语法制导的翻译

There is a level of correctness that is deeper than grammar

```
fie(a,b,c,d)
    int a, b, c, d;
{ … }

fee() {
    int f[3],g[1],
        h, i, j, k;
  char *p;
    fie(h,i,"ab",j, k);
    k = f * i + j;
    h = g[17];
    printf("<%s,%s>.\n",
        p,q);
    p = 10;
}
```

What is wrong with this program?

(let me count the ways …)

- **declared g[1], used g[17]**
- **wrong number of args to fie()**
- **"ab" is not an int**
- **wrong dimension on use of f**
- **undeclared variable q**
- **10 is not a character string**

All of these are "deeper than syntax"

To generate code, we need to understand the context !

语义分析

- ❖分析源程序的含义，并做语义处理
  - ➢数据结构的含义：主要指与标识符相关联的数据对象，如类型、值、存储地址等
  - ➢控制结构的含义：各个语法结构的含义，如if语句的执行逻辑

5.1语法制导定义 (*Syntax Directed Definition*, SDD)

❖基于上下文无关文法(CFG)，附加语义信息，没有副作用的SDD也称属性文法(*Attribute Grammar*):

➢属性(*Attributes*)，附加到文法符号

  - 值、类型等
  - 分为：综合属性、继承属性

➢语义规则 (*Semantic Rules*)，附加到语法规则

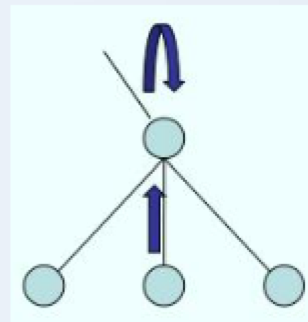
  - 属性的计算规则等，是计算属性值的一段代码

❖难题：如何自动计算属性值

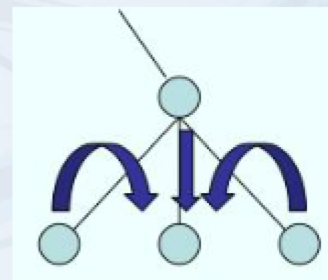
$E \rightarrow E_1 + T$ E.*code*= E_1.*code* || T.*code* || '+'

综合属性与继承属性

- 规则A→α的语义规则可表示为$b=f(c_1,\ldots,c_n)$
  - b是综合属性(*Synthesized Attribute*)：如果b是A的属性
    - ✓ S属性定义:只有综合属性的SDD
    - ✓ 终结符只有综合属性，属性值由词法分析器计算



  - b是继承属性(*Inherited Attribute*)：如果b是α中某个符号的属性，而$c_1\ldots c_n$是A或α中的文法符号的属性
    - ✓ 文法的开始符一般不得有继承属性



