编译原理

第八章语法制导翻译和中间代码生成

8.1 属性文法(Attribute Grammar)

8.2 语法制导翻译(Syntax-directed translation)

8.3 中间代码

8.4--8.8 一些语句的翻译

概述　　语义处理

语法分析后的源程序 ⇒ 语义处理

- 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。

- 属性文法和语法制导翻译方法的基本思想
- 几种典型的中间代码形式
- 对一些语法成分的翻译

属性文法和语法制导翻译

- 虽然形式语义学(如指称语义学、公理语义学、操作语义学等)的研究已取得了许多重大的进展，但目前在实际应用中比较流行的语义描述和语义处理的方法主要还是属性文法和语法制导翻译方法

- 属性文法：包含一个上下文无关文法和一系列语义规则，这些语义规则附在文法的每个产生式上。
- 语法制导翻译：指在语法分析过程中，完成附加在所使用的产生式上的语义规则描述的动作。

属性文法

- 属性文法(attribute grammar)是一个三元组: A=(G,V,F),其中

G:是一个上下文无关文法

V:有穷的属性集,每个属性与文法的一个终结符或非终结符相连，这些属性代表与文法符号相关信息，如它的类型、值、代码序列、符号表内容等等。属性与变量一样，可以进行计算和传递。属性加工的过程即是语义处理的过程。

F:关于属性的断言或一组属性的计算规则(称为语义规则) . 语义规则与一个产生式相联，只引用该产生式左端或右端的终结符或非终结符相联的属性.

- 语义规则描述的动作包括：属性计算、静态语义检查、符号表操作、代码生成等。
- 如果对于G中的某一个输入串而言， F中的所有断言对该输入串的语法树结点的属性都为真，则该输入串也是A语言中的句子。
- 编译程序的静态语义审查就是验证所编译的程序的断言是否全部为真。

- 例：类型检查的属性文法

$E \rightarrow T^1 + T^2 \{T^1.t = \text{int AND } T^2.t = \text{int}\}$

$E \rightarrow T^1 \text{ or } T^2 \{T^1.t = \text{bool AND } T^2.t = \text{bool}\}$

$T \rightarrow \text{num} \{T.t := \text{int}\}$

$T \rightarrow \text{true} \{T.t := \text{bool}\}$

$T \rightarrow \text{false} \{T.t := \text{bool}\}$

图 8.2　类型检查的属性文法

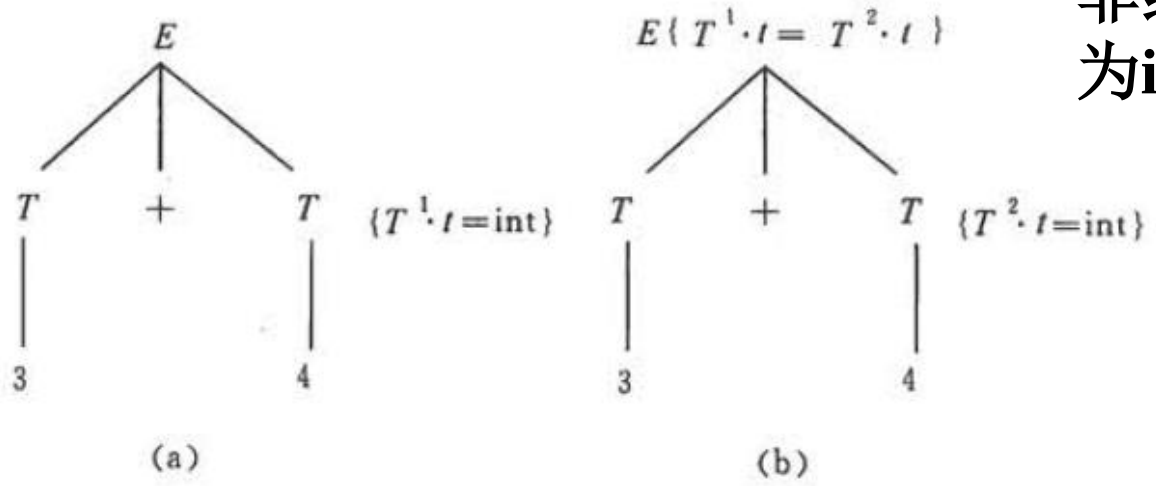


图 8.1　静态语义审查

非终结符T有属性t，为int 或bool

- 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$b:=f(c_1,c_2 \ldots c_k)$，f是函数，b和$c_1,c_2 \ldots c_k$是该产生式文法符号的属性。

