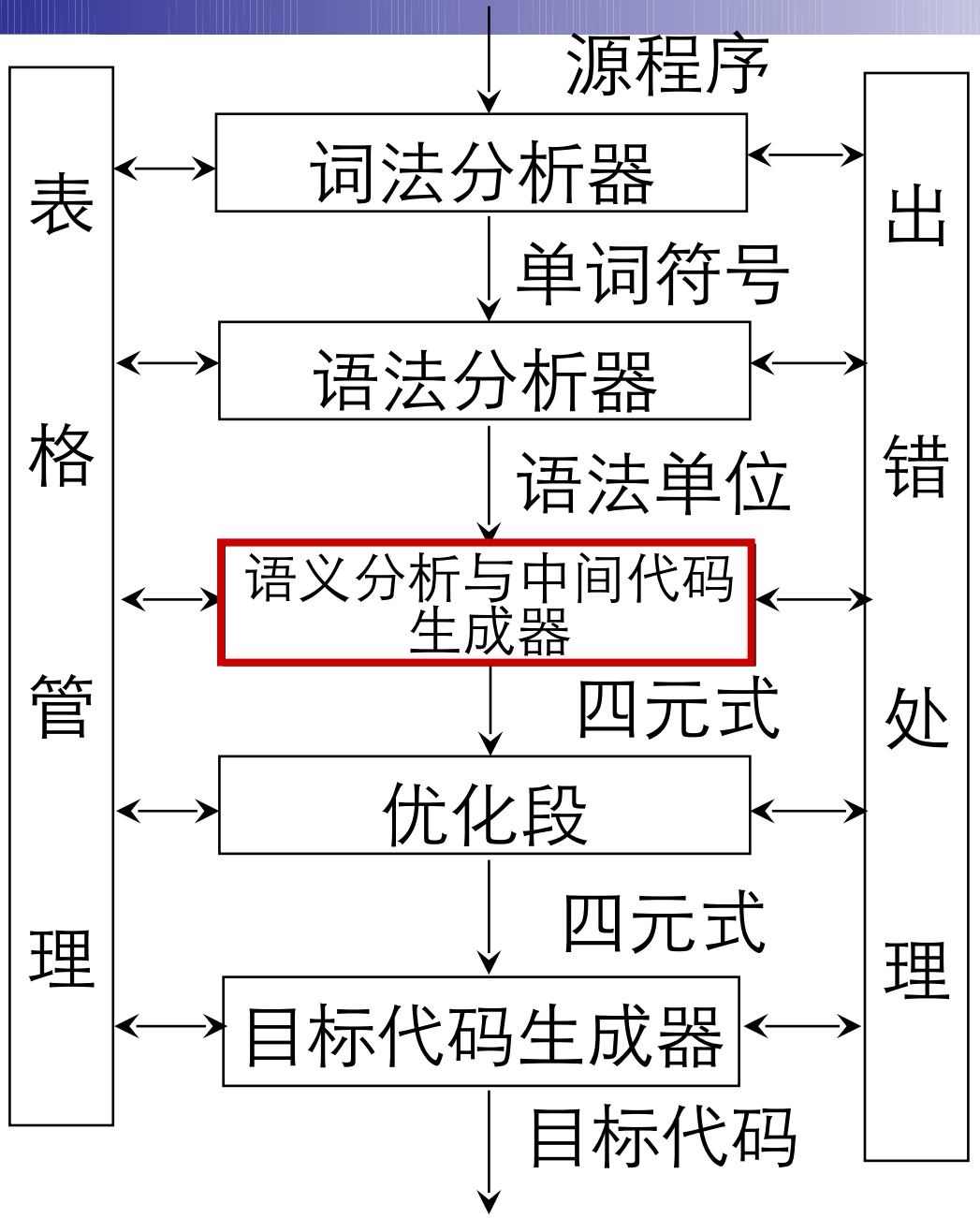
编译原理

第六章 属性文法和语法制导翻译

编译程序总框

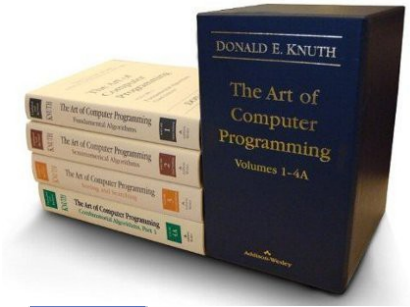

源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

第六章 属性文法和语法制导翻译

- 属性文法
- 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

6.1 属性文法

- 属性文法（也称属性翻译文法）
  - Knuth 在 1968 年提出



The Art of Computer Programming

...
Lexical scanning
Parsing techniques
Context-Free Languages
Compiler Techniques
...

Donald Ervin Knuth

TEX

属性文法

- 属性文法（也称属性翻译文法）
  - Knuth 在 1968 年提出
  - 在上下文无关文法的基础上，为每个文法符号（终结符或非终结符）配备若干相关的“值”（称为属性）
    - 属性代表与文法符号相关信息，如类型、值、代码序列、符号表内容等
    - 属性可以进行计算和传递
    - 语义规则：对于文法的每个产生式都配备了一组属性的计算规则

- 属性
  - 综合属性：“自下而上”传递信息
  - 继承属性：“自上而下”传递信息
- 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：

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

这里，f 是一个函数，而且或者

1. b 是 A 的一个综合属性并且 $c_1,c_2,\cdots,c_k$ 是产生式右边文法符号的属性，或
2. b 是产生式右边某个文 $c_1,c_2,\cdots,c_k$ 是 A 或产生

- 这两种情况下：属性 b

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→digit	F.val :=digit.lexval

属性文法

- 说明
  - 终结符只有综合属性，由词法分析器提供
    - F→digit
    - digit.lexval
  - 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
    - F→digit
    - F.val 、 digit.lexval

说明

- 对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
  - F→digit
  - F.val :=digit.lexval
- 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，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
- 语义规则所描述的工作可以包括属性计算、静态语义检查、符号表操作、代码生成等等

- 例，考虑非终结符 A， B 和 C，其中，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

产 生 式	语 义 规 则
L→En	print(E.val)
$E \rightarrow E_1+T$	E.val := E_1.val+T.val
E→T	E.val :=T.val
$T \rightarrow T_1*F$	T.val :=T_1.val* F.val
T→F	T.val :=F.val
F→ (E)	F.val :=E.val
F→digit	F.val :=digit.lexval

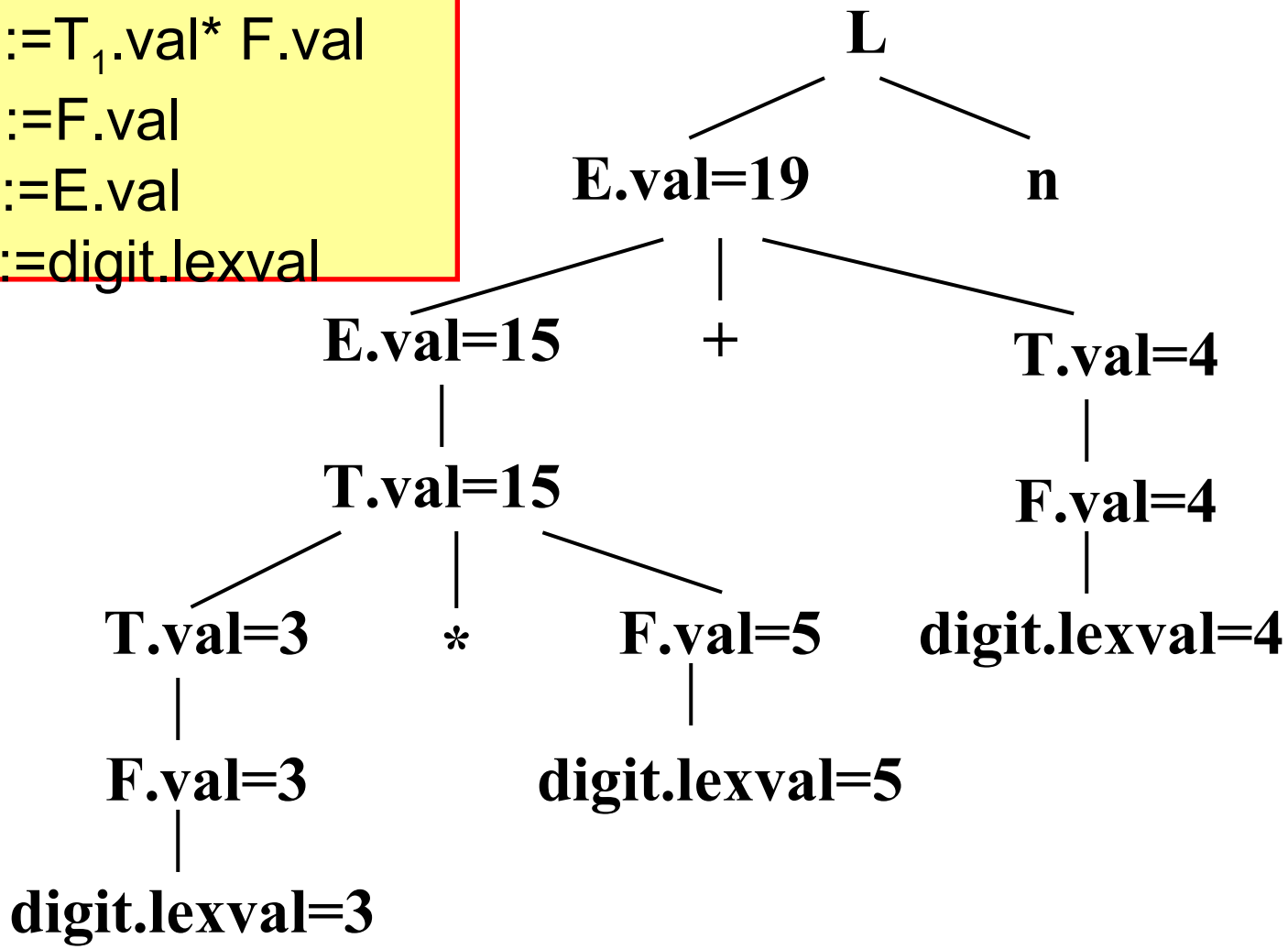
综合属性

- 综合属性
  - 在语法树中，一个结点的综合属性的值由其子结点和它本身的属性值确定
  - 使用自底向上的方法在每一个结点处使用语义规则计算综合属性的值
- 仅仅使用综合属性的属性文法称 S – 属性文法

产 生 式	语 义 规 则
L→En	print(E.val)
$E \rightarrow E_1+T$	E.val := E_1.val+T.val
E→T	E.val :=T.val
$T \rightarrow T_1*F$	T.val :=T_1.val* F.val
T→F	T.val :=F.val
F→ (E)	F.val :=E.val
F→digit	F.val :=digit.lexval

3*5+4n 的带注释的语法树

产生式	语义规则
$L \rightarrow En$	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

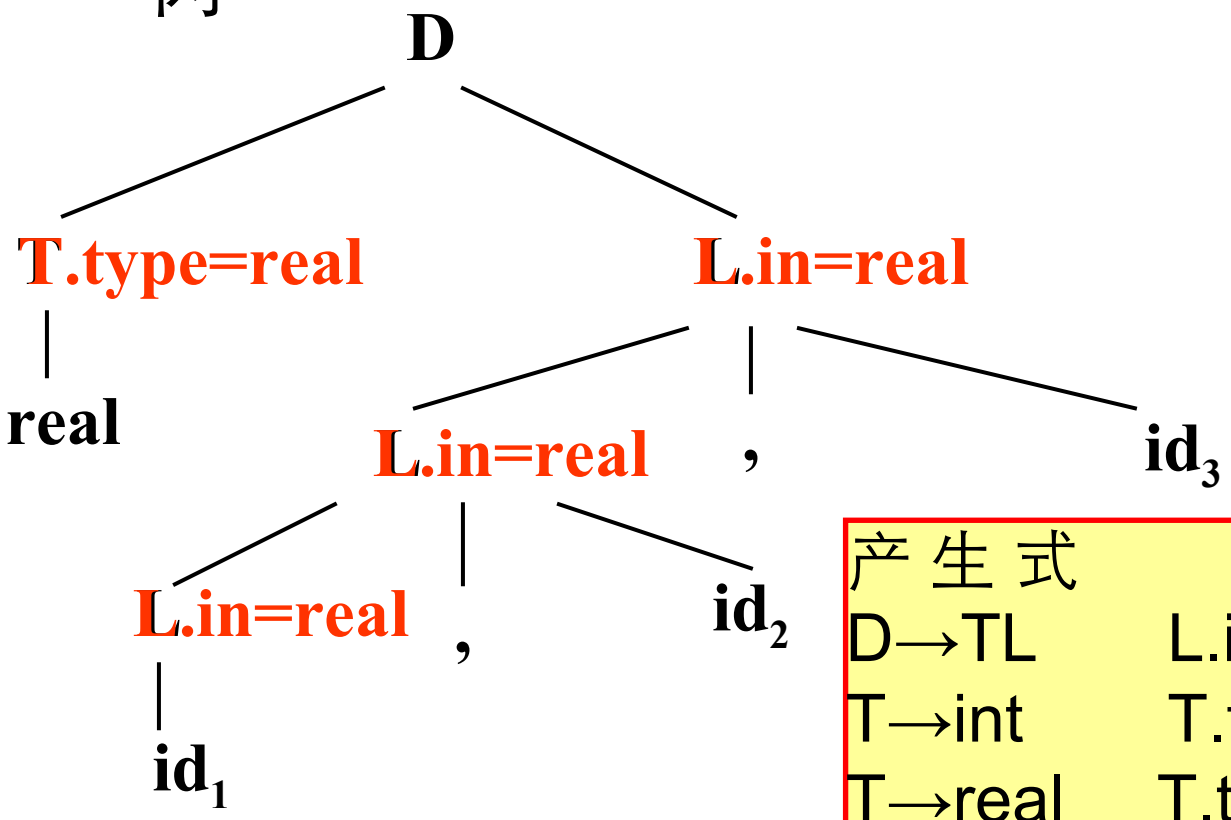


继承属性

- 继承属性
  - 在语法树中，一个结点的继承属性由其父结点、其兄弟结点和其本身的某些属性确定
  - 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

