



编译原理

习题课 (3)

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

属性文法

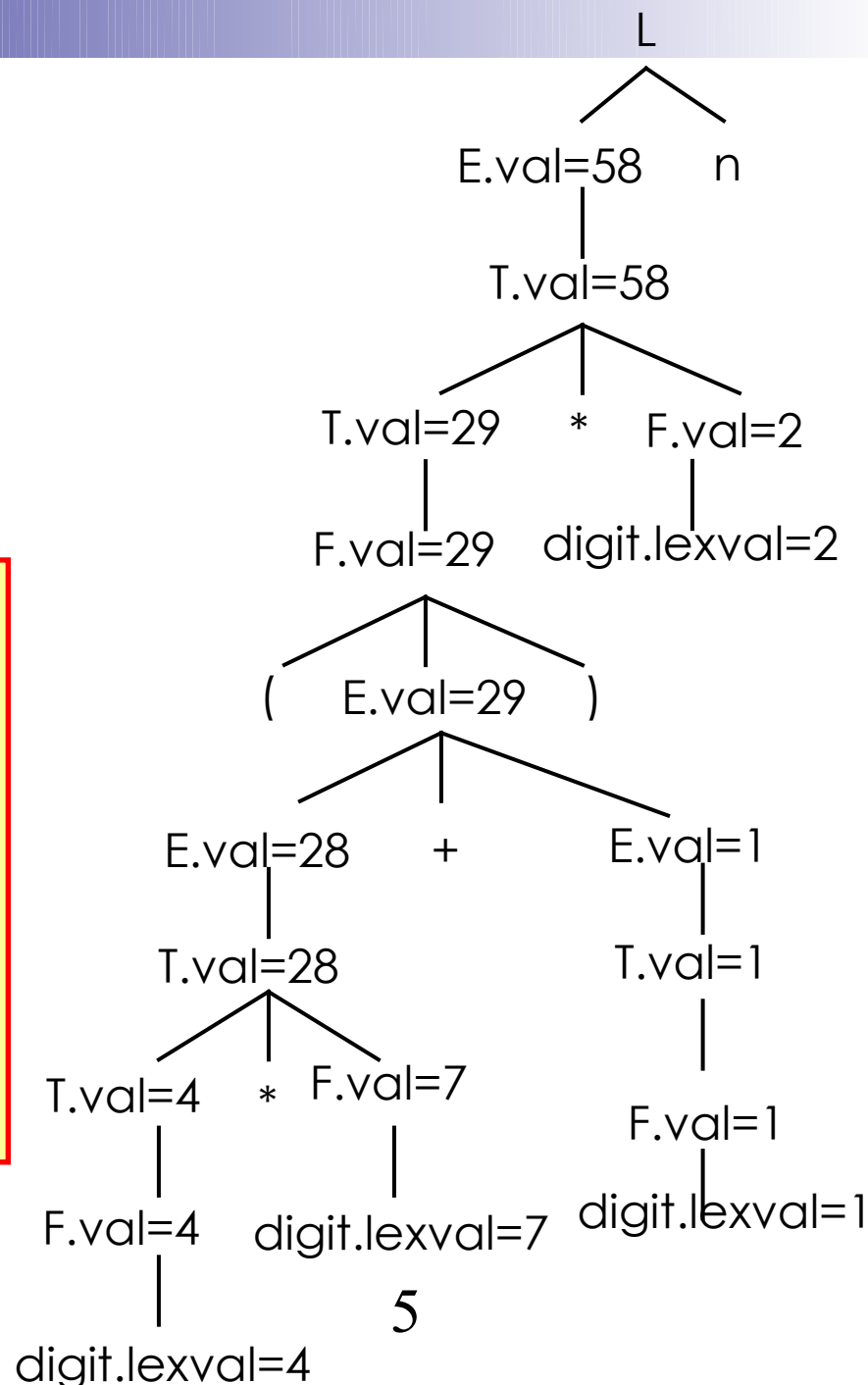
- 在上下文无关文法的基础上，为每个文法符号（终结符或非终结符）配备若干相关的“值”（称为属性）、对于文法的每个产生式都配备了一组属性的计算规则——语义规则
 - 综合属性：“自下而上”传递信息
 - 继承属性：“自上而下”传递信息