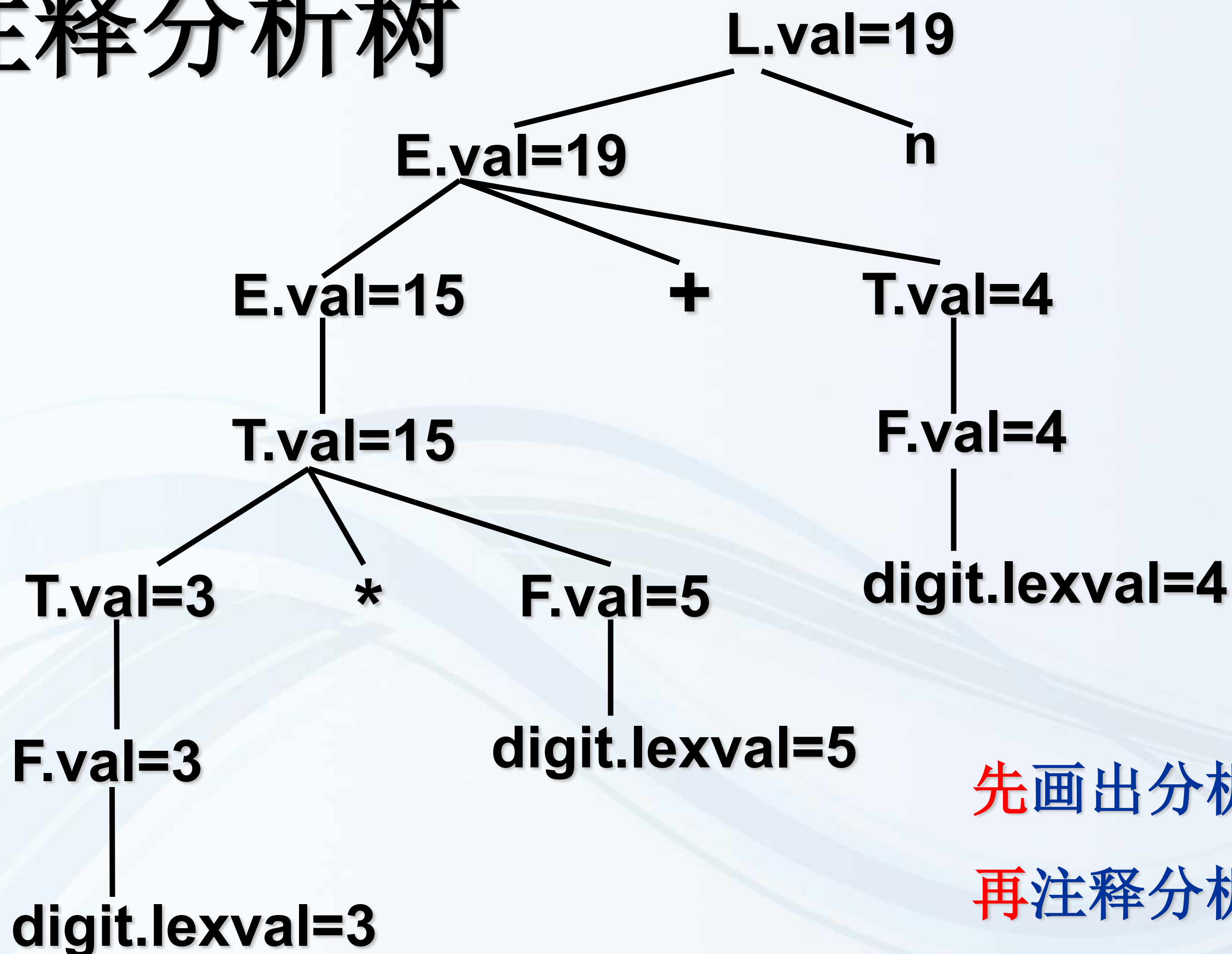
S属性定义(例)

产生式规则	语义规则
$L \rightarrow E\ n$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