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

句子 real id_1 , id_2 , id_3 的带注释的语法树



name	type
…	…
id_3	real
id_2	real
id_1	real

产生式	语义规则
D→TL	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)

小结

- 属性文法
- 综合属性
- 继承属性

作业

- P164 - 1

1. 设 AS 为文法的综合属性集, AI 为继承属性集, 则下列语法制导定义中

产生式	语义规则
P → xQR	Q.b:=R.d
	R.c:=1
	R.e:=Q.a
P → yQR	Q.b:=R.f
	R.c:=Q.a
	R.e:=2
Q → u	Q.a:=3
R → v	R.d:=R.c
	R.f:=R.e

(a) AS={ Q.a, R.c, R.e } AI={ Q.b, R.d, R.f }
(b) AS={ Q.a, Q.b } AI={ R.c, R.d, R.e, R.f }
(c) AS={ Q.b, R.c, R.f } AI={ Q.a, R.d, R.e }
(d) AS={ Q.a, R.d, R.f } AI={ Q.b, R.c, R.e }
(e) AS={ Q.b, R.d, R.e } AI={ Q.a, R.c, R.f } 哪个是正确的?

(上海交通大学 1995 年硕士生入学考试试题)

解题思路:

在语法制导定义(属性文法)中, 综合属性用于“自下而上”传递信息，而继承属性用于“自上而下”传递信息。判断综合属性和继承属性的简单方法是, 产生式左部符号的综合属性根据其右部的符号的属性和(或)左部符号自己的其他属性计算而得; 产生式右部符号的继承属性根据其左部符号的属性和(或)右部的其他符号的属性计算而得。

本题中, 从 P → xQR 的语义规则 Q.b:=R.d, R.c:=1, R.e:=Q.a 可得: Q.b, R.c 和 R.e 为继承属性，而 R.d, Q.a 的性质则需要其他产生式的语义规则一起加以分析才能确定; 从 Q → u 的语义规则 Q.a:=3 可得 Q.a 为综合属性。 依此方法, 最后可得该语法制导定义的综合属性集为 AS={ Q.a, R.d, R.f },继承属性集为 AI={ Q.b, R.c, R.e }。

解答:

(d) 是正确的

2. 文法及相应的翻译方案:

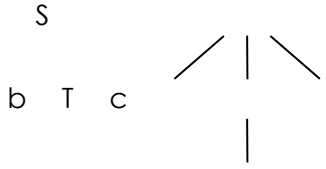
S → bTc	{print” 1” }
S → a	{print” 2” }
T → R	{print” 3” }
R → R/S	{print” 4” }
R → S	{print” 5” }

对于输入符号串 bR/bTc/bSc/ac, 该输入符号串的输出是什么?

(上海交通大学 1997 年硕士生入学考试试题)

解题思路:

该输入串对应的语法树如图 5.3



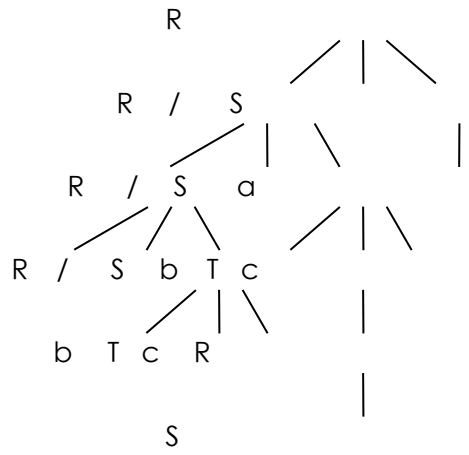


图 5.3 语法树

按深度优先遍历语法树，每当一个产生式得到匹配时则执行相应的语义动作，可得翻译结果。对本例，按深度优先遍历语法树，第一个得到匹配的产生式为 S → bTc，执行相应的语义动作，打印 1；第二个得到匹配的产生式为 R → R/S，执行相应的语义动作，打印 4；第三个得到匹配的产生式为 R → S，执行相应的语义动作，打印 5；等等。

解答：

翻译结果为 1453142431。

例题 6.1.1 下列文法对整型常数和实型常数施用加法运算符 + 生成表达式；当两个整型数相加时，结果仍为整型数，否则，结果为实型数：

E→E+T | T

T→num．num | num

(1) 试给出确定每个子表达式结果类型的属性文法；

(2) 扩充(1)的属性文法，使之把表达式翻译成后缀形式，同时也能确定结果的类型。应该注意使用一元运算符 inttoreal 把整型数转换成实型数，以便使后缀形如加法运算符的两个操作数具有相同的类型。

解题思路： 确定每个子表达式结果类型的属性文法是比较容易定义的。关键是如何扩充此属性文法，使之把表达式翻译成后缀形式。我们将不在 name 或 num.num 向 T 归约的时候输出该运算对象，而是把运算对象的输出放在 T 或 E+T 向 E 归约的时候。这是因为考虑输出类型转换算符 inttoreal 的动作可能在 E+T 向 E 归约的时候进行，如果这时两个运算对象都在前面 name 或 num.num 向 T 归约的时候已输出，需要为第一个运算对象输出类型转换算符时就已经为时太晚。

还要注意的是，在 E+T 向 E 归约时，该加法运算的第一个运算对象已经输出．所以 E→E + T 的语义规则不需要有输出 E 运算对象的动作。

解答:

(1) 为文法符号 E 和 T 配以综合属性 type，用来表示它们的类型。类型值分别用 int 和 real 来表示。确定每个子表达式结果类型的属性文法如下。

产生式　　语义规则

E->E1+T　　T.type:= if E1.type =int and T.type = int then int else real

E->T　　E.type:=T.type}

T->num.num　　T.type :=real}

T->num　　T.type:=int}

(2) 下面属性文法将表达式的后缀表示打印输出，其中 lexeme 属性表示单词的拼写。

产生式　　语义规则

E->E1+T　　if E1.type = real and T.type = int then

begin

E.type :=real;

```
                    print(T.lexeme);
                    print('inttoreal')
                  end
                else if E1.type = int and T.type=real then
                  Begin
                    E.type :=real;
                    print('inttoreal') ;
                    print(T.lexeme)
                  end
                else begin
                    E.type :=E1.type;
                    print(T.lexeme)
                  end;
    print ('+');
  E->T    E.type:=T.type;
                    print(T.lexeme)
  T->num1.num2  T.type:=real;
                    T.lexeme:=num1.lexeme|| ". "||num2.lexeme
T->num    T.type:=int;
                    T.lexeme:=num.lexeme
```

属性文法（也称属性翻译文法）是在上下文无关文法的基础上，为每个文法符号（终结符或非终结符）配备若干相关的“值”（称为属性）。这些属性代表与文法符号相关信息，如类型、值、代码序列、符号表内容等。属性可以进行计算和传递。对于文法的每个产生式都配备了一组属性的计算规则称为**语义规则**。

第六章 属性文法和语法制导翻译

- 属性文法
- 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

6.2 基于属性文法的处理方法

输入串⟶语法树 ⟶ 按照语义规则计算属性

- 由源程序的语法结构所驱动的处理办法就是语法制导翻译法
- 语义规则的计算
  - 产生代码
  - 在符号表中存放信息
  - 给出错误信息
  - 执行任何其它动作
- 对输入符号串的翻译也就是根据语义规则进行计算的结果

6.2 基于属性文法的处理方法

- 依赖图
- 树遍历
- 一遍扫描

依赖图

- 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依赖图（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

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

的形式

- 依赖图中为每一个属性设置一个结点，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。

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

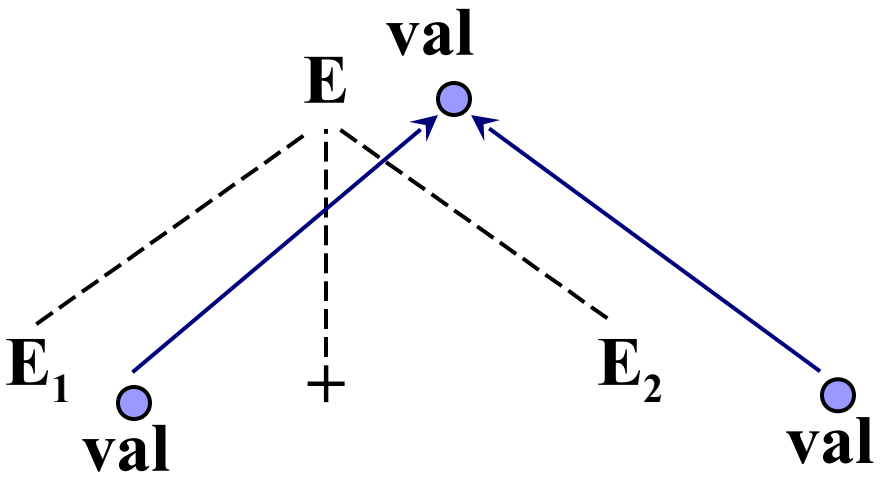
　　for　结点 n 所用产生式对应的每一个语义规则

　　$b:=f(c_1,c_2,\cdots,c_k)$ do

　　for i:=1 to k do

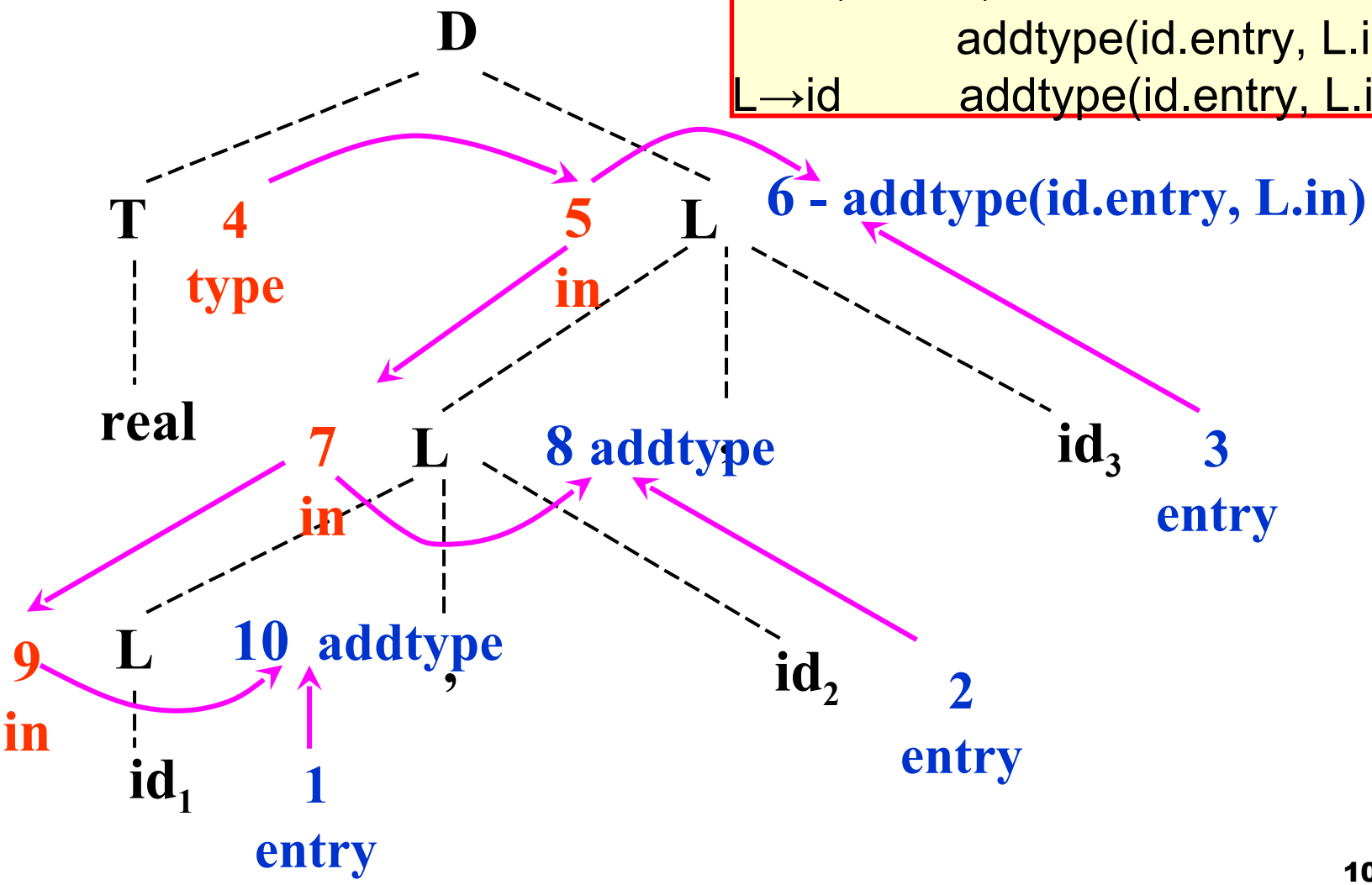
　　从 c_i 结点到 b 结点构造一条有向边；

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



句子 real id_1，id_2，id_3 的带注释的语法树的依赖图

产生式	语义规则
D→TL	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)