基于属性文法的处理方法

- 依赖图
- 树遍历
- 一遍扫描
 - 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

P164-1 按照表 6.1 所示的属性文法，构造表达式 $(4*7+1)*2$ 的附注语法树

产生式	语义规则
$L \rightarrow En$	<code>print(E.val)</code>
$E \rightarrow E_1 + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T_1 * F$	$T.val := T_1.val * F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow \text{digit}$	$F.val := \text{digit.lexval}$



P164-2. 对表达式 $((a)+(b))$

(1) 按照表 6.4 所示的属性
的抽象语法树;

(2) 按照图 6.17 所示的翻
达式的抽象语法树。

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

$$R \{ E.val := R.s \}$$

$$R \rightarrow +$$

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

$$R_1 \{ R.s := R_1.s \}$$

$$R \rightarrow -$$

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

$$R_1 \{ R.s := R_1.s \}$$

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

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

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

产生式

语义规则

$E \rightarrow E1 + T$ $E.nptr := \text{mknode}('+', E1.nptr, T.nptr)$

$E \rightarrow E1 - T$ $E.nptr := \text{mknode}('-', E1.nptr, T.nptr)$

$E \rightarrow T$ $E.nptr := T.nptr$

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

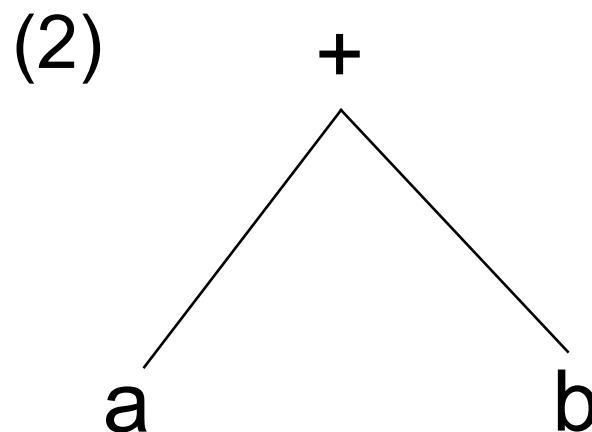
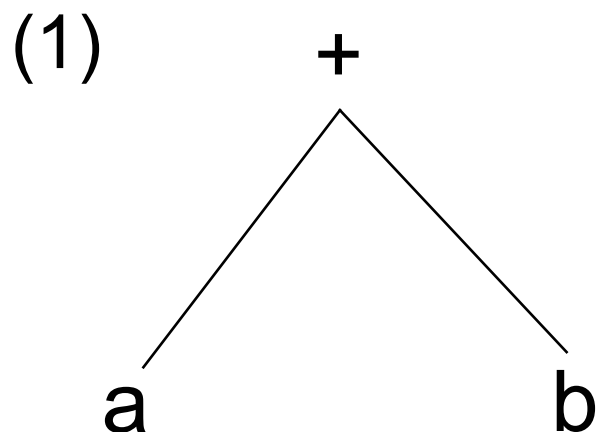
$T \rightarrow \text{id}$ $T.nptr := \text{mkleaf}(\text{id}, \text{id.entry})$

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

P164-2. 对表达式 $((a)+(b))$:

(1) 按照表 6.4 所示的属性文法构造该表达式的抽象语法树;

(2) 按照图 6.17 所示的翻译模式, 构造该表达式的抽象语法树。



P165-5. 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：

$$E \rightarrow E+T \mid T$$

$$T \rightarrow \text{num.num} \mid \text{num}$$

(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 `inttoreal num` 和 `num` 运算符的两个操作数具有相同的类型。
■ 思路：对 `E` 和 `T` 设置综合属性 `type`，表表达式的类型。

P165-5. 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：

$$E \rightarrow E + T \mid T$$
$$T \rightarrow \text{num} . \text{ num} \mid \text{num}$$

(1) 试给出确定每个子表达式结果类型的属性文法

$$\begin{array}{ll} \dot{E} \rightarrow E_1 + T & \text{if } (E_1.\text{type} = \text{int}) \text{ and } (T.\text{type} = \text{int}) \\ & \text{then } E.\text{type} := \text{int} \\ & \text{else } E.\text{type} := \text{real} \end{array}$$
$$E \rightarrow T \quad E.\text{type} := T.\text{type}$$
$$T \rightarrow \text{num} . \text{ num} \quad T.\text{type} := \text{real}$$
$$T \rightarrow \text{num} \quad T.\text{type} := \text{int}$$

