编译程序原理与技术

李赣生　王华民

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书共分 15 章。第 1、2 章介绍了编译程序的基本概念及程序的构造;第 3 章讨论了词法分析;第 4 章讨论了上下文无关文法的基本概念;第 5、6 章讨论语法分析方法;第 7 章至第 14 章讨论有关语义分析与处理的有关问题;第 15 章讨论代码生成技术。书中列出了 Lex, Yacc 和 C 的典型编译源程序,力求把理论和实现细节相结合。

本书可作为计算机软件专业大学生、研究生教材,也可供从事计算机软件研究、设计和开发的人员参考。

图书在版编目(CIP)数据

编译程序原理与技术/李赣生、王华民编著 .—北京: 清华大学出版社, 1997.7
ISBN 7-302-02632-7

Ⅰ.编… Ⅱ.李… Ⅲ.机器翻译-程序设计 Ⅳ.H085.6

中国版本图书馆 CIP 数据核字(97)第 16694 号

出版者: 清华大学出版社 (北京清华大学校内,邮编 100084)
印刷者: 中国科学院印刷厂
发行者: 新华书店总店北京科技发行所
开 本: 787× 1092 1/16 印张: 22 字数: 千字
版 次: 1997 年 10 月 第 1 版 1997 年 10 月 第 1 次印刷
书 号: ISBN 7-302-02632-7/TP·1352
印 数: 0001～4000
定 价: 28.00 元

序　　言

编译程序的基本原理与实现技术,作为计算机科学与工程专业的主要专业基础课,越来越引起人们的重视。本书的目的,是力图以一种理论与实践紧密结合的方式,将编译程序的内容阐述清楚。作者写本书的目的有两个:一是作为计算机专业与计算机软件专业的本科生与研究生的编译程序课程的教材;二是作为有志于投身计算机系统软件开发与研究的软件工作者的一本有用的参考书。

就像学习程序设计一样,掌握编译程序的构造原理与方法是自己设计与编写一个编译程序。但是,不同于一般程序设计,要自己设计与编写一个编译程序,首先必须基本掌握编译程序构造的一般原理。因此,本书仍然强调基本原理的阐述。但是,我们阐述理论主要从实现的角度出发。这就要对理论有所取舍,阐述的方式有所讲究。常用的、基本的东西要讲透,少用的、不用的东西就少讲或不讲。例如,语法分析部分我们只讲 LL 及 LR,不讲算符优先文法。代码优化部分(特别是全部优化部分)包含大量内容,如果把它纳入本书的内容,在课堂上也没有时间讨论。实际上,它远远超出了本书的内容。因此,本书只对代码优化作一些简单的论述,删去了全局代码优化和其它一些麻烦问题的讨论。实际上,从动手的角度来看,也没有精力在一门课程设计的时间内完成优化部分的实习内容。

一个完整的编译程序,再小也包括三个方面的内容:词法分析、语法分析和语义翻译。因此,为了使读者在学习本书内容的过程中,能够看书动手构造一个“五脏俱全”的编译程序,我们在全书都贯穿一条使用编译工具的线索。全书的实例与程序,都是用编译工具 Lex、Yacc 和 C 来编写的。这样做的一个明显好处是:理论联系实际,程序逻辑简单清楚。在国内,各种流行机器的 UNIX 系统及 DOS 系统上,都装有 Lex、Yacc 和 C 的编译程序。因此,学生利用这些自动化工具,完全可以在一个学期之内,独立地完成一个完整的编译程序的设计与实现。

全书共 15 章,第 1 章对编译程序的几个基本概念及编译程序的构造作一概括性的介绍。第 2 章介绍了一个简单的编译程序,读过这章之后,读者对编译程序有一个初步的感性认识,对于编译的全过程有一个初步的轮廓。对本书后面几章的内容及安排,心中基本有数。第 3 章专门讨论词法分析,这部分的理论基础是正规表达式与有限自动机,建立在这个理论基础之上的 Lex,是一个建立词法分析器的自动化工具,本章讨论了 Lex 实现及其使用的各种问题。第 4 章讨论上下文无关文法的基本概念。第 5 和第 6 章讨论语法分析方法。第 5 章讨论自上而下分析的语法分析方法。第 6 章讨论自下而上的语法分析方法。这一章中介绍了基于 LALR(1)文法的分析器自动生成工具 YACC 及其使用的问题。从第 7 章至第 14 章讨论有关语义分析与处理的有关问题。在第 7 章,首先讨论语义制导

