Chapter 6 属性文法和语法制导翻译

杨策 2022-04-24

Outlines

- 属性文法 (Attribute grammar)
- 基于属性文法的处理方法
- S-属性文法的自下而上计算
- L-属性文法和自顶向下翻译
- 自下而上计算继承属性

Overview

- 编译中的语义处理包括两个功能：
  - （1）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；
  - （2）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。
- 以上工作普遍基于属性文法和语法制导翻译方法

Overview

- 目前还不存在一种广泛接受的方式来描述为典型程序语言产生中间代码所需的邻语言的动作。原因是代码生成依赖于对语义的解释，而语义的刻划的形式化系统尚未诞生。

- 解决办法： 为每一个产生式配一个翻译子程序（语义子程序、动作），在语法分析的同时执行它。这样，配上语义动作之后，既指定了串的意义，同时又按这种意义规定了生成某种中间代码应作的基本动作。

Overview

- 属性文法和语法制导翻译方法
  - 一种语义描述和语义处理方法
  - 目前在实际应用中比较流行
- 属性文法
  - 1968年Knuth提出
  - 在CFG中
    - 为每个文法符号配备若干相关的“值”
    - 为每个产生式配备一组属性的计算规则
- 语法制导翻译方法 (Syntax-directed translation)
  - 由源程序的语法结构驱动的语义处理方法

属性

语义规则

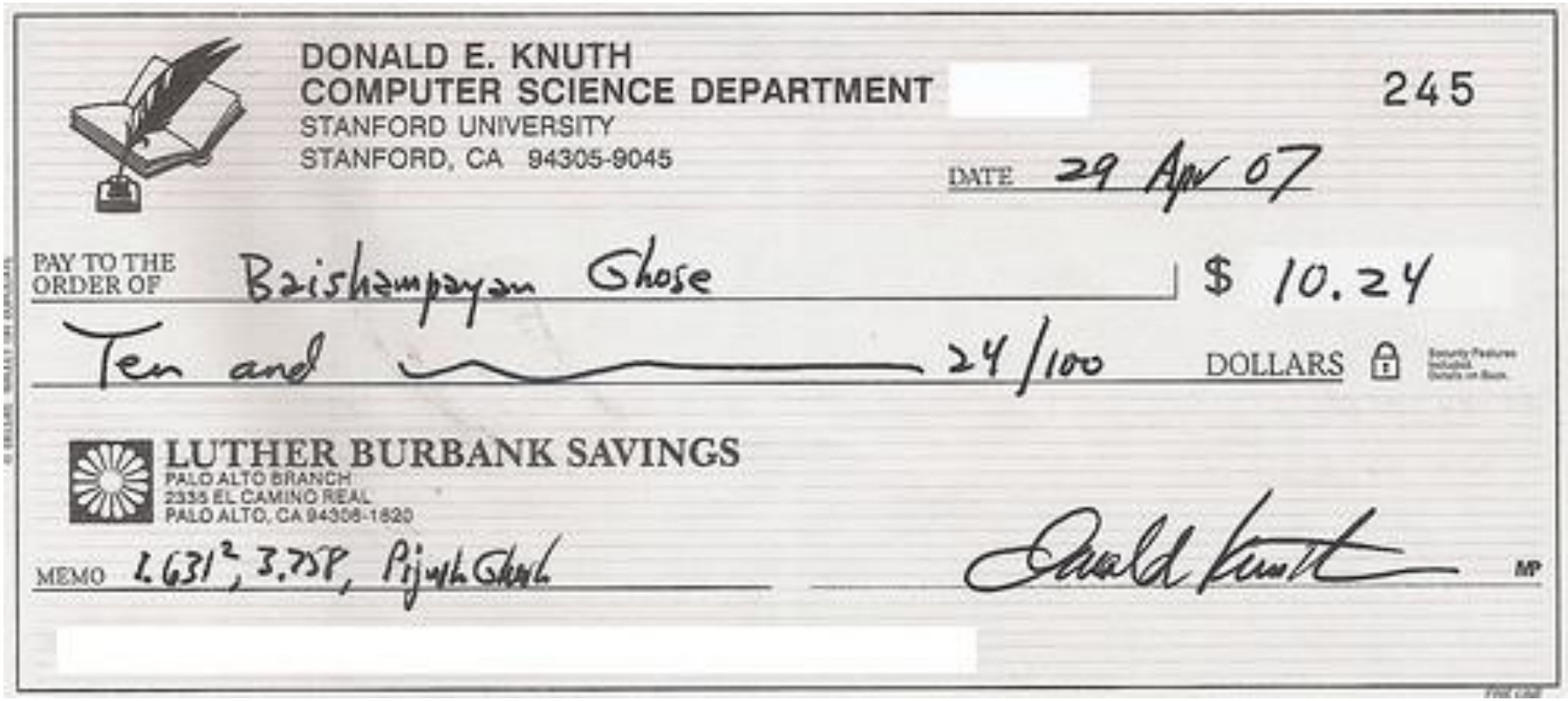
属性加工的过程就是语义处理的过程

输入串 ⟶ 语法树 ⟶ 依赖图 ⟶ 语义规则计算次序

Do you still remember?

"Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other"

- Donald Knuth



DONALD E. KNUTH
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305-9045

245

DATE 29 Apr 07

PAY TO THE ORDER OF Baishampayan Ghose

$ 10.24

Ten and 24/100 DOLLARS

LUTHER BURBANK SAVINGS
PALO ALTO BRANCH
2335 EL CAMINO REAL
PALO ALTO, CA 94306-1620

MEMO $1.631^2, 3.758, \text{Pijush Ghosh}$

Donald Knuth

MP

Attribute grammar

- 属性
  - 代表与文法符号的相关信息
    - 类型、值、代码序列、符号表内容等
    - 可以计算和传递
  - 属性分类
    - 综合属性：用于“自下而上”传递信息
    - 继承属性：用于“自上而下”传递信息
- 文法符号
  - 终结符
    - 只有综合属性，由词法分析器提供
  - 非终结符
    - 既可有综合属性也可有继承属性
    - 文法开始符号的所有继承属性作为属性计算前的初始值

Attribute grammar

- 在一个属性文法中，对应于每个产生式$A \rightarrow \alpha$都有一套与之相关联的语义规则，每条规则的形式为：

$$b:=f(c_1, c_2, \ldots, c_k)$$

这里，f是一个函数，而且或者

- b是A的一个综合属性，并且$c_1,c_2,\ldots,c_k$是产生式右边文法符号的属性
- b是产生式右边某个文法符号的一个继承属性，并且$c_1,c_2,\ldots,c_k$是A或产生式右边任何文法符号的属性

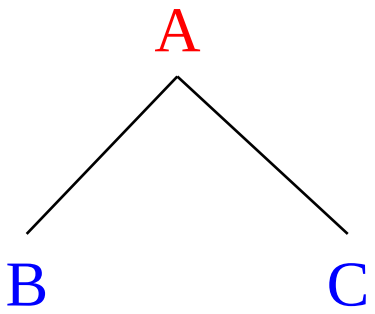
在两种情况下，我们都说

属性b依赖于属性$c_1,c_2,\ldots,c_k$

Attribute grammar

- 语法规则
  - $A \rightarrow BC$

综合属性
A.a = B.b + C.c
双亲结点属性值由孩子结点计算得到



继承属性
B.d = A.e - C.f
孩子结点属性值由双亲结点和其他孩子结点计算得到

Attribute grammar

- 属性计算的注意事项
  - 对于出现在产生式右边的继承属性和出现在产生式左边的综合属性都必须提供一个计算规则
  - 属性计算规则中只能使用相应产生式中的文法符号的属性
    - 有助于在产生式范围内“封装”属性的依赖性
  - 出现在产生式左边的继承属性和出现在产生式右边的综合属性不由所给的产生式的属性计算规则进行计算
    - 它们由其它产生式的属性规则计算或者由属性计算器的参数提供

Attribute grammar

- 非终结符A，B和C
  - A有一个继承属性a和一个综合属性b
  - B有综合属性c
  - C有继承属性d
- 产生式A→BC可能有规则

  C.d:=B.c+1

  A.b:=A.a+B.c

而属性A.a和B.c在其它地方计算

Example

属性文法的例子，例1：简单算术表达式求值的语义描述。

- 非终结符E、T及F都有一个综合属性val，
- 符号digit有一个综合属性lexval，它的值由词法分析器提供。
- 与产生式L→E对应的语义规则仅仅是打印由E产生的算术表达式的值的一个过程，我们可认为这条规则定义了L的一个虚属性。
- 某些非终结符加下标是为了区分一个产生式中同一非终结符多次出现

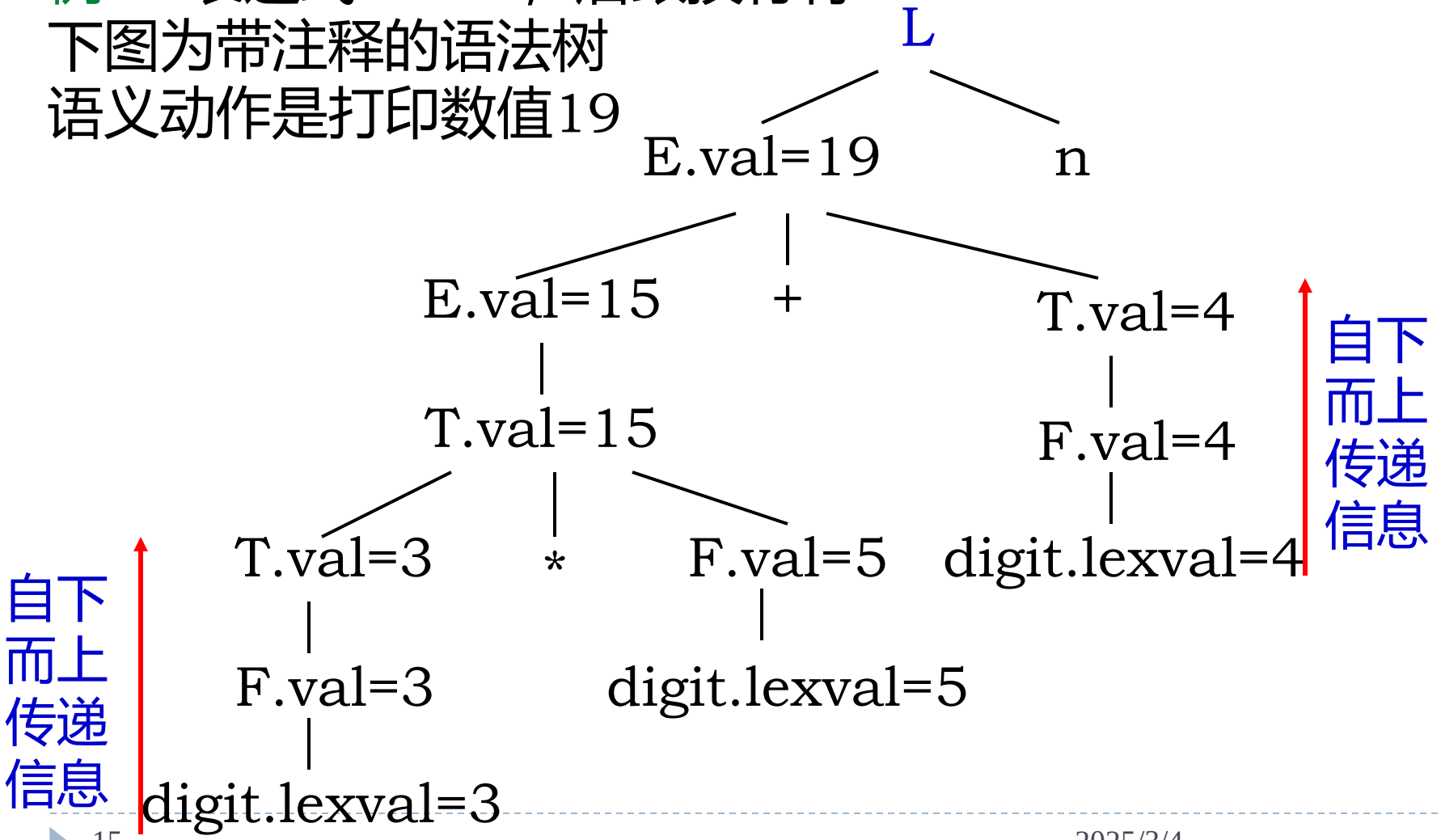
产生式	语 义 规 则
$L \longrightarrow E$	Print(E.val)
$E \longrightarrow E_1+T$	$E.val:=E_1.val+T.val$
$E \longrightarrow T$	E.val:=T.val
$T \longrightarrow T_1 * F$	$T.val:=T_1.val \times F.val$
$T \longrightarrow F$	T.val:=F.val
$F \longrightarrow (E)$	F.val:=E.val
$F \longrightarrow digit$	F.val:=digit.lexval