例 3*5+4n

输入可能是源程序（字符流），也可能是记号流。对照文法来判定。
输入为字符流时，需要转化为记号流。

注释分析树



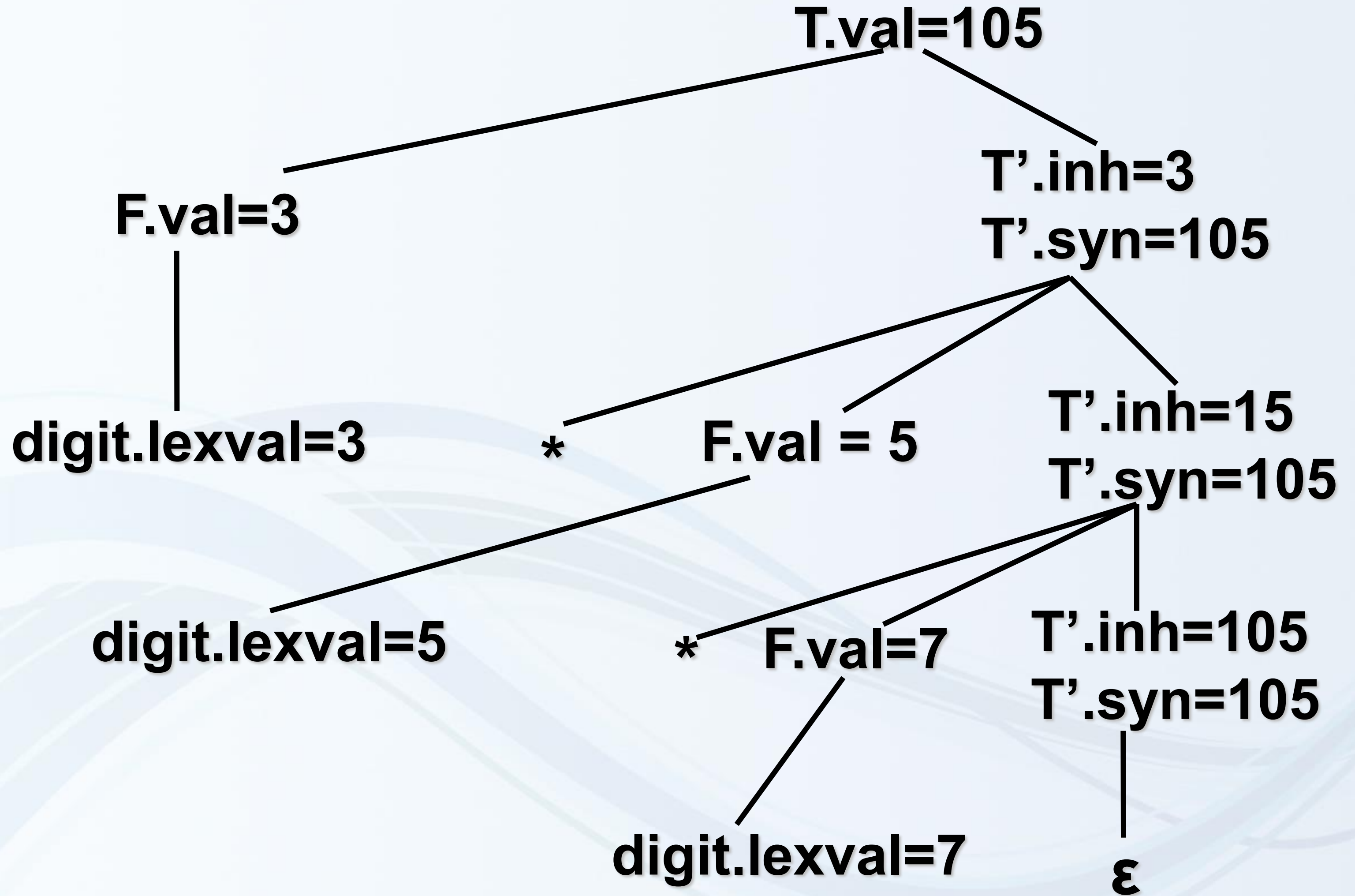
先画出分析树

再注释分析树

含有继承属性的SDD（例）

产生式	语义规则
T → FT’	T’.inh = F.val
	T.val = T’.syn
T’ →*FT$_1$’	T$_1$’.inh = T’.inh * F.val
	T’.syn = T$_1$’.syn
T’ → ε	T’.syn = T’.inh
F → digit	F.val = digit.lexval

例 3*5*7


T.val=105
F.val=3
T'.inh=3
T'.syn=105
digit.lexval=3
*
F.val = 5
T'.inh=15
T'.syn=105
digit.lexval=5
*
F.val=7
T'.inh=105
T'.syn=105
digit.lexval=7
ε