P165-5. 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：

$$E \rightarrow E + T \mid T$$


$$T \rightarrow \text{num} . \text{ num} \mid \text{num}$$

- (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。

```

E → E1 + T
    if (E1.type = int) and (T.type = int )
    then begin
        E.type := int
        E.code := E1.code || T.code || +
    end
    else if (E1.type = real) and (T.type = real)
    then begin
        E.type := real;
        E.code := E1.code || T.code || +
    End
    else if (E1.type = int)
    then begin
        E.type := real;
        E.code := E1.code || inttoreal || T.code || +
    End
    else begin
        E.type := real;
        E.code := E1.code || T.code || inttoreal || +
    end

```



$E \rightarrow T$

$E.type := T.type;$

$E.code := T.code$

$T \rightarrow num.num$

$T.type := real$

$E.code := num.num$

$T \rightarrow num$

$T.type := int$

$E.code := num$

P165-7. 下列文法由开始符号 S 产生一个二进制数，令综合属性 val 给出该数的值：

$$S \rightarrow L.L \mid L$$

$$L \rightarrow LB \mid B$$

$$B \rightarrow 0 \mid 1$$

试设计求 $S.val$ 的属性文法，其中，已知 B 的综合属性 c ，给出由 B 产生的二进位的结果值。例如，输入 101.101 时， $S.val=5.625$ ，其中第一个二进位的值是 4，最后一个二进位的值是 0.125。

- 思路：对 L 设置综合属性 val 计算二进制串 L 的值、设置综合属性 $length$ 计算 L 的长度， $L.val/2^{L.length}$ 即为小数部分 L 的值

$S \rightarrow L_1.L_2$ $S.val := L_1.val + (L_2.val / 2^{L_2.length})$

$S \rightarrow L$ $S.val := L.val$

$L \rightarrow L_1B$ $L.val := 2 * L_1.val + B.c;$
 $L.length := L_1.length + 1$

$L \rightarrow B$ $L.val := B.c;$
 $L.length := 1$

$B \rightarrow 0$ $B.c := 0$

$B \rightarrow 1$ $B.c := 1$

翻译

产生式

$E \rightarrow TR$

$R \rightarrow \text{addop } T R_1 \mid \varepsilon$

$T \rightarrow \text{num}$

语 义 规 则

`print(addop.lexeme)`

`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

$E \rightarrow TR$

$R \rightarrow \text{addop } T \{ \text{print(addop.lexeme)} \} R_1 \mid \varepsilon$

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

P165-11. 设下列文法生成变量的类型说明:

$$D \rightarrow \text{id } L$$
$$L \rightarrow , \text{id } L \mid : T$$
$$T \rightarrow \text{integer} \mid \text{real}$$

(1) 构造一个翻译模式，把每个标识符的类型存入符号表；参考例 6.2。

(2) 由 (1) 得到的翻译模式，构造一个预测翻译器。

- 思路：对 D,L,T 设置综合属性 type, 过程 addtype (id.entry,type) 用来将标识符 id 的类型 type 填入到符号表中

$D \rightarrow \text{id } L$

$D.\text{type} = L.\text{type} ;$
 $\text{addtype}(\text{id.entry}, L.\text{type})$

$L \rightarrow , \text{id } L_1$

$L.\text{type} = L_1.\text{type};$
 $\text{addtype}(\text{id.entry}, L_1.\text{type})$

$L \rightarrow :T$

$L.\text{type} = T.\text{type}$

$T \rightarrow \text{integer}$

$T.\text{type} = \text{integer}$

$T \rightarrow \text{real}$

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

第七章 语义分析和中间代码产生

- 中间语言
- 赋值语句的翻译
- 布尔表达式的翻译
- 控制语句的翻译
- 过程调用的处理

中间语言

- 后缀式，逆波兰表示
- 图表示： DAG、抽象语法树
- 三地址代码
 - 三元式
 - 四元式
 - 间接三元式

P217-1. 给出下面表达式的逆波兰表示 (后缀式) :