Synthesized attribute

- 综合属性
  - 语法树中结点的综合属性值
    - 由其子结点的属性值确定
  - 使用自底向上的方法在每一个结点处
    - 运用语义规则计算综合属性的值
- S - 属性文法
  - 仅仅使用综合属性的属性文法
  - 结点属性值的计算
    - 正好和自底向上分析建立分析树结点同步进行

Synthesized attribute

- 例2：表达式3*5+4，后跟换行符n
下图为带注释的语法树
语义动作是打印数值19



Inherited attribute

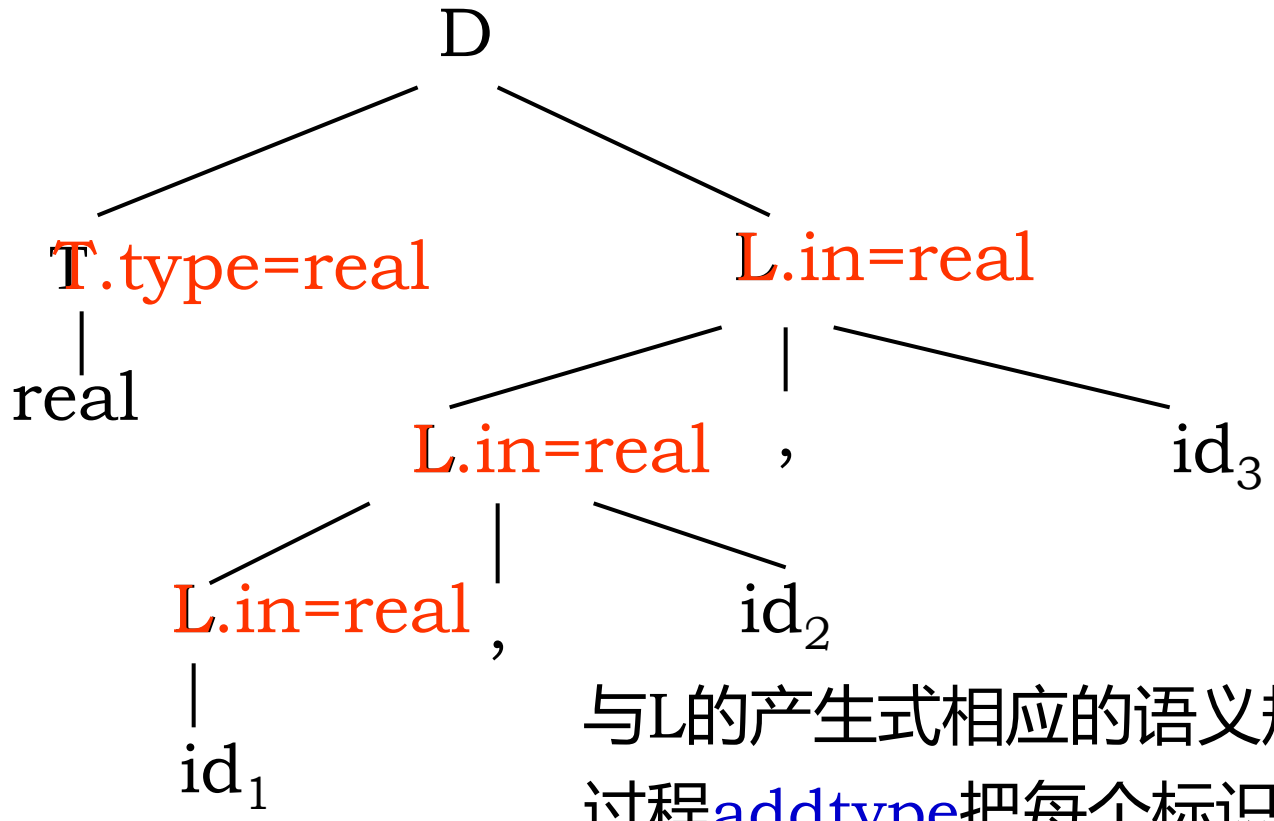
- 继承属性
  - 在语法树中，一个结点的继承属性由此结点的父结点和/或兄弟结点的某些属性确定
  - 用继承属性来表示程序设计语言结构中的上下文依赖关系很方便