翻译的基本方法,包括增广文法与属性文法。我们在这里用最少的篇幅阐述了继承属性与综合属性的概念,并讨论了它们在 LL 及 LR 分析中的处理特点。在第 7 章中还讨论了各种中间语言表示。我们在本书中讨论语义处理方法时,选择了 C 语言作样本语言,即实际讨论 C 语言的语义处理问题。因为 C 语言是当今最为流行的程序语言之一,复杂程度也适中。在第 7 章较详细描述了 C 代码,这是一种类似于 C 的三元组形式的中间语言代码,由 A.I.Holub 设计。C 代码的主要优点是易读,经简单的宏处理之后,可转换成 C 语言的语句。因此,在产生 C 代码之后,就可以利用 C 编译命令产生可执行的目标码。这对于学生的编译程序实习无疑带来了方便。第 8 章讨论符号表,特别是分程序结构的符号表。第 9 章讨论运行时存储分配的问题,Display 表的处理虽然对 C 语言并不重要,但对于一般程序设计语言是比较重要的基础。第 10 章至第 13 章都是讨论基于 C 语言的语义处理技术。第 14 章专门讨论某些常用的局部优化技术。第 15 章讨论代码生成技术。

完全地阅读一个语言的编译程序的源程序确实是掌握编译程序的好办法。正因为这样,本书中含有许多 Lex、Yacc 及 C 的源程序。但是,这样做必然会使本书的内容来得庞大且显得累赘。本书试图把理论与实现细节相结合,但不罗列全部源程序,只有典型的问题才列出其源程序,希望有助于对原理的理解又不至于太繁琐。

本书在成书过程中得到我的研究生及本科生的帮助。特别是郭飞军、李莹、李薇、钟涛和刘俊等同学做了许多工作,在此谨表示深切的谢意。本书虽几经修改,但仍然有不少错误与缺点,欢迎广大读者提出宝贵修改意见。

作　者

1997. 1 于浙大求是园

目　　录

1 引　　论

1.1 什么是编译程序

顾名思义,编译程序是一种具有编撰与翻译功能的程序。人类要用计算机来解决问题,首先面临的一个问题,就是要告诉计算机解决什么问题,或许还要告诉计算机如何解决这个问题。这就牵涉到用什么样的语言来描述的问题。人所习惯的语言与计算机的基本语言(机器指令)有很大的差别,用机器指令来描述人想解决的问题十分不便。因而,计算机科学家设计一些比较习惯的语言来描述要解决的问题。这种语言,因为面向与接近人的自然语言,表达力强,易于使用,易于为人理解与接受,称为高级程序语言。相反,能被计算机直接理解与执行的语言,即机器指令,却不易被人们理解与接受,因而被称为低级机器语言。不管用什么语言来表达的对问题的描述,通常都称为程序。例如,FORTRAN 语言,ALGOL 60 语言,PASCAL 语言,C 语言,ADA 语言,C+ + 语言等都是高级程序设计语言。各种机器的机器语言及其相应的汇编语言,都是低级程序设计语言.用高级语言写的程序,不能直接被计算机理解与执行,必须经过等价的转换,变成机器能理解与执行的机器语言的程序。进行这种等价转换的工作,就是编译程序的任务。一般地说,编译程序就是这样一种程序,它将用一种语言写的程序,等价地转换为另一种语言写的程序。因此,它也叫翻译程序。前一个程序,即被翻译的程序,叫源程序;后一个程序,即翻译成的程序,叫目的程序或目标程序。

因此,翻译程序与编译程序这两个名词并无大的区别。但是,通常把从高级语言写的源程序到机器语言表示的目标程序的转换程序称为编译程序。而把从一种语言写的源程序到另一种语言写的目标程序的转换程序称作翻译程序。例如,从 ADA 语言到 C 语言的转换,从 ADA 语言到 PASCAL 语言的转换,从 C 语言到 ADA 语言的转换都是翻译程序。

编译程序的理论与技术,在计算机科学中是比较成熟的学科,其发展历史虽然不长,但其内容却十分丰富,用途也十分广泛。它是计算机专业,特别是软件专业的主要基础课。打好这个基础,您将会站得高,看得远,解决实际问题时得心应手。