  （1）如果b是A的一个属性，并且$c_1,c_2 \ldots c_k$是产生式右边文法符号的属性或A的其他属性，则称b是A的综合属性;(Synthesized attribute)

  （2）如果b是产生式右边某个文法符号X的一个属性，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，则称b是文法符号X的继承属性。(Inherited attribute)
- 在两种情况下，我们都说属性b依赖于属性$c_1,c_2 \ldots c_k$。

- 简单地说，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。

- (1)非终结符既可有综合属性，也可有继承属性，但文法开始符号没有继承属性
- (2)终结符号只有综合属性，由词法程序提供

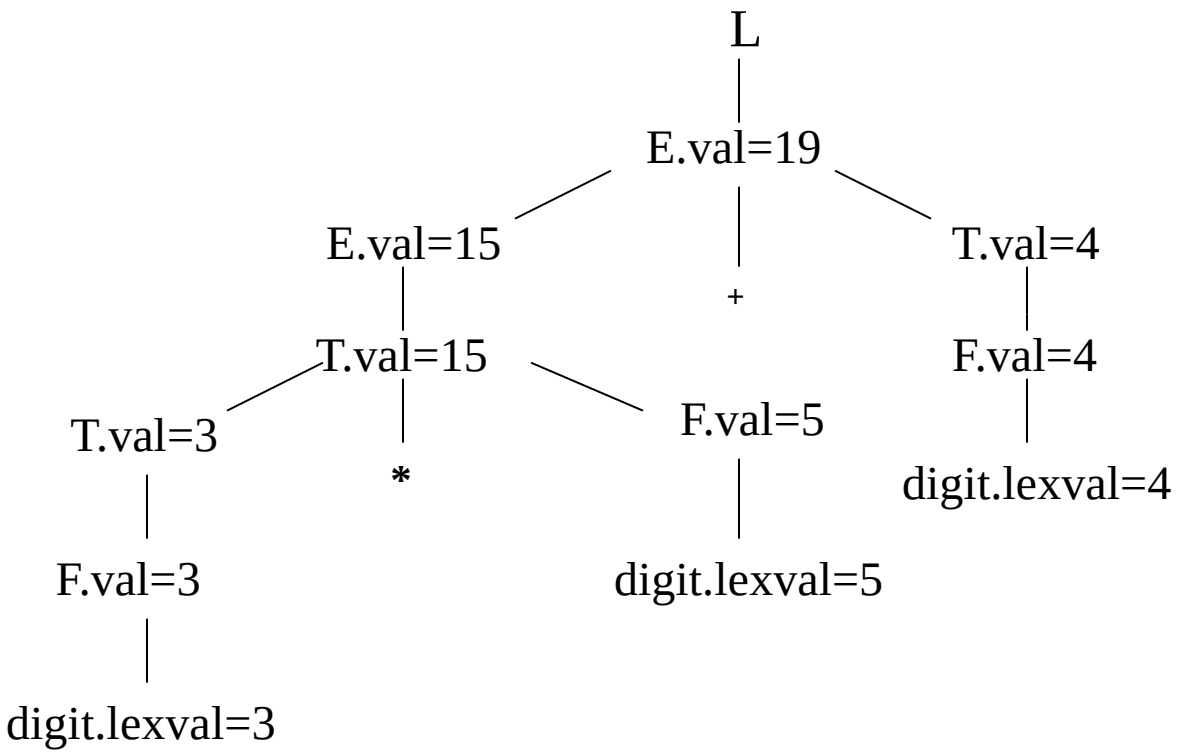
一个属性文法的例子：简单算术表达式求值的语义描述。

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，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
$L \rightarrow E$	Print(E.val)
$E \rightarrow E_1+T$	$E.val:=E_1.val+T.val$
$E \rightarrow T$	E.val:=T.val
$T \rightarrow T_1 * F$	$T.val:=T_1.val \times F.val$
$T \rightarrow F$	T.val:=F.val
$F \rightarrow (E)$	F.val:=E.val
$F \rightarrow digit$	F.val:=digit.lexval