含有继承属性的SDD（例2）

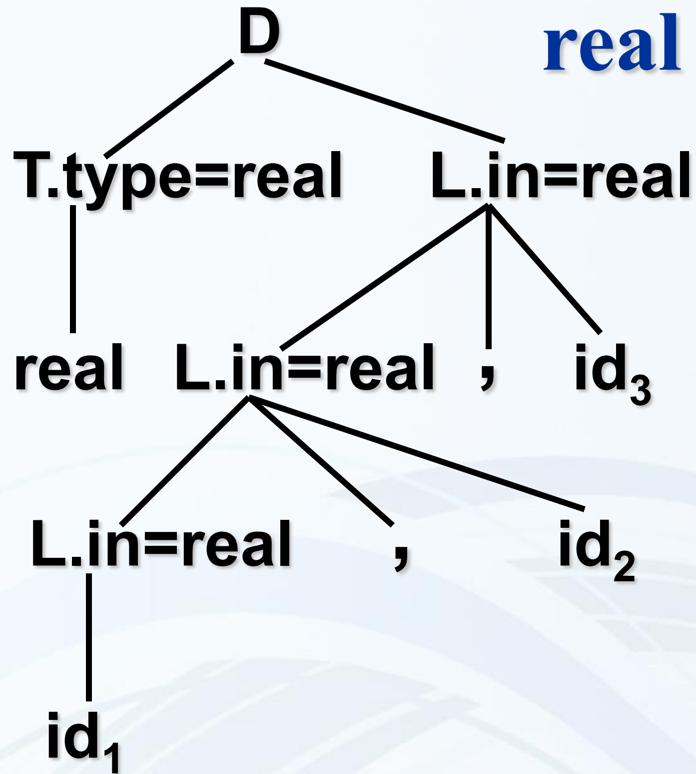
产生式	语义规则
$D \rightarrow T\ L$	$L.in = T.type$
$T \rightarrow$ **int**	$T.type$ = ***integer***
$T \rightarrow$ **real**	$T.type$ = ***real***
$L \rightarrow L_1$ **, id**	$L_1.in = L.in$
	addtype(**id**.*entry*, L.*in*)
$L \rightarrow$ **id**	*addtype*(**id**.*entry*, L.*in*)

虚综合属性

过程 ***addtype***(…)修改符号表：
在符号表相应项目插入类型信息，如***real, int***

real id_1, id_2, id_3

real id_1, id_2, id_3的注释分析树



构建语法制导定义（例）

为文法

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

(a)写一个语法制导定义，它输出括号的对数

(b)写一个语法制导定义，它输出括号嵌套的最大深度

解：构建增广文法，加上新的开始符号S'和产生式S'→S

(a)

产生式	语义规则
$S' \rightarrow S$	print(S.num)
$S \rightarrow (L)$	S.num=L.num+1
$S \rightarrow a$	S.num=0
$L \rightarrow L_1,S$	$L.num=L_1.num+S.num$
$L \rightarrow S$	L.num=S.num

一般不需要构建增广文法

构建语法制导定义（例）

为文法

S → (L) | a

L→ L,S | S

(b)写一个语法制导定义，它输出括号嵌套的最大深度

$L.num=max(L_1.num, S.num)$

也可写成

解(b) 产生式	语义规则
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

5.2 依赖图（*Dependency Graph*）

- ❖ 依赖图指出分析树中各结点属性值的计算顺序。
  - ➢ 应用场景：先有语义规则，后确定语义计算顺序
- ❖ 构建依赖图:

for 分析树中的每个结点n do　　//构建结点（标记属性）

　for 结点n对应文法符号的每个属性a do

　　在依赖图中为a构建一个结点

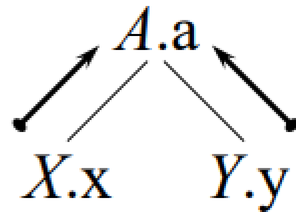
for 分析树中的每个结点n do　　//标记依赖

　for 结点n的产生式的每条语义规则$b=f(c_1,c_2,\ldots,c_k)$ do

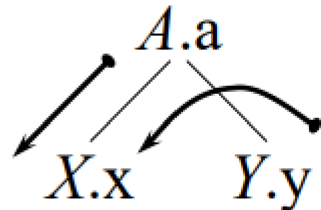
　　for(i=1 to k) do

　　　从结点c_i到结点b构造一条有向边

$A \rightarrow X\,Y$



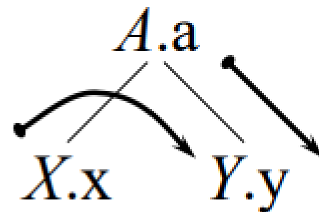
$A.\mathrm{a} = f(X.\mathrm{x},\ Y.\mathrm{y})$