$a * (-b + c)$ $\text{not } A \text{ or not } (C \text{ or not } D)$

$a + b * (c + d / e)$ $(A \text{ and } B) \text{ or } (\text{not } C \text{ or } D)$

$-a + b * (-c + d)$ $(A \text{ or } B) \text{ and } (C \text{ or not } D \text{ and } E)$

$\text{if } (x + y) * z = 0 \text{ then } (a + b) \uparrow c \text{ else } a \uparrow b \uparrow c$

$a^*(-b+c)$

$a+b^*(c+d/e)$

$-a+b^*(-c+d)$

not A or not (C or not D)

(A and B) or (not C or
D)

(A or B) and (C or not D
and E)

if $(x+y)^*z=0$ then
 $(a+b)^\uparrow c$ else $a^\uparrow b^\uparrow c$

$ab@c+^*$

$abcde/+^*+$

$a@bc@d+^*+$

A not C D not or not or

A B and C not D or or

A B or C D not E and or
and

$xy+z^*0= ab+c^\uparrow abc^\uparrow\uparrow$ if-
then-else

P217-3. 请将表达式 $-(a+b)*(c+d)-(a+b+c)$ 分别表示成三元式、间接三元式和四元式序列。

三元式序列：

- (1) +, a, b
- (2) -, (1), -
- (3) +, c, d
- (4) *, (2), (3)
- (5) +, a, b
- (6) +, (5), c
- (7) -, (4), (6)

间接三元式序列：

三元式表

- (1) +, a, b
- (2) -, (1), -
- (3) +, c, d
- (4) *, (2), (3)
- (5) +, (1), c
- (6) -, (4), (5)

间接码表

- (1)
- (2)
- (3)
- (4)
- (1)
- (5)
- (6)

P217-3. 请将表达式 $-(a+b)*(c+d)-(a+b+c)$ 分别表示成三元式、间按三元式和四元式序列。

四元式序列：

(1)	+	a	b	T1
(2)	-	T1	-	T2
(3)	+	c	d	T3
(4)	*	T2	T3	T4
(5)	+	a	b	T5
(6)	+	T5	c	T6
(7)	-	T4	T6	T7

赋值语句的翻译

- 简单算术表达式及赋值语句
- 数组元素的引用
- 产生有关类型转换的指令

P218-4. 按 7.3 节所说的办法, 写出下面赋值句

$$A:=B*(-C+D)$$

的自下而上语法制导翻译过程。给出所产生的三地址代码。

步骤	输入串	栈	PLACE	四元式
(1)	A:=B*(-C+D)			
(2)	:=B*(-C+D)	i	A	
(3)	B*(-C+D)	i:=	A-	
(4)	*(-C+D)	i:=I	A-B	
(5)	*(-C+D)	i:=E	A-B	
(6)	(-C+D)	i:=E*	A-B-	
(7)	-C+D)	i:=E*(A-B--	
(8)	C+D)	i:=E*(-	A-B---	
(9)	+D)	i:=E*(-I	A-B---C	25

P218-4. 按 7.3 节所说的办法, 写出下面赋值句

$$A:=B*(-C+D)$$

的自下而上语法制导翻译过程。给出所产生的三地址代码。

步骤	输入串	栈	PLACE	四元式
(9)	+D)	i:=E*(-I	A-B---C	
(10)	+D)	i:=E*(-E	A-B---C	(-,C,-,T1)
(11)	+D)	i:=E*(E	A-B---T1	
(12)	D)	i:=E*(E+	A-B---T1-	
(13))	i:=E*(E+I	A-B---T1-D	
(14))	i:=E*(E+E	A-B---T1-D	(+,T1,D,T2)
(15))	i:=E(E	A-B---T2	
(16)		i:=E*(E)	A-B---T2-	
(17)		i:=E+E	A-B---T2	(*,B,T2 ,T3)
(18)		i:=E	A-T3	(:=,T3 ,-,A)
(19)		A		

