第7章

语法制导的语义计算

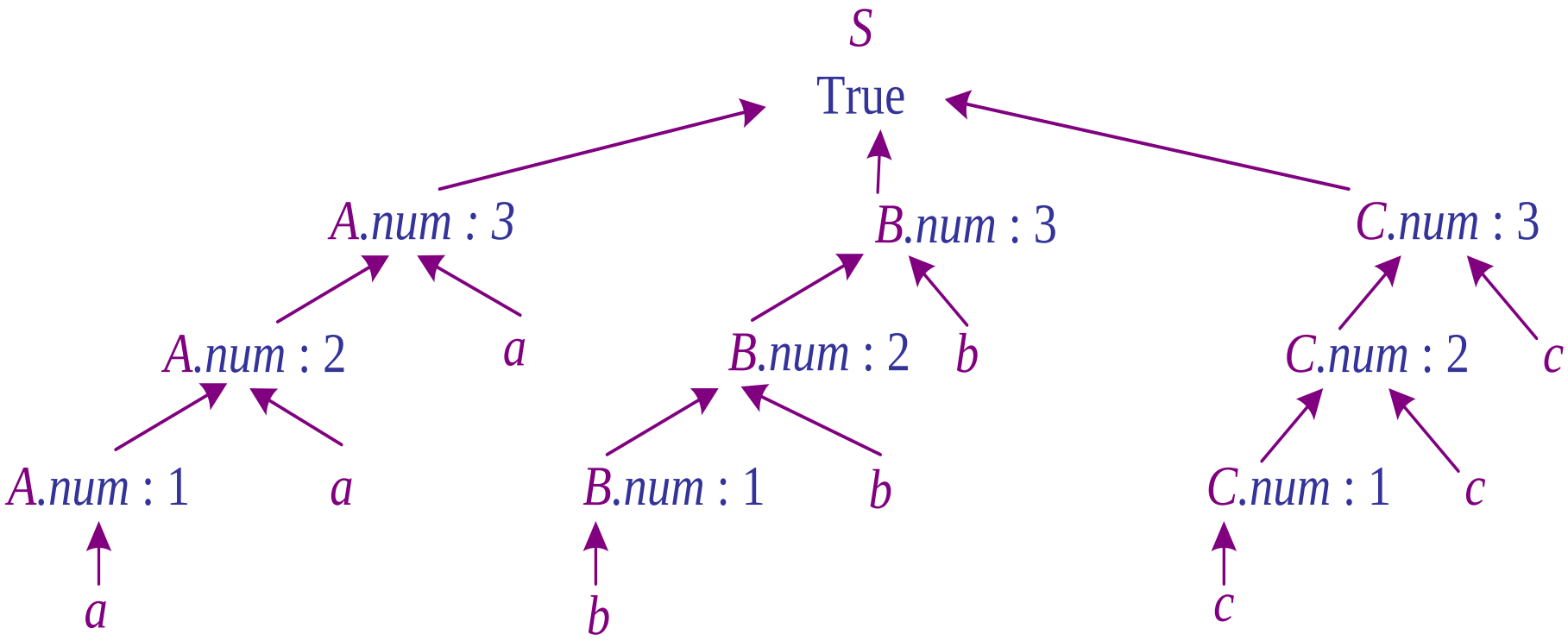
语义计算

- 两大功能：
  - 静态语义分析
    - 收集和计算源程序的上下文相关信息，进行一致性和完整性方面的检查。
  - 动态语义诠释
    - 中间代码生成
- 语法制导的语义计算
  - 属性文法
  - 翻译模式

7.1 基于属性文法的语义计算

[引例]
文法 G(E)：

产生式	语义规则
$S \rightarrow ABC$	{(A.num=B.num) and (B.num=C.num)}
$A \rightarrow Aa$	{A.num := A_1.num + 1
$A \rightarrow a$	{A.num := 1}
$B \rightarrow Bb$	{B.num := B_1.num + 1}
$B \rightarrow b$	{B.num := 1}
$C \rightarrow Cc$	{C.num := C_1.num + 1}
$C \rightarrow c$	{C.num := 1}