良定义的属性文法

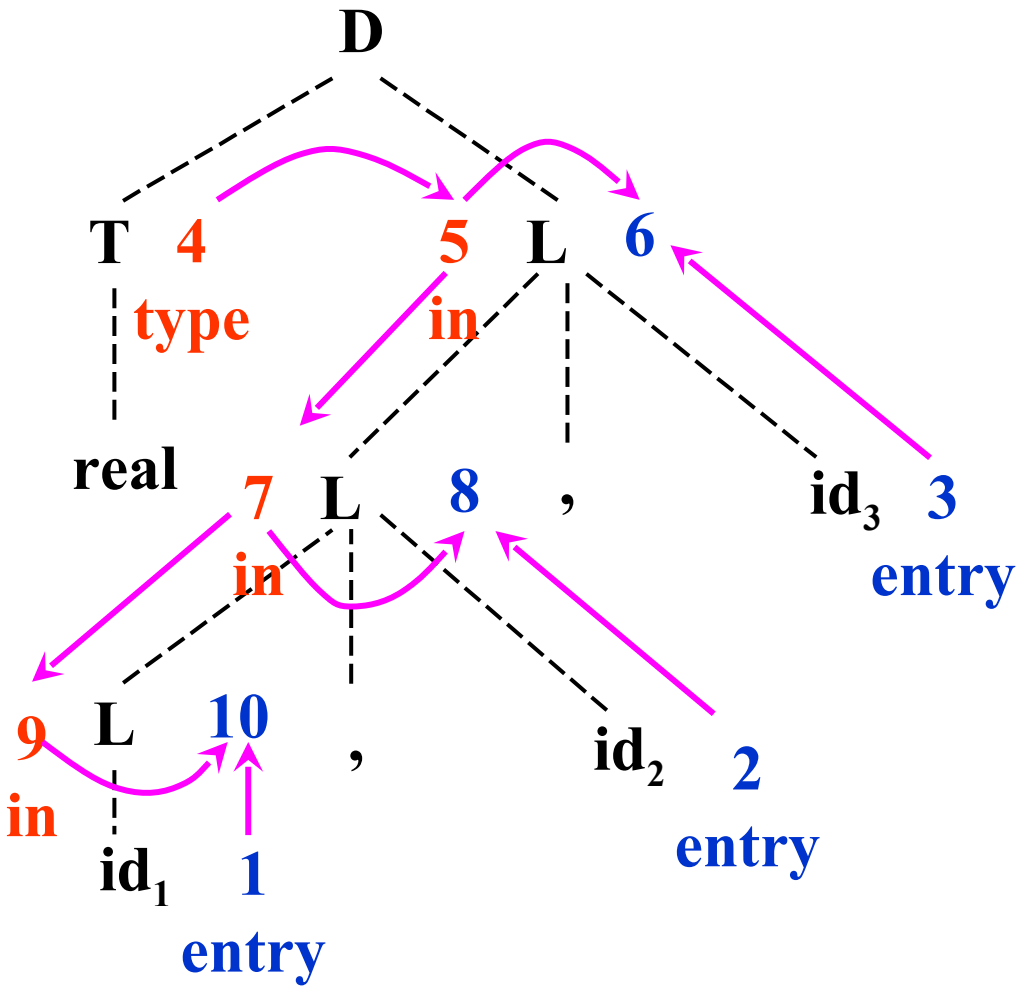
- 如果一属性文法不存在属性之间的循环依赖关系，那么称该文法为良定义的

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
  - 根据语义规则建立依赖图
  - 从依赖图的拓扑排序中，我们可以得到计算语义规则的顺序

输入串 ⟶ 语法树 ⟶ 依赖图 ⟶ 语义规则计算次序

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



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);

6.2 基于属性文法的的处理方法

- 依赖图
- 树遍历
- 一遍扫描

6.2.2 树遍历的属性计算方法

- 通过树遍历的方法计算属性的值
  - 假设语法树已建立，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

While 还有未被计算的属
VisitNode(S) /*S 是开始符号

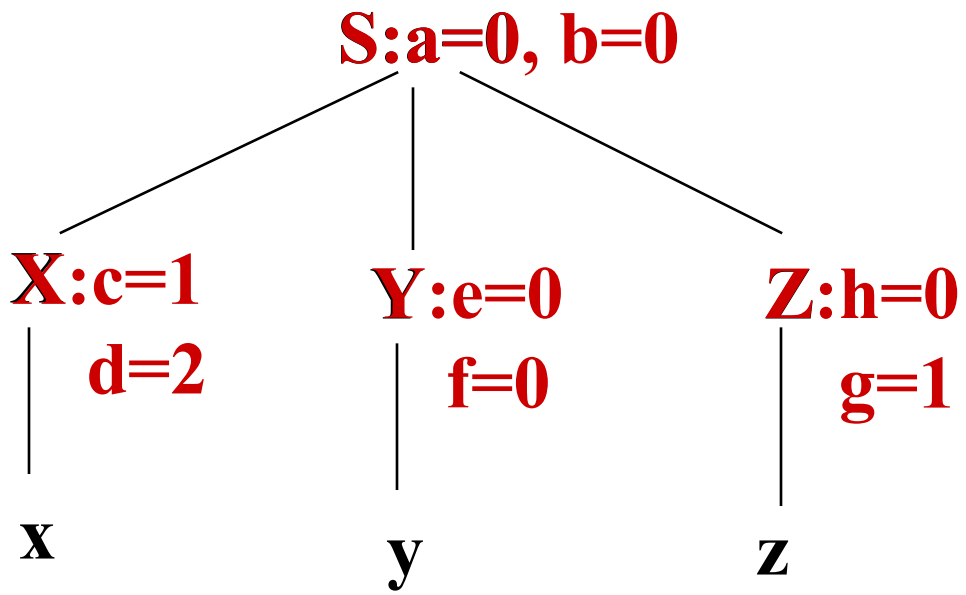
procedure VisitNode (N:
begin
if N 是一个非终结符 the
/* 假设它的产生式为 $N \rightarrow X_1 \cdots X_m$ */
for i :=1 to m do
if $X_i \in V_N$ then /* 即 X_i 是非终结符 */
begin
计算 X_i 的所有能够计算的继承属性;
VisitNode (X_i)
end;
计算 N 的所有能够计算的综合属性
end

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

假设 S.a 的初始值为 0， 输入串为

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



6.2 基于属性文法的的处理方法

- 依赖图
- 树遍历
- 一遍扫描

一遍扫描的处理方法

- 一遍扫描的处理方法是在语法分析的同时计算属性值
  - 所采用的语法分析方法
  - 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

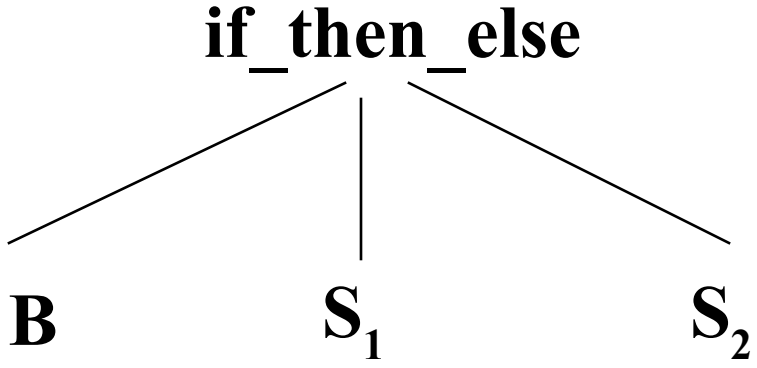
语法制导翻译法

- 所谓语法制导翻译法，直观上说就是为文法中每个产生式配上一组语义规则，并且在语法分析的同时执行这些语义规则
- 语义规则被计算的时机
  - 在自上而下语法分析中，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

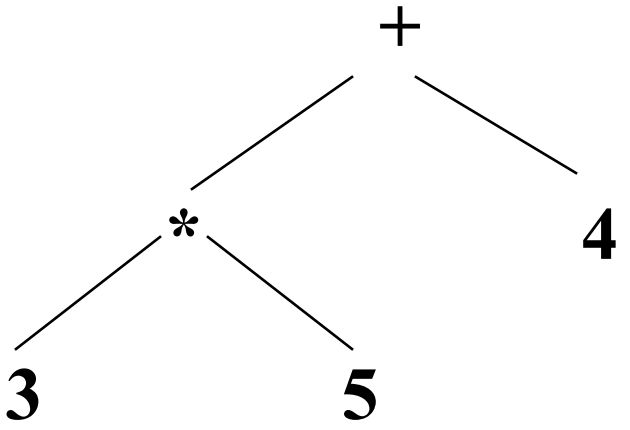
抽象语法树

- 在语法树中去掉那些对翻译不必要的信息，从而获得更有效的源程序中间表示。这种经变换后的语法树称之为抽象语法树(Abstract Syntax Tree)

- **S→if B then S_1 else S_2**



- **3*5+4**



建立表达式的抽象语法树

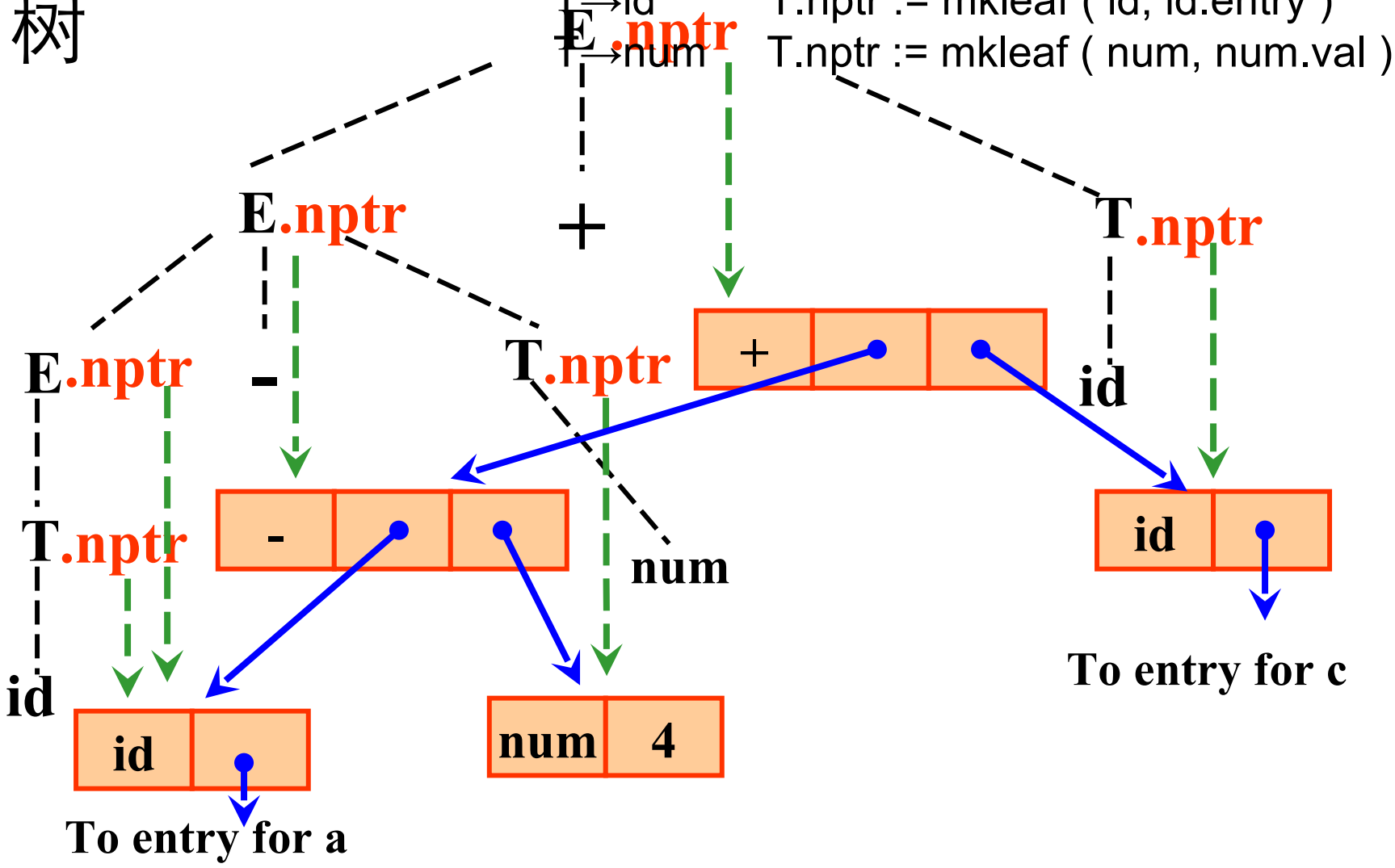
- mknode (op,left,right) 建立一个运算符号结点，标号是 op，两个域 left 和 right 分别指向左子树和右子树
- mkleaf (id,entry) 建立一个标识符结点，标号为 id，一个域 entry 指向标识符在符号表中的入口
- mkleaf (num,val) 建立一个数结点，标号为 num，一个域 val 用于存放数的值