带数组元素引用的赋值语句翻译模式

(1) $S \rightarrow L := E$

{ if $L.offset = \text{null}$ then /*L 是简单变量 */

emit($L.place := E.place$)

else emit($L.place$ '[' $L.offset$ ']' $:= E.place$)}

(2) $E \rightarrow E_1 + E_2$

{ $E.place := \text{newtemp}$;

emit($E.place := E_1.place + E_2.place$)}

(3) $E \rightarrow (E_1) \{ E.place := E_1.place \}$

(4) $E \rightarrow L$

{ if $L.offset = \text{null}$ then
 $E.place := L.place$

else begin

$E.place := \text{newtemp};$

 emit($E.place :=$ $L.place$ '[' $L.offset$ ']')

end

}

$A[i_1, i_2, \dots, i_k]$

$((\dots i_1 n_2 + i_2) n_3 + i_3) \dots n_k + i_k) \times w +$

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(8) $\text{Elist} \rightarrow \text{id} [E$

{ $\text{Elist.place} := E.\text{place};$

$\text{Elist.ndim} := 1;$

$\text{Elist.array} := \text{id.place} \}$

$A[i_1, i_2, \dots, i_k]$

$((\dots i_1 n_2 + i_2) n_3 + i_3) \dots n_k + i_k) \times w +$

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(7) $\text{Elist} \rightarrow \text{Elist}_1, E$

```
{  t:=newtemp;
   m:=Elist1.ndim+1;
   emit(t ':=' Elist1.place '*' limit(Elist1.array,m) );
   emit(t ':=' t '+' E.place);
   Elist.place:=t;
   Elist.ndim:=m
   Elist.array:= Elist1.array;
```

}

$A[i_1, i_2, \dots, i_k]$

$((\dots i_1 n_2 + i_2) n_3 + i_3) \dots n_k + i_k) \times w +$

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(5) $L \rightarrow \text{Elist}$]

{ L.place := newtemp;

emit(L.place := Elist.array ' - ' C);

L.offset := newtemp;

emit(L.offset := w '*' Elist.place) }

(6) $L \rightarrow \text{id}$ { L.place := id.place; L.offset := null }

P218-5. 按照 7.3.2 节所给的翻译模式，把下列赋值句翻译为三地址代码：

$$A[i,j]:=B[i,j]+C[A[k,l]]+D[i+j]$$

设：

A 、 B : 10*20

C 、 D : 20 ,

宽度 $w = 4$

下标从 1 开始

$A[i,j] := B[i,j] + C[A[k,l]] + D[i+j]$

T1 := i * 20

T1 := T1 + j

T2 := A - 84

T3 := 4 * T1

T4 := i * 20

T4 := T4 + j

T5 := B - 84

T6 := 4 * T4

T7 := T5[T6]

(5) L → E list] { L.place := newtemp;

emit(L.place ':=' Elist.array ' - ' C);

L.offset := newtemp;

(1) S → L := E

{ if L.offset = null then /* L 是简单变量 */

emit(L.place ':=' E.place)

else emit(L.place '[' L.offset ']' ':=' E.place) }

(4) E → L

{ if L.offset = null then

E.place := L.place

else begin

E.place := newtemp;

emit(E.place ':=' L.place '[' L.offset '])

end }



$A[i,j] := B[i,j] + C[A[k,l]] + D[i+j]$

$T1 := i * 20$

$T1 := T1 + j$

$T2 := A - 84$

$T3 := 4 * T1$

$T4 := i * 20$

$T4 := T4 + j$

$T5 := B - 84$

$T6 := 4 * T4$

$T7 := T5[T6]$

$T8 := k * 20$

$T8 := T8 + l$

$T9 := A - 84$

$T10 := 4 * T8$

$T11 := T9[T10]$

$T12 := C - 4$

$T13 := 4 * T11$

$T14 := T12[T13]$

$T15 := T7 + T14$

$T16 := i + j$

$T17 := D - 4$

$T18 := 4 * T16$

$T19 := T17[T18]$

$T20 := T15 + T19$

$T2[T3] := T20$

小结

- 属性文法和语法制导翻译
 - 属性计算
 - 根据语义设计属性文法
 - 根据语义设计翻译模式
- 语义分析和中间代码产生
 - 表达式的中间表示
 - 后缀式、DAG、抽象语法树、三地址代码
 - 翻译成四元式、构造翻译模式
 - 算术表达式