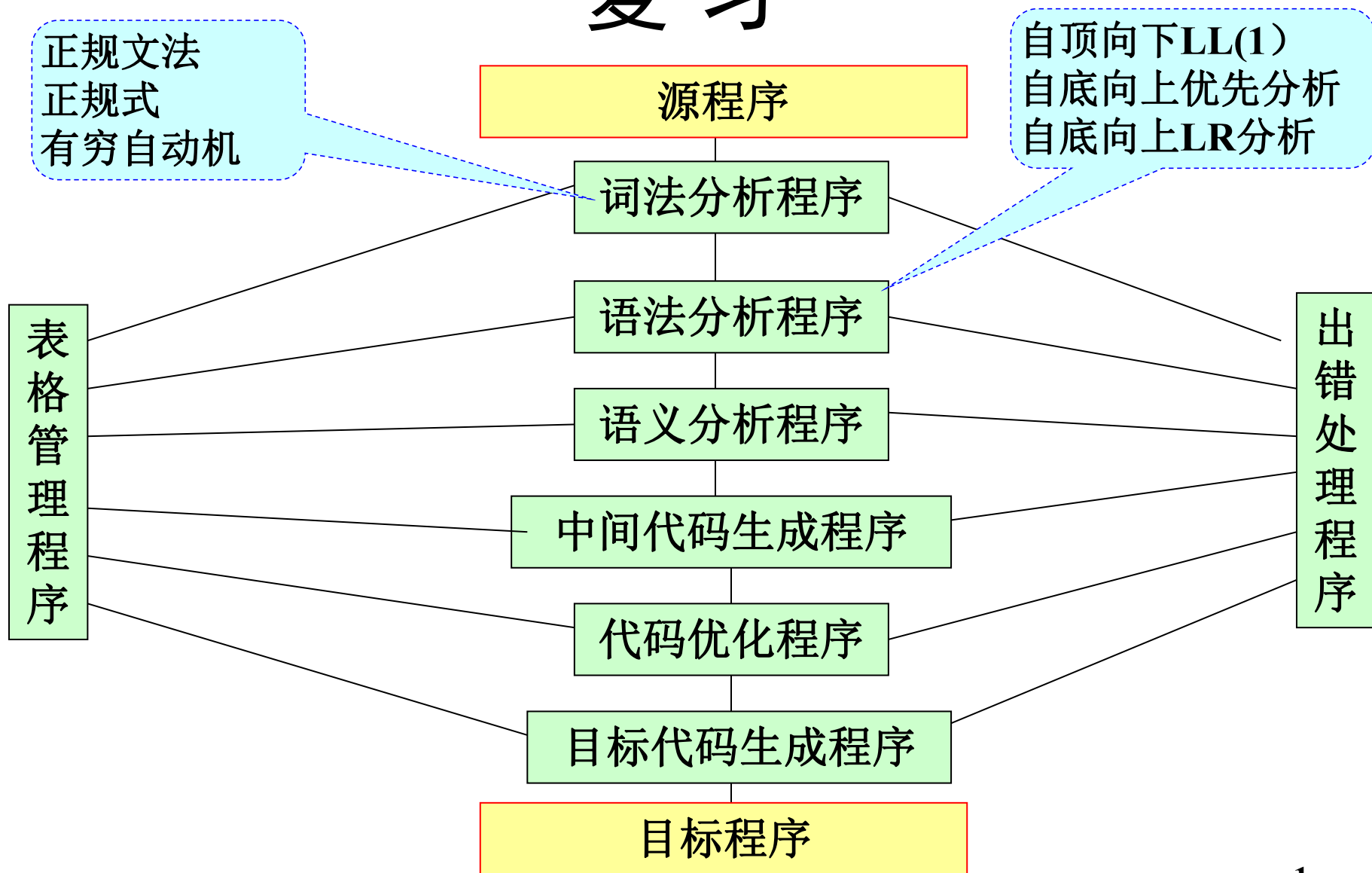


复 习



第七章语法制导的语义计算

7.1 基于属性文法的语义计算

7.1.1 属性文法

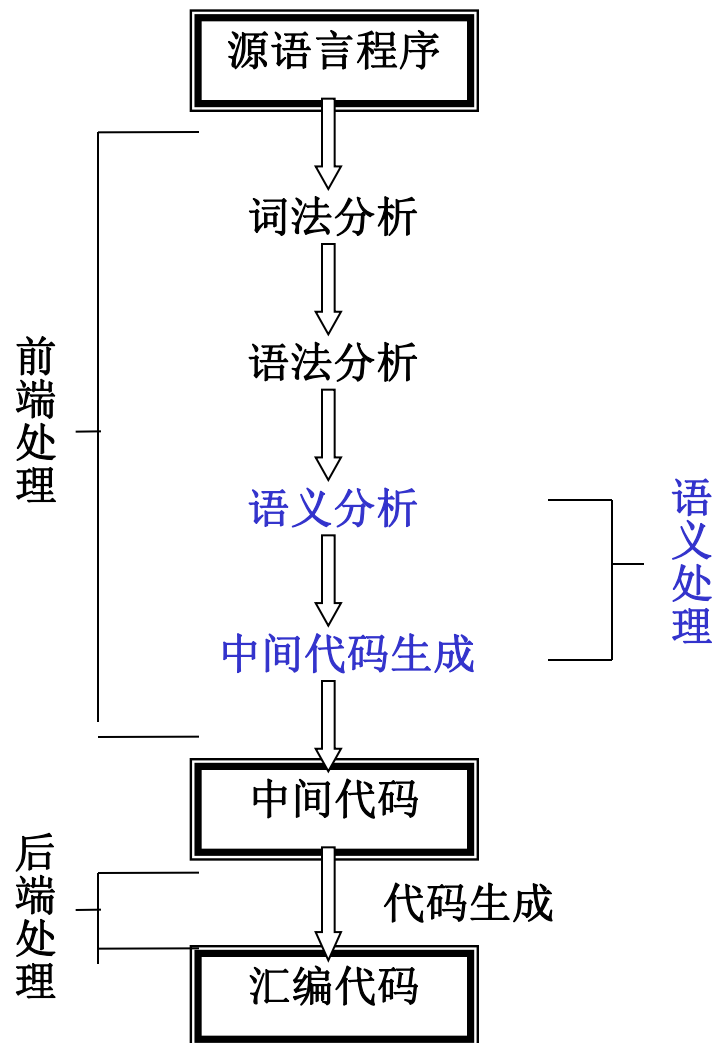
7.1.2 语义计算

7.2 基于翻译模式的语义计算

7.2.1 翻译模式

7.2.2 语义计算

在编译中的逻辑阶段



语义处理

➤ 通常在词法分析和语法分析之后，

可以由语法分析程序直接调用相应的语义子程序进行处理。

可以首先生成语法树或该结构的表示，再进行语义处理。

语义处理概述

➤ 编译中的语义处理包括两个功能：

- 审查每个语法结构的静态语义，即验证语法结构合法的程序是否真正有意义。也称为静态语义分析或静态审查；

例如：类型，维数，运算，越界

- 如果静态语义正确，则执行真正的翻译，即生成中间代码或生成实际的目标代码。

例如：变量的存储分配、表达式的求值，语句翻译

➤ 总的目标：

生成等价的代码（中间代码）。

语义处理概述

➤ 属性文法：

在语法分析的同时进行语义分析，引入属性文法的方式。

➤ 语法制导翻译：

把语法结构正确的程序的意思分析并翻译出来。

例子

If $x > 5$ then

$y = 10;$

else

$y = 0;$

end if

($>$, x , 5 , ...)

(goto)

(= , y , 10 , ——)

(goto)

(= , y , 0 , ——)

(goto)

代码结构

➤朴素的理解：编写的程序如何工作？

➤计算学科：

对信息描述（数据表示）和变换算法的系统研究。

➤变换：源、目标以及源与目标的对应关系。

□语句的代码结构（源、目标语言）

□语句分类：

说明语句————符号表的查找和填写

可执行语句————生成指令代码

典型处理方法一

- 对每一个产生式编制一个语义子程序：

当一个产生式获得匹配时，调用相应的语义子程序，实现语义检查与翻译。

$E \rightarrow E_1 + T$ $E.val := E_1.val + T.val$

$T \rightarrow T_1 * F$ $T.val := T_1.val * F.val$

$F \rightarrow id$ $F.val = id.val$

- 考虑在完成归约的时候进行。

典型处理方法二

- 在产生式的右部的适当位置，插入相应的语义动作，按照分析的进程，执行**遇到**的语义动作。

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

$T \rightarrow \text{int} \{T.type := \text{integer}\}$

$T \rightarrow \text{real} \{T.type := \text{real}\}$

$L \rightarrow \{L1.in := L.in\} L1, \text{id} \{\dots\}$

$L \rightarrow \text{id} \{\dots\}$

- 考虑在推导时候完成。

典型处理方法

➤ 对应语法基本分析方法：

Top--down: 推导过程中完成

Botton—up: 归约时完成

➤ 语义

语法成分的语义可以用相应文法符号的属性表示，通过属性的计算，来完成翻译。

7.1 属性文法

➤ 属性文法(attribute grammar)是knuth在1968年提出的

➤ 属性文法的特点：

是一种接近形式化的语义描述方法；

便于描述静态语义，不利于描述动态语义；

每个语法符号有相应的属性符号；

每个产生式有相应计算属性的规则：

属性变量：=属性表达式

属性文法举例

产生式

属性（计算）规则/语义规则

$E \rightarrow E_1 + E_2$

$E.val := E_1.val + E_2.val$

$E \rightarrow E_1 * E_2$

$E.val := E_1.val * E_2.val$

$E \rightarrow (E_1)$

$E.val := E_1.val$

$E \rightarrow id$

$E.val := id.val$

属性文法=文法+属性的有穷集+规则

7.1 属性文法

➤ 属性文法是一个三元组: $A=(G,V,F)$

G:是一个上下文无关文法

V:有穷的属性集:

语义信息作为终结符和非终结符的属性, 如它的类型、值、代码序列、符号表内容等等。

属性可以计算和传递。属性加工的过程即是语义处理的过程。

F:属性计算规则:

语义分析成为产生式相关的属性计算;

每个产生式设置语义规则, 描述各属性的关系即计算规则。

属性的设定

—属性

用来刻画一个文法符号的任何我们所关心的特性

如：符号的**值**，符号的**名字串**，符号的**类型**，符号的**偏移地址**，符号被赋予的**寄存器**，**代码片断**，等等...

—记号

文法符号 X 关联属性 a 的属性值可通过 $X.a$ 访问

– 语义规则

在属性文法中，每个产生式 $A \rightarrow \alpha$ 都关联一个语义规则的集合，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

– 属性文法中允许如下语义规则

复写 (**copy**) 规则，形如

$$X.a := Y.b$$

• 基于语义函数 (**semantic function**) 的规则，形如 $b := f(c_1, c_2, \dots, c_k)$ 或 $f(c_1, c_2, \dots, c_k)$

其中， b, c_1, c_2, \dots, c_k 是该产生式中文法符号的属性

– 实践中，语义函数的形式可以更灵活

◇ 有两种属性：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 $b := f(c_1, c_2, \dots, c_k)$,
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 $b := f(c_1, c_2, \dots, c_k)$,
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

