讲稿（教学内容、步骤）

第 7 章 语法制导的语义计算

1. 语义处理的任务和功能

(1) **静态语义分析**：收集或计算源程序的上下文相关信息，并将这些信息分配到相应的程序单元记录下来；同时审查程序的静态一致性和完整性（即静态语义检查）

- ✓ 确定标识符的数据类型
- ✓ 类型检查和转换：检查运算对象的数据类型是否合法，必要时进行类型转换
- ✓ 一致性检查：一个对象只能被声明一次
- ✓ 作用域检查
- ✓ 控制流检查：控制语句转到合法的地方继续执行

(2) 将通过静态语义检查的程序**翻译**成中间代码

2. 语义处理的方法

- **语法制导的语义计算**，又称为 **语法制导的翻译**

- 其阶段包括：
  - ✓ 语法制导的**静态语义分析**
  - ✓ 语法制导的**中间代码生成**

- 实现途径：
  - ✓ 首先，使用**属性文法**为工具，描述程序设计语言的语义规则。
  - ✓ 在语法分析时，每应用一个产生式（推导或归约），同时完成该产生式上所附的语义规则描述的动作，从而完成语义处理。

- 语义计算模型包括：
  - ① 基于属性文法的语义计算模型
  - ② 基于翻译模式的语义计算模型

3. 基于属性文法的语义计算

(1) 属性文法

- 什么是属性文法？
  - ◆ 是一种用于描述语义规则的文法。
  - ◆ 为文法符号关联有特定意义的属性（这些属性代表与文法符号相关的信息，如：类型、值、代码序列、符号表内容等），并为产生式关联相应的语义动作或条件谓词
  - ◆ 属性值可以在语法分析过程中进行计算和传递，计算和传递的过程就是语义的处理过程。

【举例】

例 1　语言 $L=\{a^nb^nc^n \mid n\geq 1\}$，写出其对应的 2 型文法

$S \rightarrow ABC$

$A \rightarrow Aa$

$A \rightarrow a$

$B \rightarrow Bb$

$B \rightarrow b$

$C \rightarrow Cc$

$C \rightarrow c$

【讨论】该文法对应的语言是 aa*bb*cc*，无法保证|A|=|B|=|C|，如何解决？

解决：采用属性文法，为每个产生式都关联一个语义计算规则的集合。

例1　语言 $L=\{a^nb^nc^n \mid n\geq 1\}$

属性文法

(1) S→ABC { if(A.num=B.num) and (B.num=C.num)
then print("Accepted!")
else print("Refused!") }

(2) $A \rightarrow A_1a$ { A.num := A_1.num+1 }

(3) A→a { A.num := 1 }

(4) $B \rightarrow B_1b$ { B.num := B_1.num+1 }

(5) B→b { B.num := 1 }

(6) $C \rightarrow C_1c$ { C.num := C_1.num+1 }

(7) C→c { C.num := 1 }

其中，

- num 为符号的**关联属性**;
- 花括号内为**语义计算规则**;
- if 后是条件谓词;
- 还有一些语句是语义动作 b:=f(c1,c2,…,ck)，包括：
  - ✓ 语义函数 f(c1,c2,…,ck)
  - ✓ 复写规则 X.a:=Y.b

- 属性的类型

- 综合属性：对关联于 A→…… 的语义函数 b:=f(c1,c2,…)，如果 b 是 左部 A 的某个属性，则称 b是A的一个综合属性。

  计算综合属性是对父结点的属性进行赋值，是自底向上传递信息。例1中的num是综合属性

- 继承属性：对关联于A→…X…的语义函数 b:=f(c1,c2,…)，如果b是产生式右部某符号X的某个属性，则称 b是X的一个继承属性。

  计算继承属性是对子结点的属性进行赋值，是自顶向下传递信息。 参见例2:

【举例】

例2 $L=\{a^nb^nc^n \mid n\geq 1\}$ 含有继承属性的属性文法

(1) S→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!")}

(2) $A\rightarrow A_1a$ { $A.num:=A_1.num+1$ }

(3) A→a { A.num:=1 }

(4) $B\rightarrow B_1b$ { $B_1.in\_num:=B.in\_num$; $B.num:=B_1.num-1$ }

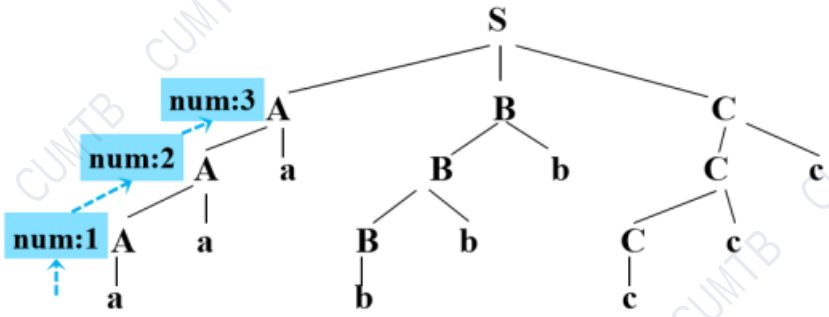
(5) B→b { B.num:=B.in_num - 1 }

(6) $C\rightarrow C_1c$ { $C_1.in\_num:=C.in\_num$; $C.num:=C_1.num-1$ }

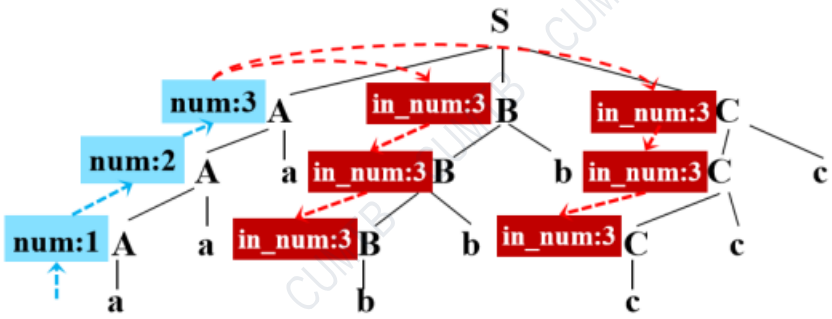
(7) C→c { C.num:=C.in_num-1 }

- num为综合属性，自底向上计算
- in_num为继承属性，自顶向下计算

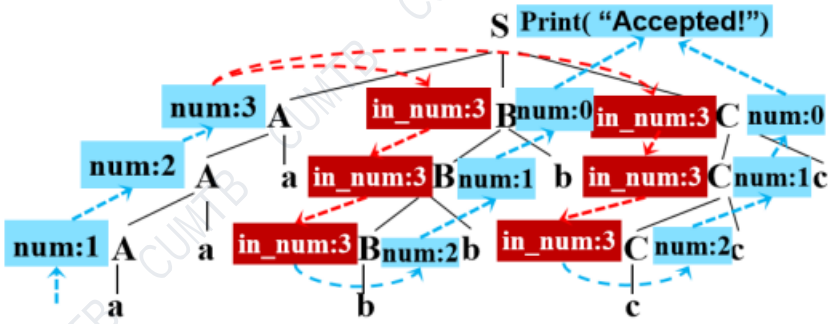
构造 aaabbbccc 带标注的语法分析树：



自底向上计算综合属性



自顶向下计算继承属性



自底向上计算综合属性

(2) 遍历分析树进行语义计算

① **构造输入串的语法分析树**

【举例】例 3　将二进制无符号小数转化成十进制小数的属性文法如下

$N \to S_1.S_2$ { N.val:=S_1.val+S_2.val;
S_1.f:=1;
S_2.f:=$2^{-S2.len}$ }

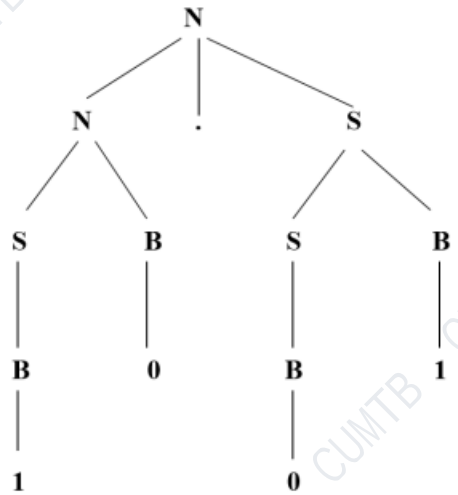
$S \to S_1B$ { S_1.f:=2S.f;
B.f:=S.f;
S.val:=S_1.val+B.val;
S.len:=S_1.len+1 }