$X.\mathrm{x} = f(A.\mathrm{a},\ Y.\mathrm{y})$

Direction of

value dependence



$Y.\mathrm{y} = f(A.\mathrm{a},\ X.\mathrm{x})$

构建依赖图与注释分析树

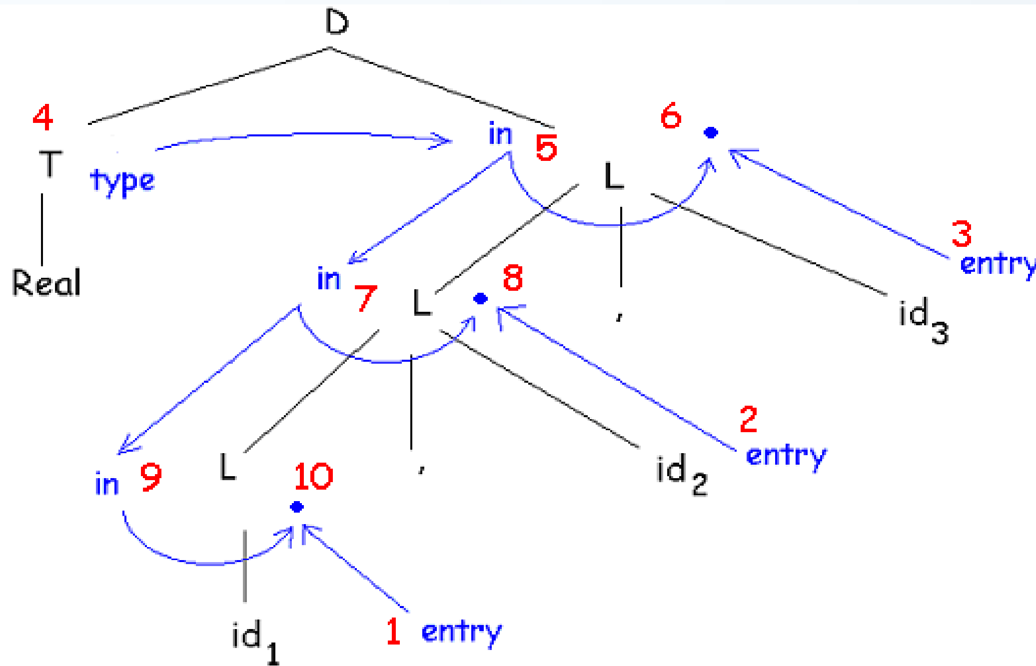
构建依赖图

1. 构建分析树（虚线）
2. 标记结点（属性）
3. 标记结点间依赖关系（实线）

注释分析树

1. 构建分析树（实线）
2. 标记属性
3. 计算属性值

基于依赖图的注释



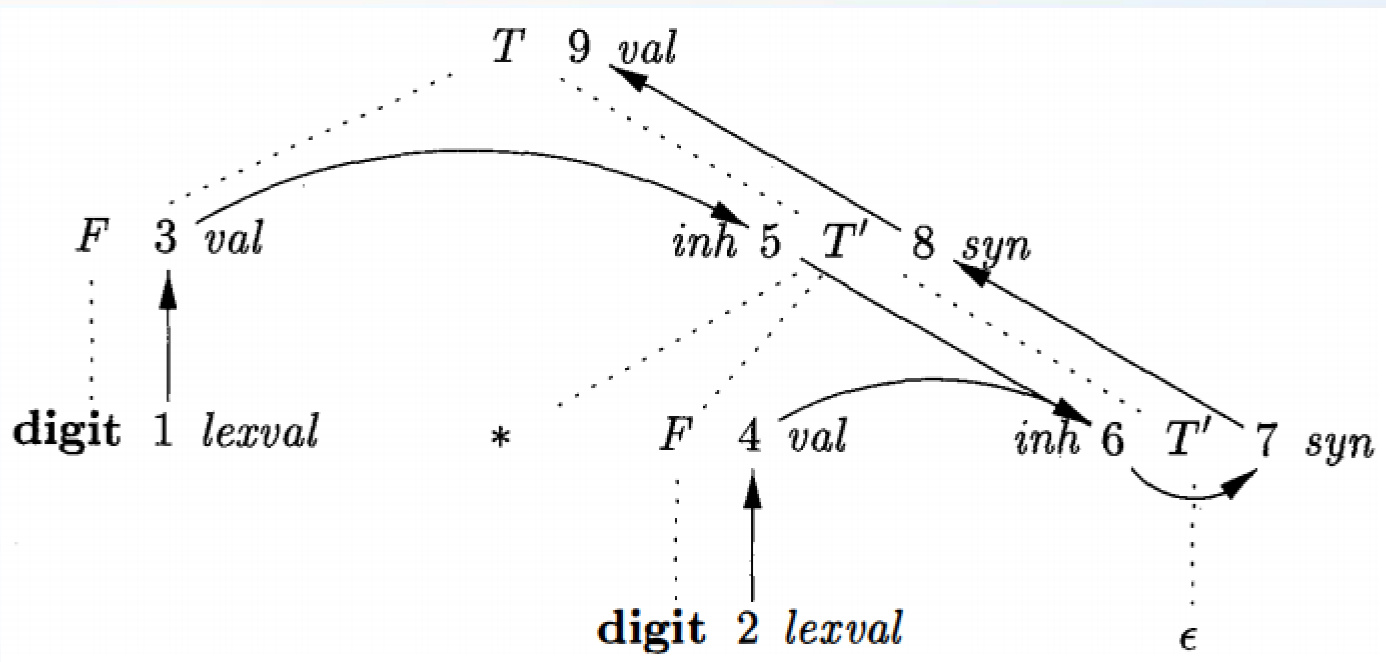
Production	Semantic Rules
D → T L	L.in := T.type
T → **Int**	T.type := **Integer**
T → **Real**	T.type := **Real**
L → L_1, **id**	L_1.in = L.in • := *addtype*(**id**.entry, L.in)