✧ 属性文法举例

– 仅含综合属性的例子（开始符号S）

产生式

语义动作

$S \rightarrow E$

$\{ \text{print}(E.val) \}$

$E \rightarrow E_1 + T$

$\{ E.val := E_1.val + T.val \}$

$E \rightarrow T$

$\{ E.val := T.val \}$

$T \rightarrow T_1 * F$

$\{ T.val := T_1.val \times F.val \}$

$T \rightarrow F$

$\{ T.val := F.val \}$

$F \rightarrow (E)$

$\{ F.val := E.val \}$

$F \rightarrow d$

$\{ F.val := d.lexval \}$

注： ***d.lexval*** 是词法分析程序确定的属性值

上传递的信息

行上传递的信息

行上传递的信息



✧ 属性文法举例

– 含继承属性的例子（开始符号S）

产生式

语义动作

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

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

$A \rightarrow \varepsilon \quad \{ A.num := 0 \}$

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

$B \rightarrow \varepsilon \quad \{ B.num := B.in_num \}$

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

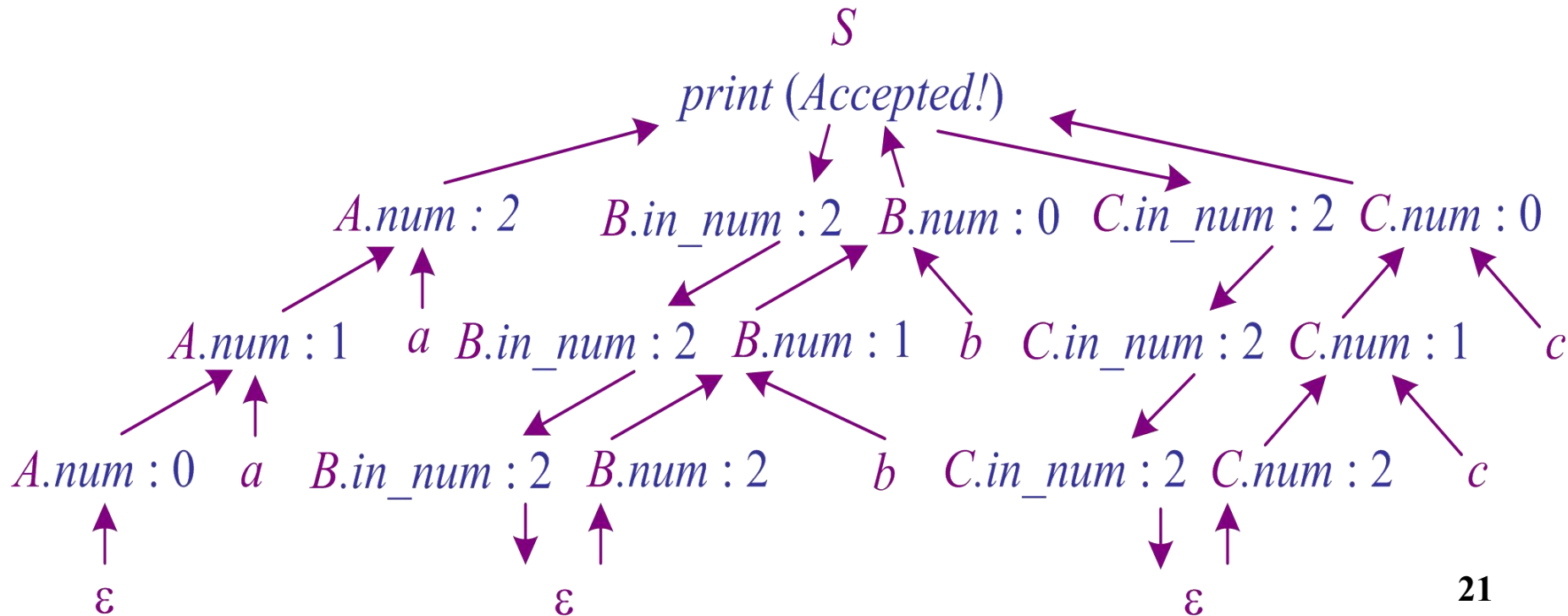
$C \rightarrow \varepsilon \quad \{ C.num := C.in_num \}$

其中， $A.num$ ， $B.num$ 和 $C.num$ 是综合属性值，而
 $B.in_num$ 和 $C.in_num$ 是继承属性值

继承属性代

–对 **aabbcc** 的分
析性相应的语义动
义动作，可以得

$S \rightarrow ABC \quad \{ B.in_num := A.num; C.in_num := A.num; \}$
 $\quad \quad \quad \text{if } (B.num=0 \text{ and } (C.num=0)) \text{ then}$
 $\quad \quad \quad \text{print("Accepted!") else print("Refused!") } \}$
 $A \rightarrow A_1 a \quad \{ A.num := A_1.num + 1 \}$
 $A \rightarrow \varepsilon \quad \{ A.num := 0 \}$
 $B \rightarrow B_1 b \quad \{ B_1.in_num := B.in_num; B.num := B_1.num - 1 \}$
 $B \rightarrow \varepsilon \quad \{ B.num := B.in_num \}$
 $C \rightarrow C_1 c \quad \{ C_1.in_num := C.in_num; C.num := C_1.num - 1 \}$
 $C \rightarrow \varepsilon \quad \{ C.num := C.in_num \}$



✧ 属性文法举例

– 更复杂的例子（开始符号 N ）

产生式 语义动作

$N \rightarrow S_1.S_2$ $\{ N.v := S_1.v + S_2.v; \quad S_1.f := 1; \quad S_2.f := 2^{-S_2.l} \}$

$S \rightarrow S_1B$ $\{ S_1.f := 2S.f, \quad B.f := S.f, \quad S.v := S_1.v + B.v; \quad S.l := S_1.l + 1 \}$

$S \rightarrow B$ $\{ S.l := 1; \quad S.v := B.v; \quad B.f := S.f \}$

$B \rightarrow 0$ $\{ B.v := 0 \}$

$B \rightarrow 1$ $\{ B.v := B.f \}$

该属性文法用于将二进制无符号小数转化为十进制小数
请思考：语义动作中涉及的属性应该如何计算？

7.1 基于属性文法的语义计算

◇ 基于属性文法的语义计算

计算方法分两类：

- 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

基于属性文法的语义计算

◇ 基于树遍历方法的语义计算

— 步骤

- 构造输入串的语法分析树
- 构造依赖图
- 若该依赖图是无圈的，则按造此无圈图的一种拓扑排序对分析树进行遍历，则可以计算所有的属性

注：若依赖图含有圈，则相应的属性文法不可采用这种方法进行语义计算，此类属性文法不是良定义的。所谓良定义的属性文法，当且仅当它的规则集合能够为所有分析树中的属性集确定唯一的值集。

◇ 依赖图是一个有向图，用来描述分析树中的属性与属性之间的相互依赖关系

—构造算法

for 分析树中每一个结点n do

for 结点n所用产生式的每个语义规则中涉及的每一个属性a do
为a在依赖图中建立一个结点；

for 结点n所用产生式中每个形如 $f(c_1, c_2, \dots, c_k)$ 的语义规则 do
为该规则在依赖图中也建立一个结点（称为虚结点）；

for 分析树中每一个结点n do

for 结点n所用产生式对应的每个语义规则 $b := f(c_1, c_2, \dots, c_k)$ do
（可以只是 $f(c_1, c_2, \dots, c_k)$ ，此时b结点为一个虚结点）

for $i := 1$ to k do

从 c_i 结点到b结点构造一条有向边

基于属性文法的语义计算

✧ 基于树遍历的计算方法举例

– 设有如下属性文法，考虑输入串 10.01 的语义计算过程

产生式

语义动作

$N \rightarrow S_1.S_2$

$\{ N.v := S_1.v + S_2.v; S_1.f := 1; S_2.f := 2^{-S_2.l} \}$

$S \rightarrow S_1B$

$\{ S_1.f := 2S.f; B.f := S.f; S.v := S_1.v + B.v; S.l := S_1.l + 1 \}$

$S \rightarrow B$

$\{ S.l := 1; S.v := B.v; B.f := S.f \}$

$B \rightarrow 0$

$\{ B.v := 0 \}$

$B \rightarrow 1$

$\{ B.v := B.f \}$

产生式 语义动作

$N \rightarrow S_1.S_2$ { $N.v := S_1.v + S_2.v$; $S_1.f := 1$; $S_2.f := 2^{-S_2.l}$ }

$S \rightarrow S_1B$ { $S_1.f := 2S.f$, $B.f := S.f$, $S.v := S_1.v + B.v$, $S.l := S_1.l + 1$ }

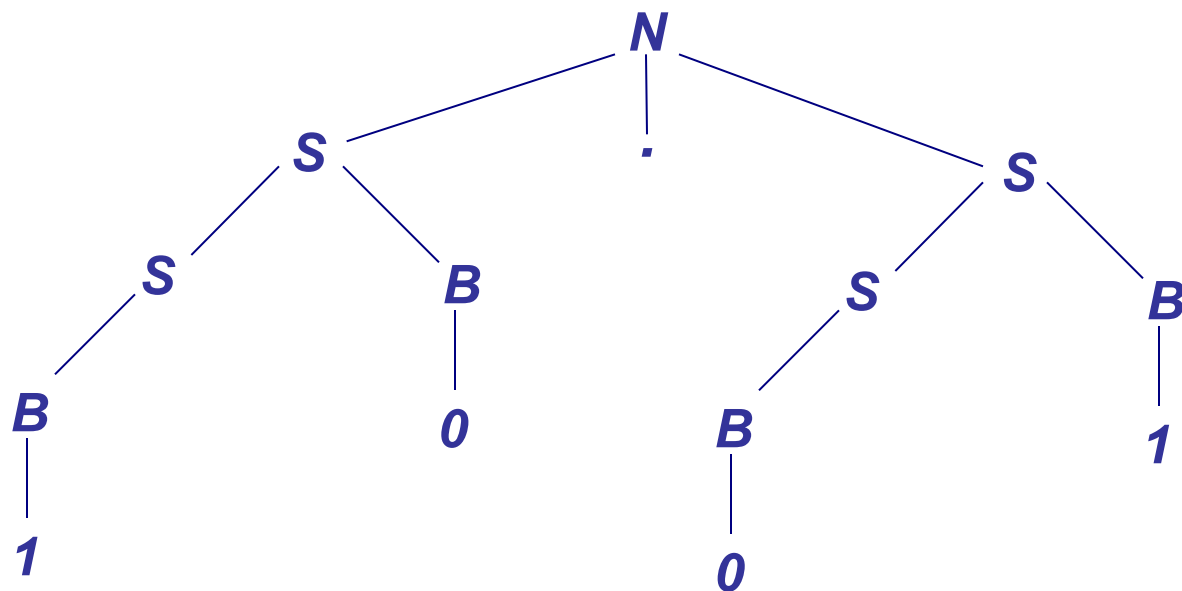
$S \rightarrow B$ { $S.l := 1$; $S.v := B.v$; $B.f := S.f$ }

$B \rightarrow 0$ { $B.v := 0$ }

$B \rightarrow 1$ { $B.v := B.f$ }

✧ 基于树遍历的计算方法举例

– 步骤一 构造输入串10.01的语法分析树



产生式 语义动作

$N \rightarrow S_1.S_2$ { $N.v := S_1.v + S_2.v$; $S_1.f := 1$; $S_2.f := 2^{-S_2.l}$ }

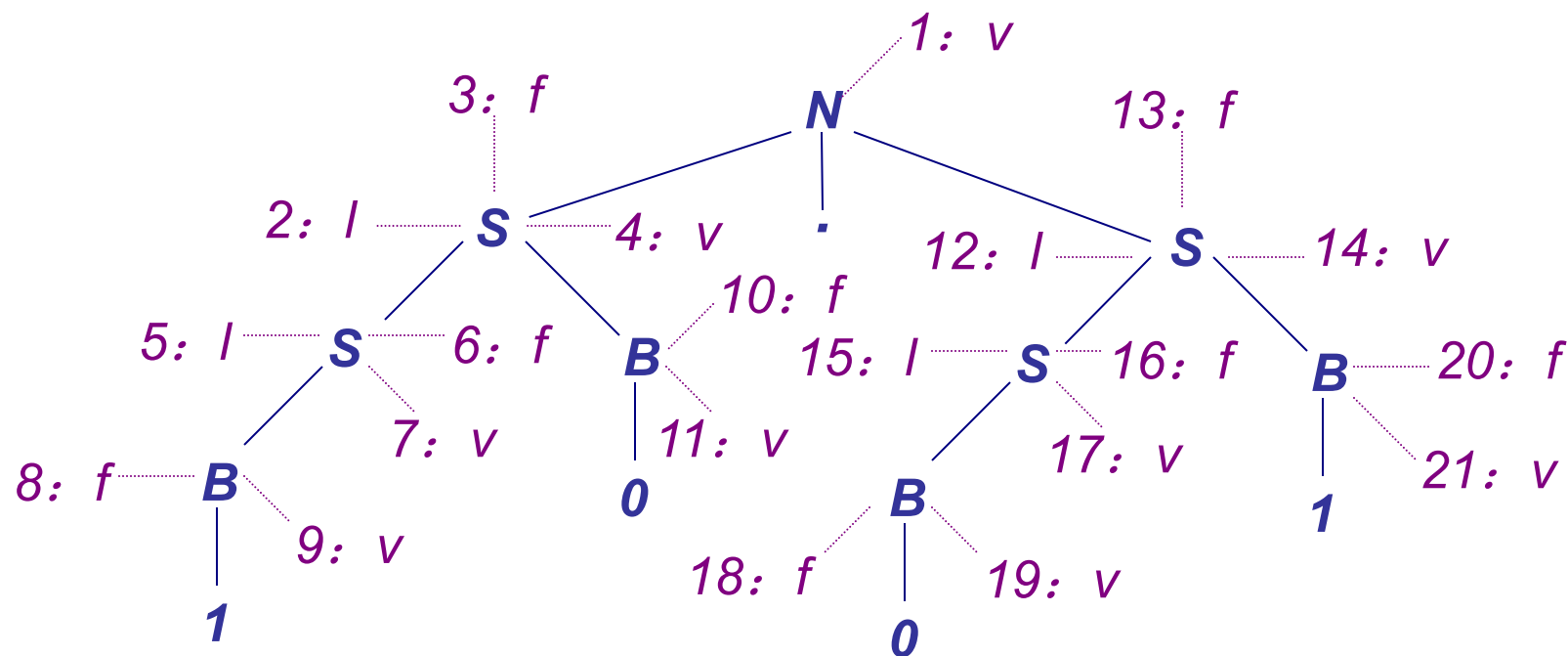
$S \rightarrow S_1B$ { $S_1.f := 2S.f$, $B.f := S.f$, $S.v := S_1.v + B.v$; $S.l := S_1.l + 1$ }