Inherited attribute – 例3

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.type, L.in)
L→id	addtype(id.type, L.in)

Inherited attribute

- **句子real id_1，id_2，id_3的带注释的语法树**



与L的产生式相应的语义规则调用过程addtype把每个标识符的类型填入符号表的相应项中

Syntax-directed translation

- 语法制导翻译
  - 即基于属性文法的处理过程
    - 对单词符号串进行语法分析，构造语法分析树
    - 根据需要遍历语法树，并在语法树的各结点处按语义规则进行计算
- 语法制导翻译方法
  - 由源程序的语法结构所驱动的处理办法
- 语义规则的计算
  - 可能产生代码、在符号表中存放信息、给出错误信息或执行其他动作
  - 根据语义规则进行计算的结果
    - 就是对输入符号串的翻译

输入串 ⟶ 语法树 ⟶ 依赖图 ⟶ 语义规则计算次序

Dependency graph

- 在一棵语法树中的结点的继承属性和综合属性之间的相互依赖关系可以由称作依赖图的一个有向图来描述
- 为每一个包含过程调用的语义规则引入一个虚综合属性b，这样把每一个语义规则都写成

$$b := f(c_1, c_2, \ldots, c_k)$$ 的形式

- 依赖图
  - 点：为每一个属性设置一个结点
  - 边：如果属性b依赖于属性c，则从属性c的结点有一条有向边连到属性b的结点
  - 作用：描述结点属性间的依赖关系

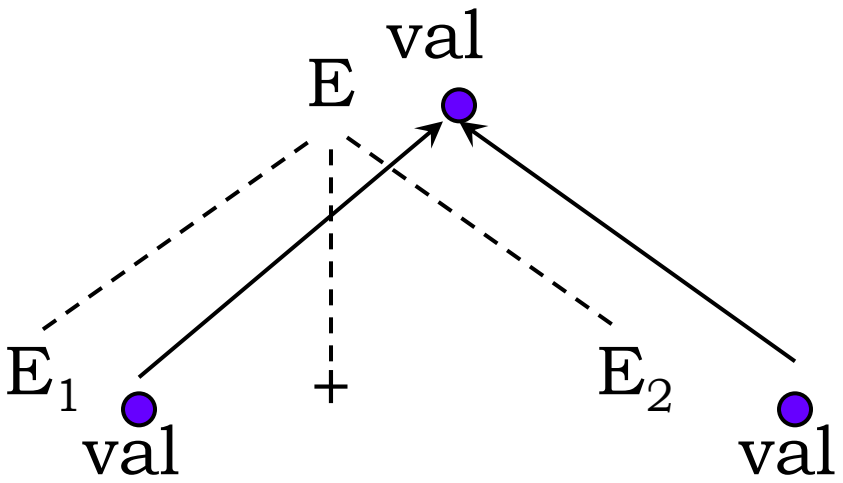
Dependency graph

- **Algorithm**

- for 语法树中每一结点n do
  for 结点n的文法符号的每一个属性a do
    为a在依赖图中建立一个结点;
- for 语法树中每一个结点n do
  for 结点n所用产生式对应的每一个语义规则
    b:=f(c_1,c_2,…,c_k) do
    for i:=1 to k do
      从c_i结点到b结点构造一条有向边;