建立抽象语法树的语义规则

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)

a – 4 + c 的抽象语法树

$E \rightarrow E_1 + T$	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
$T \rightarrow id$	T.nptr := mkleaf (id, id.entry)
$T \rightarrow num$	T.nptr := mkleaf (num, num.val)



第六章 属性文法和语法制导翻译

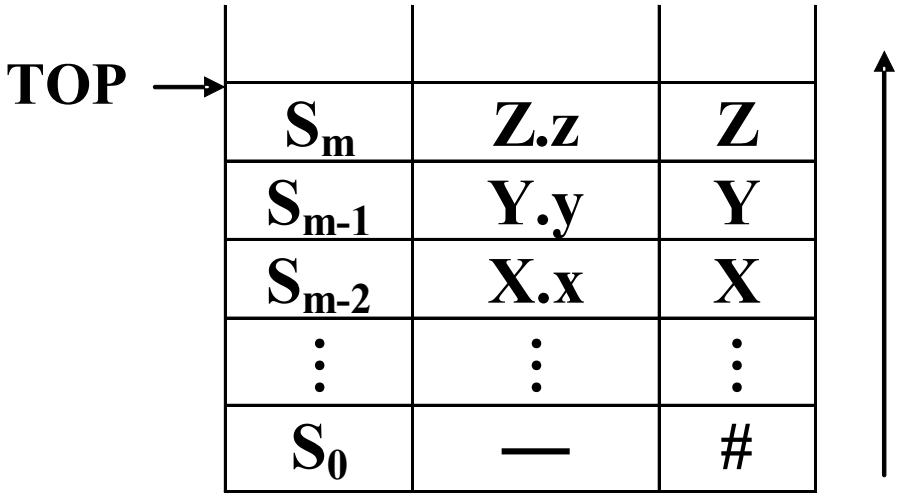
- 属性文法
- 基于属性文法的处理方法
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

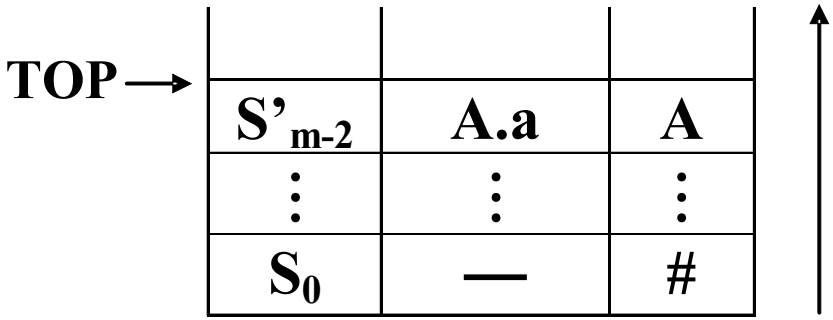
6.3 S- 属性文法的自下而上计算

- S- 属性文法：只含有综合属性
- 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
- 分析器可以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- 属性文法的计算

- 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





讨论： E 、 T 和 F 为什么没有代码段？

TOP →

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

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

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	

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	

state	sym	val	输入	用到的产生式
0	#	-	3*5+4n	
05	#3	-3	*5+4n	
03	#F	-3	*5+4n	F→digit
02	#T	-3	*5+4n	T→F
027	#T*	-3 -	5+4n	
0275	#T*5	-3 - 5	+4n	

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	

state	sym	val	输入	用到的产生式
0275	#T*5	-3 – 5	+4n	
02710	#T*F	-3 - 5	+4n	F→digit
02	#T	-15	+4n	T→T*F
01	#E	-15	+4n	E→T
016	#E+	-15-	4n	
0165	#E+4	-15- 4	n	

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	

state	sym	val	输入	用到的产生式
0165	#E+4	-15- 4	n	
0163	#E+F	-15- 4	n	F→digit
0169	#E+T	-15- 4	n	T→F
01	#E	-19	n	E→E+T
	#En	-19-		
	#L	-19	L→En	

小结

- 抽象语法树
- S- 属性文法
- S- 属性文法的自下而上计算
  - 一遍扫描

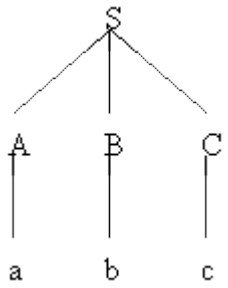
例题 6.2.1 考虑下面的属性文法

产 生 式	语 义 规 则
S→ABC	B.u := S.u
	A.u := B.v + C.v
	S.v := A.v
A→a	A.v :=2*A.u
B→b	B.v := B.u
C→c	C.v := 1

(1) 画出字符串 abc 的语法树;

(2) 对于该语法树，假设 S.u 的初始值为 3，属性计算完成后，S.v 的值为多少。

解答: (1)



(2) S.v 的值为 8

在语法树中，一个结点的**继承属性**由其父结点、其兄弟结点和其本身的某些属性确定。
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**依赖图**(有向图)来描述。
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**。
所谓**语法制导翻译法**，直观上说就是为文法中每个产生式配上一组语义规则，并且在语法分析的同时执行这些语义规则。

1.令 S.val 为文法 G[S]生成的二进制数的值，例如，对输入串 101.101 则 S.val=5.625。按照语法制导翻译方法的思想，给出计算 S.val 的相应的语义规则。

G[S]： S→L.L|L

L→LB|B

B→0|1

答案：

语义动作子程序如下:

产生式	语义动作
S'→ S	{ print(S.val) }
S → $L_1.L_2$	{ S.val:=L_1.val+L_2.val/$2^{L2.length}$}
S → L	{ S.val:=L.val }
L → L_1B	{ L.val:=L_1.val*2+B.val }
{ L.length:=L_1.length+1 }	
L → B	{ L.val:=B.val; L.length:=1 }
B → 1	{ B.val:=1 }
B → 0	{ B.val:=0 }

2.设某语言的 for 语句的形式:

S → for i:=$E^{(1)}$ to $E^{(2)}$ do $S^{(1)}$

其语义解释为

i:= $E^{(1)}$;

LIMIT := $E^{(2)}$;

again: if i<=LIMIT then

Begin

$S^{(1)}$;

i:=i+1;

goto again

End;

1 (1).写出适合语法制导翻译的产生式；

(2). 写出每个产生式对应的语义动作。

答案：

(1). 适合语法制导翻译的产生式:

F→for i:= $E^{(1)}$ to $E^{(2)}$ do

S→F $S^{(1)}$

(2). 每个产生式对应的语义动作

F→for i:= $E^{(1)}$ to $E^{(2)}$ do

{

GEN(:=， $E^{(1)}$.PLACE，-，ENTRY(i));

F.PLACE:=ENTRY(i);

LIMIT=NEWTEMP;

GEN(:=， $E^{(2)}$.PLACE， -， LIMIT);

q:=NXQ;

F.QUAD:=q; //again
GEN(j≤，F.PLACE，LIMIT，q+2);
F.CHAIN := q+1;
GEN(j，-，-，0)
}
S→F $S^{(1)}$
{
BACKPATCH($S^{(1)}$.CHAIN， NXQ);
GEN(+，F.PLACE，1，F.PALCE)
GEN(j，-，-，F.QUAD);
S.CHAIN:= F.CHAIN;
}

1. 设 AS 为文法的综合属性集, AI 为继承属性集, 则下列语法制导定义中

产生式	语义规则
P → xQR	Q.b:=R.d
	R.c:=1
	R.e:=Q.a
P → yQR	Q.b:=R.f
	R.c:=Q.a
	R.e:=2
Q → u	Q.a:=3
R → v	R.d:=R.c
	R.f:=R.e

(a) AS={ Q.a, R.c, R.e } AI={ Q.b, R.d, R.f }
(b) AS={ Q.a, Q.b } AI={ R.c, R.d, R.e, R.f }
(c) AS={ Q.b, R.c, R.f } AI={ Q.a, R.d, R.e }
(d) AS={ Q.a, R.d, R.f } AI={ Q.b, R.c, R.e }
(e) AS={ Q.b, R.d, R.e } AI={ Q.a, R.c, R.f }

答案：(d) 是正确的

在语法制导定义(属性文法)中, 综合属性用于“自下而上”传递信息，而继承属性用于“自上而下”传递信息。判断综合属性和继承属性的简单方法是, 产生式左部符号的综合属性根据其右部的符号的属性和(或)左部符号自己的其他属性计算而得; 产生式右部符号的继承属性根据其左部符号的属性和(或)右部的其他符号的属性计算而得。

本题中, 从 P → xQR 的语义规则 Q.b:=R.d, R.c:=1, R.e:=Q.a 可得: Q.b, R.c 和 R.e 为继承属性，而 R.d, Q.a 的性质则需要其他产生式的语义规则一起加以分析才能确定; 从 Q → u 的语义规则 Q.a:=3 可得 Q.a 为综合属性。 依此方法, 最后可得该语法制导定义的综合属性集为 AS={ Q.a, R.d, R.f },继承属性集为 AI={ Q.b, R.c, R.e }。

2 文法及相应的翻译方案:

S → bTc	{print”1”}
S → a	{print”2”}
T → R	{print”3”}
R → R/S	{print”4”}
R → S	{print”5”}

对于输入符号串 bR/bTc/bSc/ac, 该输入符号串的输出是什么?

答案:

该输入串对应的语法树如下图所示

S

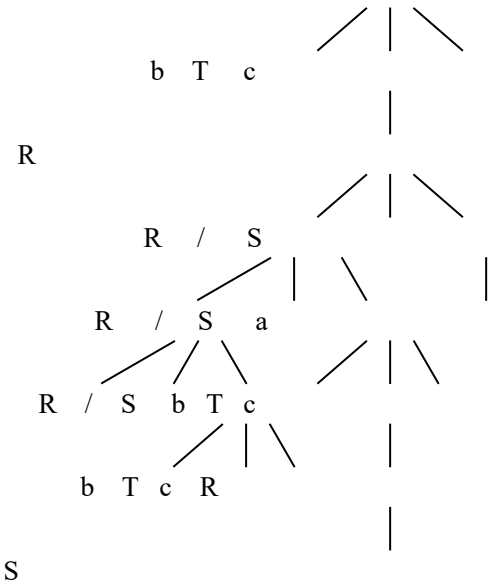


图 语法树

按深度优先遍历语法树, 每当一个产生式得到匹配时则执行相应的语义动作, 可得翻译结果。对本例, 按深度优先遍历语法树, 第一个得到匹配的产生式为 S → bTc, 执行相应的语义动作,打印 1; 第二个得到匹配的产生式为 R → R/S, 执行相应的语义动作,打印 4; 第三个得到匹配的产生式为 R → S, 执行相应的语义动作,打印 5; 等等。

因此翻译结果为 1453142431

第六章 属性文法和语法制导翻译

- 属性文法
- 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

一遍扫描的处理方法

- 一遍扫描的处理方法是在语法分析的同时计算属性值
  - 所采用的语法分析方法
  - 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

6.4 L- 属性文法和自顶向下翻译

- 通过深度优先的方法对语法树进行遍历，计算属性文法的所有属性值
- LL(1)：自上而下分析方法，深度优先建立语法树

L- 属性文法

- 一个属性文法称为 L- 属性文法，如果对于每个产生式 $A \rightarrow X_1 X_2 \cdots X_n$，其每个语义规则中的每个属性或者是综合属性，或者是 $X_j (1 \le j \le n)$ 的一个继承属性且这个继承属性仅依赖于：

  (1) 产生式中 X_j 左边符号 X_1，X_2，…，X_{j-1} 的属性

  (2) A 的继承属性

- S- 属性文法一定是 L- 属性文法（S属性文法只有综合属性）

产 生 式	语 义 规 则
A→LM	L.i := g(A.i)
	M.i :=m(L.s)
A→QR	R.i := r(A.i)
	Q.i :=q(R.s)
	A.s :=f(Q.s)