$S \rightarrow B$ { $S.l := 1$; $S.v := B.v$; $B.f := S.f$ }

$B \rightarrow 0$ { $B.v := 0$ }

$B \rightarrow 1$ { $B.v := B.f$ }

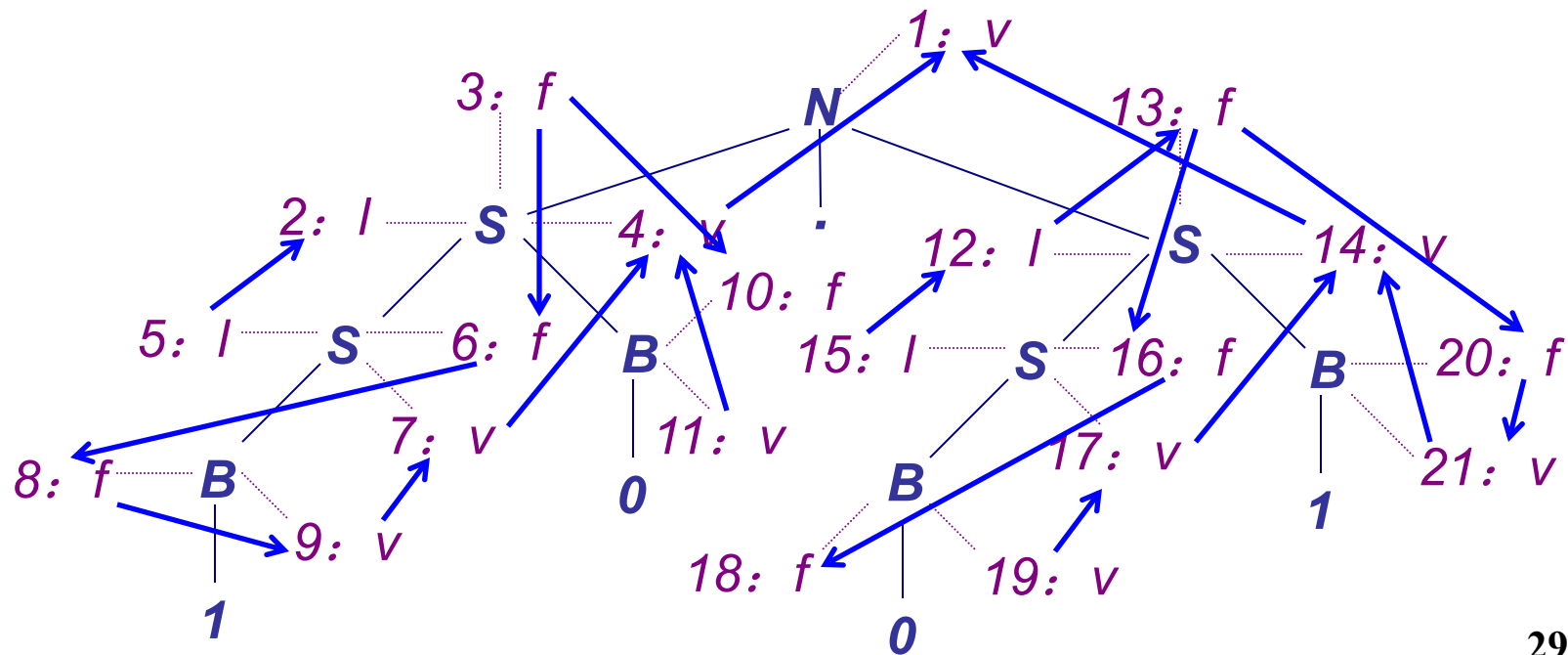
- **步骤二** 为分析树中所有结点的每个属性建立一个依赖图中的结点，并给定一个标记序号



基于属性文法的语义计算

✧ 基于树遍历的计算方法举例

步骤三 根据语义动作，建立依赖图中的有向边

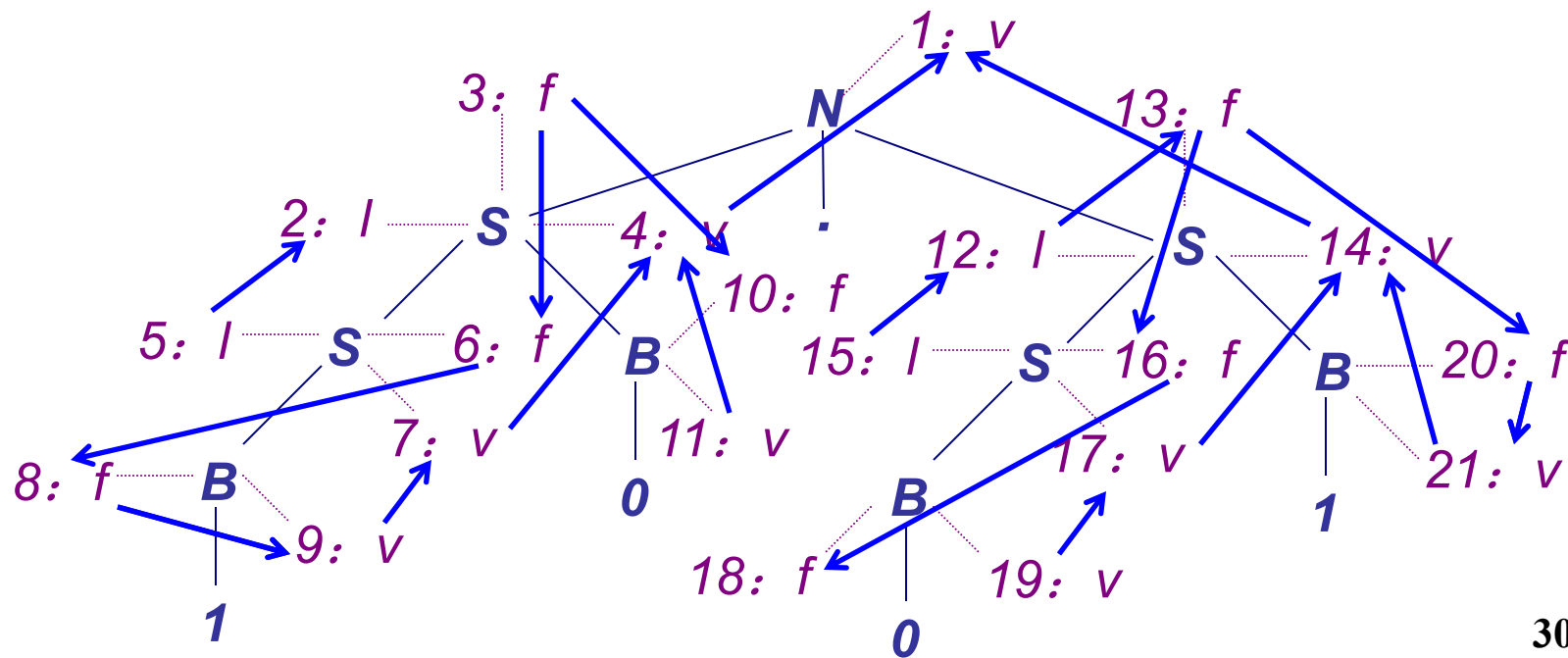
$$B \rightarrow B_1 B_2 \dots B_n \# B_{n+1} B_{n+2} \dots B_{n+m} \{ \neq S.f \}$$


基于属性文法的语义计算

✧ 基于树遍历的计算方法举例

- **步骤四** 容易看出，该依赖图是无圈的，因此存在拓扑排序. 依任何一个拓扑排序，都能够顺利完成属性值的计算. 如下是一种可能的计算次序：

3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

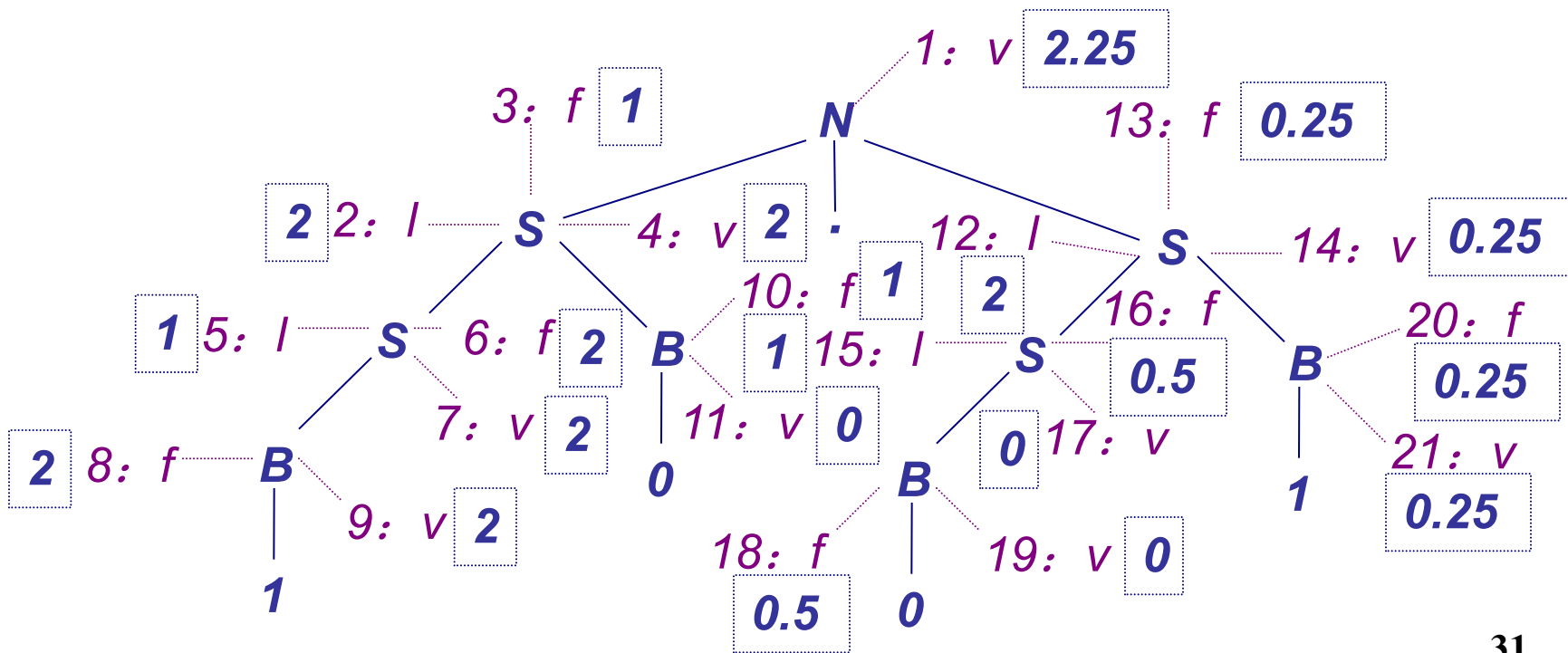


基于属性文法的语义计算

✧ 基于树遍历的计算方法举例

- **步骤五** 依计算次序，根据语义动作求出各结点对应的属性值. 对如下结点次序进行计算：

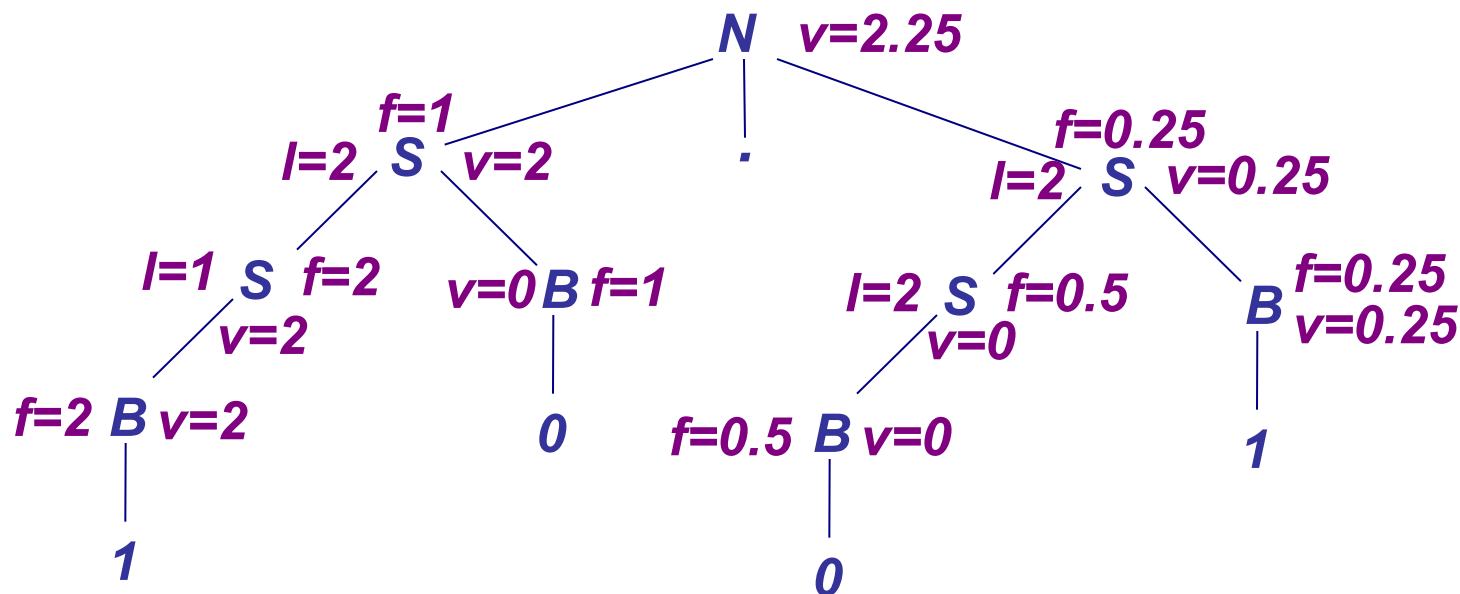
3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