1.2 解释性程序

编译程序将一种语言写的源程序转换成等价的可执行目标程序,其目标代码的格式及目标机各种各样,但编译程序的使命却始终是一种翻译工作。还有另外一种语言处理程

序,叫解释性程序,它不同于编译程序,它执行程序但不作翻译。或者说,它并不把源程序翻成可执行的目标代码。其工作过程如图 1.1 所示:

图 1.1　一个概念化的解释性程序

不同于编译程序,解释性程序把源程序(的某种中间语言表示)看成是自己的输入,源程序的原来的输入也是解释性程序的一部分输入,因而可以对源程序进行处理,就像对另一部分数据一样。程序执行时的控制点在解释性程序之中,而不在用户程序中,即用户程序是消极的,这就不同于编译程序产生的可执行的用户程序,在那里,它是积极的。

解释性程序允许:

- 在执行用户程序时修改用户程序。因此,它提供一种直接的交互调试能力。这种修改对于像 APL,BASIC 这样的非分程序结构的语言是非常容易的,因为修改个别语句并不需要重新分析整个程序。
- 对象的类型可动态地修改。随着程序的执行,符号的意义可以变化,例如,在某一点,它可以是整型变量,而在另一点,它可以是一个字符数组。这种符号意义的动态确定叫做流动绑定(Fluid Binding),对于编译程序是很头痛的事,它使得编译程序很难对其进行翻译。
- 提供良好的诊断信息。解释性程序执行时,把程序的执行与源程序行文的分析交织在一起,因而可以在诊断信息中,给出出错点的源程序行号,变量的符号名,对变量交互赋值,这些工作对于编译程序却是比较困难的。
- 解释性程序不依赖于目标机,因为它不生成目标代码。因此,其可移植性优于编译程序。

解释性程序的缺点主要是开销大,速度慢。

开销大主要体现在两个方面。一是在执行时要连续多次重新对程序行文进行分析考察。包括标识符的绑定、类型及操作的确定。这种考察所费的开销是巨大的,对于极为动态的语言,如 APL,同编译程序相比,速度上可能是 1∶ 100,甚至还要差。对于较静态的语言,如 BASIC,速度之比约为 1∶ 10。二是空间上的开销,解释性程序要保存大量支撑子程序,源程序不能按紧凑方式存放,符号表及程序行文的存放格式都要易于使用与修改,而不考虑空间上的节省。因此,一般解释性程序对于程序规模,变量个数,过程个数等都有一定的限制,超过这些内部限制的程序往往不能由解释性程序处理。

有一些语言,如 BASIC,LISP,PASCAL 等,既有解释性程序,又有编译程序。前者用于程序开发与调试,后者用于生产运行。事实上,编译与解释并非总有明显的分界线。例如,许多 BASIC 解释性程序把源程序转换成一种内部表示。在这种表示中,像 LET 及

GOTO 这样的关键字都表示为一个字节的操作码,标识符则用其在符号表中的入口号表示。这就是说,解释性程序包含有某种形式的‘翻译”。另一方面,编译程序也可包含有某种层次上的‘解释”。例如,在翻译阶段可产生某种虚拟机代码,这种代码可由软件或硬件解释。

1.3 编译程序的基本结构

编译程序的内部结构及组织方式虽是五花八门,变化多端。但是,万变不离其宗,其主要工作有两大部分:分析与综合。所谓分析,即对被编译的源程序进行分析;所谓综合,是在分析正确无误之后,综合出可以执行的机器语言程序,执行的结果应正确无误,同源程序应达到的目的完全一致。

现代编译程序都是语法制导的,也就是说,编译的过程由源程序的语法结构来控制,而语法结构通常由语法分析器(程序)来识别。语法分析器读入词法分析器输出的单词流,并由此建立源程序的语法结构。词法分析器逐一读入源程序的字符,分析字符流的词法结构,组成一个个的单词,如标识符、数、运算符等。语法结构的识别是分析部分的主要内容。根据程序的语法结构,语义分析器分析程序的语义(意义)。语义分析器包含许多语义子程序,在识别一种语法结构时,就调用与此结构相关的语义子程序,对附于该语法结构上的短语作语义检查,如一致性检查与作用域分析,确定其意义,综合出各程序部分的中间语言表示(IR)或目标代码。如果产生了IR,它作为代码生成器的输入,由此生成所需的机器语言程序。在语义分析器与代码生成器之间,可以夹一层优化,它改变IR序列,使其可以由此产生高效的代码,又与源程序的执行结果一致。一个编辑程序基本上就由这些部件组成,示于图1.2