6.4.1

产生式	语义规则
E→TR	
R→addop T R_1 | ε	print(addop.lexeme)
T→num	print(num.val)

- 语义规则：给出了属性计算的定义，没有属性计算的次序等实现细节
- 翻译模式：给出了使用语义规则进行计算的次序，这样就可把某些实现细节表示出来
- 在翻译模式中，和文法符号相关的属性和语义规则（这里我们也称语义动作），用花括号{}括起来，插入到产生式右部的合适位置上

E→TR

R→addop T {print(addop.lexeme)} R_1 | ε

T→num {print(num.val)}

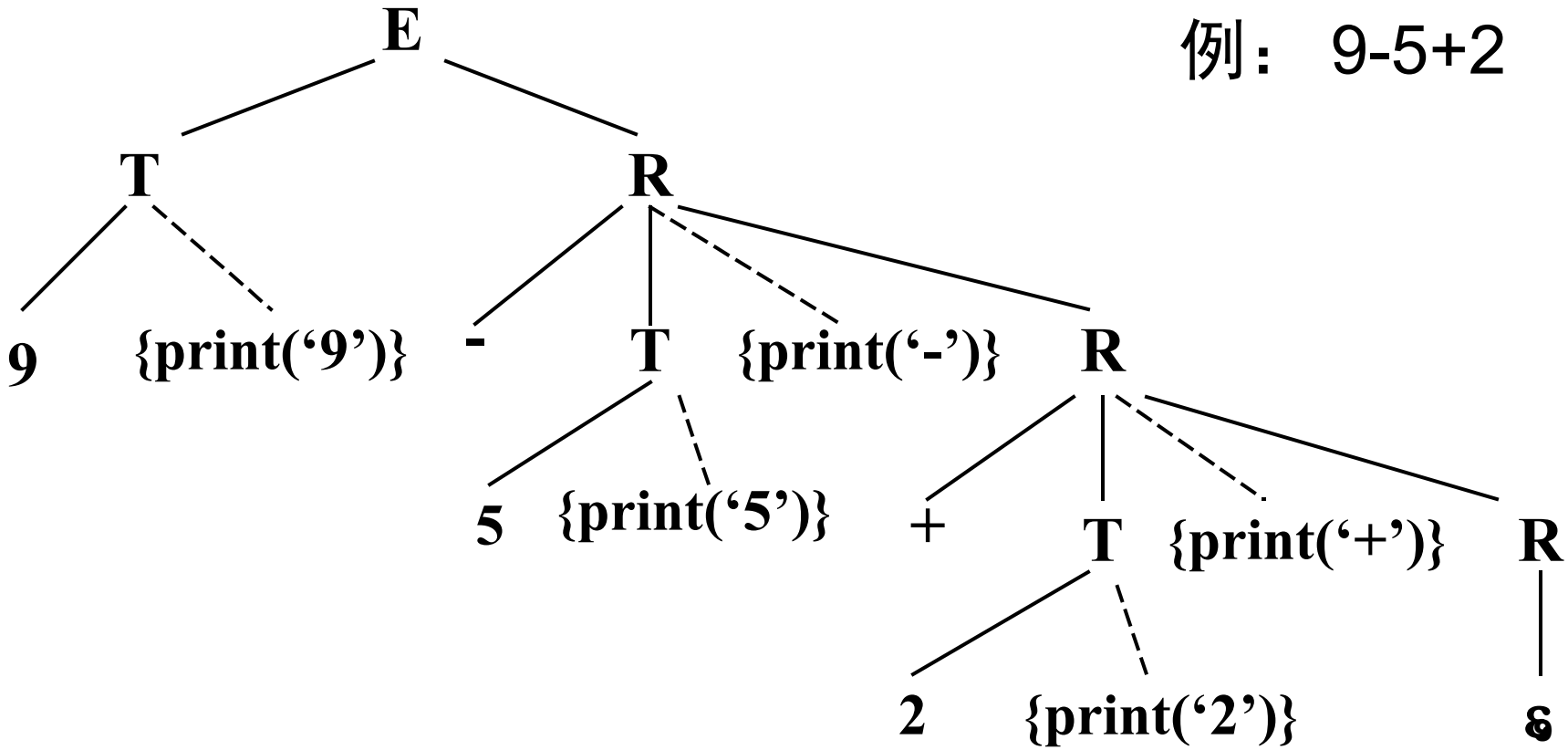
- 翻译模式示例：把带加号和减号的中缀表达式翻译成相应的后缀表达式

E→TR

R→addop T {print(addop.lexeme)} R_1 | ε

T→num {print(num.val)}

例： 9-5+2



设计翻译模式的原则

- 设计翻译模式时，必须保证当某个动作引用一个属性时它必须是有定义的
- L- 属性文法本身就能确保每个动作不会引用尚未计算出来的属性

建立翻译模式

- 当只需要综合属性时：为每一个语义规则建立一个包含赋值的动作，并把这个动作放在相应的产生式右边的末尾

产生式	语义规则
$T \rightarrow T_1 * F$	$T.val := T_1.val \times F.val$

建立产生式和语义动作：

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

建立翻译模式

- 如果既有综合属性又有继承属性，在建立翻译模式时就必须保证：

1. 产生式右边的符号的继承属性必须在这个符号以前的动作中计算出来
2. 一个动作不能引用这个动作右边的符号的综合属性
3. 产生式左边非终结符的综合属性只有在它所引用的所有属性都计算出来以后才能计算。计算这种属性的动作通常可放在产生式右端的末尾

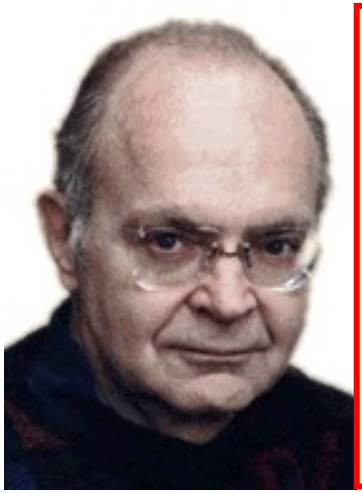
$S \rightarrow A_1 A_2$  $\{A_1.in:=1;\ A_2.in:=2\}$

$A \rightarrow a$ $\{print(A.in)\}$

排版软件 TeX 、 LaTeX

The quadratic formula is $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

The quadratic formula is ${-b\pm\sqrt{b^2-4ac} \over {2a}}$



For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

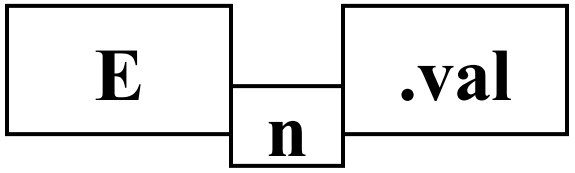
Donald Ervin Knuth

基于数学格式语言 EQN

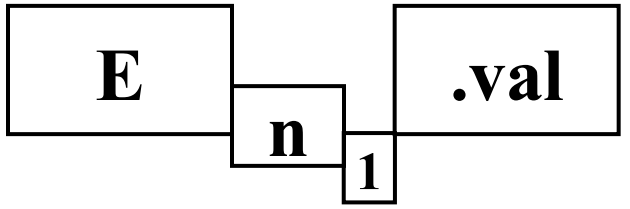
$$E_n.val$$

- 给定输入

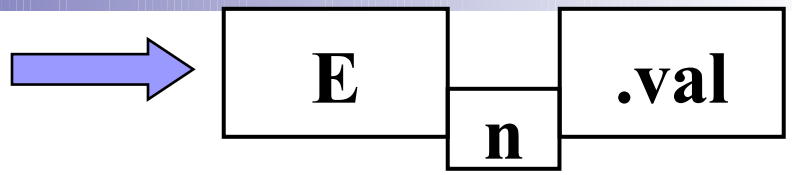
E sub n .val



E sub n sub 1.val



E sub n .val



识别输入并进行格式安放的 L- 属性文法

产生式	语义规则
$S \rightarrow B$	$B.ps := 10$
	$S.ht := B.ht$
$B \rightarrow B_1 B_2$	$B_1.ps := B.ps$
	$B_2.ps := B.ps$
	$B.ht := \max(B_1.ht, B_2.ht)$
$B \rightarrow B_1$ sub B_2	$B_1.ps := B.ps$
	$B_2.ps := shrink(B.ps)$
	$B.ht := disp(B_1.ht, B_2.ht)$
$B \rightarrow text$	$B.ht := text.h \times B.ps$

翻译模式

产生式	语义规则
$S \rightarrow B$	$B.ps := 10$
	$S.ht := B.ht$
$B \rightarrow B_1 B_2$	$B_1.ps := B.ps$
	$B_2.ps := B.ps$
	$B.ht := \max(B_1.ht, B_2.ht)$
$B \rightarrow B_1$ sub B_2	$B_1.ps := B.ps$
	$B_2.ps := shrink(B.ps)$
	$B.ht := disp(B_1.ht, B_2.ht)$
$B \rightarrow text$	$B.ht := text.h \times B.ps$

$S \rightarrow$ {B.ps:=10}
B {S.ht:=B.ht}

$B \rightarrow$ {B_1.ps:=B.ps}
B_1 { B_2.ps:=B.ps }
B_2 {B.ht:=max(B_1.ht,B_2.ht)}

$B \rightarrow$ {B_1.ps:=B.ps}
B_1
sub {B_2.ps:=shrink(B.ps)}
B_2 {B.ht:=disp(B_1.ht,B_2.ht)}

$B \rightarrow$ text {B.ht:=text.h×B.ps}

1. 产生式右边符号的继承属性必须在该符号以前的动作中计算出来
2. 一个动作不能引用该动作右边符号综合属性
3. 产生式左边非终结符的综合属性只有在它所引用的所有属性都计算出来后才能计算。

建立翻译模式

- 把所有的语义动作都放在产生式的末尾
  - 语义动作的执行时机统一
- 转换方法
  - 加入新的产生式 M→ε
  - 把嵌入在产生式中的每个语义动作用不同的标记非终结符 M 代替，并把这个动作放在产生式 M→ε 的末尾

- 翻译模式

  E → T R

  R → + T { print (‘ + ’) } R

  | – T { print (‘ – ’) } R

  | ε

  T → num { print (num.val) }

- 转换为

  E → TR

  R → + T M R | – T N R | ε

  T → num { print (num.val) }

  M → ε { print (‘ + ’) }

  N → ε { print (‘ – ’) }

小结

- L- 属性文法
- 翻译模式

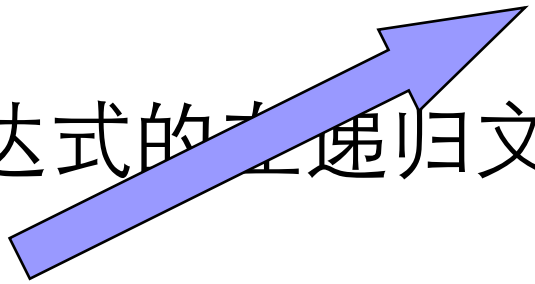
第六章 属性文法和语法制导翻译

- 属性文法
- 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

6.4.2 自顶向下翻译

- 动作是在处于相同位置上的符号被展开（匹配成功）时执行的
- 为了构造不带回溯的自顶向下语法分析，必须消除文法中的左递归
- 当消除一个翻译模式的基本文法的左递归时同时考虑属性——适合带综合属性的翻译模式

- 关于算术表达式的左递归文法模式



$E \rightarrow T\ R$
$R \rightarrow +\ T\ R_1$
$R \rightarrow -\ T\ R_1$
$R \rightarrow \varepsilon$
$T \rightarrow (\ E\ )$
$T \rightarrow num$

$E \rightarrow E_1+T$	$\{E.val:=E_1.val+T.val\}$
$E \rightarrow E_1-T$	$\{E.val:=E_1.val-T.val\}$
$E \rightarrow T$	$\{E.val:=T.val\}$
$T \rightarrow (E)$	$\{T.val:=E.val\}$
$T \rightarrow num$	$\{T.val:=num.val\}$