$S \to B$ { S.len:=1;
S.val:=B.val;
B.f:=S.f }

$B \to 0$ { B.val:=0 }

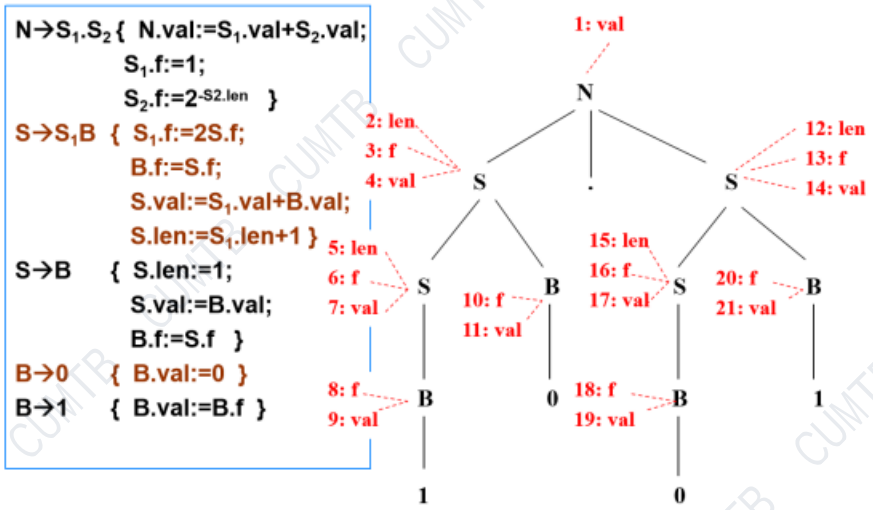
$B \to 1$ { B.val:=B.f }

构造输入串 10.01 的语法分析树：



② 构造依赖图

- 构造依赖图的结点：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。



- 构造依赖图的有向边：

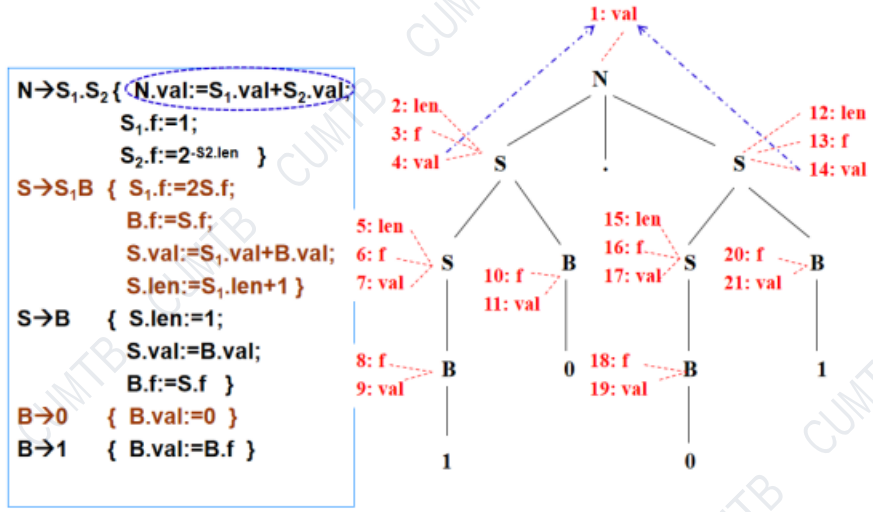
for 结点 n 所用产生式对应 b:=f(c1,c2,...,ck)

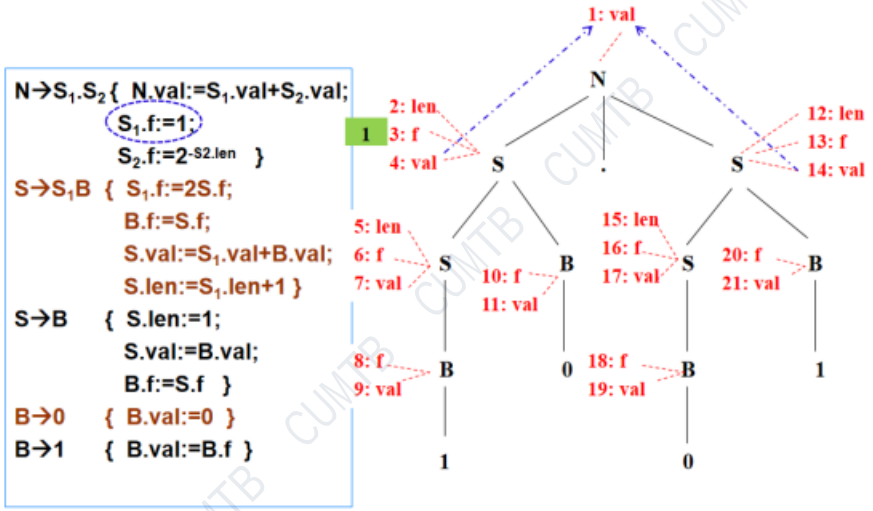
{ for i=1 to k

从结点 ci 到结点 b 构造一条有向边

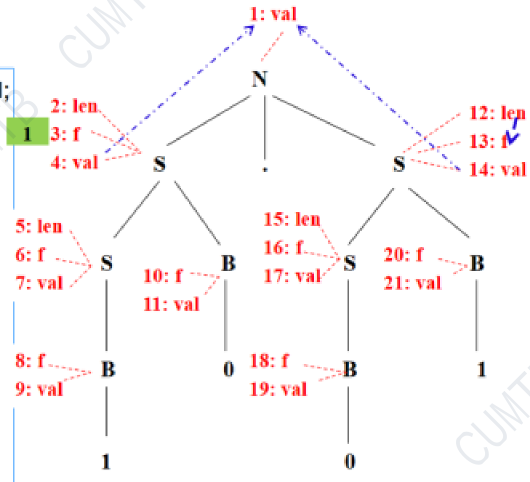
}

【举例】对上例的依赖图构造有向边

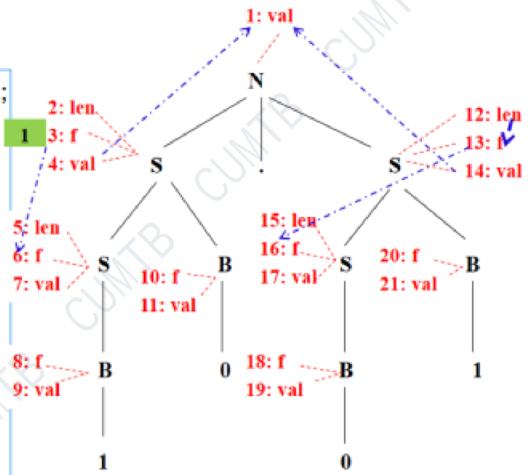




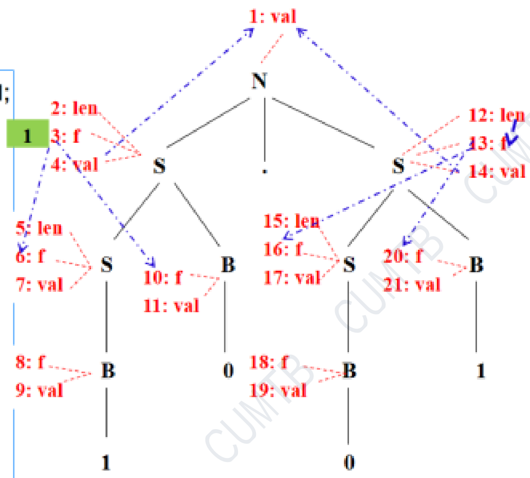
N→$S_1.S_2$ { N.val:=S_1.val+S_2.val;
S_1.f:=1;
S_2.f:=$2^{-S2.len}$ }
S→S_1B { S_1.f:=2S.f;
B.f:=S.f;
S.val:=S_1.val+B.val;
S.len:=S_1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }

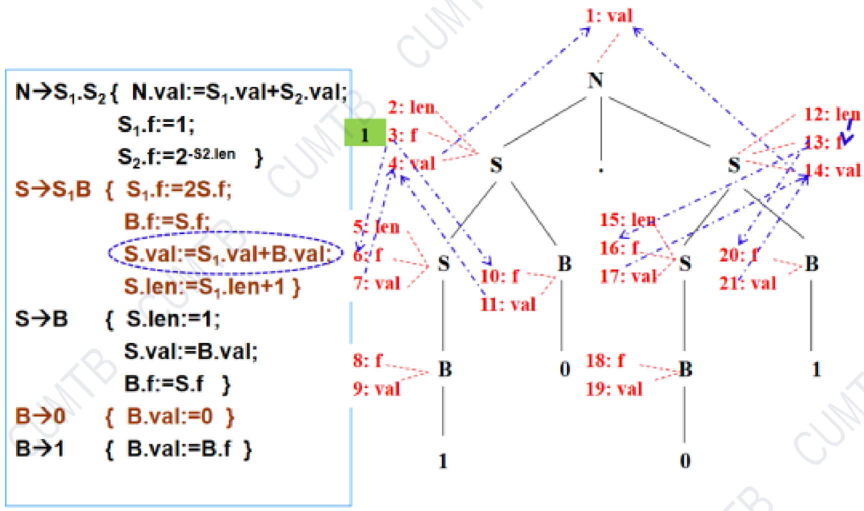


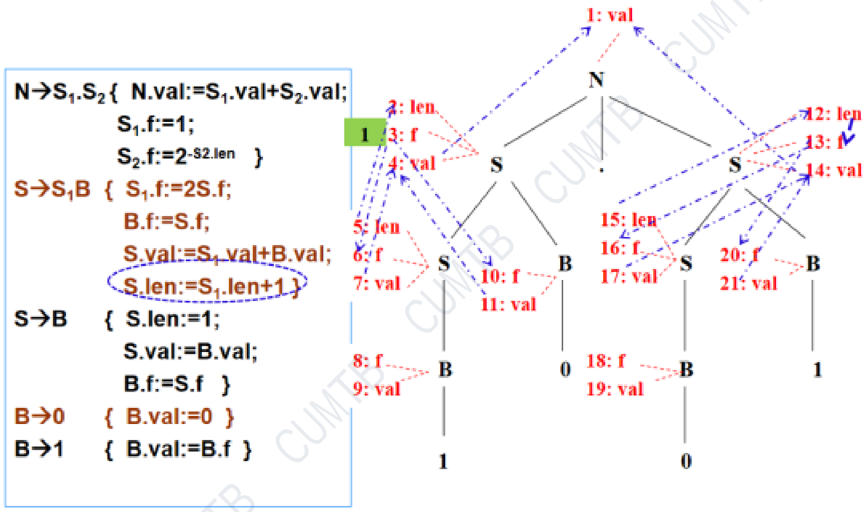
N→$S_1.S_2$ { N.val:=S_1.val+S_2.val;
S_1.f:=1;
S_2.f:=$2^{-S2.len}$ }
S→S_1B { S_1.f:=2S.f;
B.f:=S.f;
S.val:=S_1.val+B.val;
S.len:=S_1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }

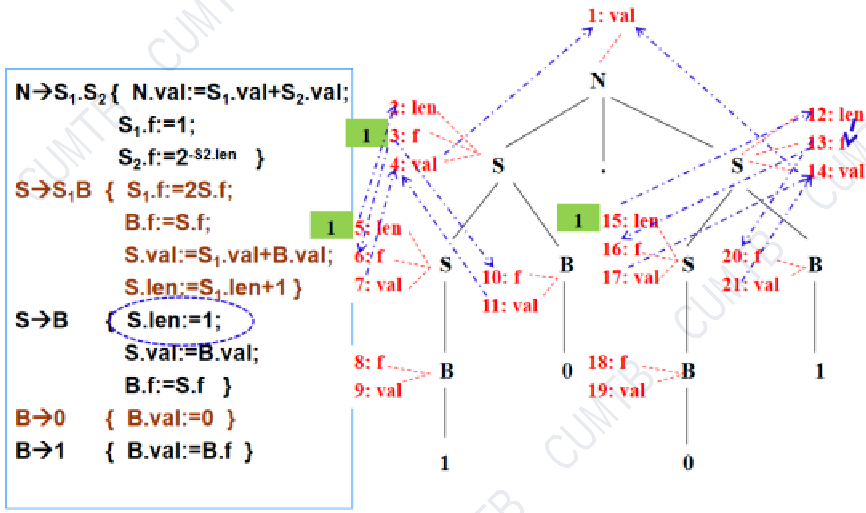


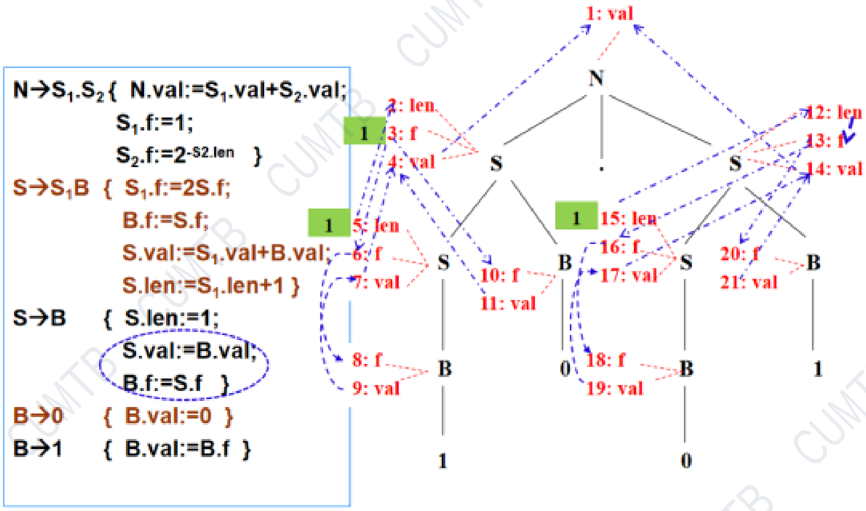
N→$S_1.S_2$ { N.val:=S_1.val+S_2.val;
S_1.f:=1;
S_2.f:=$2^{-S2.len}$ }
S→S_1B { S_1.f:=2S.f;
B.f:=S.f;
S.val:=S_1.val+B.val;
S.len:=S_1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }

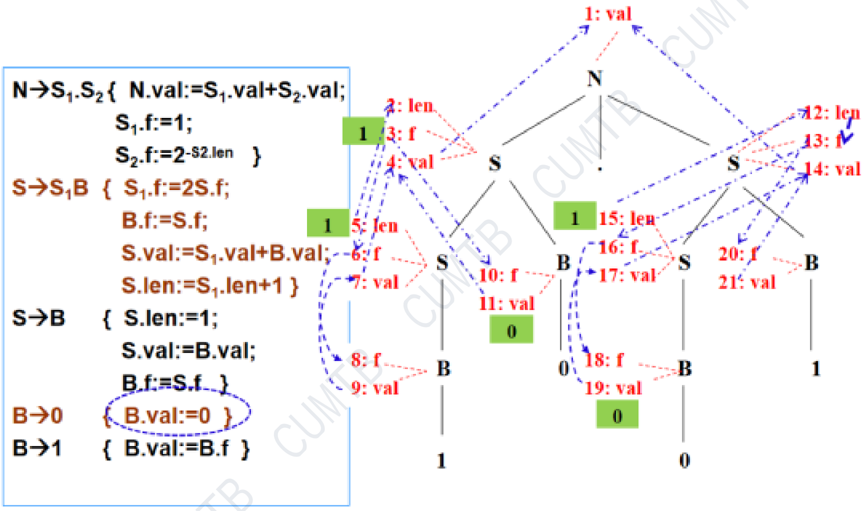


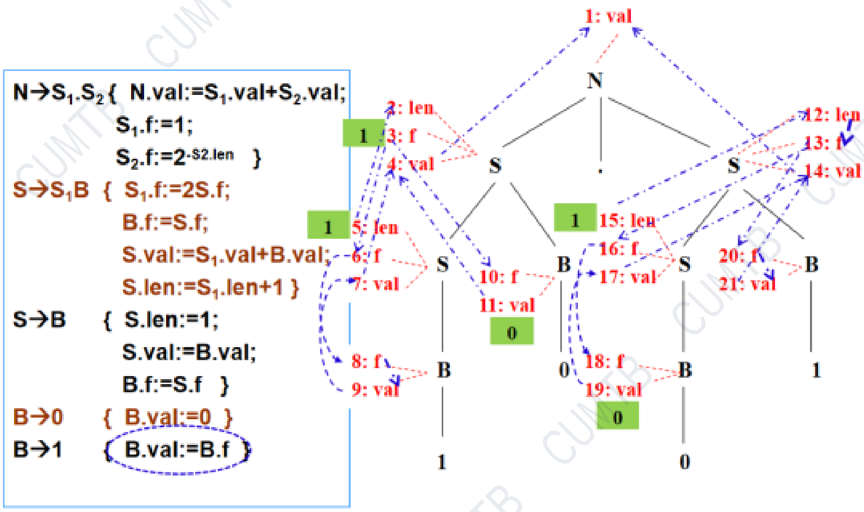

N→S1.S2 { N.val:=S1.val+S2.val;
S1.f:=1;
S2.f:=2-S2.len }
S→S1B { S1.f:=2S.f;
B.f:=S.f;
S.val:=S1.val+B.val;
S.len:=S1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }
1
1: val
2: len
3: f
4: val
5: len
6: f
7: val
8: f
9: val
10: f
11: val
12: len
13: f
14: val
15: len
16: f
17: val
18: f
19: val
20: f
21: val


N→S1.S2 { N.val:=S1.val+S2.val;
S1.f:=1;
S2.f:=2-S2.len }
S→S1B { S1.f:=2S.f;
B.f:=S.f;
S.val:=S1.val+B.val;
S.len:=S1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }
1


N→S1.S2 { N.val:=S1.val+S2.val;
S1.f:=1;
S2.f:=2-S2.len }
S→S1B { S1.f:=2S.f;
B.f:=S.f;
S.val:=S1.val+B.val;
S.len:=S1.len+1 }
S→B { S.len:=1;
S.val:=B.val;
B.f:=S.f }
B→0 { B.val:=0 }
B→1 { B.val:=B.f }
1
1
1





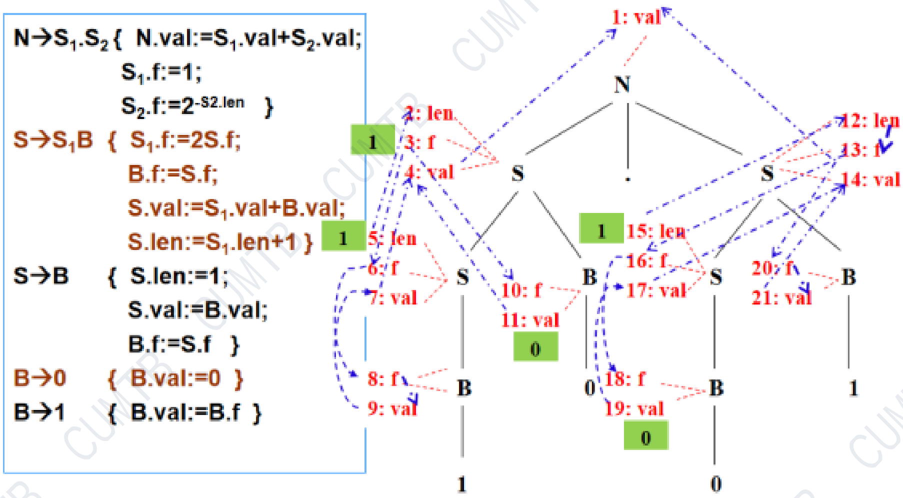


③ 计算属性值

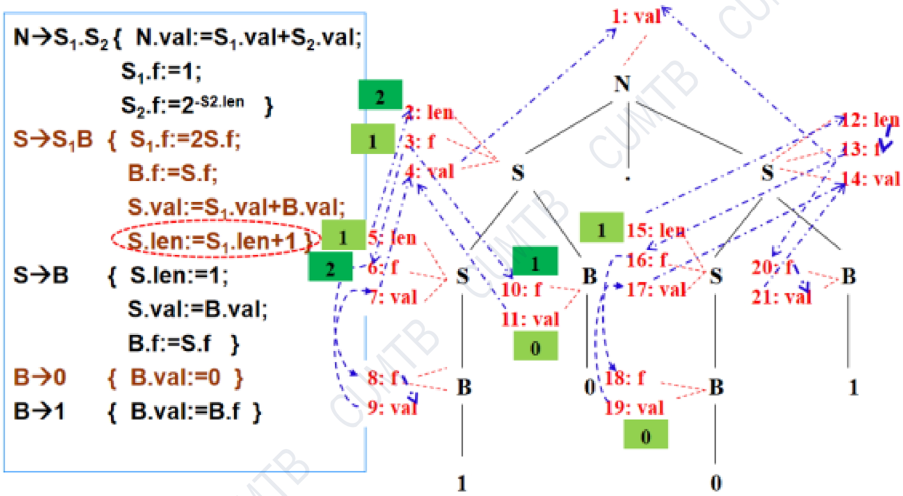
若该依赖图是无圈的，则按此图结点的任意一种拓扑排序，对分析树进行遍历，计算所有的属性值。

