编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第六章 属性文法和语法制导翻译

第六章 属性文法和语法制导翻译

- 本章目录
- 6.1 语法制导翻译概述
- 6.2 属性文法
- 6.3 S-属性文法的自下而上计算

第六章 属性文法和语法制导翻译

- 大纲要求
- 掌握：语法制导翻译的处理方法，基于属性文法的综合属性、继承属性的计算方法。
- 理解：语法制导翻译的基本思想；基于属性文法的处理方法。
- 了解：语法制导翻译、属性文法的概念和技术。

6.1 语法制导翻译概述

6.1 语法制导翻译概述

- 语法制导翻译Syntax-Directed Definitions(SDD)

  语法制导翻译是一个上下文无关文法，但具有**属性**和**规则**。

  假如X是一个符号，a是X的一个属性。假如X是语法分析树中的一个结点，X.a是指X结点的属性a的值。

6.1 语法制导翻译概述

- 语法制导翻译Syntax-Directed Definitions(SDD)

  属性有很多种，比如：数字，类型，表格操作或字符串等等。字符串甚至可能是很长的一系列代码，比如语法分析使用的中间代码。

6.1 属性文法

- 属性的规定

  终结符使用单词的属性 （id. lexval ）

  保留字：if, begin, function,……

  常数：40.12, 232, 80, “TCP/IP”

  标识符：sum, tcc, id

  非终结符根据实际需要设定属性

  表达式E：E.type， E.val

6.1 语法制导翻译概述

- 语法制导翻译的处理方法

语法制导翻译是把语法产生式跟语义规则相连接。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。

	语法产生式	语义规则
例如：	$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel$ '+'

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。

9

6.1 语法制导翻译概述

- 语法制导翻译的处理方法

例如： $E \rightarrow E_1 + T$ $E.code = E_1.code \parallel T.code \parallel$ '+'

这里有两个非终结符，E和E_1；它们的下标用于区分彼此。根据语义规则，E.code由E_1.code, T.code 和 '+'相连接组成。但语义规则明确指出，E的翻译基于E_1, T和'+'这样的字符串是不够的。

语法制导翻译，在产生式的基础上，增加了语义动作(semantic actions)，按照分析的进程，执行遇到的语义动作。

10

6.1 语法制导翻译概述

- 语法制导翻译的处理方法

例如： $E \rightarrow E_1 + T$ $\quad\quad$ $E.code = E_1.code \,||\, T.code \,||\, '+'$

语法制导翻译，在产生式的基础上，增加了语义动作(semantic actions)，按照分析的进程，执行遇到的语义动作。

$$E \rightarrow E_1 + T \quad \{print\ '+'\}$$

一般是放在大括号里面，当大括号属于语法符号时，语法符号中的大括号用单引号'{', '}'。括号放的位置，指出了语义作用执行的顺序，一般可以放在产生式的任意位置。上述例子的语义作用的执行，是在读入E_1和T以后。

6.1 语法制导翻译概述

- 语法制导翻译中3个概念的区别

输入文法：未插入动作符号时的文法。

由输入文法可以通过推导产生输入序列。

翻译文法：插入动作符号的文法。

由翻译文法可以通过推导产生活动序列。

属性翻译文法：文法符号可带有属性，并定义相应的属性求值规则，就成为**属性翻译文法**。比翻译文法能更细地描述翻译过程。（属性有**综合属性**和**继承属性**之分）

6.1 语法制导翻译概述

- 语法分析的方法

  Top-down：推导时完成

  Bottom-up：归约时完成

- 语义

  语法成分的语义可以看成是相应文法符号的属性，通过属性的计算，来完成翻译

6.1 语法制导翻译概述

- 语法制导翻译法

  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。

  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。

  语义规则的计算，在自上而下语法分析中，是在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。

6.2 属性文法

6.2 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

  非终结符的两种属性包含**继承属性**和**综合属性**。

6.2 属性文法

- 综合属性 Synthesized Attributes

语法分析树结点N的**综合属性**，是由结点N的属性值和其子结点的属性值决定。 **“自下而上”传递信息**。

语法产生式	语义规则
$A \rightarrow A_1 + T$	$A.code = A_1.code \parallel T.code \parallel$ ‘+’ ,

A是结点N处的非终结符，A的综合属性是由结点N的产生式以及相关联的语义规则综合决定。

6.2 属性文法

- 继承属性Inherited Attributes

继承属性：语法分析树结点N的继承属性，是由结点N的父结点的属性值决定。 **“自上而下”传递信息。**

语法产生式	语义规则
$B \rightarrow B_1 + T$	$B.code = B_1.code \parallel T.code \parallel$ ‘+’ ,