- 消除左递归，构造新的翻译模式

$E \rightarrow T$ {R.i:=T.val}
R {E.val:=R.s}

$R \rightarrow +$
T {R_1.i:=R.i+T.val}
R_1 {R.s:=R_1.s}

$R \rightarrow -$
T {R_1.i:=R.i-T.val}
R_1 {R.s:=R_1.s}

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

$T \rightarrow (E)$ {T.val:=E.val}

$T \rightarrow$ num {T.val:=num.val}

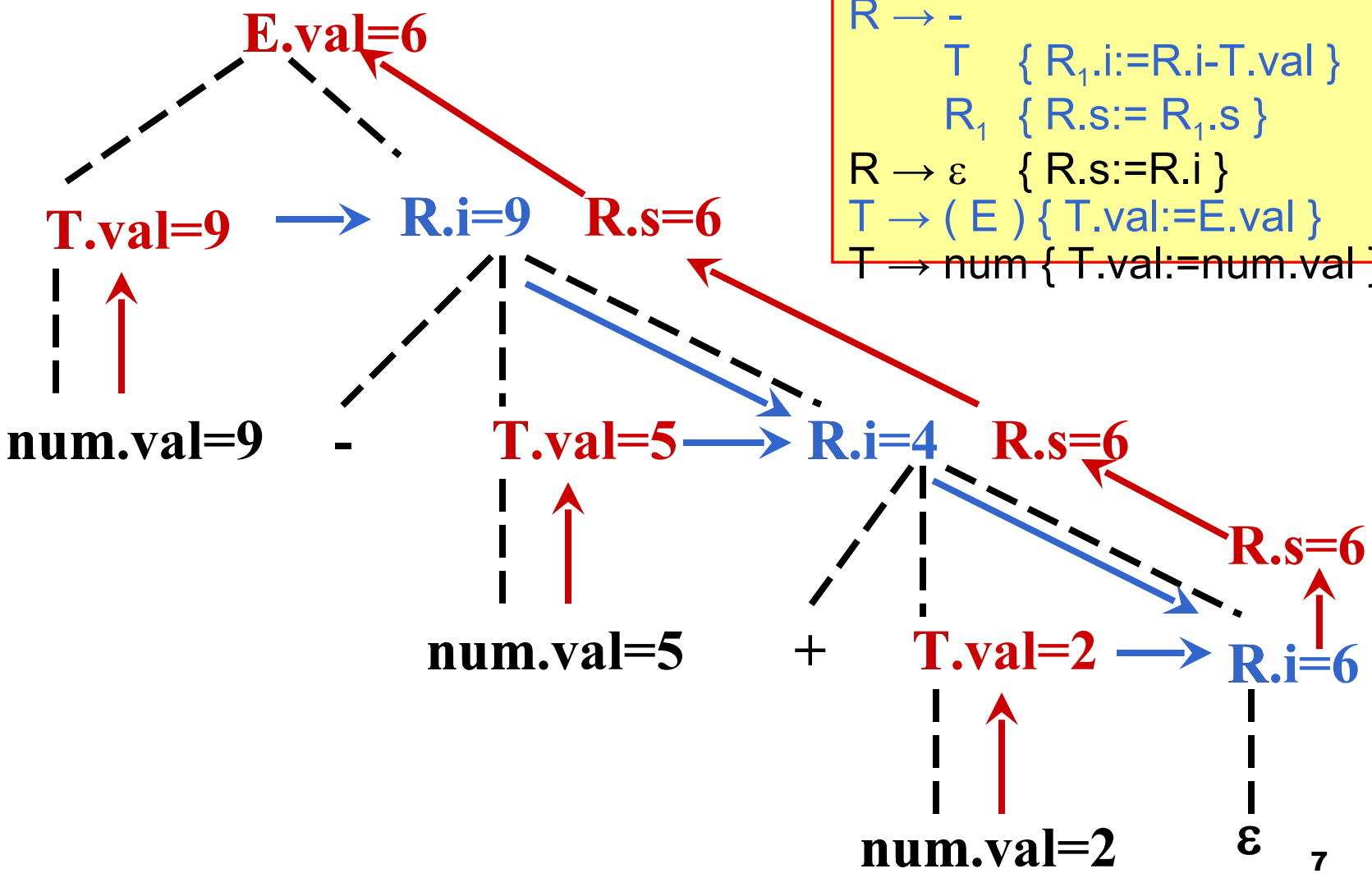
$E \rightarrow T\ R$
$R \rightarrow +TR_1$
$R \rightarrow -TR_1$
$R \rightarrow \varepsilon$
$T \rightarrow (E)$
$T \rightarrow$ num

R.i: R 前面子表达式的值

R.s: 分析完 R 时子表达式的值

计算表达式 9 – 5 + 2

$E \rightarrow T \quad \{ R.i := T.val \}$
$\quad\quad R \quad \{ E.val := R.s \}$
$R \rightarrow +$
$\quad\quad T \quad \{ R_1.i := R.i + T.val \}$
$\quad\quad R_1 \quad \{ R.s := R_1.s \}$
$R \rightarrow -$
$\quad\quad T \quad \{ R_1.i := R.i - T.val \}$
$\quad\quad R_1 \quad \{ R.s := R_1.s \}$
$R \rightarrow \varepsilon \quad \{ R.s := R.i \}$
$T \rightarrow (E) \quad \{ T.val := E.val \}$
$T \rightarrow num \quad \{ T.val := num.val \}$



一般方法

- 假设有翻译模式：

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

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

  它的每个文法符号都有一个综合属性，用小写字母表示，g 和 f 是任意函数。

- 消除左递归：

  $A \rightarrow XR$

  $R \rightarrow YR \mid \varepsilon$

R.i: R 前面子表达式的值

R.s: 分析完 R 时子表达式的值

- 翻译模式变为：

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

  R {A.a:=R.s}

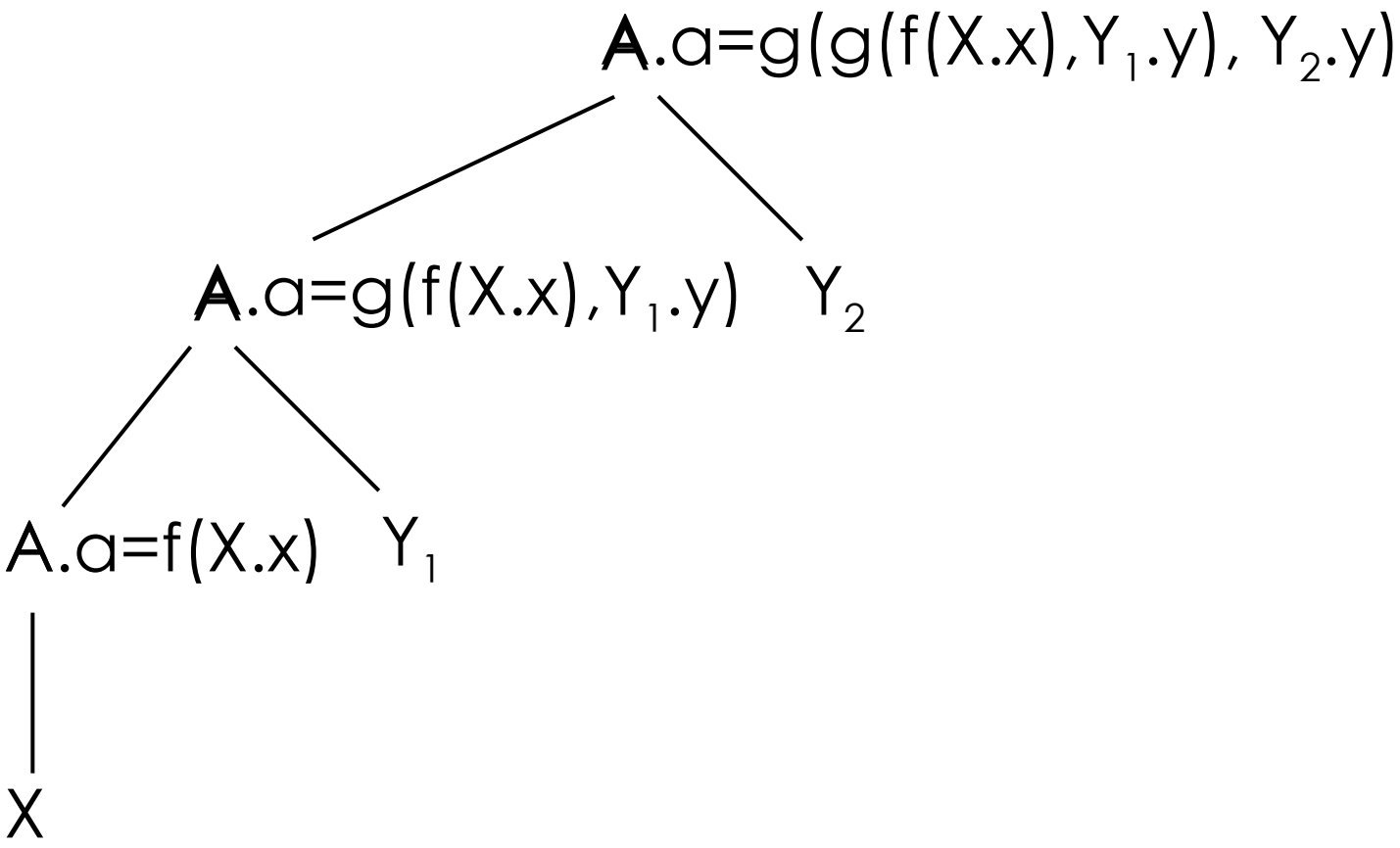
  $R \rightarrow Y$ {R_1.i:=g(R.i, Y.y) }