基于属性文法的语义计算

◇ 带标注 (annotated) 的语法分析树

- 语法分析树中各结点属性值的计算过程被称为对语法分析树的标注 (annotating) 或修饰 (decorating), 用带标注的语法分析树表示属性值的计算结果, 如:



基于属性文法的语义计算

◇ 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

本课程只讨论如下两类属性文法：

- S-属性文法
- L-属性文法

基于属性文法的语义计算

◇ 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

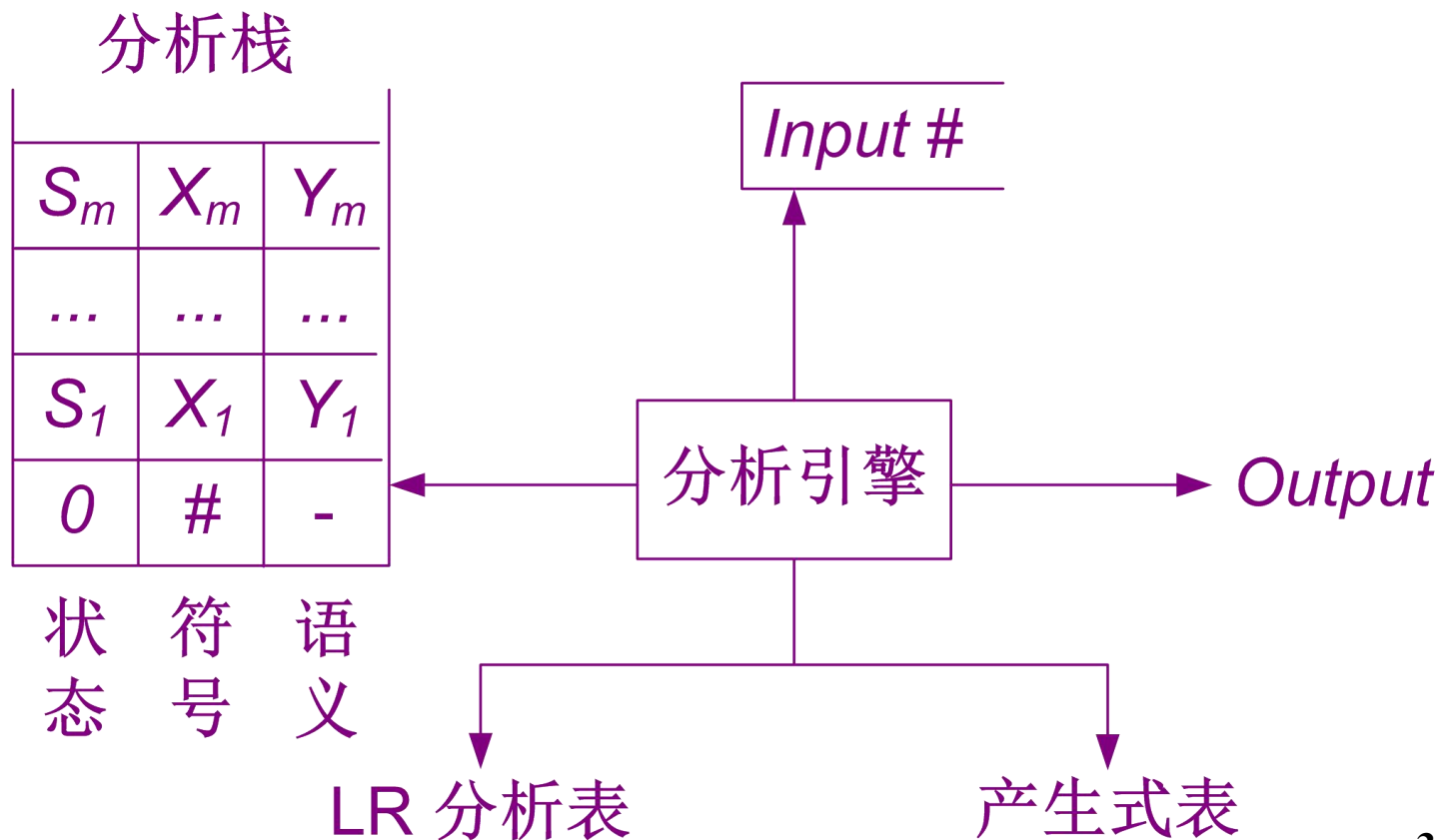
基于属性文法的语义计算

◇ 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

基于属性文法的语义计算

- ◇ 采用LR分析技术进行S-属性文法的语义计算
 - 扩充分析栈中的域形成语义栈存放综合属性的值



基于属性文法的语义计算

- ◇ 采用LR分析技术进行S-属性文法的语义计算
 - 语义动作中的综合属性可以通过存在于当前语义栈栈顶部分的属性进行计算
 - 例如，假设有相应于产生式 $A \rightarrow XYZ$ 的语义规则

$$A.a := f(X.x, Y.y, Z.z)$$

在 XYZ 归约为 A 之前， $Z.z$, $Y.y$, 和 $X.x$ 分别存放于语义栈的 **top**, **top-1** 和 **top-2** 的相应域中，因此 $A.a$ 可以顺利求出

归约后， $X.x$, $Y.y$, $Z.z$ 被弹出，而在栈顶 **top** 的位置上存放 $A.a$ 。

总结

- 属性文法
- 遍历树进行语义计算
- S属性文法和L属性文法
- 基于S属性文法的语义计算
- 基于L属性文法的语义计算

基于属性文法的语义计算

✧ 用LR分析技术进行S-属性文法的语义计算举例

– 通过下列S-属性文法G'[S]为常量表达式求值

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

基于属性文法的语义计算

☆ 文法 $G'[S]$ 的LR分析表

- (0) $S \rightarrow E$ (1) $E \rightarrow E+T$ (2) $E \rightarrow T$ (3) $T \rightarrow T*F$
 (4) $T \rightarrow F$ (5) $F \rightarrow (E)$ (6) $F \rightarrow d$

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			

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			
4	s5				s4		8	2	3
5			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		

法的语义计算

(0) $S \rightarrow E$ (1) $E \rightarrow E+T$ (2) $E \rightarrow T$
 (3) $T \rightarrow T * F$ (4) $T \rightarrow F$
 (5) $F \rightarrow (E)$ (6) $F \rightarrow d$

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	$T.val := T_1.val \times F.val$
0 # - 1 E 2 6 + - 9 T 15	#	r1	$E.val := E_1.val + T.val$
0 # - 1 E 17	#	acc	$print(E.val)$

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			

法的语义计算

(0) $S \rightarrow E$ (1) $E \rightarrow E+T$ (2) $E \rightarrow T$
 (3) $T \rightarrow T * F$ (4) $T \rightarrow F$
 (5) $F \rightarrow (E)$ (6) $F \rightarrow d$

步骤	状态栈	符号栈	语义值栈	余输入串	动作	语义动作
1	0	#	—	2+3*5#	S5	
2	05	#2	—2	+3*5#	R6	F.val=d.lexval
3	03	#F	—2	+3*5#	R4	T.val=F.val
4	02	#T	—2	+3*5#	R2	E.val=T.val
5	01	#E	—2	+3*5#	S6	
6	016	#E+	—2—	3*5#	S5	
7	0165	#E+3	—2—3	*5#	R6	F.val=d.lexval
8	0163	#E+F	—2—3	*5#	R4	T.val=F.val
9	0169	#E+T	—2—3	*5#	S7	
10	01697	#E+T*	—2—3—	5#	S5	
11	016975	#E+T*5	—2—3—5	#	R6	F.val=d.lexval
12	01697(10)	#E+T*F	—2—3—5	#	R3	T.val=T1.val*F.val
13	0169	#E+T	—2—15	#	R1	E.val=E1.val+T.val
14	01	#E	17	#	acc	Print(E.val)

基于属性文法的语义计算

◇ L-属性文法的语义计算

- 采用自上而下的方式可以较方便地进行
- 可以采用下列基于深度优先遍历的算法

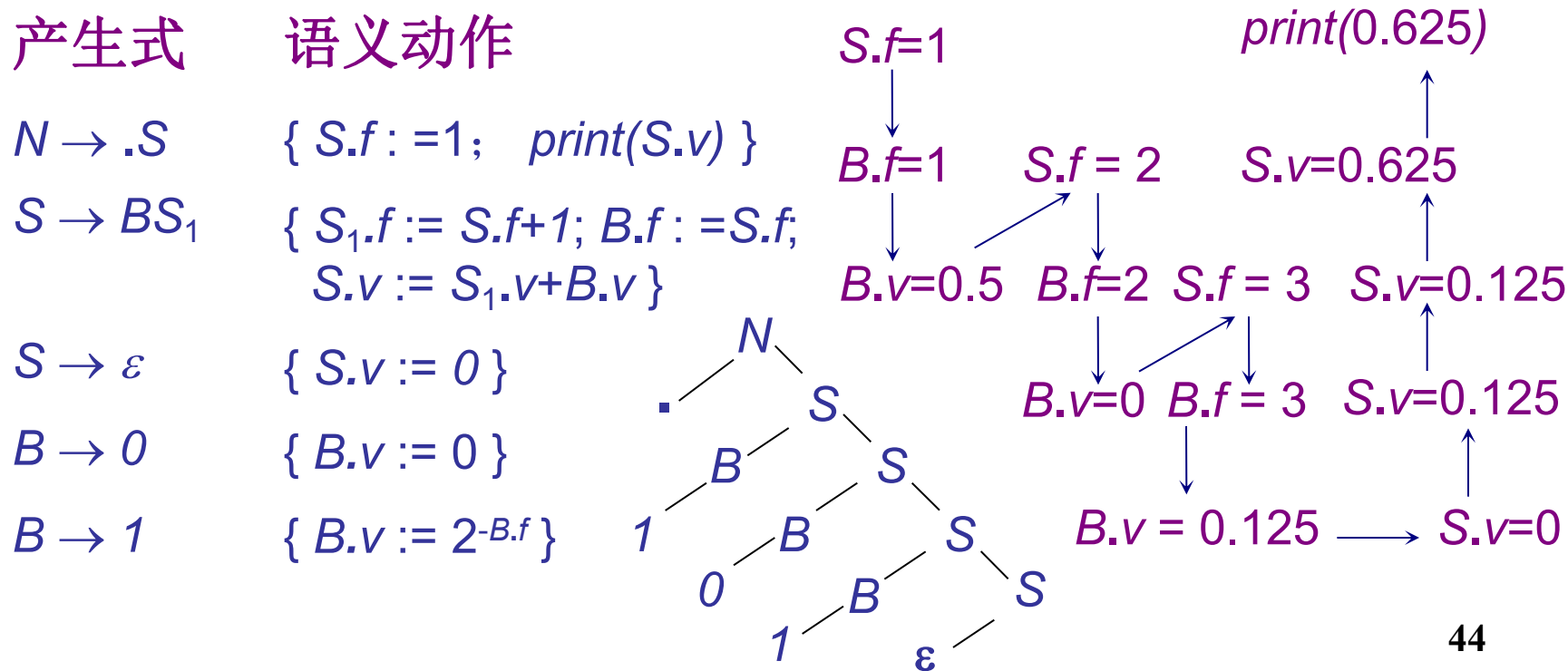
```
procedure dfvisit(n: node);  
  begin  
    for n 的每一孩子 m, 从左到右 do  
      begin  
        计算 m 的继承属性值;  
        dfvisit(m)  
      end;  
    计算 n 的综合属性值  
  end
```

- 该算法与自上而下预测分析过程对应. 因此, 基于 LL(1) 文法的 L-属性文法可以采用这种方法进行语义计算.