【举例】对上例计算属性值

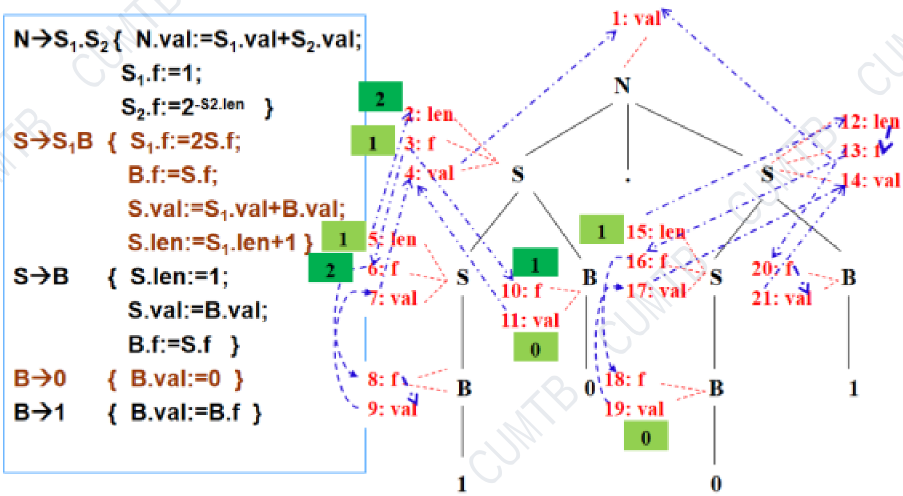
假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



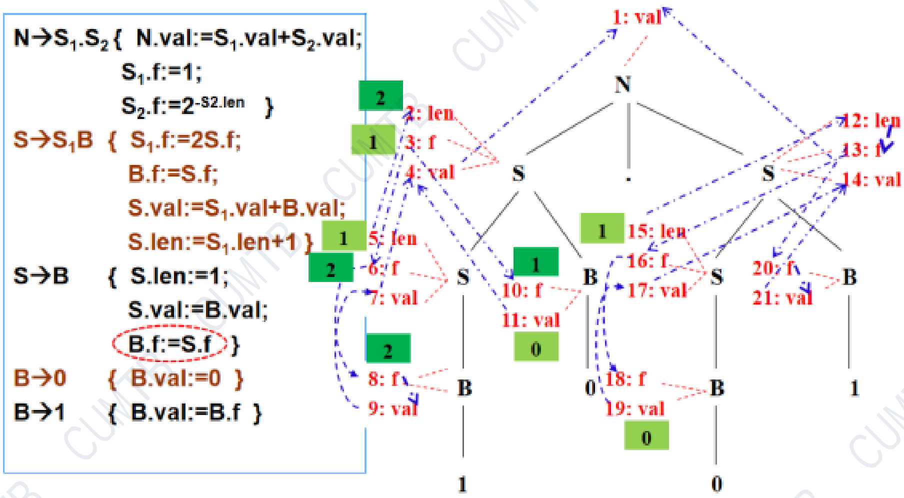
假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



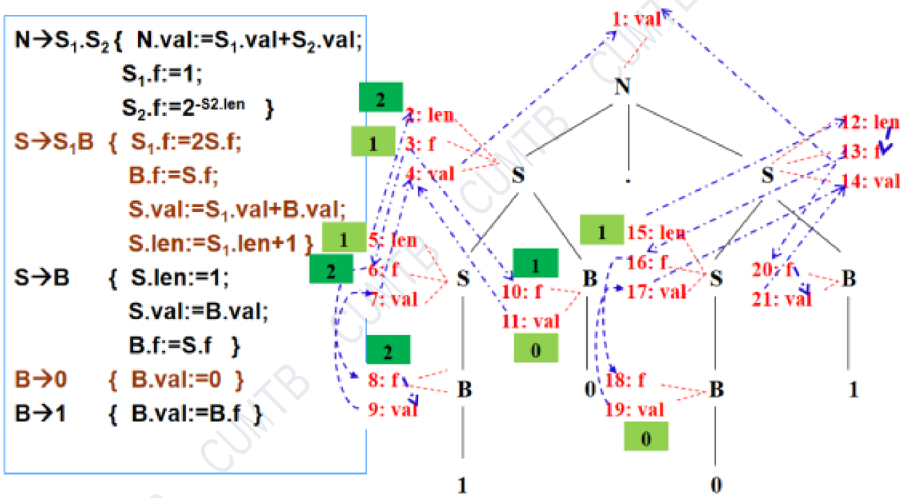
假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



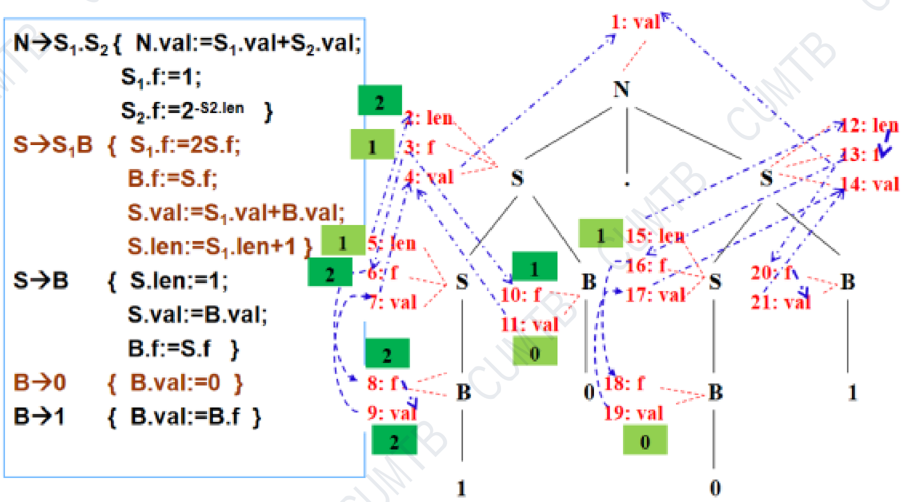
假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

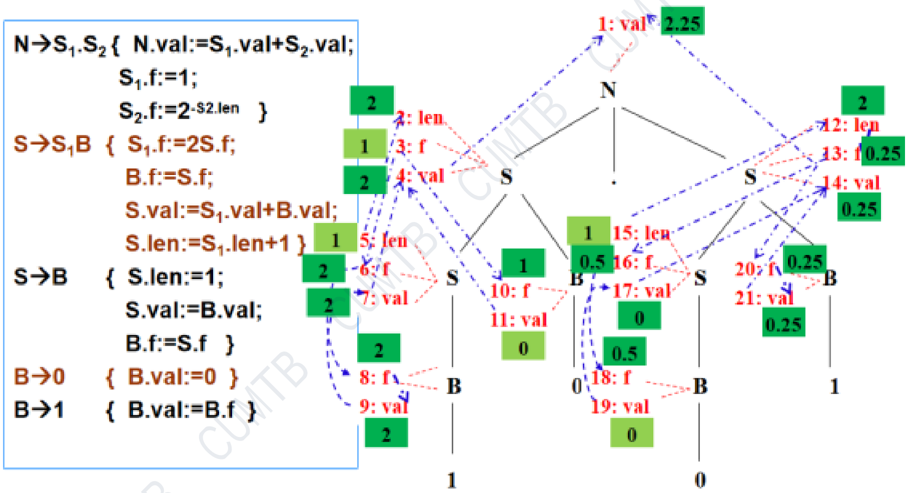
假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1

假设结点拓扑是 3,5,2,6,10,8,9,7,11,4,15,12,13,16,20,18,21,19,17,14,1



【讨论】“遍历分析树进行语义计算”存在的问题是什么？

遍历分析树进行属性计算有一定的通用性，但在语法分析之后再进行多次扫描。

解决方法：实际语法制导翻译采取单遍过程，在语法分析的同时完成语义动作。这需要对属性文法进行一定限制，两类受限的属性文法是：S-属性文法 和 L-属性文法。

(3) S-属性文法 和 L-属性文法

- S-属性文法

  只包含综合属性的文法。

S-属性文法 是 L-属性文法 的一个特例。

- L-属性文法

  如果对文法中的每一个产生式 $A \rightarrow X_1X_2 \cdots X_n$，其中每个语义动作所涉及的属性要么是综合属性，要么是某个$X_i(1 \leqslant i \leqslant n)$的继承属性，并且这个继承属性的计算只能依赖于：

  - $X_1, X_2, \cdots X_{i-1}$的属性
  - A的继承属性

(4) 基于S-属性文法的语义计算

- S-属性文法是只包含**综合属性**（自底向上传递的属性）的文法。
- S-属性文法翻译器可以**借助LR分析器**实现：

  LR分析器中增加一个栈（**语义值栈**）用来**存放综合属性的值**，进行归约的同时，栈中正在归约的产生式右部符号的综合属性值弹栈，并调用相应语义子程序进行相应计算（完成属性文法中的语义规则），产生的新值入语义值栈。