  R_1 {R.s:=R_1.s}

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

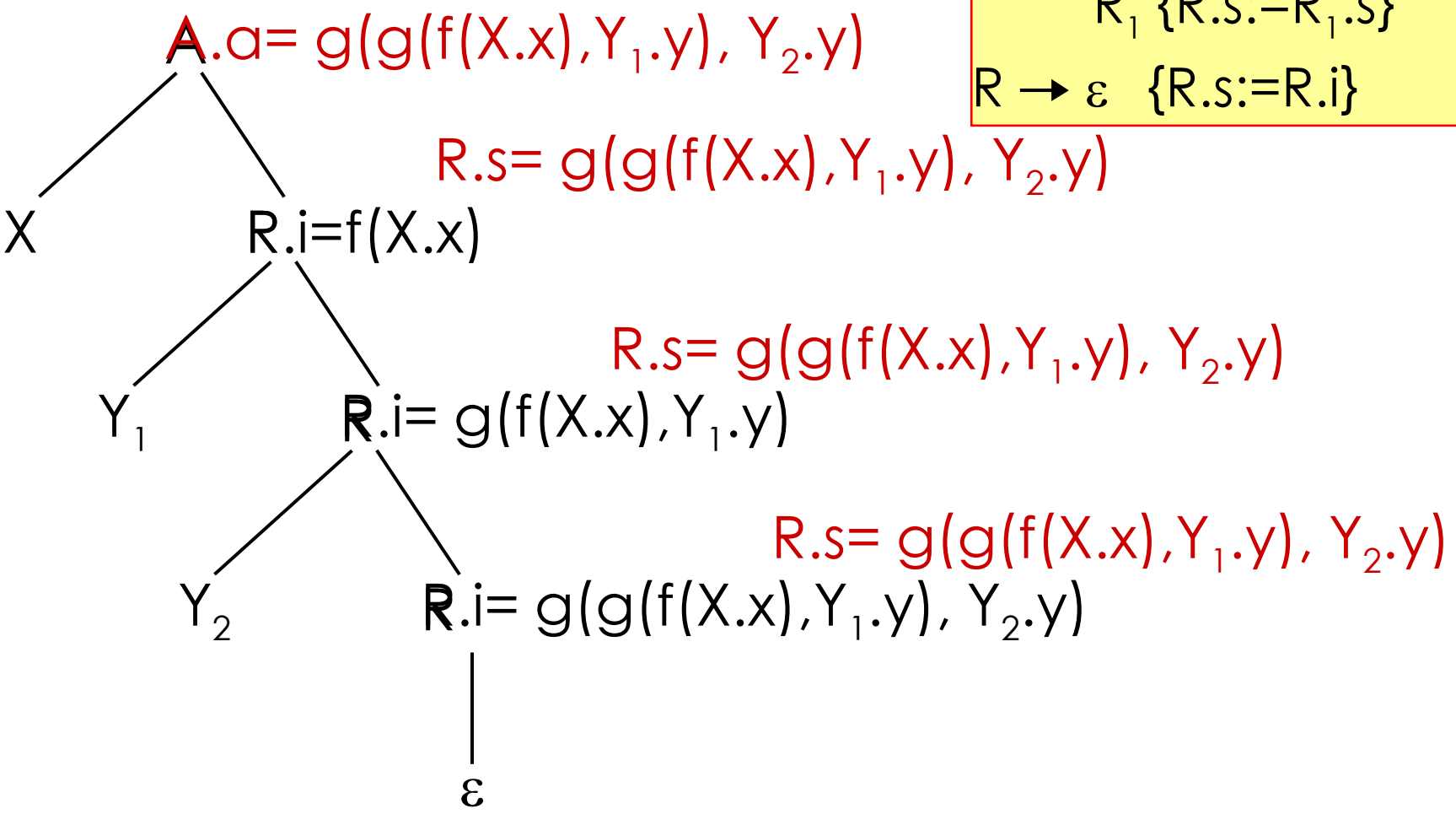
XYY

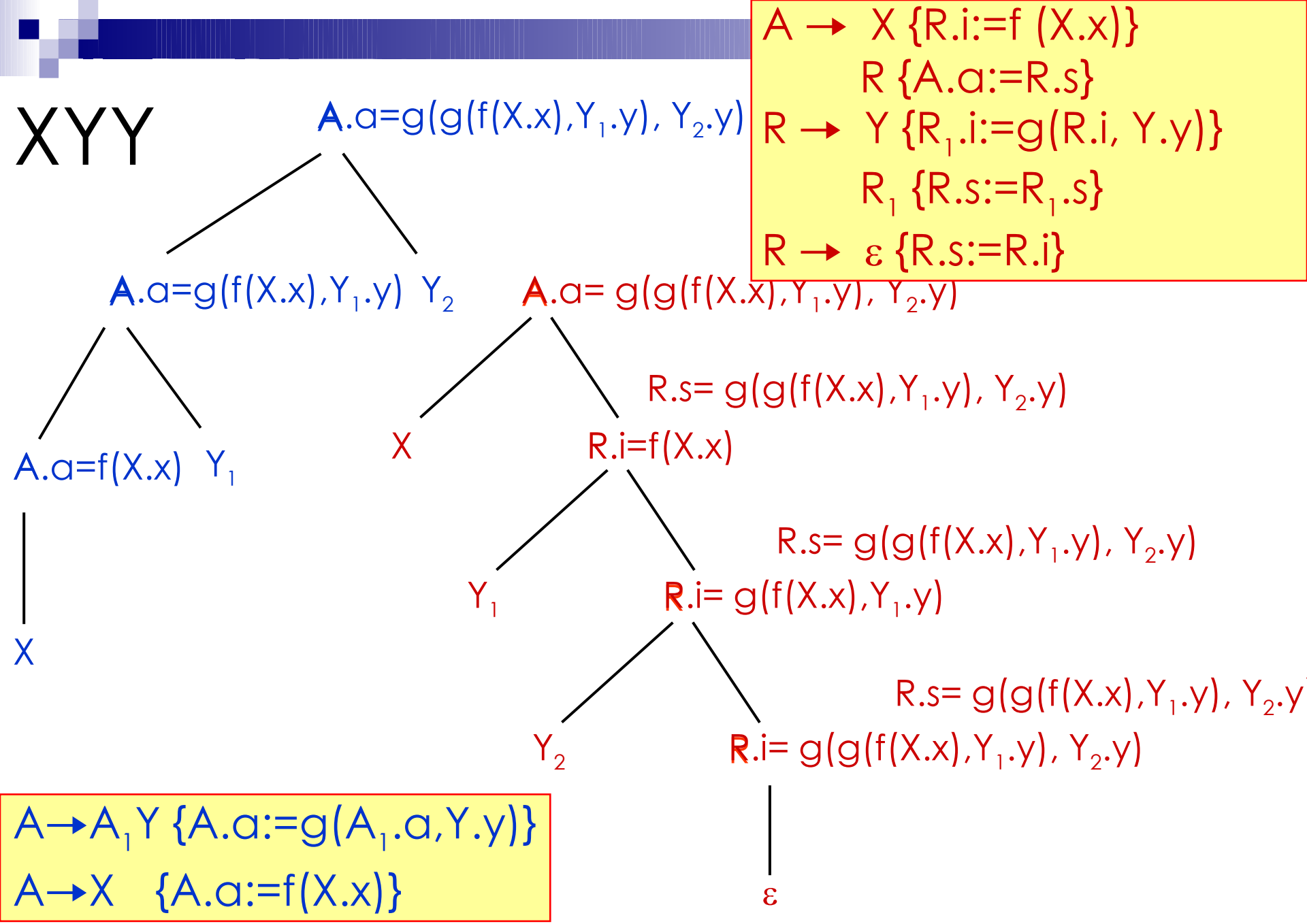
$A \rightarrow A_1 Y$	$\{A.a := g(A_1.a, Y.y)\}$
$A \rightarrow X$	$\{A.a := f(X.x)\}$



XYY

$A \rightarrow X \ \{R.i := f(X.x)\}$
$\quad R \ \{A.a := R.s\}$
$R \rightarrow Y \ \{R_1.i := g(R.i, Y.y)\}$
$\quad R_1 \ \{R.s := R_1.s\}$
$R \rightarrow \varepsilon \ \{R.s := R.i\}$




XYY
A.a=g(g(f(X.x),Y1.y), Y2.y)
A.a=g(f(X.x),Y1.y) Y2
A.a= g(g(f(X.x),Y1.y), Y2.y)
A.a=f(X.x) Y1
X
R.s= g(g(f(X.x),Y1.y), Y2.y)
X
R.i=f(X.x)
Y1
R.s= g(g(f(X.x),Y1.y), Y2.y)
R.i= g(f(X.x),Y1.y)
Y2
R.s= g(g(f(X.x),Y1.y), Y2.y
R.i= g(g(f(X.x),Y1.y), Y2.y)
ε
A → X {R.i:=f (X.x)}
R {A.a:=R.s}
R → Y {R1.i:=g(R.i, Y.y)}
R1 {R.s:=R1.s}
R → ε {R.s:=R.i}
A→A1Y {A.a:=g(A1.a,Y.y)}
A→X {A.a:=f(X.x)}

构造抽象语法树的属性文法定义转化成翻译模式

$E \rightarrow E_1+T$ {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}

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

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

$A \rightarrow X$ {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}

构造抽象语法树的属性文法定义转化成翻译模式

E →	T	{R.i:=T.nptr}
	R	{E.nptr:=R.s}
R →	+	
	T	{R_1.i:=mknode(‘+’ , R.i, T.nptr)}
	R_1	{R.s:=R_1.s}
R →	-	
	T	{R_1.i:=mknode(‘ – ’ , R.i, T.nptr)}
	R_1	{R.s:=R_1.s}
R →	ε	{R.s:=R.i}
T →	(E)	{T.nptr:=E.nptr}
T →	id	{T.nptr:=mkleaf(id, id.entry)}
T →	num	{T.nptr:=mkleaf(num, num.val)}

使用继承属性构造 a – 4 + c 的抽象语法

$E \rightarrow T\ \{R.i:=T.nptr\}\ R\ \{E.nptr:=R.s\}$

$R \rightarrow +\ T\ \{R_1.i:=mknode('+',R.i,T.nptr)\}\ R_1\ \{R.s:=R_1.s\}$

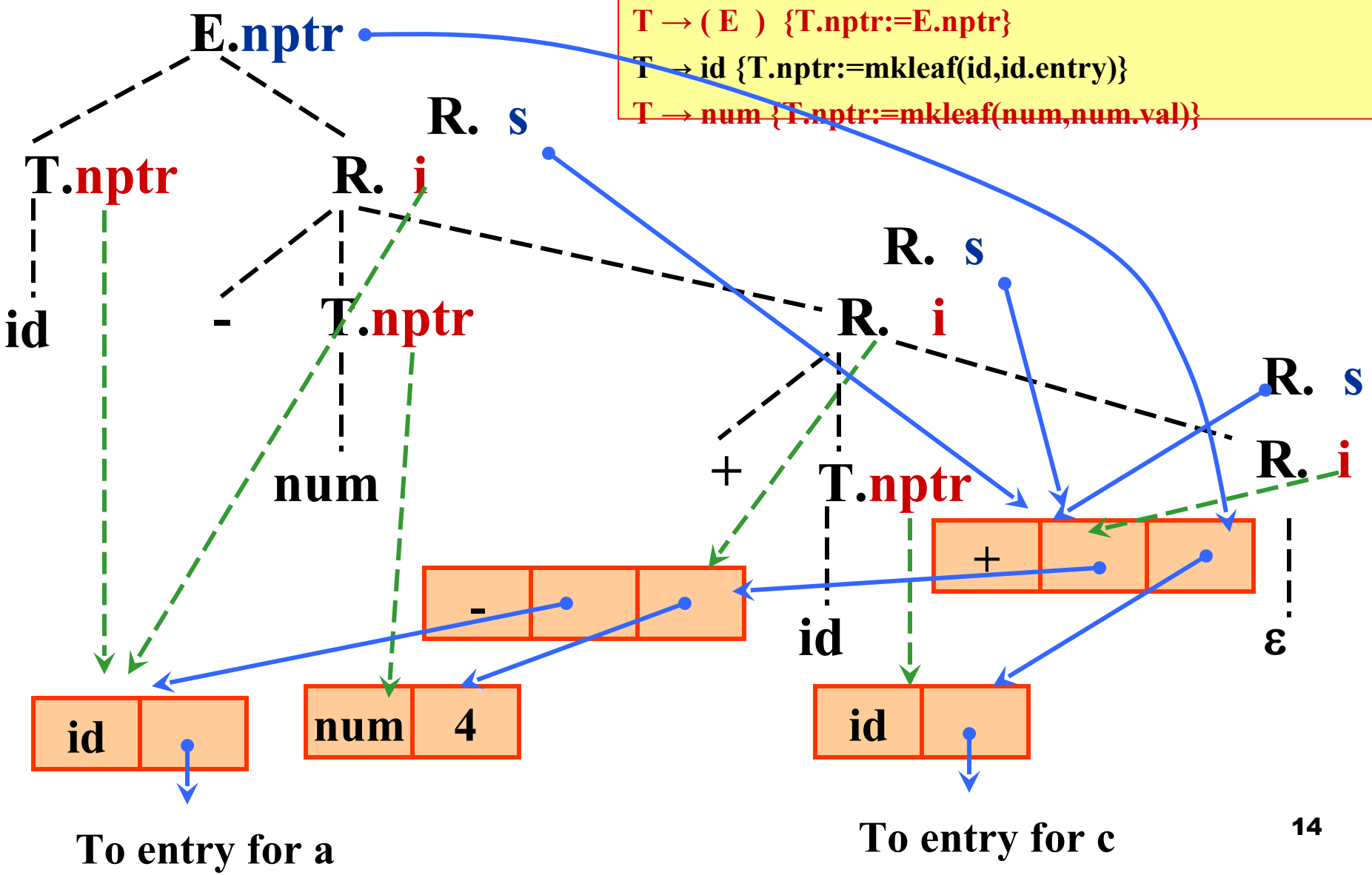
$R \rightarrow -\ T\ \{R_1.i:=mknode('-',R.i,T.nptr)\}\ R_1\ \{R.s:=R_1.s\}$

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

$T \rightarrow (\ E\ )\ \{T.nptr:=E.nptr\}$

$T \rightarrow id\ \{T.nptr:=mkleaf(id,id.entry)\}$

$T \rightarrow num\ \{T.nptr:=mkleaf(num,num.val)\}$



6.4.3 递归下降翻译器的设计

- 递归下降分析器的设计
  - 分析程序由一组递归过程组成，文法中每个非终结符对应一个过程；所以这样的分析程序称为递归下降分析器

R→addop TR|ε 的递归下降分析过程

```
procedure R;
 begin
   if sym=addop then begin
        advance;T; R
      end
   else begin /*do nothing*/
      end
 end;
```

如何在递归下降分析中实现翻译模式，构造递归下降翻译器?

R → addop
T {R_1.i:= mknode(addop.lexme, R.i, T.nptr)}
R_1 {R.s:=R_1.s}
R→ ε {R.s:=R.i}

设计递归下降翻译器的方法

- 定义参数和变量
  - 对每个非终结符 A 构造一个函数过程，对 A 的每个继承属性设置一个形式参数
  - 函数的返回值为 A 的综合属性
    - 作为记录，或指向记录的一个指针，记录中有若干域，每个属性对应一个域）
    - 为了简单，我们假设每个非终结只有一个综合属性
  - A 对应的函数过程中，为出现在 A 的产生式中的每一个文法符号的每一个属性都设置一个局部变量

设计递归下降翻译器的方法

- 函数的功能
  - 非终结符 A 对应的函数过程中，根据当前的输入符号决定使用哪个产生式候选

设计递归下降翻译器的方法

- 每个产生式对应的程序代码
  - 按照从左到右的次序，对于单词符号（终结符）、非终结符和语义动作分别作以下工作
    - 对于带有综合属性 x 的终结符 X，把 x 的值存入为 X.x 设置的变量中。然后产生一个匹配 X 的调用，并继续读入一个输入符号。
    - 对于每个非终结符 B，产生一个右边带有函数调用的赋值语句 c=B(b1,b2,…,bk)，其中，b1,b2,…,bk 是为 B 的继承属性设置的变量，c 是为 B 的综合属性设置的变量。
    - 对于语义动作，把动作的代码抄进分析器中，用代表属性的变量来代替对属性的每一次引用。

构造抽象语法树的属性文法定义转化成翻译模式

E →	T	{R.i:=T.nptr}
	R	{E.nptr:=R.s}
R →	+	
	T	{R_1.i:=mknode(‘+’ , R.i, T.nptr)}
	R_1	{R.s:=R_1.s}
R →	-	
	T	{R_1.i:=mknode(‘ – ’ ,R.i, T.nptr)}
	R_1	{R.s:=R.s}
R →	ε	{R.s:=R.i}
T →	(E)	{T.nptr:=E.nptr}
T →	id	{T.nptr:=mkleaf(id, id.entry)}
T →	num	{T.nptr:=mkleaf(num, num.val)}

- 非终结符 E 、 R 、 T 的函数及其参数类型

  function E:↑AST-node;

  function R(in:↑AST-node): ↑AST-node;

  function T: ↑AST-node;