基于属性文法的语义计算

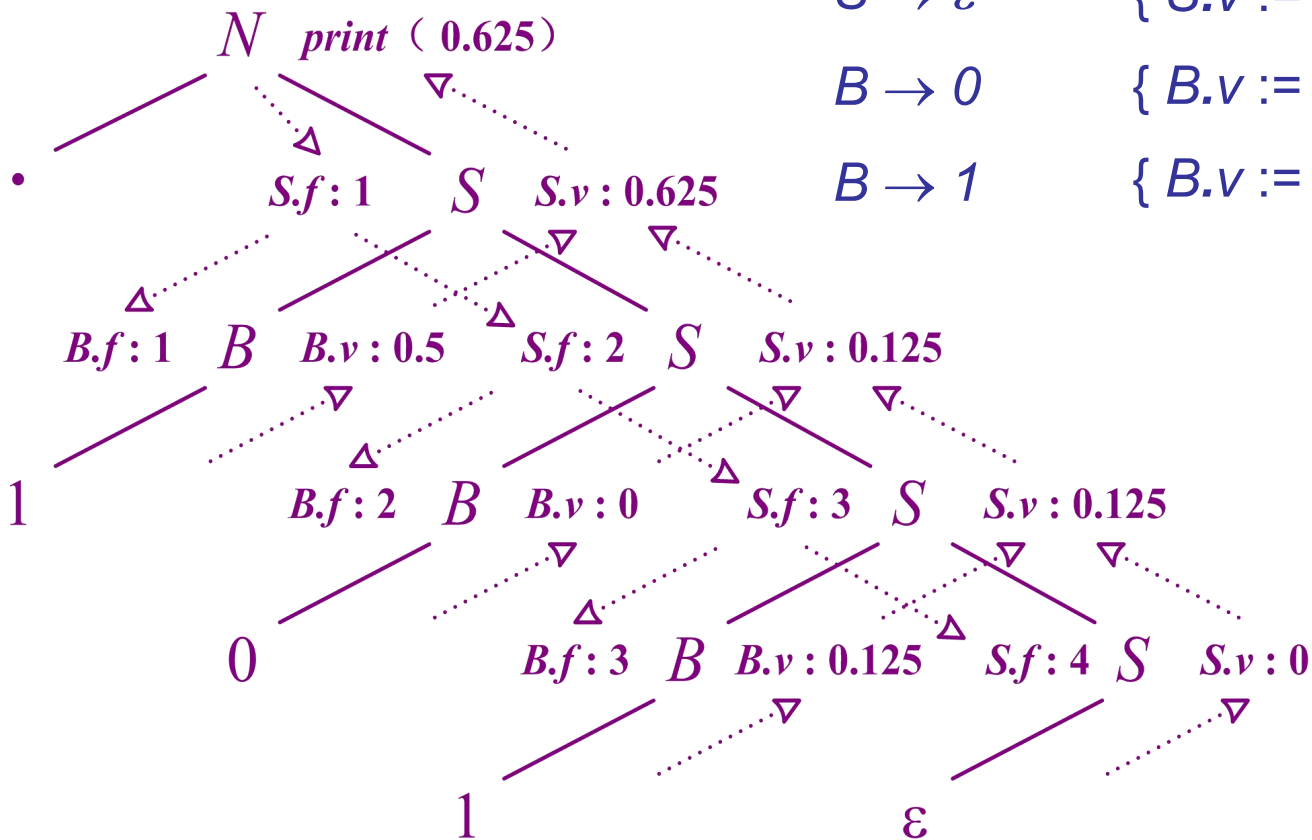
✧ 采用基于深度优先后序遍历算法进行 L-属性文法的语义计算举例

– 考虑对于下列L-属性文法，输入串为 **.101** 时的计算过程



基于属性文法的语义计算

✧ 接上页例子

$$N \rightarrow .S \quad \{ S.f := 1; \text{ print}(S.v) \}$$
$$S \rightarrow BS_1 \quad \{ S_1.f := S.f+1; B.f := S.f; \\ S.v := S_1.v+B.v \}$$
$$S \rightarrow \varepsilon \quad \{ S.v := 0 \}$$
$$B \rightarrow 0 \quad \{ B.v := 0 \}$$
$$B \rightarrow 1 \quad \{ B.v := 2^{-B.f} \}$$


7.2 基于翻译模式的语义计算

◇ 翻译模式 (Translation Scheme) 概念

- 适合语法制导语义计算的另一种描述形式
- 形式上类似于属性文法，但允许由{ }括起来的语义规则集合出现在产生式右端的任何位置. 这样做的好处是可以显式地表达动作和属性计算的次序，而在前述的属性文法中不体现这种次序

基于翻译模式的语义计算

✧ 受限的翻译模式

— 在设计翻译模式时必须作某些限制，确保每个属性值在被访问到的时候已经存在。仅讨论两类受限的翻译模式

- 受S-属性文法的启示，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即可
- 受L-属性文法的启示，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）产生式左部非终结符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。

基于翻译模式的语义计算

✧ 翻译模式举例

— 定点二进制小数转换为十进制小数

$N \rightarrow . \{ S.f := 1 \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ S_1.f := S.f + 1 \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{-B.f} \}$

$N \rightarrow .S$	$\{ S.f := 1; \text{ print}(S.v) \}$
$S \rightarrow BS_1$	$\{ S_1.f := S.f + 1; B.f := S.f, \\ S.v := S_1.v + B.v \}$
$S \rightarrow \varepsilon$	$\{ S.v := 0 \}$
$B \rightarrow 0$	$\{ B.v := 0 \}$
$B \rightarrow 1$	$\{ B.v := 2^{-B.f} \}$

基于翻译模式的语义计算

✧ 基于翻译模式的语义计算

— 仅考虑单遍的方法

- 自上而下的语义计算

借助于自上而下的预测分析技术

- 自下而上的语义计算

借助于自下而上的移进-归约分析技术

— 仅考虑上述受限的翻译模式，

基于翻译模式的语义计算

✧ 基于翻译模式的自上而下语义计算

— 对适合于自上而下预测技术的翻译模式，语法制导的语义计算程序可以如下思路构造

- 对每个非终结符 A，构造一个函数，以 A 的每个继承属性为形参，以 A 的综合属性为返回值（若有多个综合属性，可返回记录类型的值）。如同预测分析程序的构造，该函数代码的流程是根据当前的输入符号来决定调用哪个产生式。
- 与每个产生式相关的代码根据其右端的结构来构造（见下页）

基于翻译模式的语义计算

✧ 语法制导的语义计算程序的构造中，与每个产生式相关的代码根据产生式右端的终结符，非终结符，和语义规则集（语义动作），依从左到右的次序完成下列工作：

➤ 对终结符 X ，保存其综合属性 x 的值到专为 X . x 声明的变量；然后调用匹配终结符（`match_token`）和取下一输入符号（`next_token`）的函数；

➤ 对非终结符 B ，利用相应于 B 的函数 `ParseB` 产生赋值语句

$$c := B(b_1, b_2, \dots, b_k),$$

变量 b_1, b_2, \dots, b_k 对应 B 的继承属性，变量 c 对应 B 的综合属性；

➤ 对语义规则集，直接copy其中每一语义规则来产生代码，只是将对属性的访问替换为对相应变量的访问。

基于翻译模式的语义计算

✧ 基于翻译模式的自上而下语义计算举例

- 构造下列翻译模式的自上而下递归下降（预测）翻译程序（可以验证其基础文法为 LL（1）文法）

$$N \rightarrow . \{ S.f := 1 \} S \{ \text{print}(S.v) \}$$
$$S \rightarrow \{ B.f := S.f \} B \{ S_1.f := S.f + 1 \} S_1 \{ S.v := S_1.v + B.v \}$$
$$S \rightarrow \varepsilon \{ S.v := 0 \}$$
$$B \rightarrow 0 \{ B.v := 0 \}$$
$$B \rightarrow 1 \{ B.v := 2^{-B.f} \}$$

基于翻译模式的语义计算

◇ 基于翻译模式的自上而下语义计算举例

— 根据产生式

$N \rightarrow . \{ S.f := 1 \} S \{ print(S.v) \}$

对非终结符 N ，构造如下函数

```
void ParseN()
{
    MatchToken('.');    //匹配 '.'
    Sf := 1;            //变量 Sf 对应属性 S.f
    Sv := ParseS(Sf);   //变量 Sv 对应属性 S.v
    print(Sv);
}
```

基于翻译模式的语义计算

◇ 基于翻译模式的自上而下语义计算举例

– 根据产生式

$S \rightarrow \{ B.f := S.f \} B \{ S_1.f := S.f + 1 \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

对非终结符 S ，构造如下函数

```
float ParseS( int f )  
{  
    if (lookahead=='0' or lookahead=='1' ) {  
        Bf := f;  Bv := ParseB(Bf); S1f := f+1 ;  
        S1v := ParseS(S1f); Sv := S1v + Bv;  
    }  
    else if (lookahead== '#' ) Sv := 0;  
    else { printf("syntax error \n"); exit(0); }  
    return Sv;  
}
```

基于翻译模式的语义计算

✧ 基于翻译模式的自上而下语义计算举例

— 根据产生式

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{-B.f} \}$

对非终结符 B ，构造如下函数

```
float ParseB( int  $f$  )
```

```
{
```

```
    if (lookahead=='0') { MatchToken('0');  $Bv := 0$  }
```

```
    else if (lookahead== '1') {
```

```
        MatchToken('1');  $Bv := 2^{(-f)}$  }
```

```
        else { printf("syntax error \n"); exit(0); }
```

```
    return  $Bv$ ;
```

```
}
```

基于翻译模式的语义计算

◇ 消除翻译模式中左递归的一种变换方法

如下是常量表达式求值的翻译模式，但含有左递归，因而不能用 LL(1) 方法

$$S \rightarrow E \quad \{ \text{print}(E.val) \}$$
$$E \rightarrow E_1 + T \quad \{ E.val := E_1.val + T.val \}$$
$$E \rightarrow T \quad \{ E.val := T.val \}$$
$$T \rightarrow T_1 * F \quad \{ T.val := T_1.val \times F.val \}$$
$$T \rightarrow F \quad \{ T.val := F.val \}$$
$$F \rightarrow (E) \quad \{ F.val := E.val \}$$
$$F \rightarrow d \quad \{ F.val := d.lexval \}$$