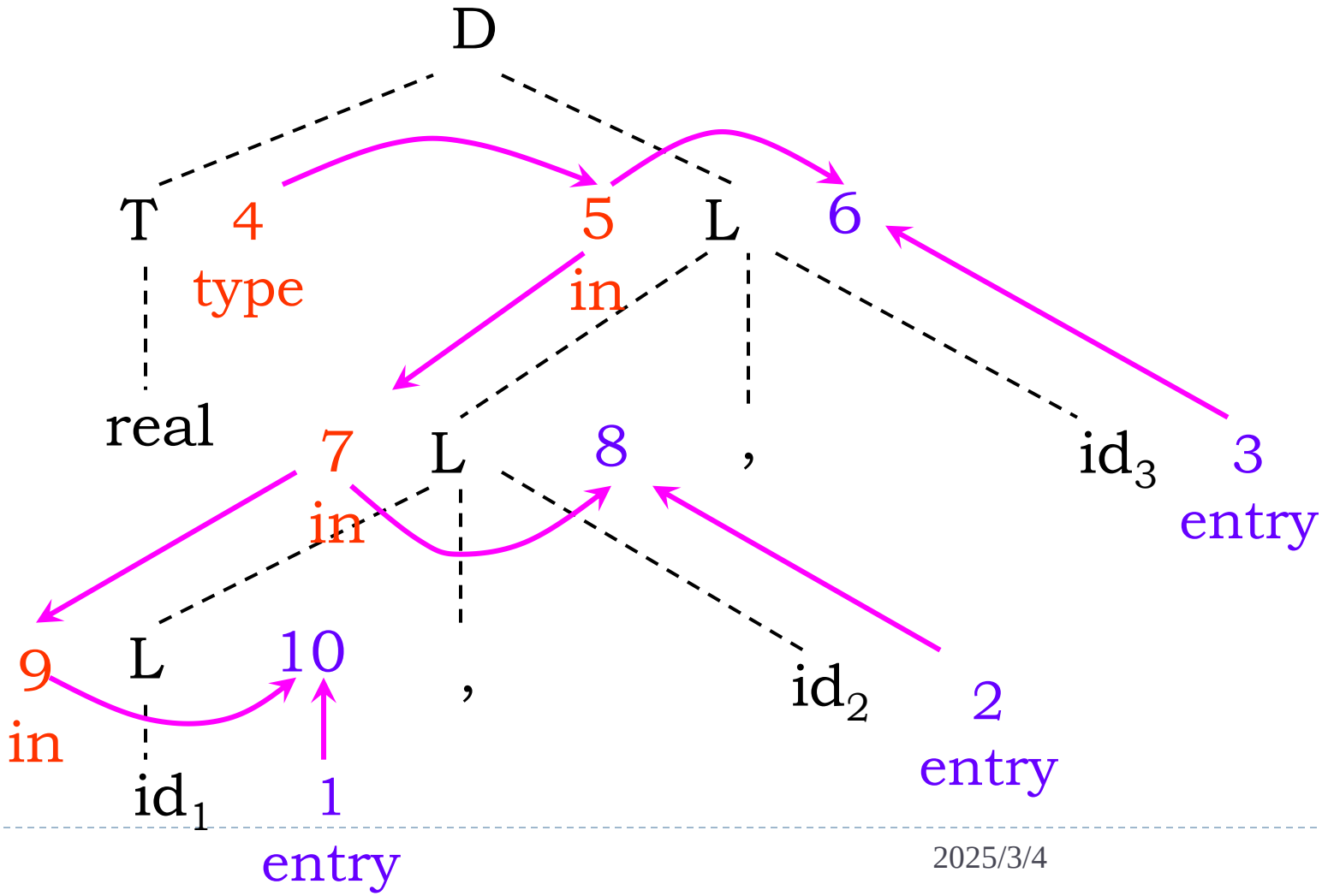
例4

- **$E \rightarrow E_1 + E_2$** **$E.val := E_1.val + E_2.val$**



例5

- **句子real id_1，id_2，id_3的带注释的语法树的依赖图**

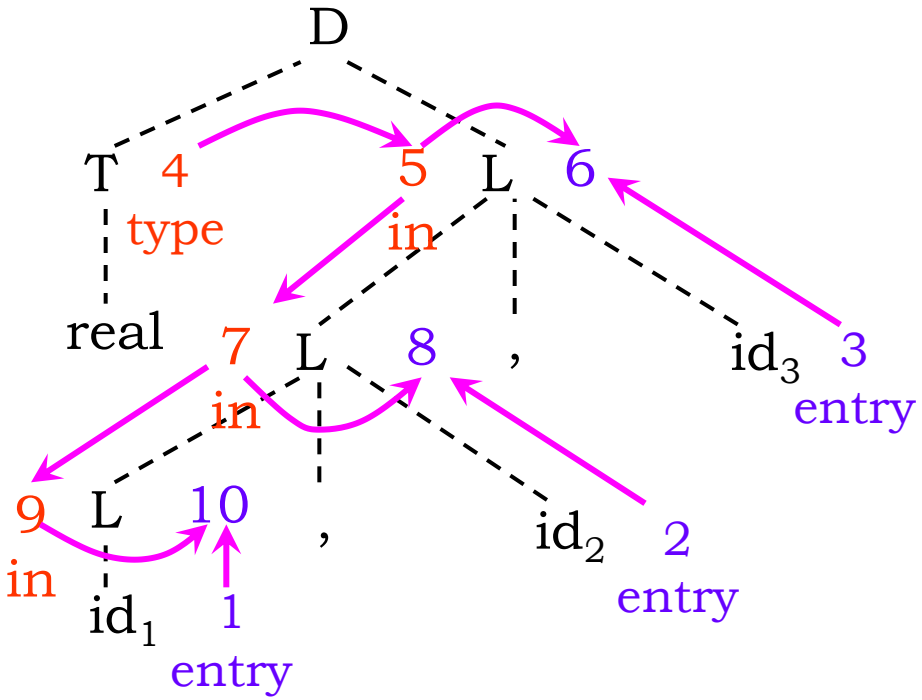


Dependency graph

- 如果一属性文法不存在属性之间的循环依赖关系，那么称该文法为良定义的
- 属性的计算次序
  - 一个依赖图的任何拓扑排序都给出一个语法树中结点的语义规则计算的有效顺序
- 属性文法说明的翻译是很精确的
  - 基础文法用于建立输入符号串的语法分析树
  - 根据计算语义规则建立依赖图
  - 从依赖图的拓扑排序中，得到计算语义规则的顺序

Dependency graph

- 每一个与L产生式有关的语义规则addtype（id.entry, L.in）都产生一个虚属性
- 为这些虚属性构造结点6、8和10



a_4:=real;

a_5:=a_4

addtype (id_3.entry,a_5);

a_7:=a_5;

addtype (id_2.entry,a_7);

a_9:=a_7

addtype (id_1.entry,a_9);

Tree traversal

- 通过树遍历的方法计算属性的值
  - 假设已经建立语法树，并且树中已带有开始符号的继承属性和终结符的综合属性
  - 以某种次序遍历语法树，直至计算出所有属性
    - 深度优先，从左到右的遍历

Tree traversal

```
While  (还有未被计算的属性){
    VisitNode(S)     //S是开始符号
}

void VisitNode (Node N) {
    if (N是一个非终结符) {
        //假设它的产生式为N→X1…Xm
        for (产生式右部的每个非终结符Xi) {
            计算Xi的所有能够计算的继承属性;
            VisitNode (Xi);
        }
    }
    计算N的所有能够计算的综合属性
}
```