B是结点N处的非终结符，B的继承属性是由结点N的父结点的产生式以及相关联的语义规则综合决定。注意B必须包含在其父结点的产生式中。

6.2 属性文法

- 综合属性和继承属性的规定

  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

  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。**

6.2 属性文法

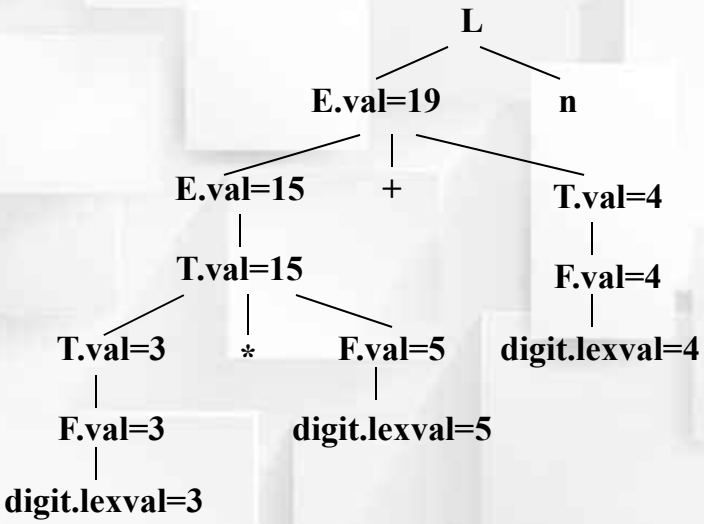
- 台式计算器的属性文法

画出计算3*5+4n的带注释的语法树

产生式	属性（计算）规则/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

digit是定义的token，是终结符，lexval是单词digit的属性

6.2 属性文法



6.2 属性文法

- 综合属性 Synthesized Attributes

  仅仅使用综合属性的属性文法称为S-属性文法。

  对于S-属性定义，通常使用自下而上的分析方法，在建立每一个结点处使用语义规则来计算综合属性值，即在用哪个产生式进行归约时，就执行那个产生式的S-属性定义计算属性的值，从叶结点到根结点进行计算。

  继承属性在什么时候使用呢?

6.2 属性文法

- 继承属性Inherited Attributes

产 生 式	语 义 规 则
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)

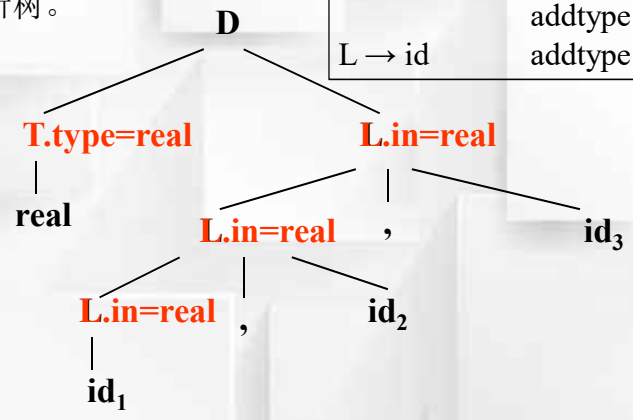
L.in := T.type 传递类型信息，T.type保存有类型信息

addtype 在符号表中为变量添加类型信息

T = real

根据产生式和对应的语义规则，可以画出语法分析树。

产 生 式	语 义 规 则
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)



6.3 S-属性文法的自下而上计算

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算
  只含有**综合属性**。

  综合属性可以在分析输入符号串的同时，在语法分析树中，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**。

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算

  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、语义子程序），用**花括号**{ }括起来，插入到产生式右部的合适位置上

  当只需要综合属性时：为每一个语义规则（这里我们也称语义动作、语义子程序）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

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算

  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。

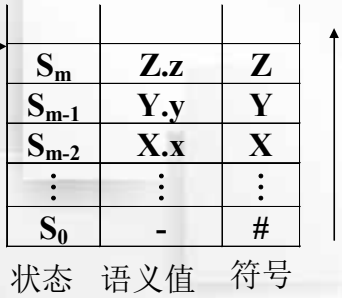
  在分析栈中使用一个附加的域来存放综合属性值，LR分析器的分析栈增加一个综合属性值的域。

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算

  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。

A→XYZ　　A·a:=*f*(X·x, Y·y, Z·z)



29

如何使用LR分析表翻译程序？

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算

计算3*5+4n

产 生 式	属性（计算）规则/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

6.3 S-属性文法的自下而上计算

- S-属性文法的自下而上计算

计算3*5+4n

状态	i	+	
0	s5		
1		s6	

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

输入	state	sym	val	用到的产生式
3*5+4n	0	#	-	
*5+4n	05	#3	-3	
*5+4n	03	#F	-3	F→digit
*5+4n	02	#T	-3	T→F
5+4n	027	#T*	-3 -	
+4n	0275	#T*5	-3 - 5	T.val :=T_1.val*F.val
+4n	0275	#T*5	-3 - 5	
+4n	02710	#T*F	-3 - 5	F→digit
+4n	02	#T	-15	T→T*F
+4n	01	#E	-15	E→T
4n	016	#E+	-15-	
n	0165	#E+4	-15- 4	E.val := E_1.val+T.val
n	0165	#E+4	-15- 4	
n	0163	#E+F	-15- 4	F→digit
n	0169	#E+T	-15- 4	T→F
n	01	#E	-19	E→E+T
		#En	-19-	
		#L	-19	L→En

33

第六章 小结

第六章 小结

- 6.1 语法制导翻译概述
- 6.2 属性文法
- 6.3 S-属性文法的自下而上计算

Coursework

- 6.1 按照书上表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	属性（计算）规则/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