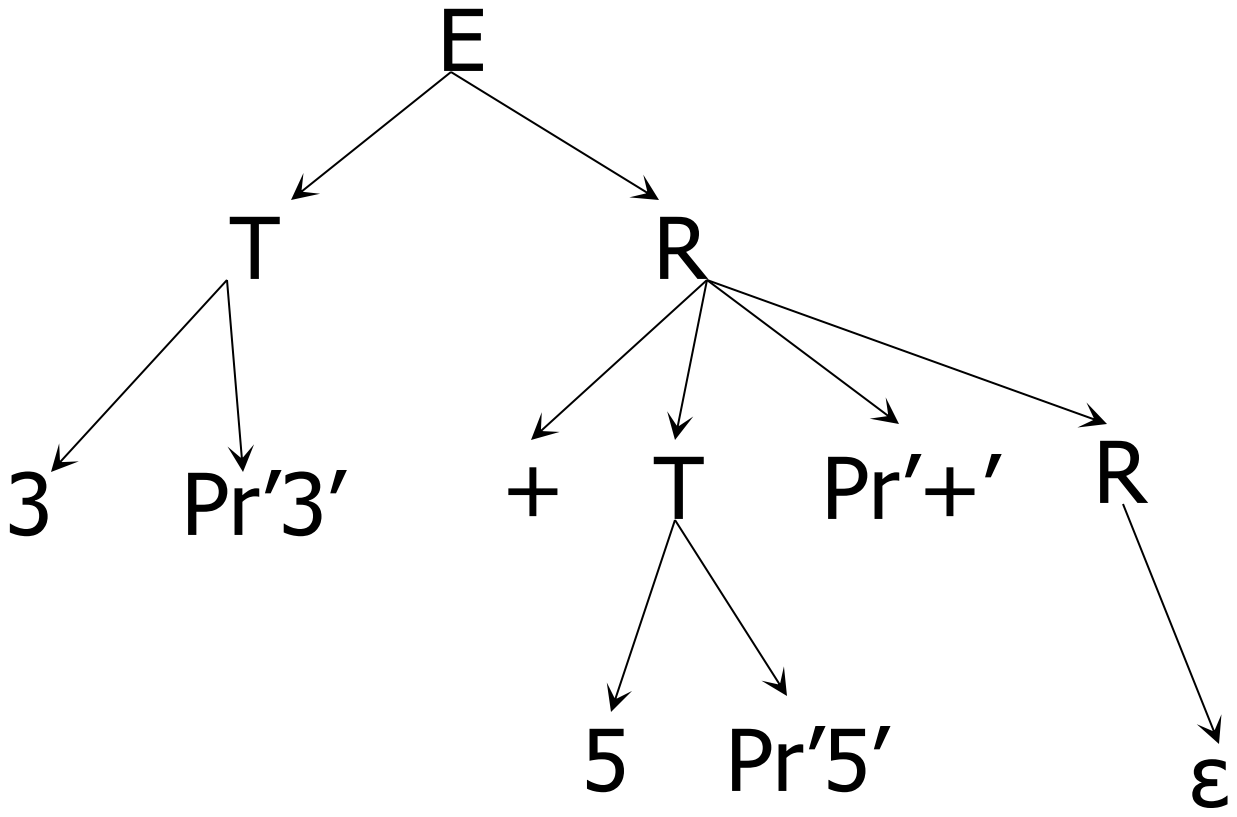
一个简单的翻译模式（中缀变后缀）

$E \rightarrow TR$

$R \rightarrow addop\ T$ {print(addop.lexeme)} $R_1 \mid \varepsilon$

$T \rightarrow num$ {print(num.val)}

3+5的语义翻译过程



结果：35+

翻译方案不仅要考虑“做什么”，还要考虑“怎么做”