- 设表达式为3＊5+4，则语义动作是什么？

设表达式为3＊5+4，则语义动作打印数值19



3*5+4的带注释的分析树

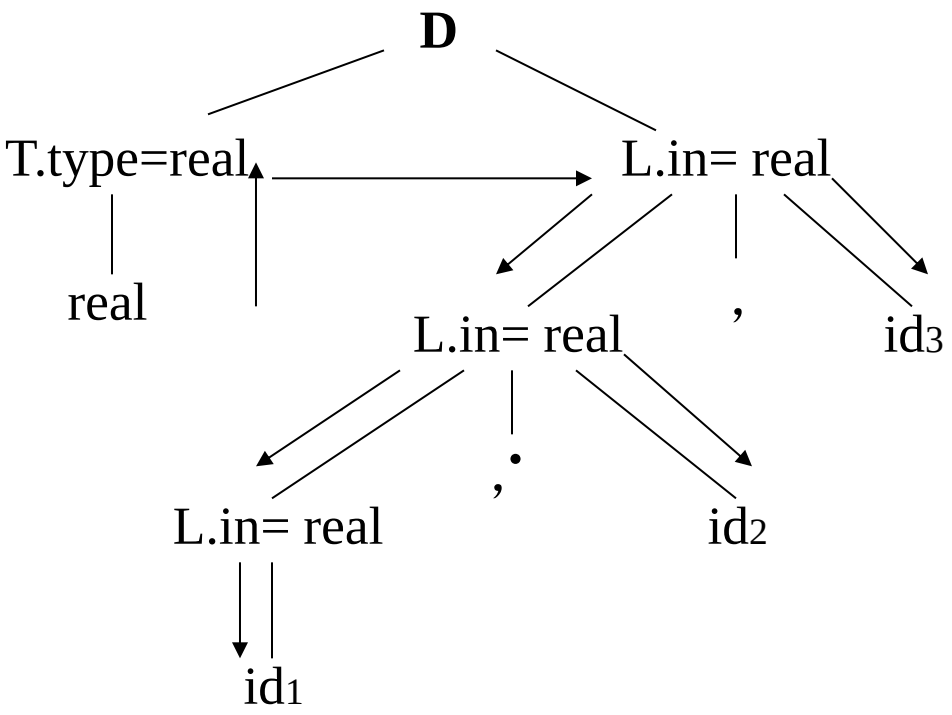
继承属性(Inherited attribute)

- 一个结点的继承属性值是由此结点的父结点和/或兄弟结点的某些属性来决定的。

例8.2　描述说明语句中各种变量的类型信息的语义规则（**继承属性L.in**）

产 生 式	语 义 规 则
D →TL	L.in:=T.type
T→ int	T.type=integer
T →real	T.type:=real
L → L1,id	L1.in:=L.in addtype(id.entry,L.in)
L→ id	addtype(id.entry,L.in)

Real id1,id2,id3



addtype把每个标识符的类型信息登录在符号表的相关项中

语法制导的翻译

- 语法制导翻译：基于属性文法的处理过程
- 直观地说，一个语法制导翻译的基础是一个属性文法，其中翻译成分依附在每一产生式上。

语法制导翻译实现

- 语法制导翻译即基于属性文法的处理过程通常是这样的：对单词符号串进行语法分析，构造语法分析树，然后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。对输入符号串的翻译就是根据语义规则进行计算的结果。

→输入符号串

→ 语法分析树

→ 属性依赖图

→ 语义规则的计算顺序

依赖图(Dependency graph)