下面,把图1.2中的每个方框的程序的工作内容,作一概括性的介绍。本书以后的各章也大致按此思路对各个程序涉及的工作内容逐一展开,进行更具体的介绍和讨论。

词法分析

源程序可以简单地被看成是一个多行的字符串。词法分析器逐一从前到后(从左到右)读入字符,按照源语言规定的词法规则,拼写出一个一个的单词(TOKEN)。此后,编译程序就把单词当作源语言的最小单位。例如,标识符、数、保留字、运算符、分隔符等,都是单词。在对下面的赋值语句

X: = 2* X + Y (1.1)

中的字符进行词法分析之后,把这一段字符串分割成下面的单词:

1. 标识符 X
2. 赋值号: =

图1.2 语法制导的编译程序的基本结构

3. 整数 2
4. 乘号*
5. 标识符 X
6. 加号+
7. 标识符 Y
8. 分号;

虽然单词的内部表示随编译器不同而不同,但基本上呈如下格式

TOKEN	LEXEME
单词名	单词值(或称词文)

例如,标识符、数、赋值号、逗号、分号、加号、乘号都是单词名。对于标识符 X、Y,单词名都是 id,而具体的字符串值 X,Y 都是单词值;对于数 2,数是单词名,而 2 是其单词值。对于运算符+、*,只有单词名+、*,无须单词值。这里,用 id1,id2 分别代表 X、Y,强调标识符的内部表示由于组成该标识符的字符串不同而不同。

因此,经过词法分析之后,(1.1)的表示不妨被写成

$$id1:=2*id1+id2; \tag{1.2}$$

当然,对于赋值号,+ 及数 2,也应有相应的单词反映其内部表示,这将在后面专讲词法分析时讨论。

语法分析

源语言的语法由以产生式为主体的上下文无关文法描述,据此,语法分析器读入单词,将它们组合成按产生式规定的词组(短语)。例如,根据赋值语句的产生式

〈赋值语句〉→〈变量〉:=〈表达式〉

及〈表达式〉的产生式

〈表达式〉→〈表达式〉+〈项〉
〈表达式〉→〈项〉
〈项〉→〈项〉*〈因式〉
〈项〉→〈因式〉
〈因式〉→标识符
〈因式〉→数

每个产生式都有形式:左部→右部。左部只有一个符号,叫非终结符,表示一个语法单位。右部是由非终结符与终结符组成的串。只在右部出现的符号叫终结符。非终结符即可在左部又可在右部中出现。例如,在上述产生式中,〈表达式〉、〈项〉和〈因式〉都是非终结符,+、*、标识符与数都是终结符。产生式中的箭头→可读为"定义为",例如产生式:〈表达式〉→〈表达式〉+〈项〉可表示:〈表达式〉定义为〈表达式〉后接+再后接〈项〉。

赋值语句 X:=2*X+Y 被分析成如图 1.3 层次结构的分析树所示。

图 1.3 中的分析树描述了输入的语法结构。在图 1.4(a)中给出了这种语法结构的更一般的内部表示语法树。语法树是分析树的压缩表示形式。语法树中运算符以内部节点的名义出现,而不是作叶节点出现。

图 1.3　x: = 2* X+ Y 的分析树

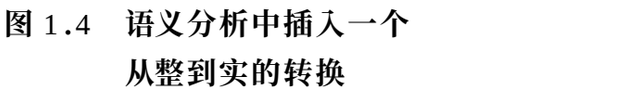

图 1.4　语义分析中插入一个从整到实的转换

在语法分析中,如果源程序没有语法错,就能正确地画出其分析树,否则,就指出语法错,给出相应的诊断信息。

语义分析

语义分析阶段主要检查源程序是否包含语义错误,并收集类型信息供后面的代码生成阶段使用。只有语法、语义正确的源程序才可被翻译成正确的目标代码。

语义分析的一个重要内容是类型检查。定义一种类型一般包含两个方面的内容,类型的载体及其上的操作。例如,整型是容纳整数的空间及定义在存于此空间上的整数的操作的统一体。这个空间是 32 位或 64 位的字,操作是+ 、- 、* 、/ 等 32 位或 64 位整数操作。因此,每个操作符(运算符)的操作数都必须符合源语言的规定。例如,有的程序语言要求整数加法只施用于两个整数,有的程序语言允许将实数自动转换为整数并进行整数加。有的程序语言把做数组下标用的实数视为一个错误并作出错报告。对表达式及语句中的对象作类型检查与分析,是语义分析的重要内容。