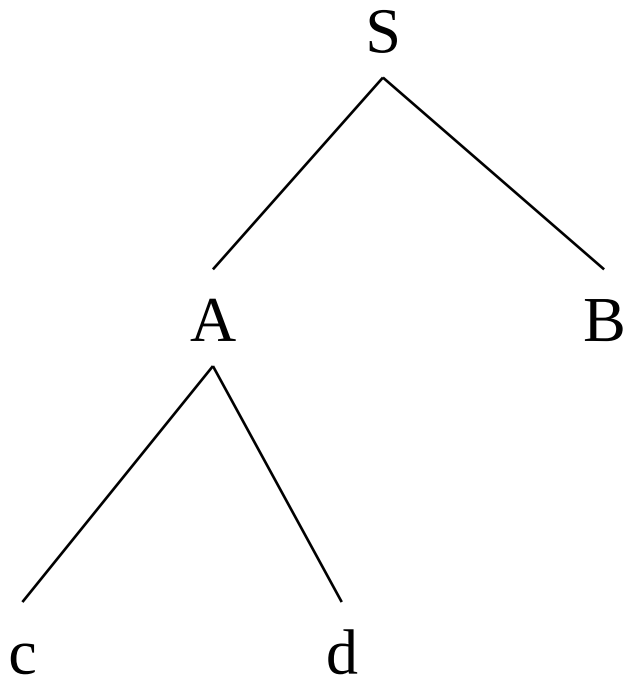
Tree traversal

VisitNode(S)

- 计算A的继承属性
- VisitNode(A)
  - 计算A的综合属性
- 计算B的继承属性
- VisitNode(B)
- 计算S的综合属性

多趟

如果有值没有算出，继续VisitNode(S)



Example

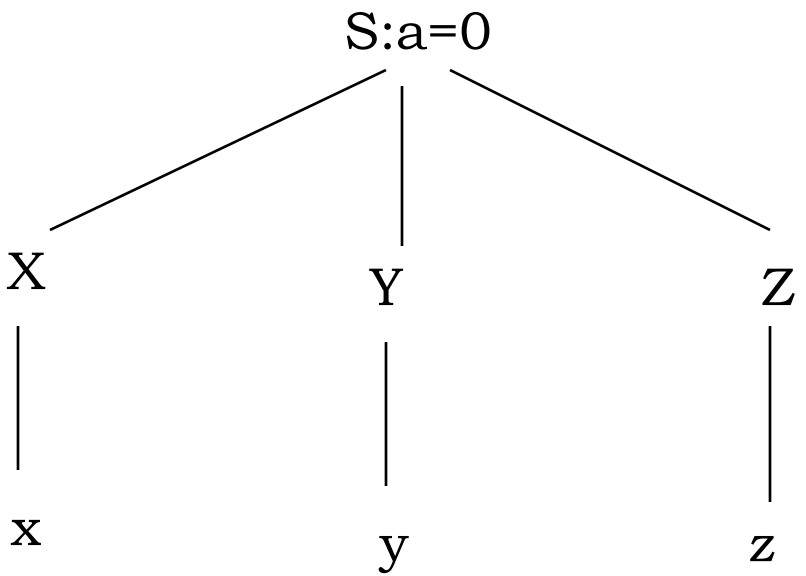
- 例:考虑属性的文法G。其中:
  - S有继承属性a，综合属性b
  - X有继承属性c、综合属性d
  - Y有继承属性e、综合属性f
  - Z有继承属性h、综合属性g

产 生 式	语 义 规 则
S→XYZ	Z.h := S.a
	X.c := Z.g
	S.b := X.d - 2
	Y.e := S.b
X→x	X.d :=2×X.c
Y→y	Y.f := Y.e×3
Z→z	Z.g :=Z.h + 1

Example

- 假设S.a的初始值为0，输入串为xyz

产生式	语义规则
S→XYZ	Z.h := S.a
	X.c := Z.g
	S.b := X.d -2
	Y.e := S.b
X→x	X.d :=2*X.c
Y→y	Y.f := Y.e*3
Z→z	Z.g :=Z.h+1



X继承属性c
X综合属性d

Y继承属性e
Y综合属性f

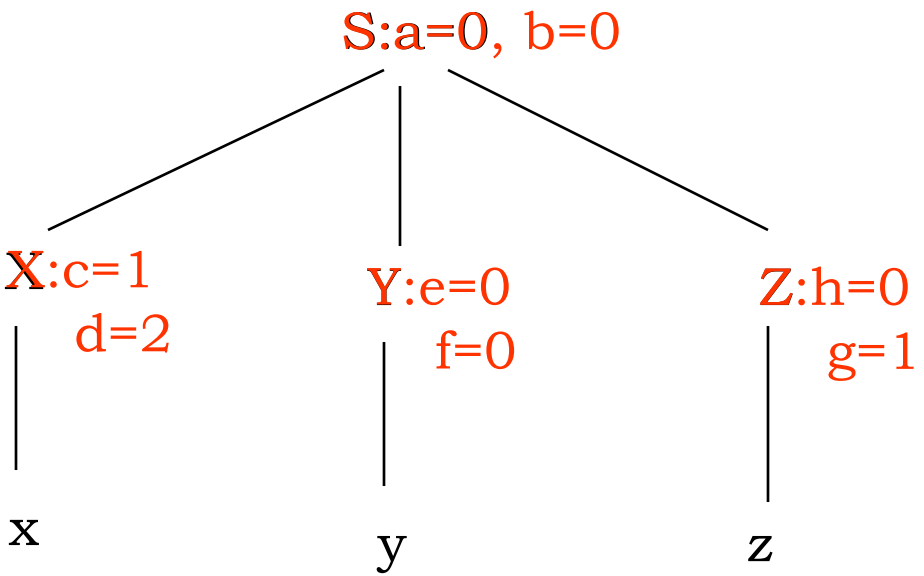
Z继承属性h
Z综合属性g

S综合属性b

Example

- 假设S.a的初始值为0，输入串为xyz

产生式	语义规则
S→XYZ	Z.h := S.a
	X.c := Z.g
	S.b := X.d -2
	Y.e := S.b
X→x	X.d :=2*X.c
Y→y	Y.f := Y.e*3
Z→z	Z.g :=Z.h+1