aaabbbccc的语法制导语义计算

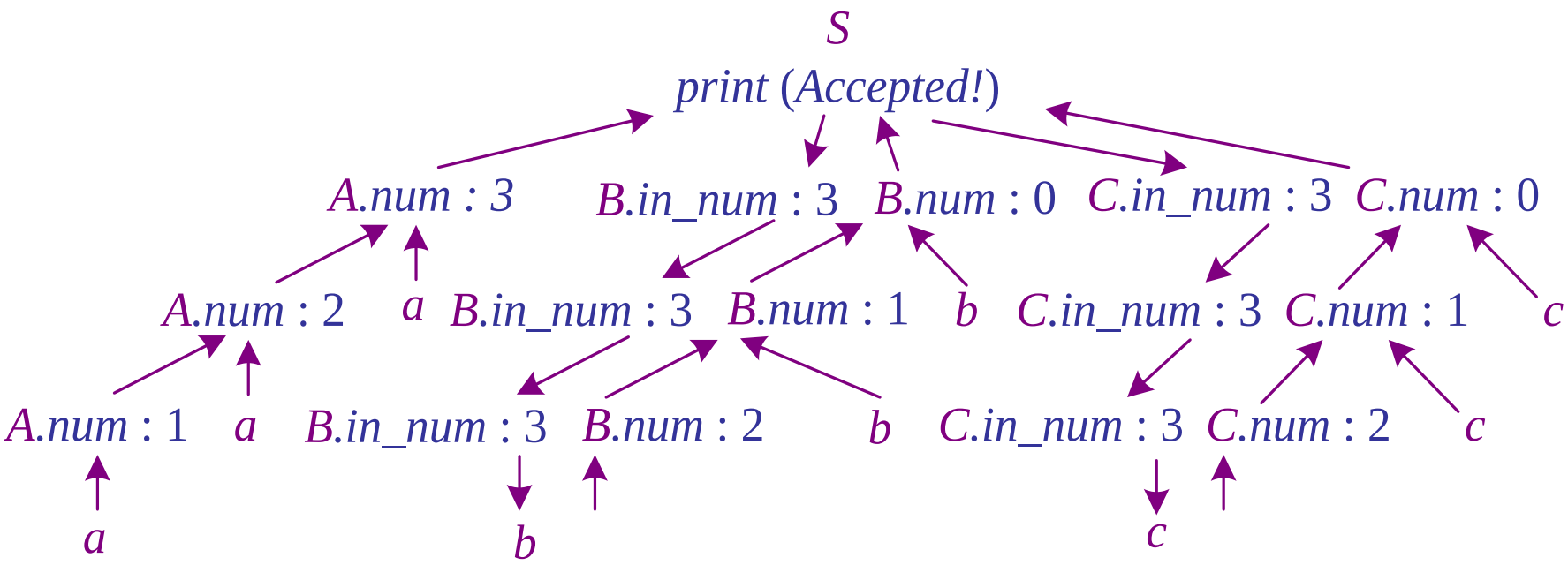
属性文法

- 属性文法（**Attribute Grammar**）在上下文无关文法的基础上进行如下扩展：
  - 为每个文法符号关联多个属性（**Attribute**）
    - 用来刻画一个文法符号的 任何我们所关心的特性
    - 文法符号 **X** 关联属性 **a** 的属性值可通过 **X.a** 访问
  - 为文法的每个产生式关联一个语义规则集合或称为语义动作。
    - 用于描述如何计算当前产生式中文法符号的属性值或附加的语义动作
    - 复写规则，形如 **X.a := Y.b**
      基于语义函数的规则，形如 **b:=f(c_1, c_2, …, c_k)**

例1

产生式	语义动作
$S \rightarrow ABC$	{B.in_num := A .num; C.in_num := A .num; if (B.num=0 and (C.num=0) then print("Accepted!") else print("Refused!") }
$A \rightarrow A_1a$	{ A.num := A_1.num + 1 }
$A \rightarrow a$	{ A.num := 1 }
$B \rightarrow B_1b$	{ B_1.in_num := B.in_num; B.num := B_1.num-1 }
$B \rightarrow b$	{ B.num := B.in_num -1 }
$C \rightarrow C_1c$	{ C_1.in_num := C.in_num; C.num := C_1.num-1 }
$C \rightarrow c$	{ C.num := C.in_num -1 }