- 若需要消除翻译模式之基础文法中的左递归，那么翻译模式应该如何变化呢？

随后介绍较简单但常用的一种情形

基于翻译模式的语义计算

◇ 消除翻译模式中左递归的一种变换方法

— 假设有如下翻译模式：

$$A \rightarrow A_1 Y \quad \{ A.a: = g(A_1.a, Y.y) \}$$

$$A \rightarrow X \quad \{ A.a: = f(X.x) \}$$

消去关于 A 的直接左递归，基础文法变换为

$$A \rightarrow XR \quad R \rightarrow YR \mid \varepsilon$$

再考虑语义动作，翻译模式变换为

$$A \rightarrow X \{ R.i: = f(X.x) \} R \{ A.a: = R.s \}$$

$$R \rightarrow Y \{ R_1.i: = g(R.i, Y.y) \} R_1 \{ R.s: = R_1.s \}$$

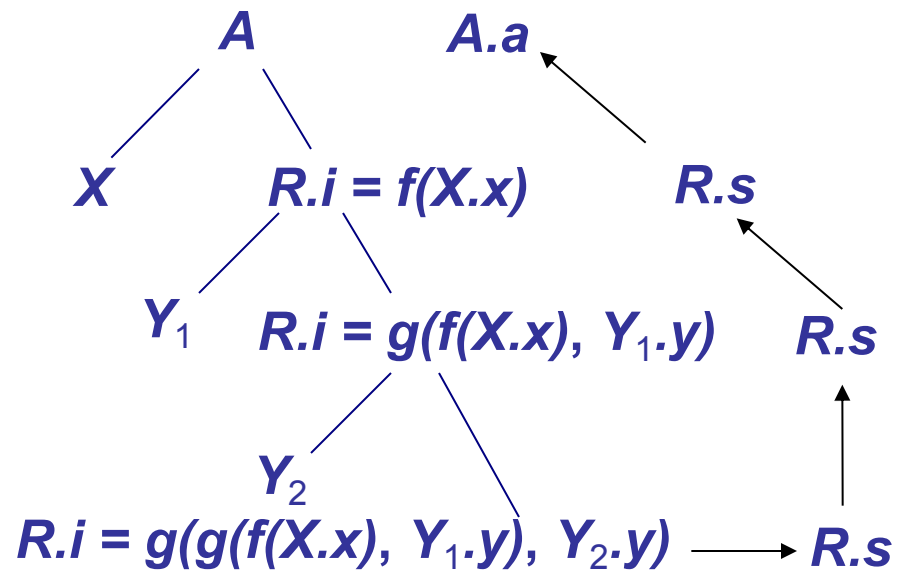
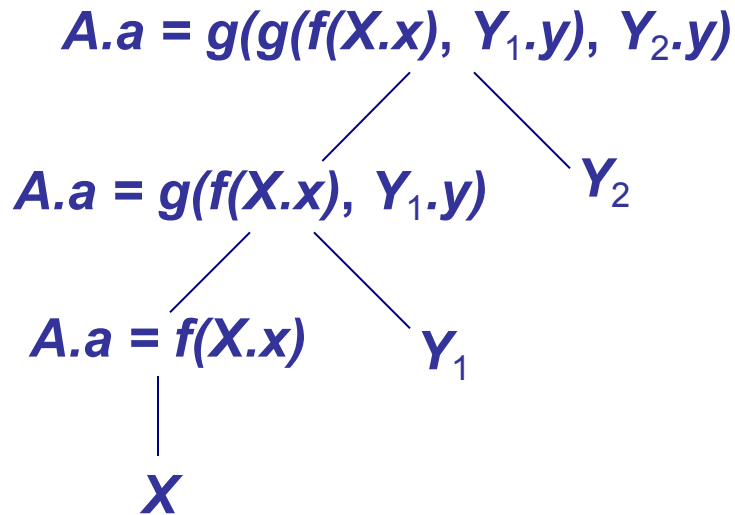
$$R \rightarrow \varepsilon \{ R.s: = R.i \}$$

基于翻译模式的语义计算

✧ 消除翻译模式中左递归的一种变换方法

— 理解这种变换方法

变换前后代表两种不同的计算方式



基于翻译模式的语义计算

✧ 消除翻译模式中左递归的一种变换方法举例

— 消除右边翻译模式中的左递归

$S \rightarrow E$	$\{ \text{print}(E.val) \}$
$E \rightarrow E_1 + T$	$\{ E.val := E_1.val + T.val \}$
$E \rightarrow T$	$\{ E.val := T.val \}$
$T \rightarrow T_1 * F$	$\{ T.val := T_1.val \times F.val \}$
$T \rightarrow F$	$\{ T.val := F.val \}$
$F \rightarrow (E)$	$\{ F.val := E.val \}$
$F \rightarrow d$	$\{ F.val := d.lexval \}$

\Rightarrow

$S \rightarrow E$	$\{ \text{print}(E.val) \}$
$E \rightarrow T$	$\{ R.i := T.val \} \quad R \{ E.val := R.s \}$
$R \rightarrow + T$	$\{ R_1.i := R.i + T.val \} \quad R_1 \{ R.s := R_1.s \}$
$R \rightarrow \varepsilon$	$\{ R.s := R.i \}$
$T \rightarrow F$	$\{ P.i := F.val \} \quad P \{ T.val := P.s \}$
$P \rightarrow * F$	$\{ P_1.i := P.i \times F.val \} \quad P_1 \{ P.s := P_1.s \}$
$P \rightarrow \varepsilon$	$\{ P.s := P.i \}$
$F \rightarrow (E)$	$\{ F.val := E.val \}$
$F \rightarrow d$	$\{ F.val := d.lexval \}$

复 习

基于翻译模式的语义计算

- 自上向下的分析：

非终结符：A.综合属性=函数（A.继承属性）

终结符：值存变量，匹配，读下一个符号

- 自下向上的分析：

翻译模式中去掉嵌在产生式中间的语义动作

依产生式归约时语义计算的代码片断

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

- 扩展前述的关于S-属性文法的自下而上计算技术（即在分析栈中增加存放属性值的域）
- 翻译模式中综合属性的求值采用前述的计算方法
- 对于前述受限的翻译模式，核心问题实际上是L-属性文法的自下而上计算，该问题的讨论较复杂，本节仅涉及如下3个方面的简介
 - 翻译模式中去掉嵌在产生式中间的语义动作
 - 分析栈中继承属性的访问及继承属性的模拟求值
 - 用综合属性代替继承属性

◇ 基于翻译模式的自下而上语义计算

— 从翻译模式中去掉嵌在产生式中间的语义规则集

- 若语义规则集中未关联任何属性，引入新的非终结符 N 和产生式 $N \rightarrow \varepsilon$ ，把嵌入在产生式中间的动作用非终结符 N 代替，把该语义规则集放在产生式后面
- 若语义规则集中有关联的属性，引入新的非终结符 N 和产生式 $N \rightarrow \varepsilon$ ，以及把该语义规则集放在产生式后面的同时，需要在适当的地方增加复写规则
- 目的：使所有嵌入的除复写规则外的语义规则都出现在产生式的末端，以便自下而上计算综合属性

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 从翻译模式中去掉嵌在产生式中间的语义规则集举例

$E \rightarrow T R$
 $R \rightarrow + T \{ \text{print}(' + ') \} R_1$
 $R \rightarrow - T \{ \text{print}(' - ') \} R_1$
 $R \rightarrow \varepsilon$
 $T \rightarrow \underline{\text{num}} \{ \text{print}(\underline{\text{num.val}}) \}$



$E \rightarrow T R$
 $R \rightarrow + T M R_1$
 $R \rightarrow - T N R_1$
 $R \rightarrow \varepsilon$
 $T \rightarrow \underline{\text{num}} \{ \text{print}(\underline{\text{num.val}}) \}$
 $M \rightarrow \varepsilon \{ \text{print}(' + ') \}$
 $N \rightarrow \varepsilon \{ \text{print}(' - ') \}$

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问

- 自下而上语义计算程序根据产生式 $A \rightarrow XY$ 的归约过程中，假设 X 的综合属性 $X.s$ 已经出现在语义栈上. 因为在 Y 以下子树的任何归约之前， $X.s$ 的值一直存在，因此它可以被 Y 继承. 如果用复写规则 $Y.i := X.s$ 来定义 Y 的继承属性 $Y.i$ ，则在需要 $Y.i$ 时，可以使用 $X.s$

将继承属性的访问落实到对其他综合属性的访问。

✧ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问举例

翻译模式

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

$T \rightarrow \underline{int} \{ T.type := integer \} \mid \underline{real} \{ T.type := real \}$

$L \rightarrow \{ L_1.in := L.in \} L_1, v \{ addtype(v.entry, L.in) \}$

$L \rightarrow v \{ addtype(v.entry, L.in) \}$

产生式

依产生式归约时语义计算的代码片断

$D \rightarrow T L$

$T \rightarrow \underline{int}$

$val [top] := integer$

$T \rightarrow \underline{real}$

$val [top] := real$

$L \rightarrow L, v$

$addtype(val [top].entry, val [top-2])$

$L \rightarrow v$

$addtype(val [top].entry, val [top-1])$

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

✧ 基于翻译模式的自下而上语义计算

— 继承属性的模拟求值

- 从上面的讨论可知，分析栈中继承属性的访问是通过栈中已有文法符号的综合属性值间接进行的，因此设计翻译模式时需要做到的一点就是要保证继承属性总可以通过某个文法符号的综合属性体现出来
- 必要时，通过增加新的文法符号以及相应的复写规则常常可以达到上述目的

✧ 基于翻译模式的自下而上语义计算

— 继承属性的模拟求值举例

考虑如下翻译模式：

$$\begin{aligned} S &\rightarrow a A \{C.i := A.s\} C \mid b A B \{C.i := A.s\} C \\ C &\rightarrow c \{C.s := g(C.i)\} \end{aligned}$$

若直接应用上述复写规则的计算方法，则在使用 $C \rightarrow c$ 进行归约时， $C.i$ 的值或存在于次栈顶 ($top-1$)，或存在于次次栈顶 ($top-2$)，不能确定用哪一个。一种可行的做法是引入新的非终结符 M ，将以上翻译模式改造为：

$$\begin{aligned} S &\rightarrow a A \{C.i := A.s\} C \mid b A B \{M.i := A.s\} M \{C.i := M.s\} C \\ C &\rightarrow c \{C.s := g(C.i)\} \\ M &\rightarrow \varepsilon \{M.s := M.i\} \end{aligned}$$

这样，在使用 $C \rightarrow c$ 进行归约时， $C.i$ 的值就一定可以通过访问次栈顶 ($top-1$) 得到

◇ 基于翻译模式的自下而上语义计算

— 继承属性的模拟求值举例

考虑如下翻译模式：

$$S \rightarrow a A \{C.i := f(A.s)\} C$$

这里，继承属性 $C.i$ 不是通过复写规则来求值，而是通过普通函数 $f(A.s)$ 调用来计算。在计算 $C.i$ 时， $A.s$ 在语义栈上，但 $f(A.s)$ 并未存在于语义栈。同样，一种做法是引入新的非终结符 M ，将以上翻译模式改造为：

$$S \rightarrow a A \{M.i := A.s\} M \{C.i := M.s\} C$$

$$M \rightarrow \varepsilon \{M.s := f(M.i)\}$$

这样，就解决了上述问题。

✧ 基于翻译模式的自下而上语义计算

— 继承属性的模拟求值 (较复杂的例子)

要求：产生式中间的语义动作中仅含复写规则

$N \rightarrow . \{ S.f := 1 \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ S_1.f := S.f + 1 \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

基于翻译模式的语义计算

✧ 基于翻译模式的自下而上语义计算

—分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$
 $S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$
 $S \rightarrow \varepsilon \{ S.v := 0 \}$
 $B \rightarrow 0 \{ B.v := 0 \}$
 $B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$
 $M \rightarrow \varepsilon \{ M.s := 1 \}$
 $P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

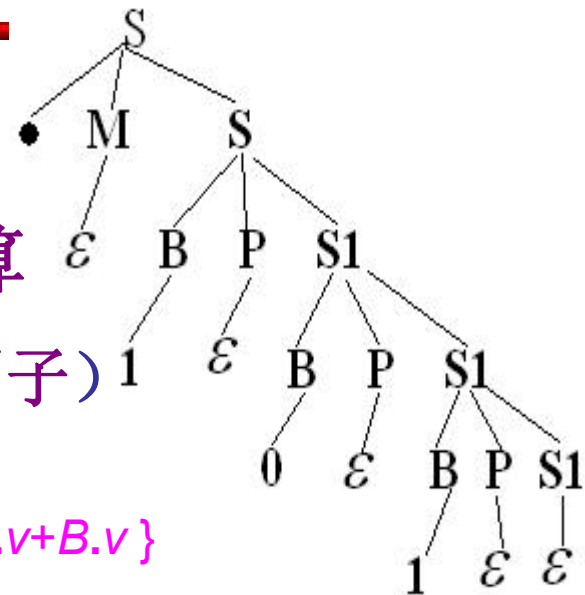
产生式

$N \rightarrow . M S$
 $S \rightarrow B P S_1$
 $S \rightarrow \varepsilon$
 $B \rightarrow 0$
 $B \rightarrow 1$
 $M \rightarrow \varepsilon$
 $P \rightarrow \varepsilon$

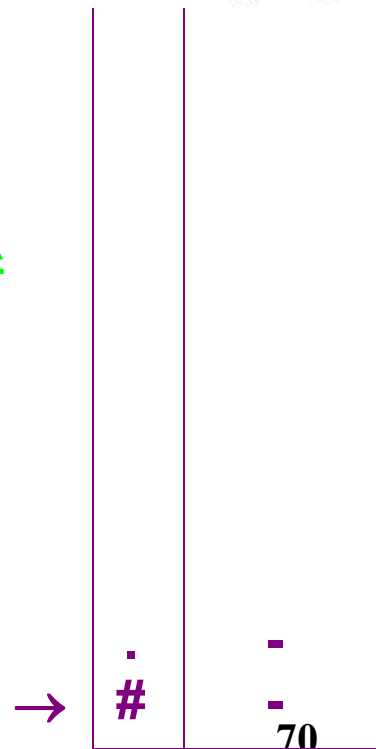
依产生式归约时语义计算的代码片断

$\text{print}(\text{val}[\text{top}].v)$
 $\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$
 $\text{val}[\text{top}+1].v := 0$
 $\text{val}[\text{top}].v := 0$
 $\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$
 $\text{val}[\text{top}+1].s := 1$
 $\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)