X继承属性c X综合属性d	Y继承属性e Y综合属性f	Z继承属性h Z综合属性g	S综合属性b
第二遍	第四遍	第一遍	第三遍

Example

- 能否只运行一遍就计算完成？
- $S \rightarrow AB$

A继承属性
↓
A综合属性
↓
B继承属性
↓
B综合属性
↓
S综合属性

⇦ S综合属性，S继承属性
B综合属性，B继承属性

L文法：从左到右计算
A的继承属性只依赖于S的继承属性；
B的继承属性依赖于S的继承属性和A的属性；

S文法：只有综合属性

One pass

- 一遍扫描的处理方法
  - 在语法分析的同时计算属性值
    - 语法分析方法
    - 属性的计算次序
- L - 属性文法
  - 适合于一遍扫描的自上而下分析
- S - 属性文法
  - 适合于一遍扫描的自下而上分析

Syntax-directed translation

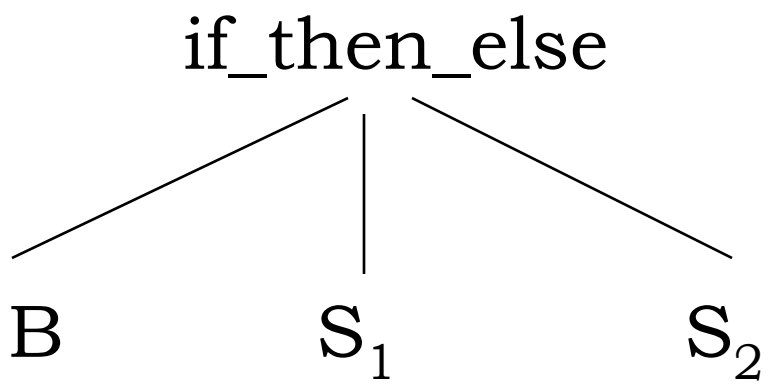
- 所谓语法制导翻译法
  - 就是为文法中每个产生式配上一组语义规则
  - 在语法分析的同时执行这些语义规则
- 计算语义规则的时机
  - 在自上而下分析中
    - 一个产生式匹配输入串成功时
  - 在自下而上分析中
    - 当一个产生式被用于进行归约时

语法分析工作和语义规则计算是穿插进行的

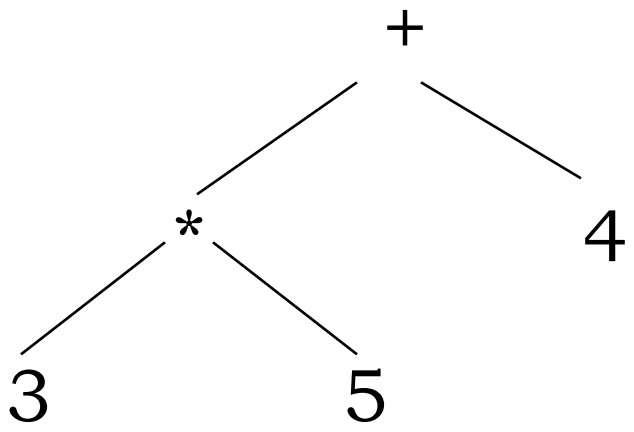
Abstract Syntax Tree

- 抽象语法树
  - 在语法树中去掉那些对翻译不必要的信息
  - 目的：获得更有效的源程序中间表示

- $S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$



- 3*5+4



Abstract Syntax Tree

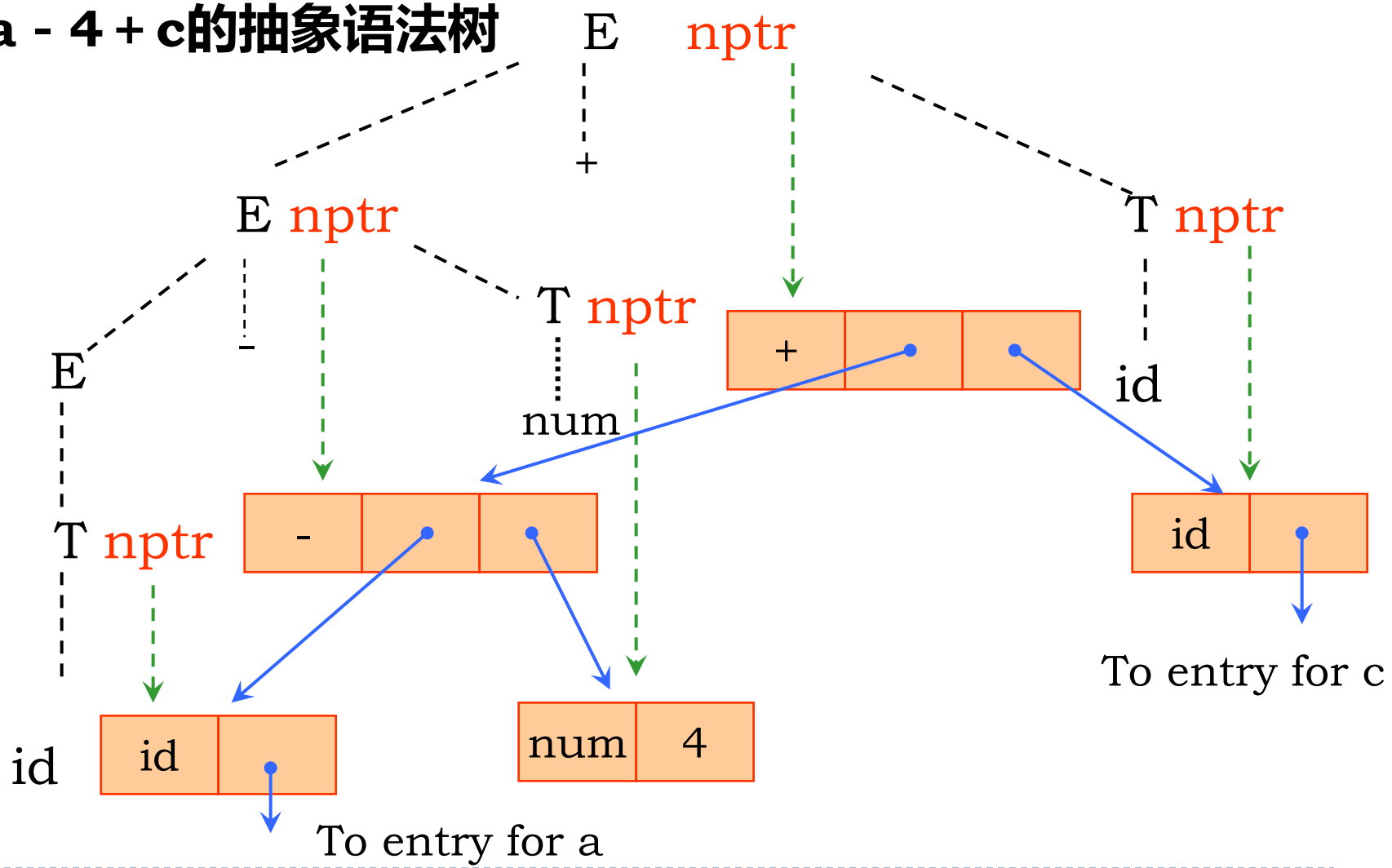
- mknode（op，left，right)
  - 建立一个运算符号结点
  - 标号是op
  - 两个域left和right分别指向左子树和右子树
- mkleaf（id，entry)
  - 建立一个标识符结点
  - 标号为id
  - 一个域eutry指向标识符在符号表中的入口
- mkleaf（num，ral)
  - 建立一个数结点
  - 标号为num
  - 一个域ral用于存放数的值