L → **id**	• := *addtype*(**id**.entry, L.in)

dummy attributes (nodes 6,8,10)

```
a4 := real;
a5 := a4;
addtype(id3.entry, a5);
a7 := a5;
addtype(id2.entry, a7);
a9 := a7;
addtype(id1.entry, a9);
```

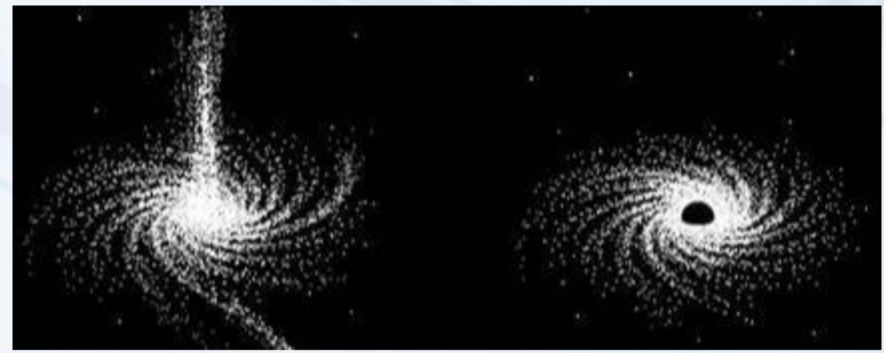
拓扑排序：对于节点$m_1,m_2,\ldots,m_k$，若$m_i \to m_j$是从m_i到m_j的边，那么在此排序中m_i先于m_j

依赖图中的属性位置（建议）：左继承，右综合



产生式	语义规则
T → FT’	T’.inh = F.val T.val = T’.syn
T’ →*FT₁’	T_1’.inh = T’.inh * F.val T’.syn = T_1’.syn
T’ → ε	T’.syn = T’.inh
F → digit	F.val = digit.lexval