【举例】对 2+3*5 进行LR分析和语义计算（LR分析表参见P167）

	状态栈	符号栈	语义栈	待分析串	动作
1	0	#	-	2+3*5#	S5
2	05	#2	-2	+3*5#	r6
3	03	#F	-2	+3*5#	r4
4	02	#T	-2	+3*5#	r2
5	01	#E	-2	+3*5#	S6
6	016	#E+	-2-	3*5#	S5
7	0165	#E+3	-2-3	*5#	r6
8	0163	#E+F	-2-3	*5#	r4
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
12	0169710	#E+T*F	-2-3-5	#	r3
13	0169	#E+T	-2-15	#	r1
14	01	#E	-17	#	acc

(5) 基于L-属性文法的语义计算

L-属性文法可采用 **自顶向下** **深度优先** **从左至右** 遍历分析树，计算出所有属性值。

function visit(n: node)

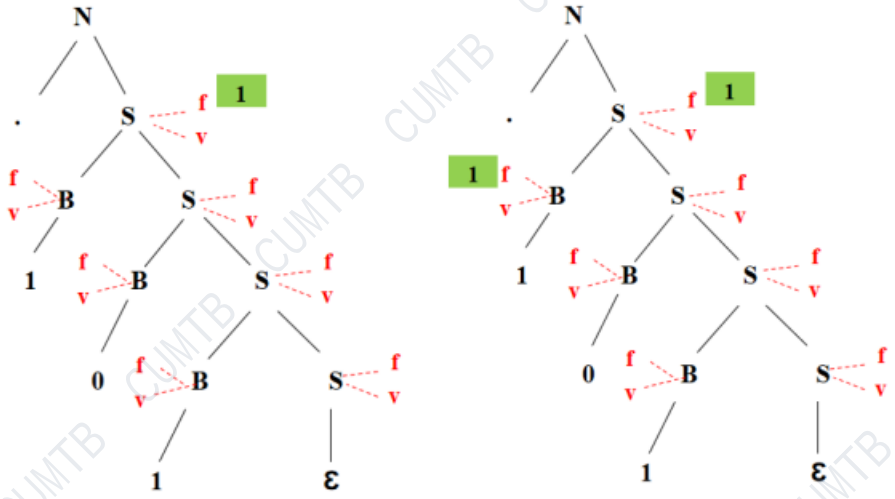
```
{   for n 的每一个孩子 m，从左到右
    {   计算 m 的继承属性值；
        visit(m);
    }
    计算 n 的综合属性值；
}
```

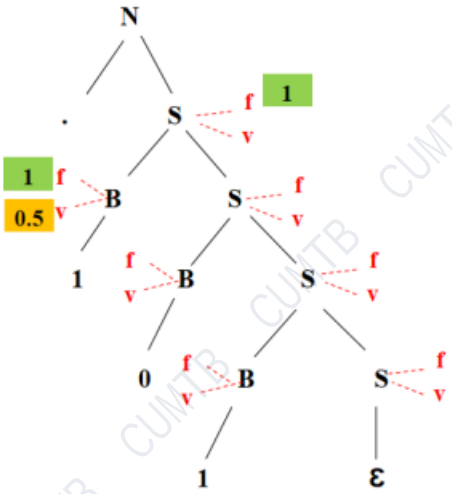
【举例】例 7.6　二进制无符号定点小数 0.101 转化为十进制小数。

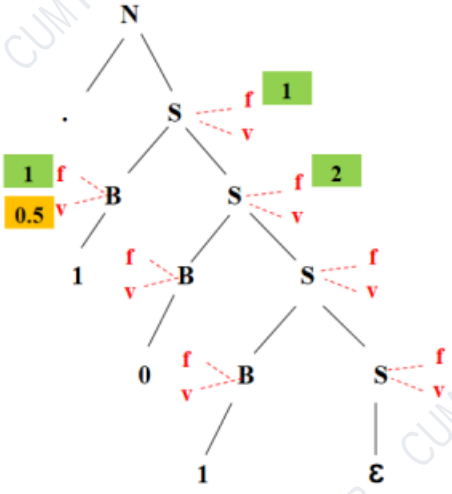
N→.S	{ S.f:=1; print(S.v) }
S→BS$_1$	{ S$_1$.f:=S.f + 1; B.f:=S.f; S.v:=B.v+ S$_1$.v }
S→ε	{ S.v:=0 }
B→0	{ B.v:=0 }
B→1	{ B.v:= 2$^{-B.f}$ }