- 用 addop 代表 + 和 –

R → addop

T {R_1.i := mknode(addop.lexme, R.i,T.nptr) }

R_1 {R.s:=R_1.s}

R→ ε { R.s:=R.i }

产生式 R→addop TR|ε 的分析过程

```
procedure R;
 begin
   if sym=addop then begin
         advance;T; R
       end
    else begin /*do nothing*/
       end
end;
```

R → addop
T {R_1.i:= mknode(addop.lexme, R.i, T.nptr)}
R_1 {R.s:=R_1.s}
R→ ε {R.s:=R.i}

```
function R (in:↑AST-node): ↑AST-node;
  var nptr, i1,s1,s: ↑AST-node;
  addoplexeme: char;
begin
   if sym=addop then begin
        /* 产生式       R→addop TR */
        addoplexeme:=lexval;
        advance;
        nptr:=T;
        i1:=mknode (addoplexeme, in, nptr);
        s1:=R (i1)
        s:=s1
    end
   else  s:= in;
   return  s
end;
```

小结

- 翻译模式的改造——消除左递归
- 自顶向下翻译——构造递归下降翻译器

作业

- P165 - 11， 12 （选作）

第六章 属性文法和语法制导翻译

- 属性文法与翻译模式
- 基于属性文法的的处理方法
  - 依赖图
  - 树遍历
  - 一遍扫描
- 抽象语法树
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

1.给出下列表达式的逆波兰表示（后缀式）：

a+b*c

a≤b+c∧a>d∨a+b≠e

1　写出下列语句的逆波兰表示（后缀式）：

<变量>：= <表达式>

IF <表达式> THEN <语句 1>ELSE<语句 2>

2　写出算术表达式：

A+B*（C-D）+E/（C-D）**N

的四元式，三元式和间接三元式序列。

（中国科学院软件研究所 1996 年硕士生入学考试试题）

解题思路：

把中缀式转换后缀式的简单方法：按中缀式中各运算符的优先规则，从最先执行的部分开始写，一层层套。如对 a≤b+c∧a>d∨a+b≠e ,先把 b+c 写为 bc+，然后把 a≤套上去，成为 abc+≤；再把 a>d 表示为 ad>，然后把∧套上去，成为 abc+≤ad>∧，依此类推。

四元式的由 4 个部分组成：算符 op、第一和第二运算量 arg1 和 arg2，以及运算结果 result。运算量和运算结果有时指用户自定义的变量，有时指编译程序引进的临时变量。如果 op 是一个算术或逻辑算符，则 result 总是一个新引进的临时变量，用于存放运算结果。

三元式只需三个域：op、arg1 和 arg2。与四元式相比，三元式避免了临时变量的填入，而是通过计算这个临时变量值的语句的位置来引用这个临时变量。我们很容易把一个算术表达式或一个赋值句表示为四元式序列或三元式序列。

间接三元式是指用一张间接码表辅以三元式表的办法来表示中间代码。间接码表按运算的先后顺序列出有关三元式在三元表中的位置，对于相同的三元式无需重复出现在三元表中。

解答

①

a+b*c 的后缀式为　　abc*+

a≤b+c∧a>d∨a+b≠e 的后缀式为　abc+≤ad>∧ab+e≠∨

②

赋值语句<变量>：= <表达式>的逆波兰表示为：

<变量> <表达式> :=

条件语句 IF <表达式> THEN <语句 1>ELSE<语句 2>的逆波兰表示为：

<表达式> <语句 1> <语句 2> IF

或

<表达式> P_1 jez <语句 1> P_2 j P_1:<语句 2> P_2:

其中：jez 是<表达式>和 P_1 这两个运算对象的二元运算符，表示当<表达式>等于 0(即取假值时)转去执行由 P_1 标领的<语句 2>；否则，执行<语句 1>。j 是无条件转移的一元运算符，P_2 j 表示无条件转至 P_2 所指地方。

3　四元式序列：

(1) (-,C,D,T_1)
(2) (*,B,T_1,T_2)
(3) (+,A,T_2,T_3)
(4) (-C,D,T_4)
(5) (**,T_4,N,T_5)
(6) (/,E,T_5,T_6)
(7) (+,T_3,T_6,T_7)

三元式序列：

(1) (-,C,D)
(2) (*,B,(1))
(3) (+,A,(2))
(4) (-,C,D)
(5) (**,(4),N)
(6) (/,E,(5))
(7) (+,(3),(6))

间接三元式序列：

三元式表	间接码表
(1) (-,C,D)	(1)
(2) (*,B,(1))	(2)
(3) (+,A,(2))	(3)
(4) (**,(1),N)	(1)
(5) (/,E,(4))	(4)
(6) (+,(3),(5))	(5)
	(6)

例题 6.3.1 设下列文法生成变量的类型说明：

D→id L

L→, id L | : T

T→integer | real

(1) 构造一个翻译模式，把每个标识符的类型存入符号表；

(2) 由(1)得到的翻译模式，构造一个预测翻译器。

解题思路: 这是一个对说明语句进行语义分析的题目, 不需要产生代码, 但要求把每个标识符的类型填入符号表中。

对给定的适合于自顶向下翻译的翻译模式，第 5.1.4 节给出了设计预测翻译器（或称递归下降翻译器）的方法。按照此方法，不难构造所求的预测翻译器。

解答:

对 D,L,T 设置综合属性 type。过程 addtype(id, type)用来把标识符 id 及其类型 type 填入到符号表中。

（1）翻译模式如下:

D→id L {addtype(id.entry, L.type)}

L→,id L1 { addtype(id.entry, L1.type); L.type:=L1.type;}

L→:T {L.type:=T.type}

T→integer {T.type:=integer}

T→real {T.type:=real}

（2）假设 Ttype 为已定义的表示“类型”的数据结构，预测翻译器如下:

```
procedure D;
  var l_type:Ttype
begin
     if sym= “id” then
        begin
          advance;
```

```
            l_type:=L;
            addtype(id.entry, l_type)
          end
       else error
end;
procedure L;
       var l_type:Ttype;
begin
       if sym= “,” then
          begin
            advance;
            if sym= “id” then
              begin
                advance;
                l_type:=L;
                addtype(id.entry, l_type)
              end
                else error;
     end
       else if sym= “:” then
          begin
            advance;
            l_type:=T;
          end
```

```
        else error;
        return(l_type);
  end;
  procedure T;
       var t_type:Ttype
      begin
         if sym= “integer” then
            begin
              advance;
              t_type:=integer;
            end
        else if sym= “real” then
            begin
              advance;
              t_type:=real;
            end
        else error;
        return(t_type);
  end;
```

例题 6.3.2 按照 EQN 语言的功能，给出识别输入并进行格式安放的 L-属性文法。

解题思路：

文法中， 非终结符 B（表示盒子）代表一个公式， 产生式 B→BB 代表两个盒子并置，B→B1 sub B2 代表 B2 的大小比 B1 的小，并且放在下角标的位置。继承属性 ps 将影响公式的高度。产生式 B→text 对应的语义规则使得 text 的标准高度乘以 ps 得到 text 的实际高度。

关于 text 的属性 h 通过查表获得，由单词 text 所表示的字符给出。当应用产生式 B→B1B2 时，B1 和 B2 通过复写规则继承 ps 的值。综合属性 B.ht 代表 B 的高度，它取 B1.ht 和 B2.ht 的最大值。

当使用产生式 B→B1 sub B2 时，函数 shrink 使 B2.ps 减少 30％。函数 disp 把盒子 B2 向下放置，并计算 B 的高度。在这里，产生实际排字命令的规则没有给出来。

文法中，唯一的继承属性是非终结符 B 的 ps 属性，每条定义 ps 属性的语义规则只依赖于产生式左边非终结符的继承属性。因此它是一个 L 属性文法。

解答:

所求 L-属性文法如下:

产 生 式	语 义 规 则
S→B	B.ps :=10
	S.ht :=B.ht
B→B1B2	B1.ps :=B.ps
	B2.ps :=B.ps
	B.ht := max(B1.ht, B2.ht)
B→B1 sub B2	B1.ps :=B.ps
	B2.ps :=shrink(B.ps)
	B.ht :=disp(B1.ht, B2.ht)
B→text	B.ht :=text.h×B.ps

一个属性文法称为 **L-属性文法**，如果对于每个产生式 $A \rightarrow X_1X_2 \ldots X_n$，其每个语义规则中的每个属性或者是综合属性，或者是 $X_j(1 \leq j \leq n)$的一个继承属性且这个继承属性仅依赖于：

(1) 产生式中 X_j 左边符号 X_1，X_2，…，X_{j-1} 的属性

(2) A 的继承属性

1. 中缀逻辑表达式-a*(b+c)<d+e AND t。

(1). 写出等价的逆波兰后缀表示；

(2). 写出四元式表示。

答案：

(1)a@bc+*de+<tAND

(2)

(@,a,-,T1)

(+,b,c,T2)

(*,T1,T2,T3)

(+,d,e,T4)

(<,T3,T4,T5)

(AND,T5,t,T6)

2.有如下表达式

-(a+(b-c))+d

分别写出其等价的逆波兰表示（后缀式）和四元式表示。

答案：

逆波兰表示为 abc-+@d+

四元式表示为:

(1) (-,b, c, T1)

(2) (+, a, T1,T2)

(3) (-, T2, -,T3)

(4) (+, T3, d,T4)

3. 将下列语句翻译为逆波兰表示（后缀式），三元式和间接三元式序列和四元式表示：

a:=(b+c)*e+(b+c)/f

答案：

逆波兰表示为：bc+e*bc+f/+:=

三元式序列为:

(1) (+, b, c)

(2) (*,(1),e)

(3) (+, b, c)

(4) (/,(3),f)

(5) (+,(2),(4))

(6) (:=,a,(5))

间接三元式表示为：

三元式表	间接码表
(1) (+, b, c)	(1)
(2) (*,□, e)	(2)
(3) (/,□, f)	(1)
(4) (+,□,□)	(3)
(5) (:=,a,□)	(4)
	(5)

四元式表示为:

(1) (+, b, c, T1)
(2) (*, T1,e, T2)
(3) (+, b, c, T3)
(4) (/, T3, f, T4)
(5) (+, T2,T4, T5)
(6) (:=,T5, -, a)

4. 现有文法 G_1、G_2 如下：欲将 G_1 定义的 expression 转换如 G_2 的 E 所描述的形式。给出其语法制导翻译的语义描述。（提示：可采用类似 yacc 源程序的形式，所涉及的语义函数须用自然语言给予说明，不用抄写产生式，用产生式编号表示。）

G_1

⑴ <PROGRAM>→<decl statement>;BEGIN<statement list>END
⑵<decl statement>→VAR<iddecl>
⑶<iddecl>→<iddecl>,id:<type decl>
⑷<iddecl>→id:<type decl>
⑸<type decl>→int
⑹<type decl>→bool
⑺<statement list>→<expression>
⑻<statement list>→<statement list>;<expression>
⑼<expression>→<expression>AND<expression>
⑽<expression>→<expression>*<expression>
⑾<expression>→id
⑿<expression>→NUM
⒀<expression>→true
⒁<expression>→false

要求：⑴ <expression>中的 id 必须在<decl statement>中先声明。
⑵ AND 和*分号是常规的布尔和算术运算符，要求其运算对象的相应类型匹配。

G_2： E→EE* | EEand | id | num | true | false

答案：

从题目分析, 翻译的关键是把 G1 描述的中缀形式的表达式转换为 G2 描述的后缀形式的表达式, 并要进行一定的类型检查。 在下面的语义动作中, 采用类似 yacc 源程序的形式, $$代表相应产生式的左部符号, $1 代表产生式右部的第一个文法符号, $2 代表产生式右部的第二个文法符号, 依此类推。 在下面语义动作中, lookup(name)的功能是对 name 查找符号表并返回其类型,如果查不到则报错; lexeme(token)给出 token 的词法值。

⑴ <PROGRAM>→<decl statement>;BEGIN<statement list>END
{ $$.code := $4.code;}
⑵<decl statement>→VAR<iddecl>
{ }
⑶<iddecl>→<iddecl>,id:<type decl>
{ enterid($3,$5.type);}
⑷<iddecl>→id:<type decl>
{ enterid($1,$3.type);}

⑸<type decl>→int
{ $$.type := int ;}
⑹<type decl>→bool
{ $$.type := bool;}
⑺<statement list>→<expression>
{ $$.code := $1.code;}
⑻<statement list>→<statement list>;<expression>
{ $$.code := $1.code ‖ $3.code;}
⑼<expression>→<expression>AND<expression>
{ if $1.type = bool and $3.type = bool
then $$.type := bool
else typeerror;
$$.code := $1.code ‖$3.code ‖'and' ; }
⑽<expression>→<expression>*<expression>
{ if $1.type = int and $3.type = int
then $$.type := int
else typeerror;
$$.code := $1.code ‖$3.code ‖'*' ; }
⑾<expression>→id
{ $$.type := lookup($1);
$$.code := lexeme($1); }
⑿<expression>→NUM
{ $$.type := int;
$$.code := lexeme($1); }
⒀<expression>→true
{ $$.type := bool;
$$.code := lexeme($1); }
⒁<expression>→false
{ $$.type := bool;
$$.code := lexeme($1); }