注意白洞、黑洞

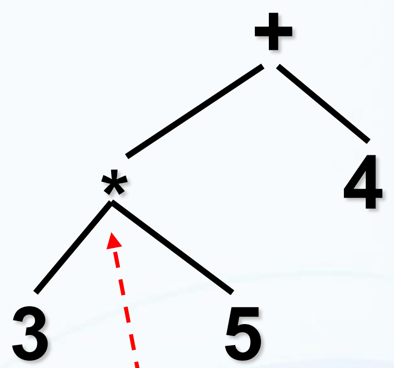


计算属性值过程（依赖图方法）

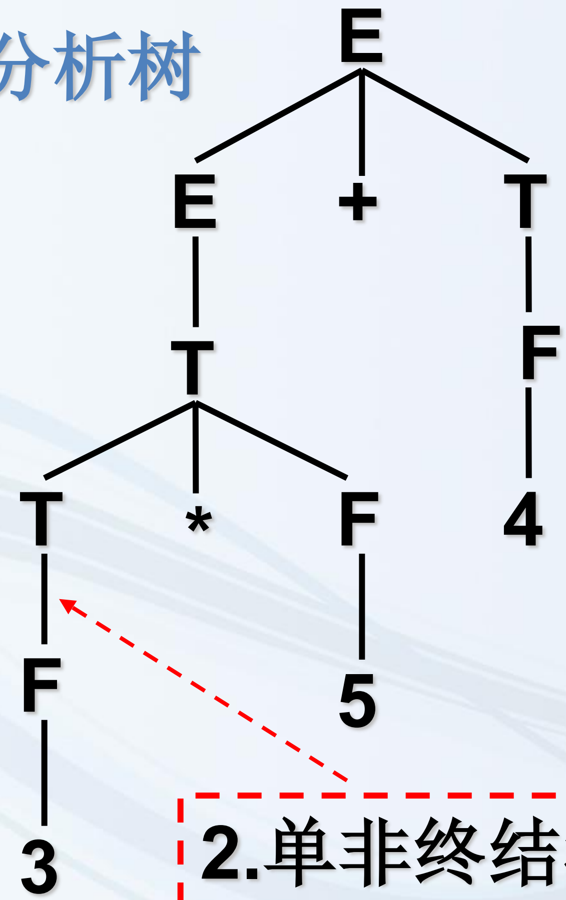
- 创建分析树（自顶向下或自底向上）
- 构建依赖图（基于分析树和语义规则）
- 对依赖图，进行拓扑排序
- 计算属性值