某种意义上说，语法制导定义类似于算法，而翻译方案更象程序

例：文法G的产生式如下：

S→（L）

S →a

L →L, S

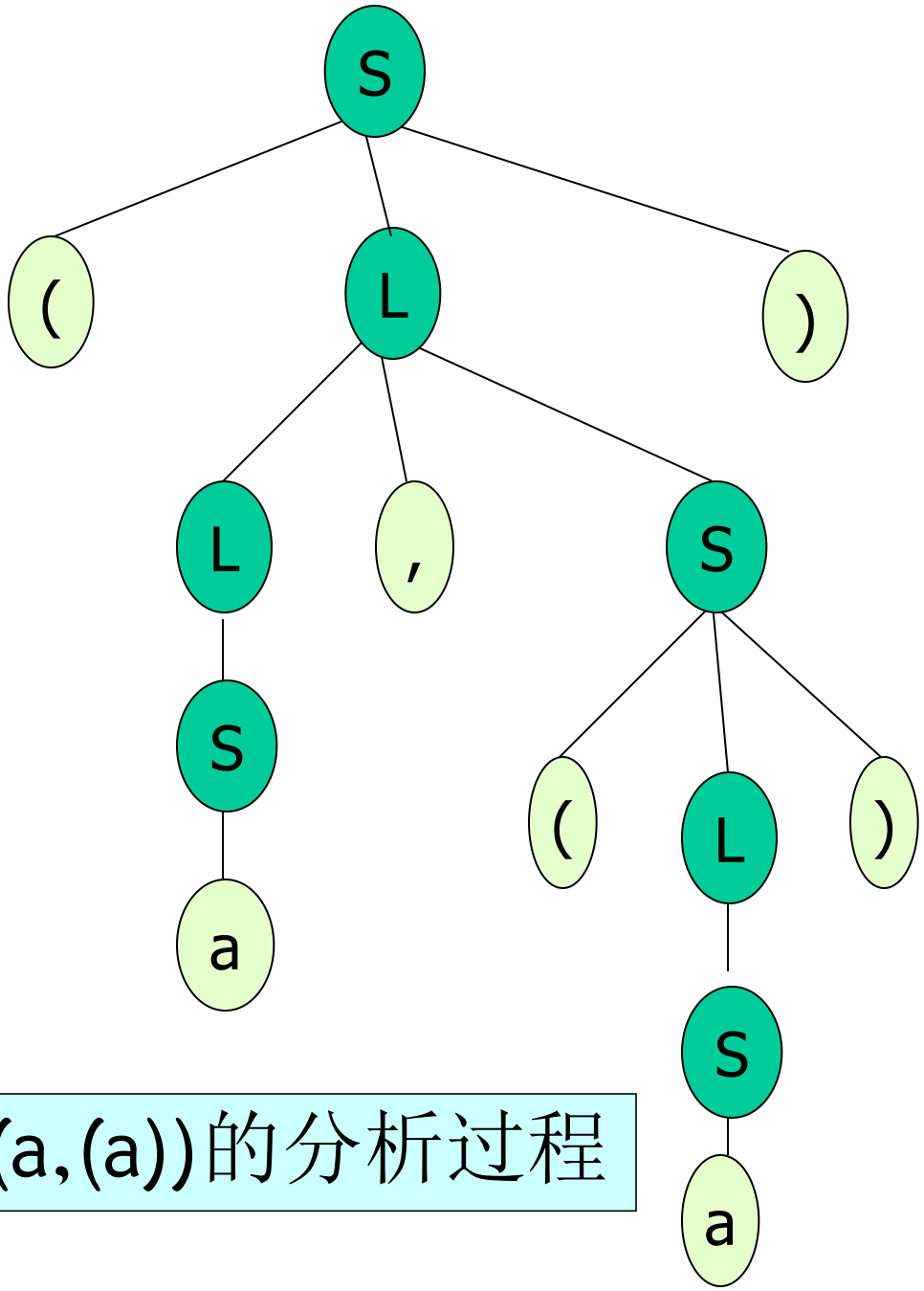
L →S

1.试写出一个语法制导定义，输出配对括号个数

2.写一个翻译方案，打印每个a的嵌套深度

解：1.为S，L引入属性h

产生式	语法制导定义
S→（L）	S.h=L.h+1
S →a	S.h=0
L →L1,S	L.h=L1.h+S.h
L →S	L.h=s.h
S’ →S	print(S.h)



S.h=0

L.h=0

S.h=0

L.h=0

S.h=1

L.h=1

S.h=2

(a,(a))的分析过程

2.为S，L引入属性d,翻译方案如下

S’ →{S.d=0} S

S→（ {L.d=S.d+1} L）

S →a {print(s.d)}

L →{L1.d=L.d} L1, {S.d=L.d}S

L →{S.d=L.d} S

(a,(a))的分析过程