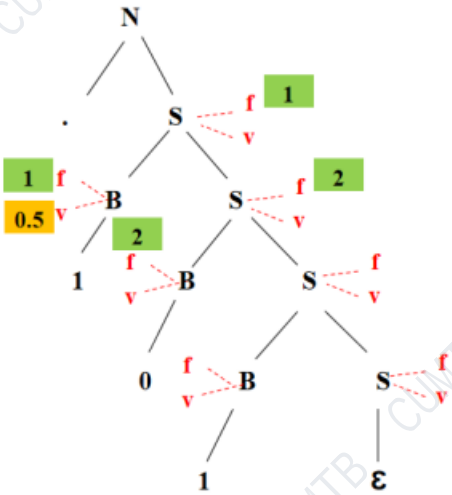
利用倒序入栈的表驱动（预测）分析法，实现语法分析和语义分析。

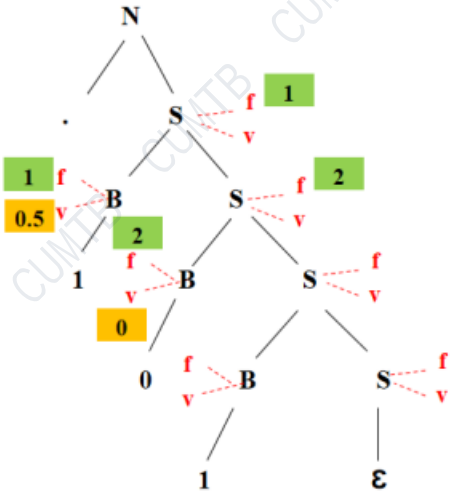
解：

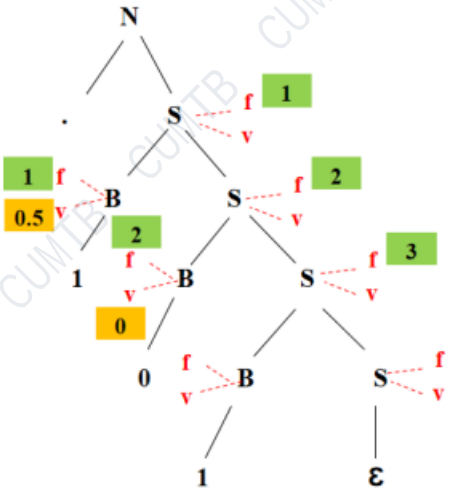


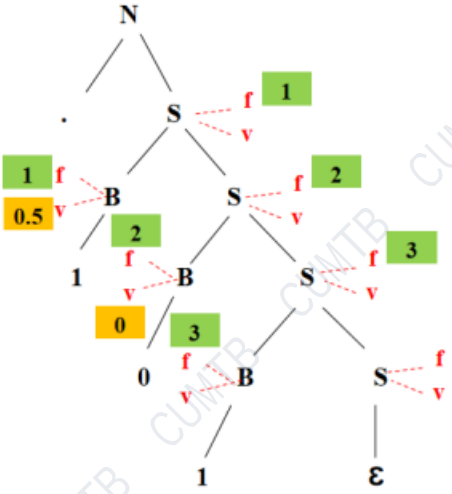

N
.
S
1
f
v
1
f
0.5
v
B
S
1
f
B
S
0
f
B
S
1
ε

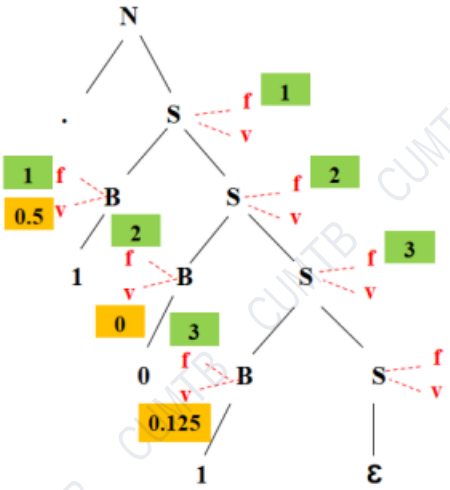

N
.
S
1
f
v
1
f
0.5
v
B
S
2
1
f
B
S
0
f
B
S
1
ε

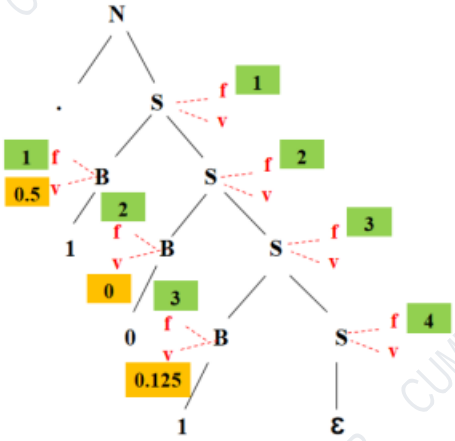

N
.
S
1
f
v
1
f
0.5
v
B
S
2
2
1
f
B
S
0
f
B
S
1
ε

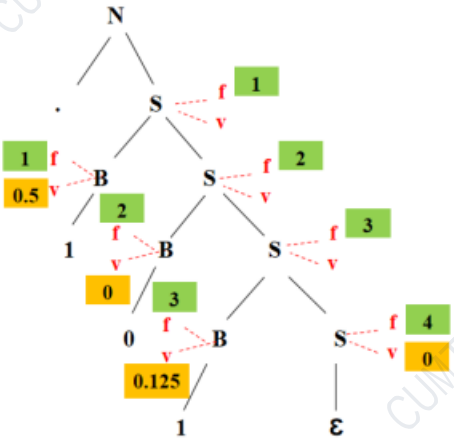

N
.
S
1
f
v
1
f
0.5
v
B
S
2
2
1
f
B
S
0
0
f
B
S
1
ε

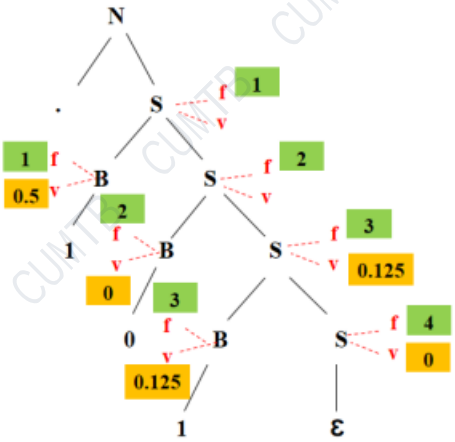

N
.
S
1
f
v
1
f
0.5
v
B
S
2
2
1
f
B
S
3
0
0
f
B
S
1
ε

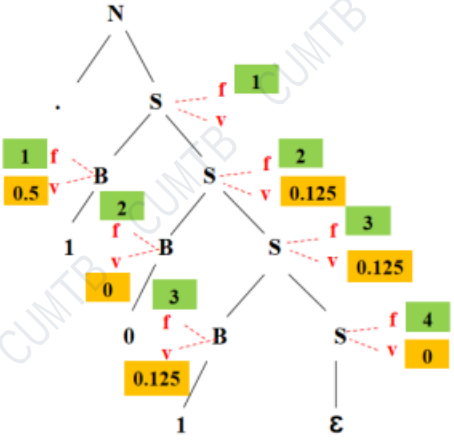

N
.
S
1
f
v
1
f
0.5
v
B
S
2
2
1
f
B
S
3
0
3
0
f
B
S
1
ε

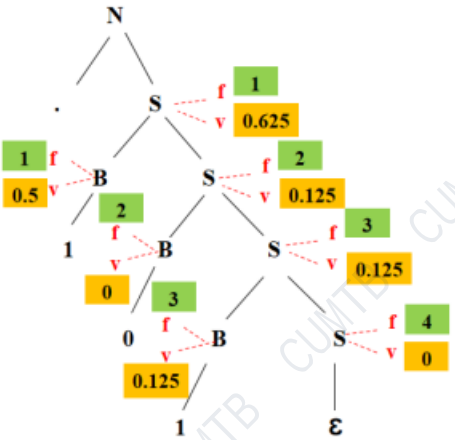

N
.
S
f
v
1
1
f
0.5
v
B
S
2
2
f
v
1
B
S
3
0
3
f
v
0
B
S
0.125
1
ε

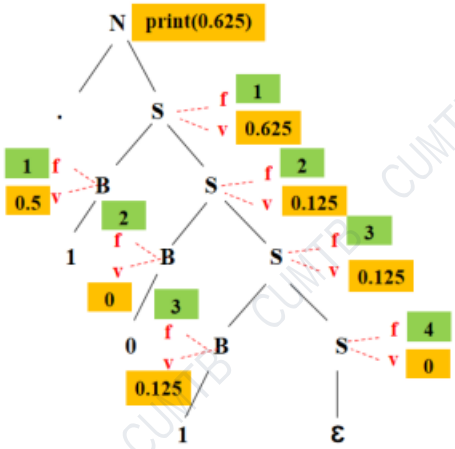

N
.
S
f
v
1
1
f
0.5
v
B
S
2
2
f
v
1
B
S
3
0
3
f
v
0
B
S
4
0.125
1
ε


N
.
S
f
v
1
1
f
0.5
v
B
S
2
2
f
v
1
B
S
3
0
3
f
v
0
B
S
4
0.125
0
1
ε


N
.
S
f
v
1
1
f
0.5
v
B
S
2
2
f
v
1
B
S
3
0.125
0
3
f
v
0
B
S
4
0.125
0
1
ε


N
.
S
f
v
1
1
f
0.5
v
B
S
2
0.125
2
f
v
1
B
S
3
0.125
0
3
f
v
0
B
S
4
0.125
0
1
ε


N
.
S
f
v
1
0.625
1
f
0.5
v
B
S
2
0.125
2
f
v
1
B
S
3
0.125
0
3
f
v
0
B
S
4
0.125
0
1
ε



4. 基于翻译模式的语义计算

(1) 翻译模式

- 什么是翻译模式？
  - ✓ 形式上类似于属性文法，但语义动作允许出现在产生式右部的任意位置，以此显式表达属性的计算次序。
  - ✓ 翻译模式是适合语法制导语义计算的另一种描述形式。

【举例】

属性文法：	翻译模式：
N→.S { S.f:=1; print(S.v) }	N→. {S.f:=1} S {print(S.v)}
$S \rightarrow BS_1$ { S_1.f:=S.f + 1; B.f:=S.f; S.v:=B.v+ S_1.v}	S→ {B.f:=S.f}B{S1.f:=S.f+1}S_1{S.v:=B.v+S_1.v}
S→ε { S.v:=0 }	S→ε { S.v:=0 }
B→0 { B.v:=0 }	B→0 { B.v:=0 }
B→1 { B.v:= $2^{-B.f}$ }	B→1 { B.v:= $2^{-B.f}$ }

- 为什么需要翻译模式？
  - ✓ 给出了使用语义规则进行计算的次序，可把某些实现细节表现出来。
  - ✓ 在语义分析过程中，为明确表达语义栈上的操作，可以将属性文法转化成翻译模式。
- 翻译模式的类型
  - ✓ **S-翻译模式**：仅涉及综合属性
  - ✓ **L-翻译模式**：既可以包含综合属性，又可以包含继承属性

【举例】

L-翻译模式：

N→. {S.f:=1} S {print(S.v)}

S→ {B.f:=S.f} B {S_1.f:=S.f+1} S_1 {S.v:=B.v+S_1.v}

S→ε { S.v:=0 }

B→0 { B.v:=0 }

B→1 { B.v:= $2^{B.f}$ }

(2) 基于S-翻译模式的语义计算

- S-翻译模式：仅涉及综合属性，通常与将语义动作置于相应的产生式右端末尾。

- 将属性文法改写为S-翻译模式的方法：

  假设语义栈由向量v表示，栈顶为top，栈中存放对应的属性值。

  x用v[i].x表示

  ① A→XY {A.a := f(X.x, Y.y); … }

  改为 A→XY {v[top-1].a:=f(v[top-1].x, v[top].y); …}

  ② A→XYZ {A.a := f(X.x, Y.y, Z.z); …}

  改为A→XYZ {v[top-2].a := f(v[top-2].x, v[top-1].y, v[top].z); …}

【举例】例7.8 给出下列属性文法的S-翻译模式

属性文法	S-翻译模式
S→E { print(E.val) }	{ print(v[top].val) }
E→E_1+T{ E.val:=E_1.val+T.val }	{ v[top-1].val:=v[top-1].val + v[top].val }
E→T { E.val:=T.val }	{ v[top].val:=v[top].val }
T→T_1*F{ T.val:=T_1.val*F.val }	{ v[top-1].val:=v[top-1].val * v[top].val }
T→F { T.val:=F.val }	{ v[top].val:=v[top].val }
F→(E) { F.val:=E.val }	{ v[top].val:=v[top].val }
F→d { F.val:=d.lexval }	{ v[top].val:=d.lexval }

(3) 基于L-翻译模式的自顶向下语义计算

- L-翻译模式：既可以包含**综合属性**，又可以包含**继承属性**，但要满足两个条件：

  ① 符号的**继承属性**必须位于**该符号之前**，语义动作仅访问它**左边符号的属性**，不访问位于它右边符号的属性；

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全部计算出来之后进行，通常将这样的语义动作置于**产生式的末尾**。

- L-翻译模式规定好产生式右部文法符号和语义动作的处理次序。

- 语义计算通常基于**自顶向下**分析过程（如预测分析、递归下降分析）

- 递归下降中每个非终结符对应一个分析子函数，分析程序从开始符号对应的分析子函数开始执行。

- 在自顶向下分析过程，根据所选产生式右部依次出现的符号设计其行为：
  ① 遇到终结符 a：首先将其综合属性 x 保存在专门为 a.x 声明的变量里；然后判断 a 是否与当前读入的单词符号匹配，若匹配则继续读取下一个单词符号；若不匹配，则报告语法错误。
  ② 遇到非终结符 B：利用 B 对应的子函数 ParseB 产生赋值语句 c:=ParseB(b1, b2, …, bk)，其中参量 b1, b2, …, bk 对应 B 的各继承属性，变量 c 对应 B 的综合属性。
  ③ 遇到一个语义动作集合：直接复制其中每一个语义动作所对应的代码，只需要注意将属性的访问替换成相应变量的访问。

- 经上述方法改造后的分析子函数称为**语义计算子函数**，改造后的递归下降分析程序称为**递归下降翻译程序**。

【举例】例 7.9 对于如下 L-翻译模式，二进制无符号定点小数转换为十进制小数，构造相应的递归下降翻译程序。

N→. {S.f:=1} S {print(S.v)}
S→ {B.f:=S.f} B {S_1.f:=S.f+1} S_1 {S.v:=B.v+S_1.v}
S→ε { S.v:=0 }
B→0 { B.v:=0 }
B→1 { B.v:= $2^{-B.f}$ }