计算aaabbbccc的语义



aaabbbccc的语法制导语义计算

综合属性和继承属性

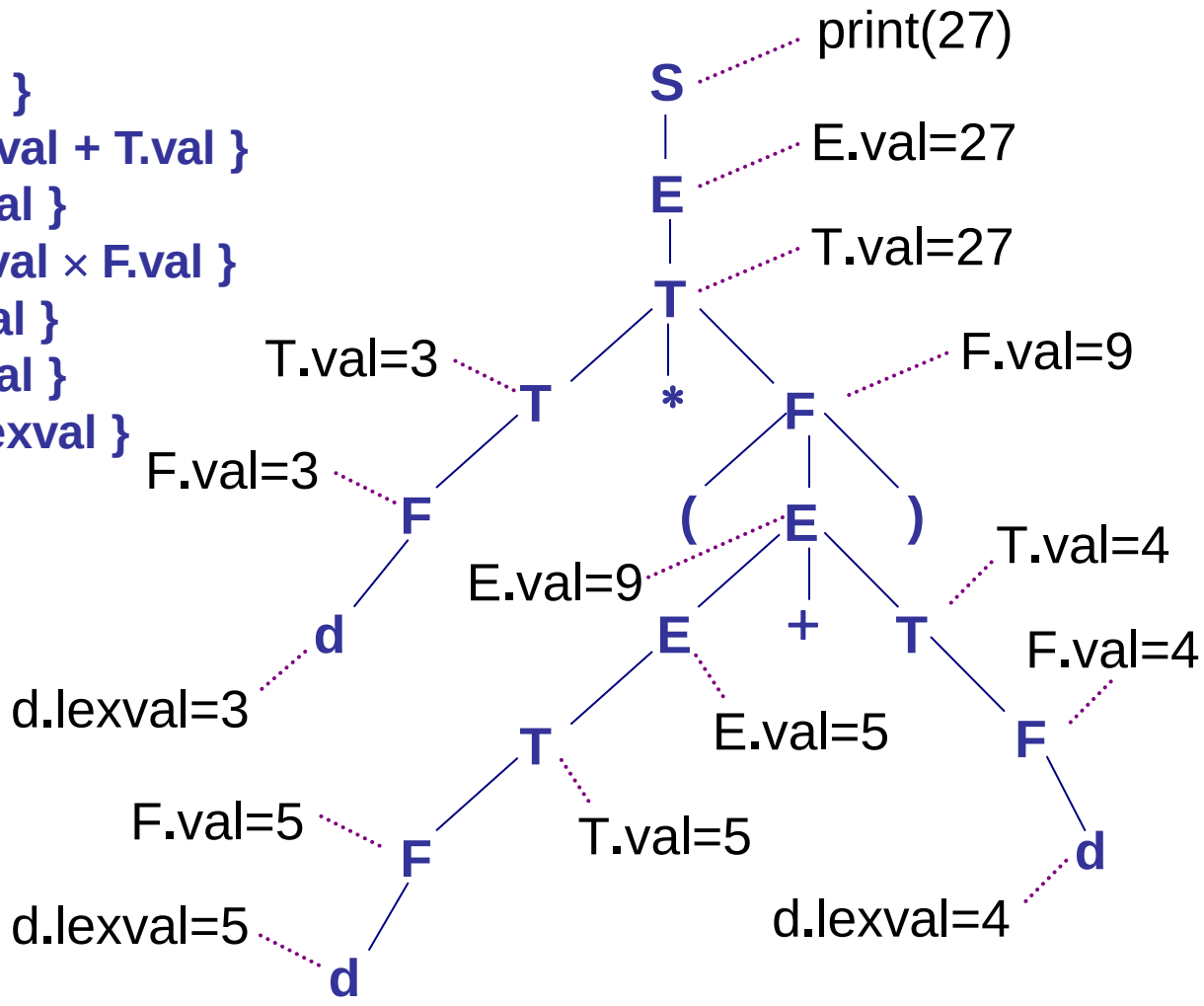
- 综合属性（Synthesized Attribute）
  - 用于“自下而上”传递信息
  - 对关联于产生式 $A\rightarrow\alpha$ 的语义规则 $b:=f(c_1, c_2, \ldots, c_k)$，如果 b 是 A 的某个属性，则称 b 是 A 的一个综合属性。
- 继承属性（Inherited Attribute）
  - 用于“自上而下”传递信息
  - 对关联于产生式 $A\rightarrow\alpha$ 的语义规则 $b:=f(c_1, c_2, \ldots, c_k)$，如果 b 是产生式右部某个文法符号 X 的某个属性，则称 b 是文法符号 X 的一个继承属性。