例：处理输入串 .101



基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

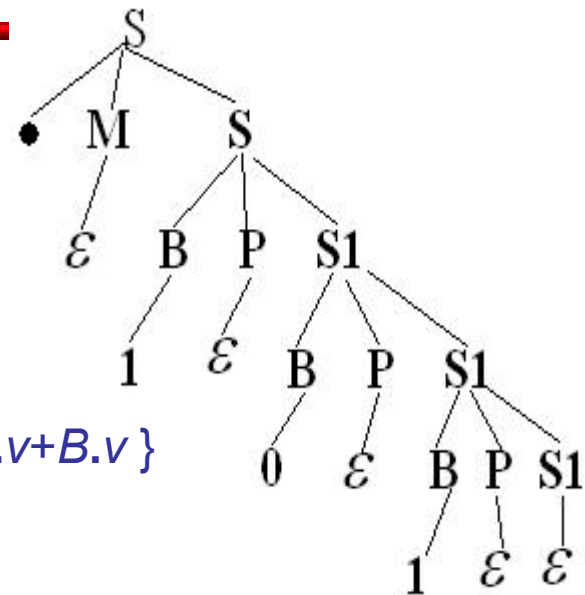
$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$



例: 处理输入串 .101

产生式

依产生式归约时语义计算的代码片断

$N \rightarrow . M S$

`print(val [top] .v)`

$S \rightarrow B P S_1$

`val [top-2].v := val [top].v + val [top-2].v`

$S \rightarrow \varepsilon$

`val [top+1].v := 0`

$B \rightarrow 0$

`val [top].v := 0`

$B \rightarrow 1$

`val [top].v := 2(-val [top-1].s)`

$M \rightarrow \varepsilon$

`val [top+1].s := 1`

$P \rightarrow \varepsilon$

`val [top+1].s := val [top-1].s+1`

(分析栈 **val** 存放文法符号的综合属性, **top** 为栈顶指针)

→

M	1
.	-
#	-
	71

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例：处理输入串 **.101**

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

print(val [top] .v)

val [top-2].v := val [top].v + val [top-2].v

val [top+1].v := 0

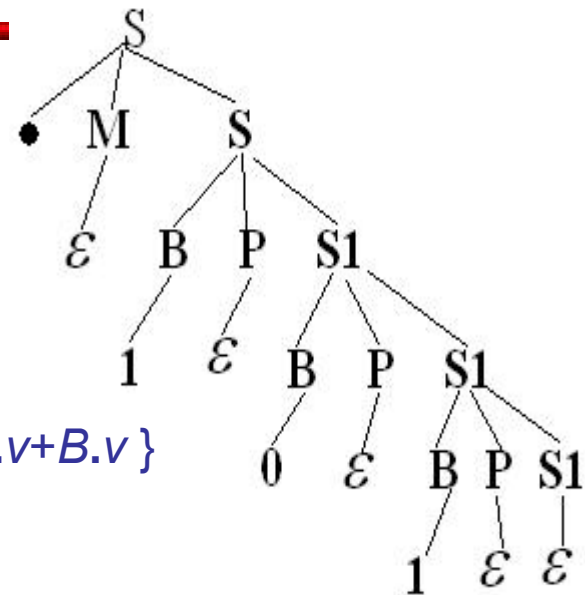
val [top].v := 0

val [top].v := 2^(-val [top-1].s)

val [top+1].s := 1

val [top+1].s := val [top-1].s+1

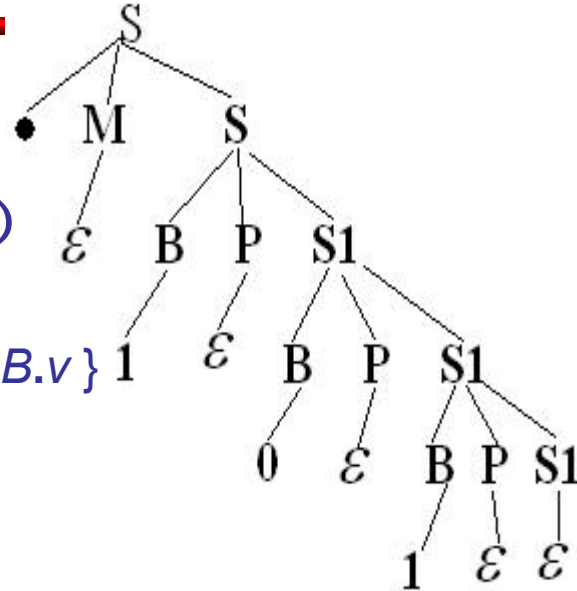
(分析栈 *val* 存放文法符号的综合属性, *top* 为栈顶指针)



→

1	-
M	1
.	-
#	-
	72

基于翻译模式的自下而上语义计算



—分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$
 $S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$
 $S \rightarrow \varepsilon \{ S.v := 0 \}$
 $B \rightarrow 0 \{ B.v := 0 \}$
 $B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$
 $M \rightarrow \varepsilon \{ M.s := 1 \}$
 $P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

产生式

依产生式归约时语义计算的代码片断

$N \rightarrow . M S$	<code>print(val [top] .v)</code>
$S \rightarrow B P S_1$	<code>val [top-2].v := val [top].v + val [top-2].v</code>
$S \rightarrow \varepsilon$	<code>val [top+1].v := 0</code>
$B \rightarrow 0$	<code>val [top].v := 0</code>
$B \rightarrow 1$	<code>val [top].v := 2^(-val [top-1].s)</code>
$M \rightarrow \varepsilon$	<code>val [top+1].s := 1</code>
$P \rightarrow \varepsilon$	<code>val [top+1].s := val [top-1].s+1</code>

(分析栈val 存放文法符号的综合属性, top为栈顶指针)

→

1	-
M	1
.	-
#	-
	73

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$
 $S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$
 $S \rightarrow \varepsilon \{ S.v := 0 \}$
 $B \rightarrow 0 \{ B.v := 0 \}$
 $B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$
 $M \rightarrow \varepsilon \{ M.s := 1 \}$
 $P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

产生式

$N \rightarrow . M S$
 $S \rightarrow B P S_1$
 $S \rightarrow \varepsilon$
 $B \rightarrow 0$
 $B \rightarrow 1$
 $M \rightarrow \varepsilon$
 $P \rightarrow \varepsilon$

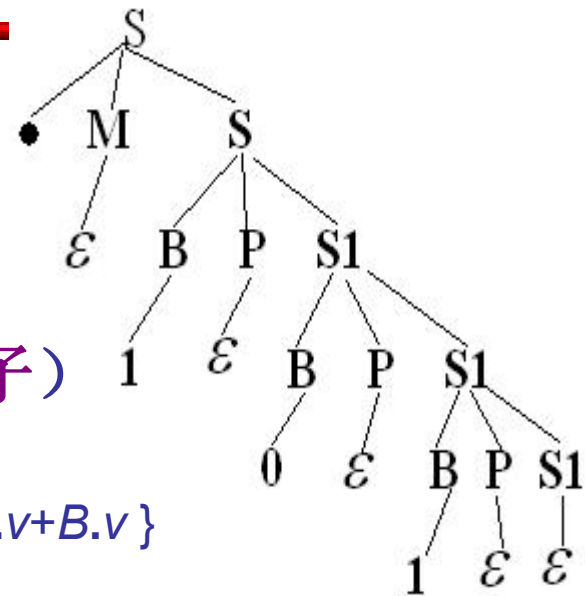
依产生式归约时语义计算的代码片断

$print(val[top].v)$
 $val[top-2].v := val[top].v + val[top-2].v$
 $val[top+1].v := 0$
 $val[top].v := 0$
 $val[top].v := 2^{(-val[top-1].s)}$
 $val[top+1].s := 1$
 $val[top+1].s := val[top-1].s + 1$

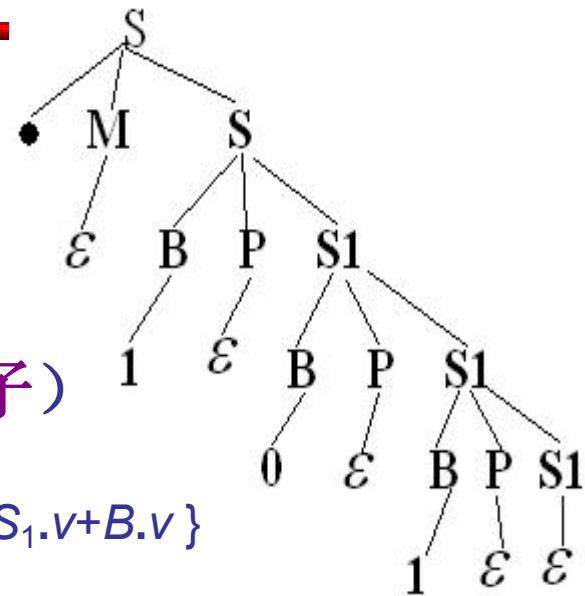
→

P	2
B	0.5
M	1
.	-
#	-
	74

(分析栈 *val* 存放文法符号的综合属性, *top* 为栈顶指针)



基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v \rightarrow$

$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

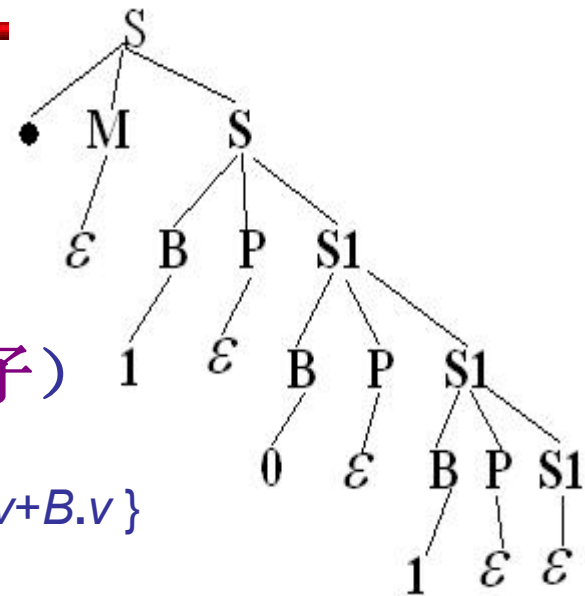
$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

0	-
P	2
B	0.5
M	1
.	-
#	-
	75

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$print(val [top] .v)$

$val [top-2].v := val [top].v + val [top-2].v$

$val [top+1].v := 0$

$val [top].v := 0$

$val [top].v := 2^{(-val [top-1].s)}$

$val [top+1].s := 1$

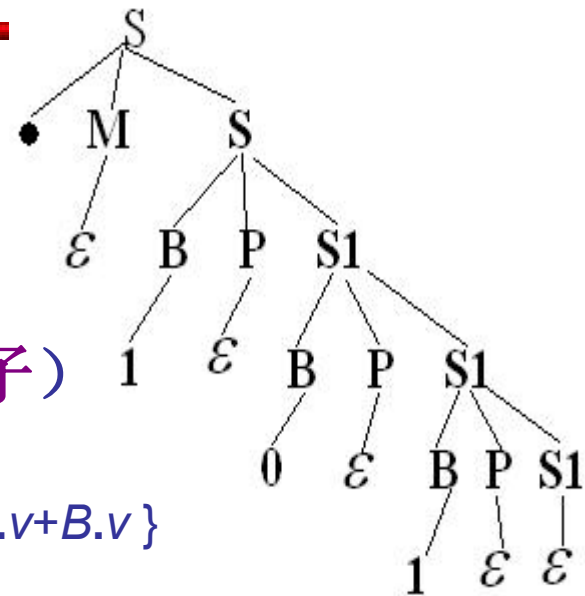
$val [top+1].s := val [top-1].s + 1$

→

0	-
P	2
B	0.5
M	1
.	-
#	-
	76

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$print(val[top].v)$

$val[top-2].v := val[top].v + val[top-2].v$

$val[top+1].v := 0$

$val[top].v := 0$

$val[top].v := 2^{(-val[top-1].s)}$

$val[top+1].s := 1$

$val[top+1].s := val[top-1].s + 1$

→

P	3
B	0
P	2
B	0.5
M	1
.	-
#	-

77

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断 →

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

$\text{val}[\text{top}+1].v := 0$

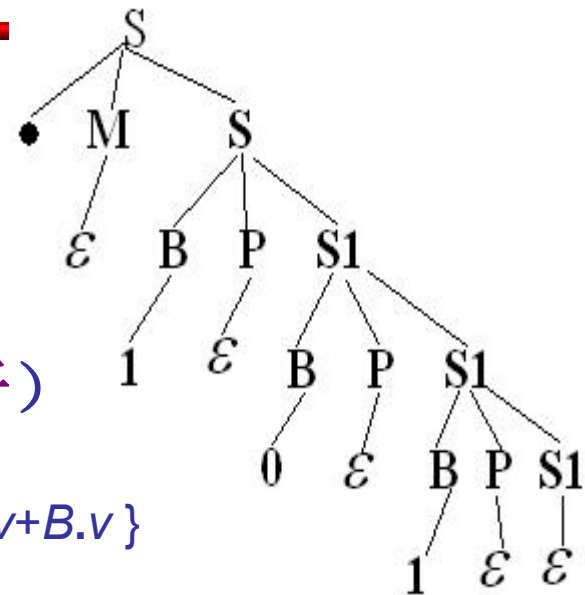
$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)