Abstract Syntax Tree – 例7

产生式	语义规则
$E \rightarrow E_1 + T$	E.nptr := mknode('+', E_1.nptr, T.nptr)
$E \rightarrow E_1 - T$	E.nptr := mknode('-', E_1.nptr, T.nptr)
$E \rightarrow T$	E.nptr := T.nptr
$T \rightarrow (E)$	T.nptr := E.nptr
$T \rightarrow id$	T.nptr := mkleaf(id, id.entry)
$T \rightarrow num$	T.nptr := mkleaf(num, num.val)

例8

- **a - 4 + c的抽象语法树**



S-Attribute grammar

- S-属性文法：只含有综合属性
- S-属性文法的翻译器通常可借助于LR分析器实现
- 综合属性
  - 可以在分析输入符号串的同时由自下而上的分析器来计算
- 分析器
  - 保存与栈中文法符号有关的综合属性值
  - 每当进行归约时，新的属性值就由栈中正在归约的产生式右边符号的属性值来计算

S-Attribute grammar

- 在分析栈中使用一个附加的域来存放综合属性值
- 假设语义规则A.a:=f(X.x,Y.y,Z.z)是对应于产生式A→XYZ的

S_m	Z. z	Z
S_{m-1}	Y. y	Y
S_{m-2}	X. x	X
⋮	⋮	⋮
S_0	–	#

S'_{m-2}	A. a	A
⋮	⋮	⋮
S_0	–	#

S-Attribute grammar – 例9

产生式	代 码 段
L→En	print(val[top])
$E \rightarrow E_1+T$	val[ntop] := val[top-2]+val[top]
E→T	
$T \rightarrow T_1*F$	val[ntop] := val[top-2]*val[top]
T→F	
F→ (E)	val[ntop] :=val[top-1]
F→digit	

S-Attribute grammar – 例9

输入	state	val	用到的产生式
3*5+4n	—	—	
*5+4n	3	3	
*5+4n	F	3	F→digit
*5+4n	T	3	T→F
5+4n	T*	3 -	
+4n	T*5	3 - 5	
+4n	T*F	3 - 5	F→digit
+4n	T	15	T→T*F
+4n	E	15	E→T
4n	E+	15-	
n	E+4	15- 4	
n	E+F	15- 4	F→digit
n	E+T	15- 4	T→F
n	E	19	E→E+T
	En	19-	
	L	19	L→En

Exercise

- 例10：有文法： S→(L)|a　　L→L,S|S

  给此文法配上语义动作子程序(或者说为此文法写一个语法制导定义)，它输出配对括号的个数。如对于句子(a,(a,a))，输出是2。

Exercise

- 例10：有文法： S→(L)|a　　L→L,S|S

  给此文法配上语义动作子程序(或者说为此文法写一个语法制导定义)，它输出配对括号的个数。如对于句子(a,(a,a))，输出是2。

  - 解：加入新开始符号S'和产生式S'→S，设num 为综合属性，代表值属性，则语法制导定义如下：

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

Exercise

- 例11：构造属性文法，能对下面的文法，只利用综合属性获得类型信息。

D → L,id | L　　L → T id　　T → int | real

Exercise

- 例11：构造属性文法，能对下面的文法，只利用综合属性获得类型信息。

D → L,id | L　　L → T id　　T → int | real

- 解：属性文法（语法制导）定义：

产生式	语义规则
D → L,id	D.type:=L.type addtype(id.entry,L.type)
D → L	D.type:=L.type
L → T id	L.type:=T.type addtype(id.entry,T.type)
T → int	T.type:=integer
T → real	T.type:=real