- 例2

G[S]:

产生式	语义动作
$S \rightarrow E$	{ print(E.val) }
$E \rightarrow E_1 + T$	{ E.val := E_1.val + T.val }
$E \rightarrow T$	{ E.val := T.val }
$T \rightarrow T_1 * F$	{ T.val := T_1.val × F.val }
$T \rightarrow F$	{ T.val := F.val }
$F \rightarrow (E)$	{ F.val := E.val }
$F \rightarrow d$	{ F.val := d.lexval }

产生式	语义动作
$S \rightarrow E$	{ print(E.val) }
$E \rightarrow E_1 + T$	{ E.val := E_1.val + T.val }
$E \rightarrow T$	{ E.val := T.val }
$T \rightarrow T_1 * F$	{ T.val := T_1.val × F.val }
$T \rightarrow F$	{ T.val := F.val }
$F \rightarrow (E)$	{ F.val := E.val }
$F \rightarrow d$	{ F.val := d.lexval }



3 * (5+4)

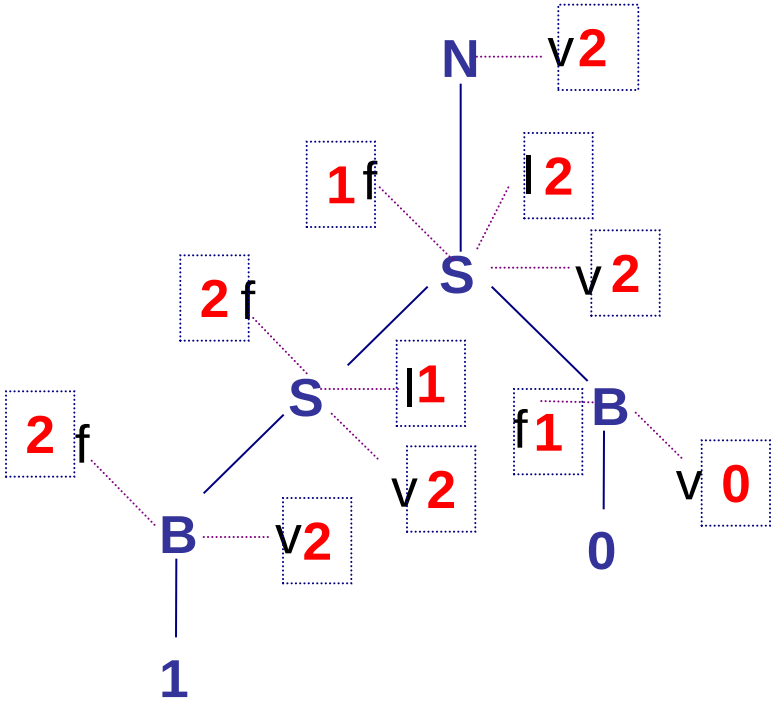
例3

G[N]:

产生式	语义动作
$N \rightarrow S$	{ N.v := S.v; S.f :=1 }
$S \rightarrow S_1B$	{ S_1.f := 2S.f; B.f : =S.f; S.v := S_1.v+B.v; S.l := S_1.l +1 }
$S \rightarrow B$	{ S.l := 1 ; S.v := B.v ; B.f := S.f }
$B \rightarrow 0$	{ B.v := 0 }
$B \rightarrow 1$	{ B.v := B.f }

产生式	语义动作
$N \rightarrow S$	{ N.v := S.v; S.f : =1 }
$S \rightarrow S_1B$	{ S_1.f := 2S.f; B.f : =S.f; S.v := S_1.v+B.v; S.l := S_1.l +1 }
$S \rightarrow B$	{ S.l := 1; S.v := B.v ; B.f := S.f }
$B \rightarrow 0$	{ B.v := 0 }
$B \rightarrow 1$	{ B.v := B.f }