在确认源程序的语法与语义(类型一致性)之后,就可以对其进行翻译,改变源程序的内部表示,使其逐步变成可在目标机上运行的目标代码。经语义分析中的类型检查之后,语句(1.1)的中间表示转换成另一棵树,如图 1.4(b),其中增加了一个 inttoreal 的内部节点。

经过类型一致性检查之后,有的编译程序把源程序转换为明显的中间语言表示。不妨把这种中间语言表示看成是一种抽象机的程序。一般地说,这种中间表示应有两个重要的性质:易于产生;易于译成目标程序。产生中间语言表示的工作也可以看成是一个独立的编译阶段,也可以看成是语义分析阶段的一部分工作,即把类型一致性检查看成静态语义分析,产生中间语言表示看成是动态语义分析。

中间语言表示有多种形式,这里用的是一种“三地址指令”形式。它很像是一台机器的汇编语言,机器的内存单元的作用同寄存器一样。(1.1)中的源程序可转换成下面的三地址代码:

```
temp1: = inttoreal(2)
temp2: = id1* temp1                                    (1.3)
temp3: = id2+ temp2
  id1 : = temp3
```

在第 7 章,将讨论编译程序中使用的几种中间语言表示及关于典型的程序设计语言构造的中间语言代码的生成算法。

符号表管理

在编译过程中,源程序中的标识符及其各种属性都要记录在符号表中。这些属性可以提供标识符的存储分配信息、类型信息、作用域信息等等。对于过程标识符,还有参数信息,包括参数的个数及类型,实形参结合的方式等等。

符号表是一种含记录的数据结构,通常一个标识符在符号表中占一个记录,记录中除了标识符的名字域之外,还有记录该标识符的属性的域。符号表在编译过程中使用频繁,是影响编译速度的主要因素。因此,对符号表的组织的要求是存储与查找的效率。

在词法分析阶段就可识别出标识符,因而就可将其填入符号表。但是,标识符的属性在词法分析时一般不能确定。例如,在 PASCAL 语言程序的声明中:

```
VAR     X, Y: REAL
```

标识符在前,类型在后,且标识符后不直接跟类型,因此,在词法分析看到一个标识符时,其类型还看不到。

在语义分析阶段及其以后,都要使用标识符的属性,因此,都要涉及查询与填表的问题。

出错处理

编译的每个阶段都会发现源程序中的错误。在发现错误之后,一般要对其有一定的处理措施,因而编译还可继续进行,不会一有错误就停止编译工作。

在语法与语义分析阶段,一般都处理编译中可发现的大部分错误。词法分析中可能发现字符拼写错误,但单词串违反语言的结构规则(语法)的错误要由语法分析阶段来查。在语义分析中,编译程序进一步查出语法上虽正确但含有无意义的操作的成分,如两个标识符相加,一个是数组名,一个是过程名,虽然语法上允许,但语义上不允许。诸如此类的各种错误,都在相应的阶段进行处理。

代码优化

代码优化的阶段力图提高中间代码的质量,使之运行速度提高,经语义分析之后,对于树表示中的每一内部节点都生成一条指令的直接代码生成算法,为源程序(1.1)生成中间代码(1.3),可以改为

```
temp1: = id1* 2.0
id1: = id2 + temp1                                     (1.4)
```

这里把转换 inttoreal(2) 直接写成 2.0,放在编译时做了,因而节省了运行时间。此外,由

于(1.3)的 temp3 只用一次,用 id1 代替 temp1 也没问题,这就省掉(1.3)中的最后的赋值。得到(1.4)的代码。

代码优化阶段的设计随编译程序不同而异。进行了大量优化工作的编译程序通常称作"优化编译程序",这种编译程序的大部分编译时间都花在这个阶段。但是,在实用上仍有许多简单有效的优化算法,能够显著地提高目标程序的运行速度而不明显地降低编译速度。

代码生成

编译的最后一个阶段,生成目标机代码,通常是可供装配的机器代码或汇编码。在代码生成过程中,对源程序中使用的变量要分配寄存器或内存单元,然后,为每一条中间语言指令选择合适的机器指令,包括确定机器指令的操作码或编址模式。

例如,利用寄存器 1 与寄存器 2,由(1.4)可生成代码

```
MOVF    id1         R2
MULF    # 2.0      R2
MOVF    id2         R1
ADDF    R2          R1
MOVF    R1          id1
```