5.3 语法制导翻译的应用（构建语法树）

语法树



分析树



1.运算符作为内部结点

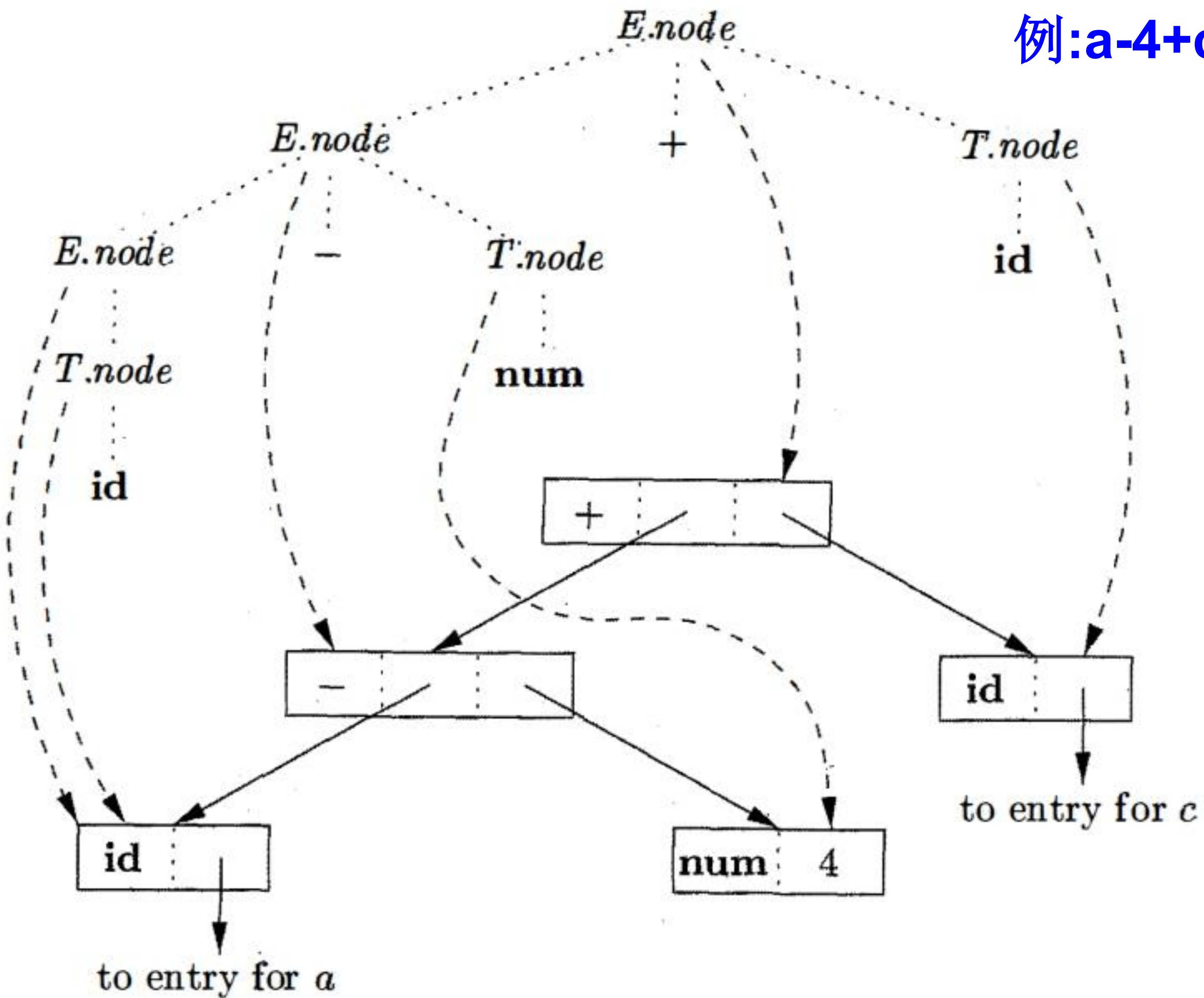
2.单非终结符产生式链可能消失

语法树

- 两个建立结点的函数（返回值为指向新建立结点的指针）：
  - *Node*(op,c_1,c_2,…,c_k) 建立内部结点，op为运算符，其余k个字段的值为指向其它结点的指针.
  - *Leaf(op,val)* 建立叶结点，op为记号，*val*为指向符号表相关项目的指针。
- 应用综合属性 *node* (语法树的结点)

产生式	语义规则
E → E_1 + T	E.*node* = new *node*("+",E_1.*node* ,T.*node*)
E → E_1 - T	E.*node* = new *node*("-",E_1. *node* ,T. *node*)
E → T	E.*node* = T.*node*
T → (E)	T.*node* = E.*node*
T → id	T.*node* = new *leaf*(id, id.*lexval*)
T → num	T.*node* = new *leaf*(num, num.*val*)

例:a-4+c


例:a-4+c
E.node
E.node
+
T.node
E.node
−
T.node
id
T.node
num
id
+
−
id
id
num
4
to entry for c
to entry for a

移入/归约时的S-属性定义计算

- 扩展栈的项目以包括属性
- 归约时，修改栈

符号	值
X	X.*x*
Y	Y.*y*
Z	Z.*z*

栈顶

A → XYZ

符号	值
A	A.*a*

新栈顶

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

栈上的语义代码

stack[top-2].val=*f*(stack[*top*-2].val, stack[*top*-1].val, stack[*top*].val)
top=top-2

产生式	语义规则
$L \rightarrow E\ n$	*print*(E.*val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	F.*val* = digit .*lexval*

Example:
3*5+4n

产生式	代码段
$L \rightarrow E\ n$	*print*(stack[*top-1*].val); top=top-1;
$E \rightarrow E_1 + T$	stack[*top-2*].val = stack[*top*-2].val+stack[*top*].val; top=top-2;
$E \rightarrow T$	
$T \rightarrow T_1 * F$	stack[*top-2*].val = stack[*top*-2].val*stack[*top*].val; top=top-2;
$T \rightarrow F$	
$F \rightarrow (E)$	stack[*top-2*].val =stack[*top*-1].val; top=top-2;
$F \rightarrow digit$	