依赖图是一个**有向图**，用来描述分析树中的属性之间的相互**依赖关系**。

依赖图的构造算法：

for分析树中每一个结点n **do**

for 结点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 分析树中每一个结点n **do**

for结点n所用产生式对应的每一个语义规则

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

for i :=1 to k **do**

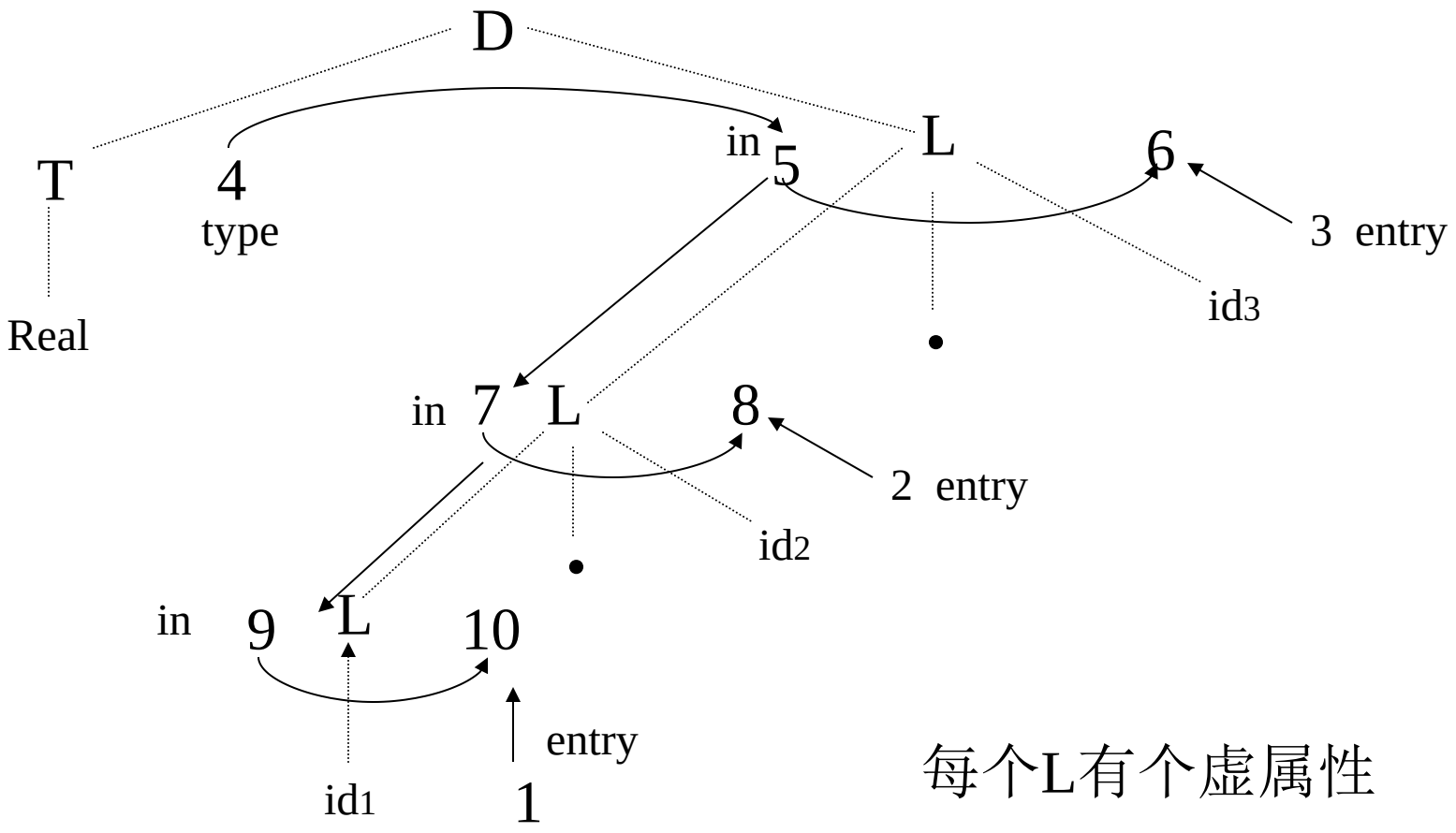
从c_i结点到b结点构造一条有向边

依赖图----例8.2

例8.2 **继承属性L.in**

产生 式	语 义 规 则
D →TL	L.in:=T.type
T→ int	T.type=integer
T →real	T.type:=real
L → L1,id	L1.in:=L.in addtype(id.entry,L.in)
L→ id	addtype(id.entry,L.in)