每条指令的第一操作数是源操作数,第二操作数是目的操作数。指令中的操作码中的 F 表示此指令是浮点指令。本书在第 15 章讨论代码生成。

作为本节的一个小结,把上述对赋值语句(1.1)经历的整个编译过程绘于图 1.5。

编译过程的前面几个阶段,包括词法分析、语法分析、符号表管理、语义分析直至中间代码生成,依赖于被编译的源语言,因此,将这几个阶段归并在一起,称为编译程序的前端。代码优化中与目标机无关的部分也属于前端。前端中各个阶段中的错误处理部分当然也属于前端。

编译过程中同目标机有关的部分属于编译程序的后端。一般地说,它同源语言无关,只同中间语言有关。因此,后端包括代码优化中涉及目标机的部分,代码生成以及相关的错误处理及符号表操作。

为一个源语言做好一个前端以后,可以为若干个不同的目标机做不同的后端,由此可以得到一个语言在几个不同的目标机上的编译程序。

如果设计了一个高质量的后端,如果要在同一目标机上为另一种源语言设计编译程序时,也只要设计其前端,而无须重新设计后端。这些问题都涉及到大量语言的大量编译程序的实现问题。

图 1.5　一个语句的编译过程

1.4 程序语言的语法与语义

程序语言同编译程序有直接的关系。程序语言的完整定义包括它的语法描述及语义描述。语法是关于什么样的字符串才是该语言在组成结构上合法的程序的法则，语义是规定在结构上合法的程序的意义的法则。现代程序语言的语法一般都采用上下文无关文法作为描述机构。例如，上下文无关文法可以描述 A：= B+ C; 是一个合法的语句，而 A：= B+ 却不是。但是，上下文无关文法并不能描述全部程序结构特征。像类型一致性及作用域规则就不能用上下文无关文法描述。例如，A：= B+ C; 这个字符串虽然在语法上是正确的，但是，如果标识符 A，B，C 中有未经声明的或者 B，C 中有 BOOLEAN 类型的，A：= B+ C就是没有意义的。这种描述用上下文无关文法是无法做到的。

由于上下文无关文法的局限性，程序设计语言的语义成分通常分为两类：静态语义与动态语义。静态语义是一组限制，用以限定某些在语法上合法的程序的进一步合法性。例如，程序中的所有标识符在使用之前都要先行声明，表达式中的标识符与数的类型与其运算符的类型有兼容性(一致性)的要求，都属于静态语义中的限制。这些限制不能由上下文无关文法来刻划，对上下文无关文法是一种补充，完善对程序设计语言的定义的描述。静态语义的实施，即语法正确的程序行文是否符合静态语义的要求，通过程序行文的检查就可看出，因而可以在编译时进行，正因为如此，才称其为静态语义。程序的含义还包括程序是做什么的，刻划程序是做什么的，通常要通过程序的执行来表达，这就是动态语义，又称执行语义，运行时语义。不管是静态语义还是动态语义，其描述可以是非形式的，也可以是形式的。实际上，大部分程序设计语言的参考手册都包含了关于各种语言构造的语义描述，描述用的语言是普通自然语言。这种描述称为非形式描述。其特点是，易懂但不够精确，有时有歧义。语义的形式化描述是另一种重要的描述方法，它对于程序语言的设计与实现都有重要的意义。形式描述的特点是简练且准确，但难阅读，难理解。语义的形式描述有几种重要的方法：1. 属性文法与操作语义学；2. 指称语义学；3. 公理语义学；4. 代数语义学。

属性文法广泛用于描述程序语言的静态语义。例如，对产生式 E→E + T，可以为 E 及 T 设置一个表示类型的属性和一个要求类型一致性的谓词，如 E. type = t. type = numeric. 有一些编译程序自动生成系统是基于属性文法的，它们具有关于属性值的自动计算能力。

动态语义是描述程序是做什么的。其描述常常通过抽象机的操作或解释模型而进行。在这种模型中，定义程序的状态，程序的执行就是改变状态。维也纳定义语言 VDL[17] 就基于这样一种操作模型。VDL 曾用来定义 PL/I 语言的语义。

操作语义的描述建立在某种机器模型之上，因而语义的精确性受到限制。指称语义的描述方法摆脱了这个限制，建立在抽象、精确的数学概念的基础上，同操作语义方法相比，它显得更紧凑与准确。指称语义方法曾成功地运用于 ADA 语言的非并发成分的语义描