递归下降语法分析程序	递归下降翻译程序

void ParseN(){
MatchToken('.');
ParseS();
}
	void ParseN(){
	MatchToken('.');
	Sf:=1;
	Sv:=ParseS();
	print(Sv);
	}

void ParseS(){
if(lookahead== '0' or lookahead== '1'){
ParseB();
	float ParseS(int Sf){
	if(lookahead== '0' or lookahead== '1'){
	Bf:=Sf;

```
                ParseS();
        } else if(lookahead== '#' ){
        } else {
                print( "syntax error" );
                exit(0);
        }
}
```

```
                Bv:=ParseB(Bf);
                S1f:=Sf+1;
                S1v:=ParseS(S1f);
                Sv:=S1v+Bv;
        }else if(lookahead== '#' )
                Sv:=0;
        else{
                print( "syntax error" );
                exit(0);
        }
        return Sv;
}
```

```
float ParseB( ){
        if(lookahead== '0' ){
                MatchToken( '0' );
        }else if(lookahead== '1' )
                MatchToken( '1' );
        else{
                print( "syntax error" );
                exit(0);
        }
}
```

```
float ParseB(int Bf){
        if(lookahead== '0' ){
                MatchToken( '0' );
                Bv:=0;
        }else if(lookahead== '1' )
                MatchToken( '1' );
                Bv:=2^(-Bf);
        else{
                print( "syntax error" );
                exit(0);
        }
        return Bv;
}
```

- 关于递归下降翻译程序的一点说明：

如果文法不是 LL(1)文法，则不能使用 L-翻译模式，需要将基础文法消除左递归和左公共因子，变成 LL(1)文法后，才可以构造 L-翻译模式，继而构造递归下降翻译程序。

(4) 基于 L-翻译模式的自底向上语义计算

- L-翻译模式中既有继承属性，又有综合属性。
  - ✓ 情况 1：L-翻译模式只包含综合属性
  - ✓ 情况 2：L-翻译模式中包含继承属性（略，参见 P179-182）

- 针对情况 1：L-翻译模式只包含综合属性（综合属性是自底向上传递信息）

处理方法：可以将 **L-翻译模式**处理成 **S-翻译模式**，然后利用 LR 分析，增加一个语义栈，进行语义分析。 （参见 7.2.2 基于 S-翻译模式的语义计算）

【举例】将 L-翻译模式处理成 S-翻译模式

L-翻译模式：

$E \rightarrow TR$

$R \rightarrow +T\ \{ print('+') \}\ R_1$

$R \rightarrow -T\ \{ print('-') \}\ R_1$

$R \rightarrow \varepsilon$

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

S-翻译模式：

$E \rightarrow TR$

$R \rightarrow +T\ M\ R_1$

$R \rightarrow -T\ N\ R_1$

$R \rightarrow \varepsilon$

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

$M \rightarrow \varepsilon\ \{ print('+') \}$

$N \rightarrow \varepsilon\ \{ print('-') \}$

【课堂练习】

1、补充 1、补充 2、补充 3、补充 4、补充 5 （给出属性文法）

2 、3 、4 （构造语法分析树）

5 （LR 分析和求值过程）

1. 设计属性文法，可以计算出每个二值布尔表达式的取值。例如 ¬true∨ ¬ false∧true, 输出true.

S' →S	{ print(S.val) }
S→S_1∨T	{ if(S_1.val=true or T.val=true) S.val:=true else S.val:=false }
S→T	{ S.val:=T.val }
T→T_1∧S	{ if(T_1.val=true and S.val=true) T.val:=true else T.val:=false }
T→F	{ T.val:=F.val }
F→ ¬F_1	{ if(F_1.val=true) F.val:=false else F.val:=true }
F→true	{ F.val:=true }
F→false	{ F.val:=false }

补充1. 构造下属文法的属性文法，它可以输出句子中括号的对数。

S' →S	{ print(S.num) }
S→ (L)	{ S.num := L.num + 1 }
S→a	{ S.num := 0 }
$L \rightarrow L_1 , S$	{ $L.num := L_1.num + S.num$ }
L→S	{ L.num := S.num }

补充2. 构造如下文法的属性文法，它可以输出句子中括号的对数。

S' →S	{ print(S.num) }
$S \rightarrow S_1 (S_2) S_3$	{ $S.num := S_1.num + S_2.num + S_3.num + 1$ }
S→ε	{ S.num := 0 }

补充3. 构造如下文法的属性文法，它可以输出句子中括号的最大嵌套层数。

S' →S	{ print(S.num) }
S→ (L)	{ S.num := L.num + 1 }
S→a	{ S.num := 0 }
$L \rightarrow L_1 , S$	{ if ($L_1.num > S.num$) $L.num := L_1.num$ else L.num := S.num }
L→S	{ L.num := S.num }

补充4. 构造如下文法的属性文法，它可以输出句子中括号的最大嵌套层数。

S' →S	{ print(S.num) }

S→S_1 (S_2) S_3	{ if (S_1.num > S_2.num+1 and S_1.num > S_3.num)
	S.num := S_1.num
	else if(S_2.num+1 > S_3.num)
	S.num := S_2.num+1 }
	else
	S.num := S_3.num
	}

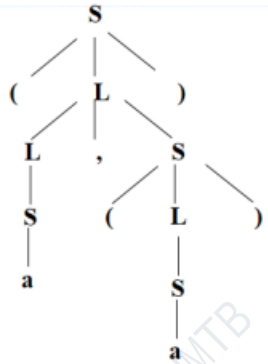
S→ε	{ S.num := 0 }

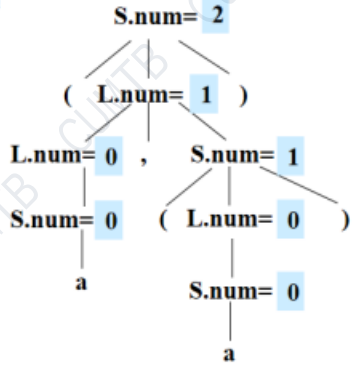
补充5. 构造如下文法的属性文法，它可以输出算术表达式的值。

S' →E	{ print(E.val) }
E→E_1+T	{ E.val := E_1.val + T.val }
E→T	{ E.val := T.val }
T→T_1*F	{ T.val := T_1.val * F.val }
T→F	{ T.val := F.val }
F→(E)	{ F.val := E.val }
F→d	{ F.val := d.lexval }

2. 给定 S-属性文法，输入串(a, (a))的语法分析树和对应的带标注语法树如下，试标出空缺属性值。

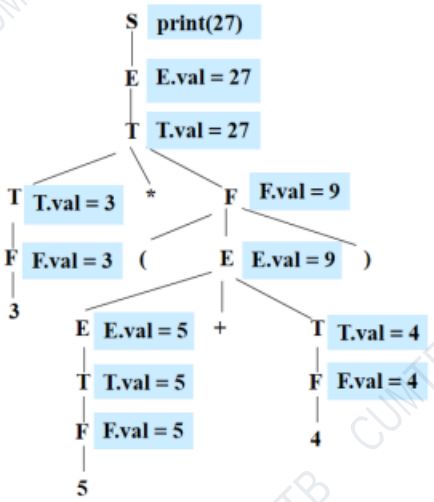
S→ (L)	{ S.num := L.num + 1 }
S→a	{ S.num := 0 }
L→L_1 , S	{ L.num := L_1.num + S.num }
L→S	{ L.num := S.num }





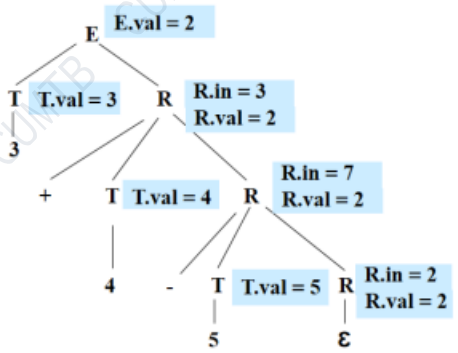
3. 给定 S-属性文法，试给出 3*(5+4) 的带标注语法分析树。

S' →E { print(E.val) }
E→E_1+T { E.val := E_1.val + T.val }
E→T { E.val := T.val }
T→T_1*F { T.val := T_1.val * F.val }
T→F { T.val := F.val }
F→(E) { F.val := E.val }
F→d { F.val := d.lexval }



4. 给定 L-属性文法，试给出 3+4-5 的带标注语法分析树。

E→TR { R.in := T.val;
E.val := R.val }
R→+TR_1 { R_1.in := R.in + T.val;
R.val := R_1.val }
R→-TR_1 { R_1.in := R.in - T.val;
R.val := R_1.val }
R→ε { R.val := R.in }
T→num { T.val := lexval(num) }



5. 题2给定的 S-属性文法，LR分析表如下表，请补全输入串(a, (a))的语法分析和求值过程。

① S→ (L) { S.num := L.num + 1 }
② S→a { S.num := 0 }
③ L→L_1 , S { L.num := L_1.num + S.num }
④ L→S { L.num := S.num }

状态	Action					Goto	
	a	,	(	)	#	S	L
0	S3		S2			1	
1					acc		
2	S3		S2			5	4
3		r2		r2	r2		
4		S7		S6			
5		r4		r4			
6		r1		r1	r1		
7	S3		S2				
8		r3		r3			