例8.2 **Real id1,id2,id3**分析树的依赖图



每个L有个虚属性

从依赖图的拓扑排序中，可以得到计算语义规则的顺序。用这个顺序来计算语义规则就得到输入符号串的翻译。

例8.2Real id1,id2,id3分析树的依赖图

每一条边都是从序号较低的结点指向序号较高的结点。历此，依赖图的一个拓扑排序可以从低序号到高序号顺序写出。从这个拓扑排序中我们可以得到下列程序，用a_n来代表依赖图中与序号n的结点有关的属性：

a_4: = real

a_5: = a_4

(a6) addtype (id_3.entry, a_5);

a_7: = a_5;

(a8) addtype (id_2.entry, a_7)

a_9: = a_7

(a10) addtype (id_1.entry, a_9)

这些语义规则的计算将把real类型填入到每个标识符对应的符号表项中。

属性计算方法

- 树遍历的属性计算方法
  设语法树已经建立起了，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。然后以某种次序遍历语法树，直至计算出所有属性。最常用的遍历方法是深度优先，从左到右的遍历方法。如果需要的话，可使用多次遍历（或称遍）。
- 一遍扫描的处理方法

  与树遍历的属性计算文法不同，一遍扫描的处理方法是在语法分析的同时计算属性值，而不是语法分析构造语法树之后进行属性的计算，而且无需构造实际的语法树。

- 在某些情况下可用一遍扫描实现属性文法的语义规则计算。也就是说在语法分析的同时完成语义规则的计算，无须明显地构造语法树或构造属性之间的依赖图。因为单遍实现对于编译效率非常重要。

具体的实现希望在单遍扫描中完成翻译

怎样实现这种翻译器？一个一般的属性文法的翻译器可能是很难建立的，然而有一大类属性文法的翻译器是很容易建立的

S-属性文法　适用于自底向上的计算

L-属性文法　适用于自顶向下的分析，也可用于自底向上。

S-属性文法是L-属性文法的一个特例。

- 用一遍扫描的编译程序模型来理解语法制导的翻译方法：
- 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。
- 在自上而下的语法分析中，若一个产生式匹配输入串成功（对非终结符推导一次）
- 自下而上分析中，当一个产生式被用于进行归约时，此产生式相应的语义规则就被计算，完成相关的语义分析和代码产生等工作。在这种情况下，语法分析工作和语义规则的计算是穿插进行的。

S－属性文法的自下而上计算

S－属性文法，它只含有综合属性。

- 综合属性可以在分析输入符号串的同时自下而上来计算。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－属性文法的翻译器通常可借助于LR分析器实现。在S－属性文法的基础上，LR分析器可以改造为一个翻译器，在对输入串进行语法分析的同时对属性进行计算。

S_m	y. Val	y
S_{m-1}	x. Val	x
⋮	⋮	⋮
S_0	—	#
状态栈	语义值栈	符号栈

图 8.6　扩充的分析栈

产生式	语义规则
0）L→E	print（E.val）
1）E→E¹＋T	E.val：＝E¹.val＋T.val
2）E→T	E.val：＝T.val
3）T→T¹*F	T.val：＝T¹.val×F.val
4）T→F	T.val：＝F.val
5）F→（E）	F.val：＝E.val
6）F→digit	F.val:＝digit.lexval

LR分析器可以改造为一个翻译器，在对输入串进行语法分析的同时对属性进行计算。

LR分析器**增加语义栈**

２＋３*５的分析和计值过程

步骤	归约动作	状态栈	语义栈(值栈)	符号栈	留余输入串
1)		0	—	#	2+3*5#
2)		05	——	#2	+3*5#
3)	r_6	03	—2	#F	+3*5#
4)	r_4	02	—2	#T	+3*5#
5)	r_2	01	—2	#E	+3*5#
6)		016	—2—	#E+	3*5#
7)		0165	—2——	#E+3	*5#
8)	r_6	0163	—2—3	#E+F	*5#
9)	r_4	0169	—2—3	#E+T	*5#
10)		01697	—2—3—	#E+T*.	5#
11)		016975	—2—3——	#E+T*5	#
12)	r_6	01697(10)	—2—3—5	#E+T*F	#
13)	r_3	0169	—2—(15)	#E+T	#
14)	r_1	01	—(17)	#E	#
15)	接受				

总结

- 属性文法
- 语法制导翻译
- S-属性文法

- 作业
  - P203:3