10



S-属性文法和L-属性文法

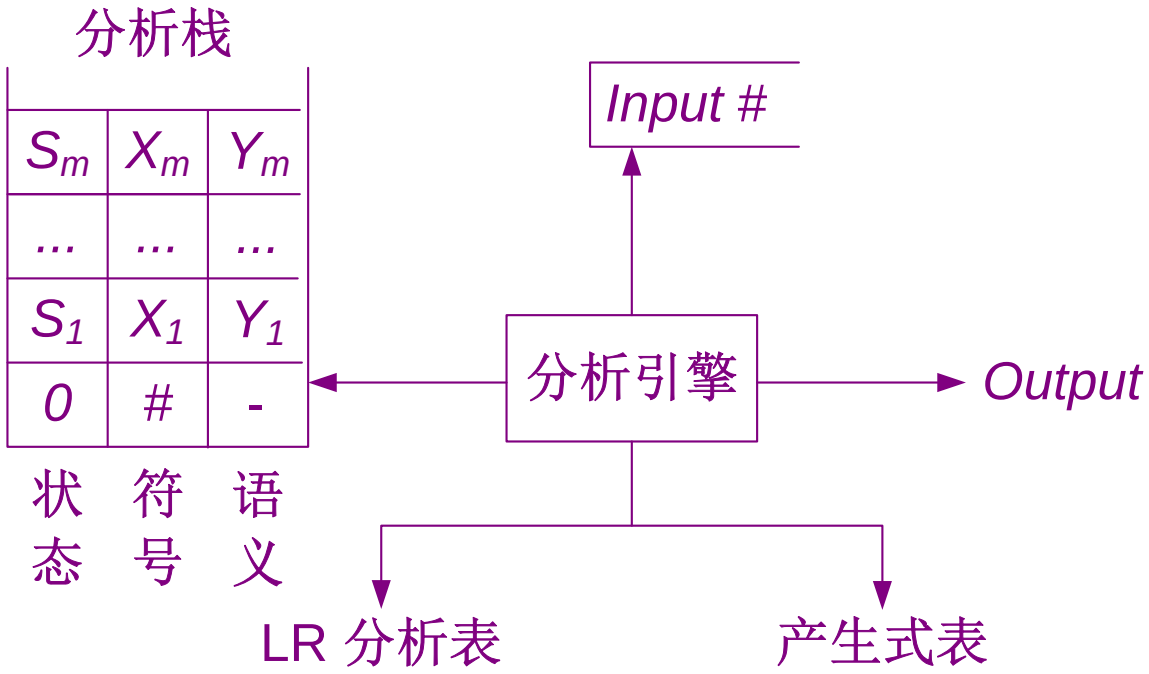
- **S-属性文法**
  - 只包含综合属性
- **L-属性文法**
  - 可以包含综合属性，也可以包含继承属性。
  - 产生式右端某文法符号的继承属性的计算只取决于该符号左边文法符号的属性（对于产生式左边 文法符号，只能是继承属性）。
  - **S-属性文法是L-属性文法的一个特例。**

基于属性文法的语义计算

- 树遍历方法
  通过遍历分析树进行属性计算
- 单遍的方法
  - 语法分析遍的同时进行属性计算
    - 自下而上方法
    - 自上而下方法
  - 只适用于特定文法
    - **S-**属性文法
    - **L-**属性文法

S-属性文法的语义计算

- 通常采用自下而上的方式进行
- 若采用**LR**分析技术，可以通过扩充分析栈中的域，形成语义栈来存放综合属性的值，计算相应产生式左部文法符号的综合属性值刚好发生在每一步归约之前的时刻。



产生式	语义动作
(0) $S \rightarrow E$	{ print(E.val) }
(1) $E \rightarrow E_1 + T$	{ E.val := E_1.val + T.val }
(2) $E \rightarrow T$	{ E.val := T.val }
(3) $T \rightarrow T_1 * F$	{ T.val := T_1.val × F.val }
(4) $T \rightarrow F$	{ T.val := F.val }
(5) $F \rightarrow (E)$	{ F.val := E.val }
(6) $F \rightarrow d$	{ F.val := d.lexval }

状态	ACTION						GOTO		
	d	*	+	(	)	#	E	T	F
0	s5			s4			1	2	3
1			s6			acc			
2		s7	r2		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8			s6		s11				
9		s7	r1		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

（0）S→E　　{ print(E.val) }
（1）E→E+T　　{ E.val := E_1.val + T.val }
（2）E→T　　{ E.val := T.val }
（3）T→T*F　　{ T.val := T_1.val × F.val }
（4）T→F　　{ T.val := F.val }
（5）F→(E)　　{ F.val := E.val }
（6）F→d　　{ F.val := d.lexval }

状态	ACTION						GOTO		
	d	*	+	(	)	#	E	T	F
0	s5			s4			1	2	3
1			s6			acc			
2		s7	r2		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8			s6		s11				
9		s7	r1		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