	状态栈	符号栈	语义栈	待分析串	动作
1	0	#	-	(a,(a))#	S3
2	02	#(	- -	a,(a))#	S3
3	023	#(a	- - -	,(a))#	r2
4	025	#(S	- - 0	,(a))#	r4
5	024	#(L	- - 0	,(a))#	S7
6	0247	#(L,	- - 0 -	(a))#	S2
7	02472	#(L,(	- - 0 - -	a))#	S3
8	024723	#(L,(a	- - 0 - - -	))#	r2
9	024725	#(L,(S	- - 0 - - 0	))#	r4
10	024724	#(L,(L	- - 0 - - 0	))#	S6
11	0247246	#(L,(L)	- - 0 - - 0 -	)#	r1
12	02478	#(L,S	- - 0 - 1	)#	r3
13	024	#(L	- - 1	)#	

答：

	状态栈	符号栈	语义栈	待分析串	动作
1	0	#	-	(a,(a))#	S2
2	02	#(	- -	a,(a))#	S3
3	023	#(a	- - -	,(a))#	r2
4	025	#(S	- - 0	,(a))#	r4
5	024	#(L	- - 0	,(a))#	S7
6	0247	#(L,	- - 0 -	(a))#	S2
7	02472	#(L,(	- - 0 - -	a))#	S3
8	024723	#(L,(a	- - 0 - - -	))#	r2
9	024725	#(L,(S	- - 0 - - 0	))#	r4
10	024724	#(L,(L	- - 0 - - 0	))#	S6
11	0247246	#(L,(L)	- - 0 - - 0 -	)#	r1
12	02478	#(L,S	- - 0 - 1	)#	r3
13	024	#(L	- - 1	)#	S6
14	0246	#(L)	- - 1 -	#	r1
15	01	#S	- 2	#	**acc**

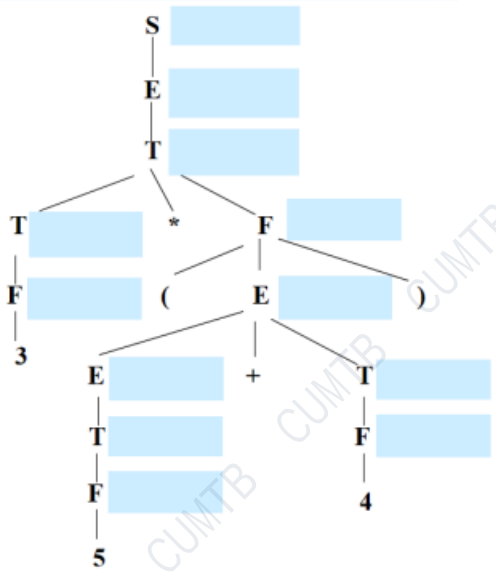
【课后作业】

3（构造语法分析树）、5（LR 分析和求值过程）

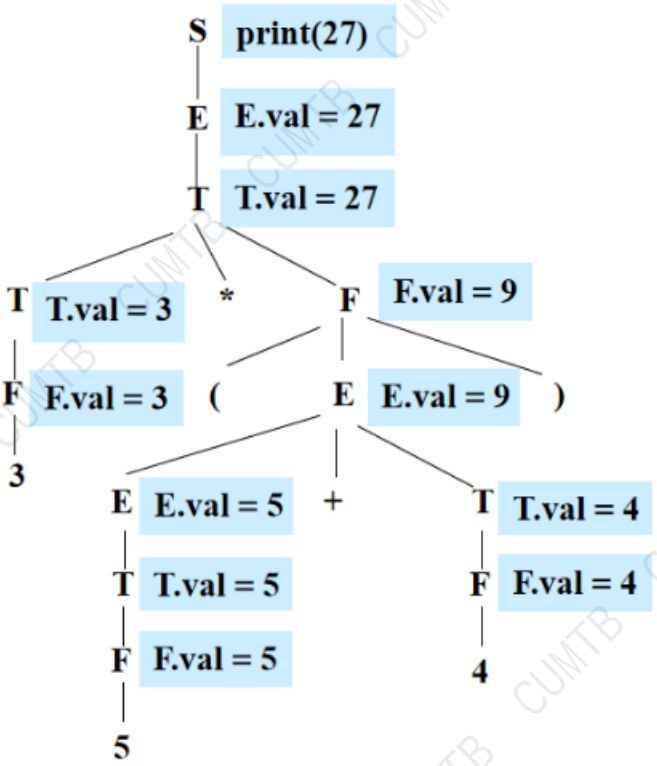
补充 1（给出属性文法）、补充 3（给出属性文法）、补充 6（给出属性文法）

3. 给定 S-属性文法，试给出 3*(5+4) 的带标注语法分析树。

S' →E	{ print(E.val) }
E→E_1+T	{ E.val := E_1.val + T.val }
E→T	{ E.val := T.val }
T→T_1*F	{ T.val := T_1.val * F.val }
T→F	{ T.val := F.val }
F→(E)	{ F.val := E.val }
F→d	{ F.val := d.lexval }



答：



5. 题2给定的 S-属性文法，LR分析表如下表，请补全输入串(a, (a))的语法分析和求值过程。

① S→ (L) { S.num := L.num + 1 }
② S→a { S.num := 0 }
③ L→L_1 , S { L.num := L_1.num + S.num }
④ L→S { L.num := S.num }

状态	Action					Goto	
	a	,	(	)	#	S	L
0	S3		S2			1	
1					acc		
2	S3		S2			5	4
3		r2		r2	r2		
4		S7		S6			
5		r4		r4			
6		r1		r1	r1		
7	S3		S2				
8		r3		r3			

	状态栈	符号栈	语义栈	待分析串	动作
1	0	#	-	(a,(a))#	S3
2	02	#(	- -	a,(a))#	S3
3	023	#(a	- - -	,(a))#	r2
4	025	#(S	- - 0	,(a))#	r4
5	024	#(L	- - 0	,(a))#	S7
6	0247	#(L,	- - 0 -	(a))#	S2
7	02472	#(L,(	- - 0 - -	a))#	S3
8	024723	#(L,(a	- - 0 - - -	))#	r2
9	024725	#(L,(S	- - 0 - - 0	))#	r4
10	024724	#(L,(L	- - 0 - - 0	))#	S6
11	0247246	#(L,(L)	- - 0 - - 0 -	)#	r1
12	02478	#(L,S	- - 0 - 1	)#	r3
13	024	#(L	- - 1	)#	

答：

	状态栈	符号栈	语义栈	待分析串	动作
1	0	#	-	(a,(a))#	S2
2	02	#(	- -	a,(a))#	S3
3	023	#(a	- - -	,(a))#	r2
4	025	#(S	- - 0	,(a))#	r4
5	024	#(L	- - 0	,(a))#	S7
6	0247	#(L,	- - 0 -	(a))#	S2
7	02472	#(L,(	- - 0 - -	a))#	S3
8	024723	#(L,(a	- - 0 - - -	))#	r2
9	024725	#(L,(S	- - 0 - - 0	))#	r4
10	024724	#(L,(L	- - 0 - - 0	))#	S6
11	0247246	#(L,(L)	- - 0 - - 0 -	)#	r1
12	02478	#(L,S	- - 0 - 1	)#	r3
13	024	#(L	- - 1	)#	S6
14	0246	#(L)	- - 1 -	#	r1
15	01	#S	- 2	#	**acc**

补充1. 构造下属文法的属性文法，它可以输出句子中括号的对数。

S’ →S
S→ (L)
S→a
$L \rightarrow L_1$, S

L→S

答：

S’ →S	{ print(S.num) }
S→ (L)	{ S.num := L.num + 1 }
S→a	{ S.num := 0 }
$L \rightarrow L_1$, S	{ L.num := L_1.num + S.num }
L→S	{ L.num := S.num }

补充3. 构造如下文法的属性文法，它可以输出句子中括号的最大嵌套层数。

S’ →S
S→ (L)
S→a
$L \rightarrow L_1$, S

L→S

答：

S’ →S	{ print(S.num) }
S→ (L)	{ S.num := L.num + 1 }
S→a	{ S.num := 0 }
$L \rightarrow L_1$, S	{ if (L_1.num > S.num) L.num := L_1.num else L.num := S.num }
L→S	{ L.num := S.num }

补充6. 将二进制无符号小数转化成十进制小数。

$N' \rightarrow N$

$N \rightarrow S_1.S_2$

$S \rightarrow S_1B$

$S \rightarrow B$

$B \rightarrow 0$

$B \rightarrow 1$

答：

$N' \rightarrow N$	{ print(N.val) }
$N \rightarrow S_1.S_2$	{ N.val := S_1.val + S_2.val * $2^{-S2.len}$; }
$S \rightarrow S_1B$	{ S.val := 2*S_1.val + B.val; S.len := S_1.len + 1 }
$S \rightarrow B$	{ S.val := B.val; S.len := 1 }
$B \rightarrow 0$	{ B.val := 0 }
$B \rightarrow 1$	{ B.val := 1 }

【本章小结】

1. 语义处理的任务：
   (1) 静态语义分析
   (2) 翻译

2. 语义处理的实现途径：
   首先，使用属性文法为工具，描述程序设计语言的语义规则。在语法分析时，每应用一个产生式（推导或归约），同时完成该产生式上所附的语义规则描述的动作，从而完成语义处理。

3. 语义计算模型：
   (1) 基于**属性文法**的语义计算模型
   (2) 基于**翻译模式**的语义计算模型

4. 属性文法 （综合属性、继承属性）

5. 遍历分析树进行语义计算
   (1) 构造分析树
   (2) 构造依赖图
   (3) 计算属性值

6. S-属性文法、基于 S-属性文法的自底向上语义计算

7. L-属性文法、基于 L-属性文法的自顶向下语义计算