例: 处理输入串 **.101**

1	-
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	78

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片段

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

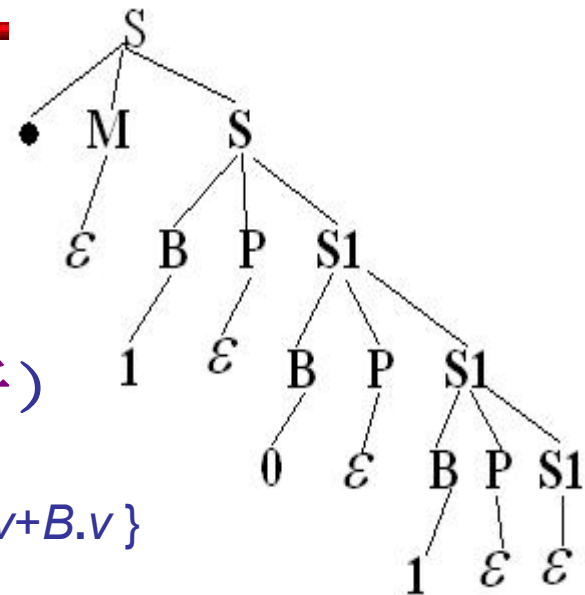
$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

例：处理输入串 **.101**

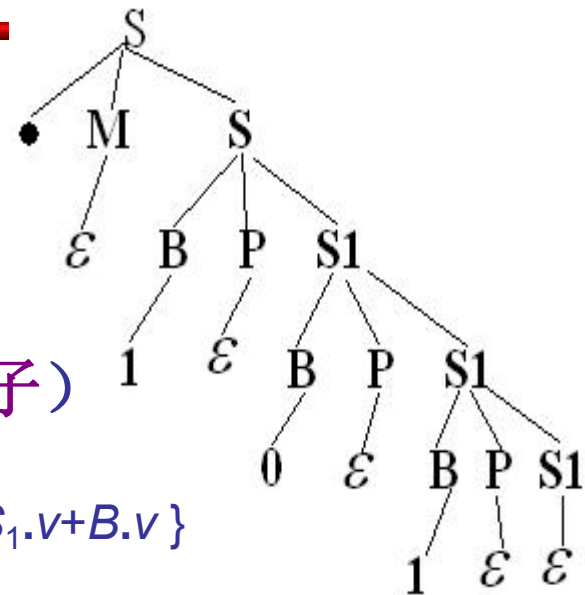
→

1	-
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	79

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)



基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例: 处理输入串 **.101**

→

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片段

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

P	4
B	0.125
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	80

(分析栈 *val* 存放文法符号的综合属性, *top* 为栈顶指针)

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片段

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

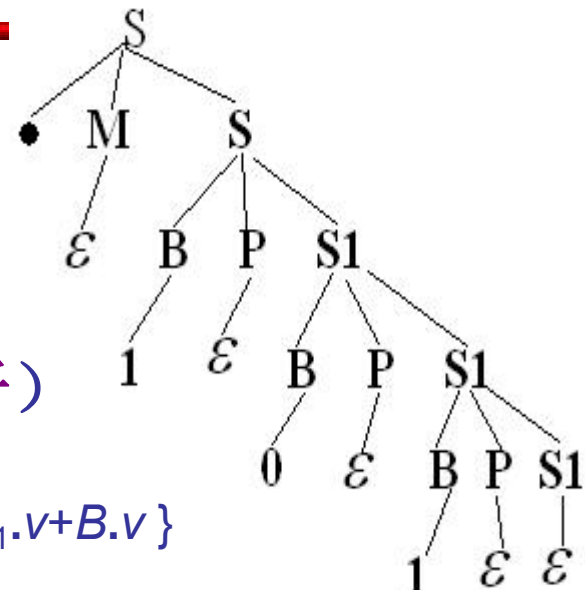
$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$



例: 处理输入串 **.101** →

S	0
P	4
B	0.125
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	81

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

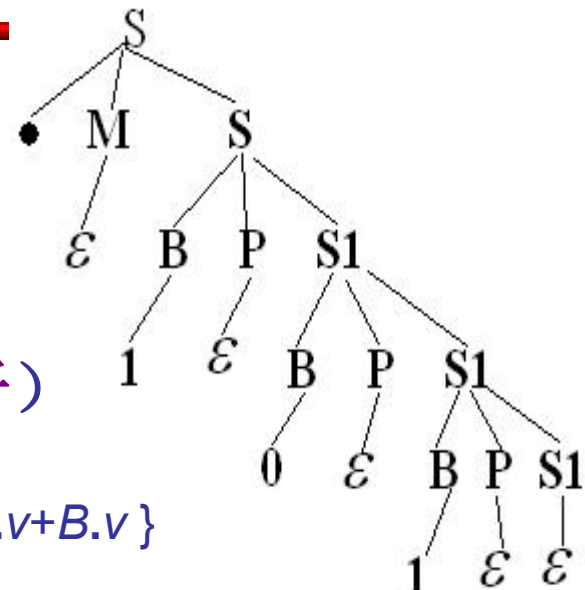
$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

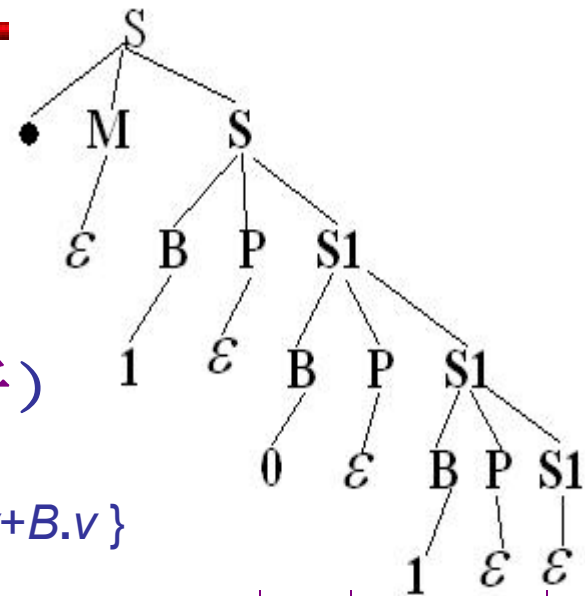


例: 处理输入串 .101

S	0
P	4
B	0.125
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	82

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例：处理输入串 **.101**

→

S	0.125
P	3
B	0
P	2
B	0.5
M	1
.	-
#	-
	83

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片段

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

$\text{val}[\text{top}+1].v := 0$

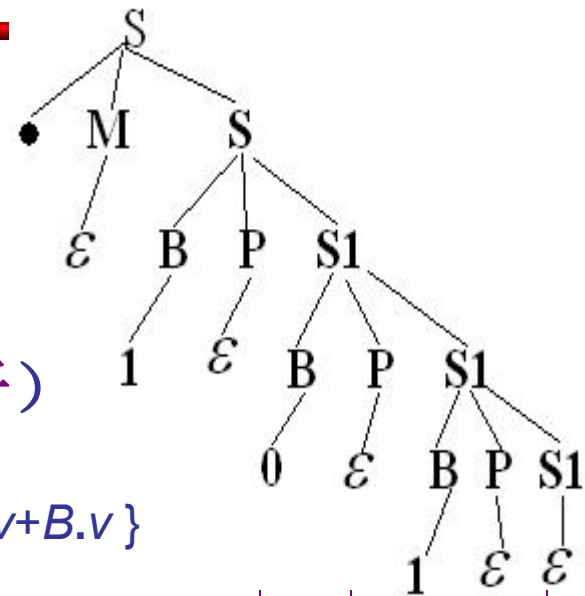
$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$

例：处理输入串 **.101**



S	0.125
P	2
B	0.5
M	1
.	-
#	-
	84

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算

◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

$\text{print}(\text{val}[\text{top}].v)$

$\text{val}[\text{top}-2].v := \text{val}[\text{top}].v + \text{val}[\text{top}-2].v$

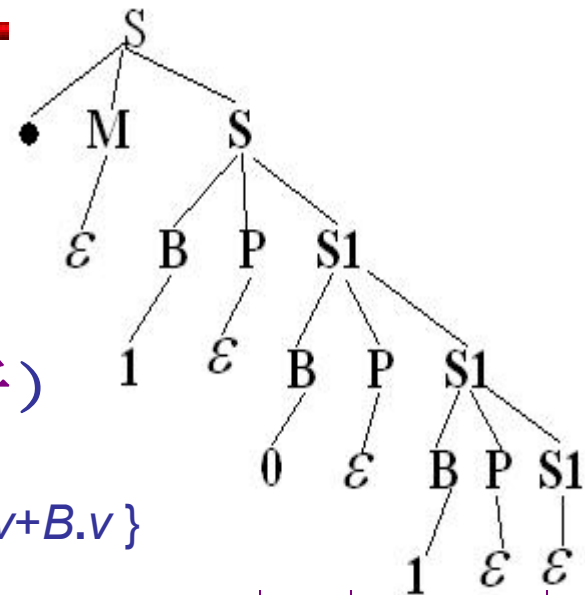
$\text{val}[\text{top}+1].v := 0$

$\text{val}[\text{top}].v := 0$

$\text{val}[\text{top}].v := 2^{(-\text{val}[\text{top}-1].s)}$

$\text{val}[\text{top}+1].s := 1$

$\text{val}[\text{top}+1].s := \text{val}[\text{top}-1].s + 1$



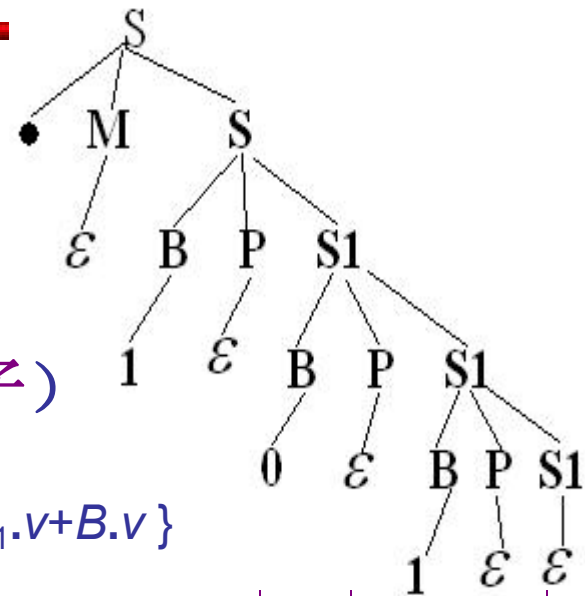
例: 处理输入串 **.101**

print 0.625 →

S	0.625
M	1
.	-
#	-
	85

(分析栈 val 存放文法符号的综合属性, top 为栈顶指针)

基于翻译模式的语义计算



◇ 基于翻译模式的自下而上语义计算

— 分析栈中继承属性的访问（较复杂的例子）

$N \rightarrow . M \{ S.f := M.s \} S \{ \text{print}(S.v) \}$

$S \rightarrow \{ B.f := S.f \} B \{ P.i := S.f \} P \{ S_1.f := P.s \} S_1 \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon \{ S.v := 0 \}$

$B \rightarrow 0 \{ B.v := 0 \}$

$B \rightarrow 1 \{ B.v := 2^{(-B.f)} \}$

$M \rightarrow \varepsilon \{ M.s := 1 \}$

$P \rightarrow \varepsilon \{ P.s := P.i + 1 \}$

例：处理输入串 **.101**

产生式

$N \rightarrow . M S$

$S \rightarrow B P S_1$

$S \rightarrow \varepsilon$

$B \rightarrow 0$

$B \rightarrow 1$

$M \rightarrow \varepsilon$

$P \rightarrow \varepsilon$

依产生式归约时语义计算的代码片断

`print(val [top] .v)`

`val [top-2].v := val [top].v + val [top-2].v`

`val [top+1].v := 0`

`val [top].v := 0`

`val [top].v := 2(-val [top-1].s)`

`val [top+1].s := 1`

`val [top+1].s := val [top-1].s + 1`

acc →

N	-
#	-
	86

(分析栈 *val* 存放文法符号的综合属性, *top* 为栈顶指针)

总 结

- 基于属性文法的语义计算
- 基于翻译模式的语义计算