分析栈（状态，符号，语义值）	余留输入串	动作	语义动作
0 # –	2 + 3 * 5 #	s5	
0 # – 5 2 2	+ 3 * 5 #	r6	F.val := d.lexval
0 # – 3 F 2	+ 3 * 5 #	r4	T.val := F.val
0 # – 2 T 2	+ 3 * 5 #	r2	E.val := T.val
0 # – 1 E 2	+ 3 * 5 #	s6	
0 # – 1 E 2 6 + –	3 * 5 #	s5	
0 # – 1 E 2 6 + – 5 3 3	* 5 #	r6	F.val := d.lexval
0 # – 1 E 2 6 + – 3 F 3	* 5 #	r4	T.val := F.val
0 # – 1 E 2 6 + – 9 T 3	* 5 #	s7	
0 # – 1 E 2 6 + – 9 T 3 7 * –	5 #	s5	
0 # – 1 E 2 6 + – 9 T 3 7 * – 5 5 5	#	r6	F.val := d.lexval
0 # – 1 E 2 6 + – 9 T 3 7 * – 10 F 5	#	r3	T.val:=T_1.val×F.val
0 # – 1 E 2 6 + – 9 T 15	#	r1	E.val:=E_1.val+T.val
0 # – 1 E 17	#	acc	print(E.val)

7.2 基于翻译模式的语义计算

- 显式地表达动作和属性计算的次序
- 两类受限的翻译模式
  - **S-翻译模式**

    对于仅需要综合属性的情形，只要创建一个语义规则集合，放在相应产生式右端的末尾，把属性的计算规则加入其中。
  - **L-翻译模式**

    对于既包含继承属性又包含综合属性的情形，但需要满足：

    （1）产生式右端某个符号继承属性的计算必须位于该符号之前，其语义动作不访问位于它右边符号的属性，只依赖于该符号左边符号的属性（对于产生式 左部的符号，只能是继承属性）；

    （2）产生式左部符号的综合属性的计算只能在所用到的属性都已计算出来之后进行，通常将相应的语义动作置于产生式的尾部。

产生式	语义动作
$S \rightarrow ABC$	{B.in_num := A .num; C.in_num := A .num; if (B.num=0 and (C.num=0) then print("Accepted!") else print("Refused!") }
$A \rightarrow A_1a$	{ A.num := A_1.num + 1 }
$A \rightarrow a$	{ A.num := 1 }
$B \rightarrow B_1b$	{ B_1.in_num := B.in_num; B.num := B_1.num-1 }
$B \rightarrow b$	{ B.num := B.in_num -1 }
$C \rightarrow C_1c$	{ C_1.in_num := C.in_num; C.num := C_1.num-1 }
$C \rightarrow c$	{ C.num := C.in_num -1 }

$S \rightarrow A$ {B.in_num := A .num} B { C.in_num := A .num} C
{if (B.num=0 and (C.num=0)
then print("Accepted!")
else print("Refused!") }

$A \rightarrow A_1 a$ {A.num := A1.num + 1}

$A \rightarrow a$ {A.num := 1}

$B \rightarrow$ {B1.in_num := B.in_num} B_1 b {B.num := B1.num-1}

$B \rightarrow b$ {B.num := B.in_num -1}

$C \rightarrow$ {C1.in_num := C.in_num} $C_1 c$ {C.num := C1.num-1}

$C \rightarrow c$ {C.num := C.in_num -1}

✧ 基于S-翻译模式的语义计算

–S-翻译模式在形式上与S-属性文法是一致的，可以采用同样的语义计算方式。

产生式	语义动作
$S \rightarrow E$	{ print(E.val) }
$E \rightarrow E_1 + T$	{ E.val := E_1.val + T.val }
$E \rightarrow T$	{ E.val := T.val }
$T \rightarrow T_1 * F$	{ T.val := T_1.val × F.val }
$T \rightarrow F$	{ T.val := F.val }
$F \rightarrow (E)$	{ F.val := E.val }
$F \rightarrow d$	{ F.val := d.lexval }

翻译模式

{ print(v[top].val) }
{ v[top-2].val := v[top-2].val +[top].val }
{ v[top].val := v[top].val }
{ v[top-2].val := v[top-2].val * [top].val }
{ v[top].val := v[top].val }
{ v[top].val := v[top-1].val }
{ v[top].val := d.lexval }