普通高等教育规划教材

程序分析技术

刘磊　等编著

CHENGXU FENXI JISHU

机械工业出版社
CHINA MACHINE PRESS

普通高等教育规划教材

程序分析技术

刘磊　张晶　单鄣　黄毅　编著

机械工业出版社

程序分析技术是以程序为处理对象，按需求对其进行各种分析的方法，在程序理解、程序测试、程序优化和程序重构等方面有着重要的应用。本书把当前常用的程序分析方法和作者的科研成果相结合，着重介绍了元程序设计、信息流分析、形式概念分析、别名分析、程序分片和部分求值等内容。本书可作为计算机本科生、研究生教材，或计算机专业高年级选修课教材，也可作为计算机科研与开发人员的参考书。

图书在版编目（CIP）数据

程序分析技术 / 刘磊等编著 .—北京：机械工业出版社，2005.7

普通高等教育规划教材

ISBN 7-111-16786-4

Ⅰ.程 ... Ⅱ.刘 ... Ⅲ.程序分析－高等学校－教材

Ⅳ.TP311.11

中国版本图书馆 CIP 数据核字（2005）第 067537 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）

责任编辑：王保家　版式设计：霍永明　责任校对：王　欣

封面设计：张　静　责任印制：杨　曦

北京机工印刷厂印刷·新华书店北京发行所发行

2005 年 8 月第 1 版第 1 次印刷

787mm × 1092mm $^{1}/_{16}$ · 10 印张 · 245 千字

定价：16.00 元

前　言

“程序分析技术”是一门非常实用的技术，深入地理解并熟练地掌握这门技术，对增强对程序设计语言的理解，掌握先进的程序设计方法，更好地分析和理解程序是有很大帮助的。因此，程序分析技术应是计算机软件专业人员必备的一门技术，同时也是计算机软件专业学生（本科生、研究生）今后进行科学研究的一门专业必修课。

本书的作者多年来一直为吉林大学计算机专业的研究生开设程序分析技术的课程，深受学生欢迎。遗憾的是，这门课缺少相应的教材，国内这方面的参考书也非常少，急需这方面的教材。我们根据自己多年的教学和科研的经验，在给研究生授课讲义的基础上，编写了此书。

程序分析技术包含的范围非常广泛，考虑到教学方便和学生所需，我们尽可能选取比较实用、应用范围较广、内容跟得上时代发展的程序分析技术。本书选取的程序分析技术有：元程序设计、数据流分析、信息流分析、控制流分析、部分求值、形式概念分析、程序分片、程序转换等。程序是与程序设计语言紧密相关的，为此，我们又特别增加了程序设计语言一章。

本书具有如下特点：

1. 在内容安排上，每一章介绍一个专题，每个专题自成体系，读者可以根据需要选取感兴趣的专题进行阅读。根据专题内容的不同，介绍的侧重点也各有不同：有的侧重原理的讲解、有的侧重应用实例的分析、有的侧重算法的设计。可以说做到了重点突出、讲解透彻，非常适合本科生高年级和研究生学习；

2. 对于书中的许多专题，在介绍基本内容的同时，又融入了许多编著者的科研成果。如元程序设计专题中，面向对象元程序设计方法；部分求值专题中，过程式语言的动静态部分求值技术；程序分片专题中，过程间的程序分片技术等；

3. 力争把最新的技术和方法介绍给广大的读者，如近年来比较热门的形式概念分析技术等；

4. 将程序分析技术的几方面的知识合为一体，读者可以通过学习一本书看到多方面的内容，免去了查找相关资料之苦。

本书可作为高等院校计算机软件专业的研究生和本科生高年级的教材，也可作为软件专业人员的参考书。

本书第1、5章由张晶编写；第2、3、7章由刘磊、单郸编写；第4、6章由刘磊、黄毅编写；全书由刘磊统稿。金成植教授对于本书的编写给予了大力支持和帮助，李碧涛等同学对全书进行了认真的校对，谨向他们表示衷心的感谢。

由于作者水平和时间限制，书中难免存在疏漏和不足之处，殷切希望广大读者批评指正。

编著者

目　录

第1章　程序设计语言

程序是一个指令序列。计算机程序是用计算机指令为计算机排定的工作顺序、工作步骤。计算机的控制器从程序的第一条指令开始，逐条取出指令进行解释，然后按指令的规定和要求指挥整个计算机系统工作。人通过程序的形式向计算机提出服务要求，计算机按程序自动进行工作，这是计算机系统最基本的原理。为计算机编排程序的过程称为程序设计。程序设计语言是指用于编写、描述计算机程序的语言。

对于从事计算机科学的人来说，懂得程序设计语言是十分重要的，因为当今所有的计算都需要通过程序设计语言才能完成。从第一台计算机诞生到现在的六十年间，大量的程序设计语言被发明、被取代、被修改或被组合在一起。尽管人们多次试图创造一种通用的程序设计语言，却没有一次尝试是成功的。之所以有那么多种不同的编程语言存在，是因为编写程序的初衷各不相同；新手与老手之间技术的差距非常大，而有许多语言对新手来说太难学；还有不同程序之间的运行成本（Runtime Cost）各不相同。

1.1　程序设计语言的四个发展阶段

迄今，程序设计语言的发展经过了机器语言、汇编语言、高级语言、第四代语言四个阶段。每一个阶段都使程序设计的生产率得到大大的提高。我们常常把机器语言称为第一代程序设计语言，汇编语言为第二代语言，高级语言为第三代语言，还有近代的第四代语言。

1.1.1　机器语言

一台计算机的指令系统是计算机最原始的，也是最基本的程序设计语言，我们称它为“机器语言”。机器语言是一台计算机的功能体现。

机器语言由能被计算机直接执行的机器指令组成，每条机器指令是一串二进制代码。用机器语言编出的程序是一串二进制代码序列。

例 1.1.1　若计算

$$Y=\begin{cases}X+15 & X<Y\\ X-15 & X\geqslant Y\end{cases}$$

用 Pentium 机器语言可编出如下程序片段（设程序从 100 号单元开始；X、Y 分别占用 116、118 号单元）。

```
1010 1001 0001 0110 0000 0001
0011 1100 0001 1000 0000 0001
0111 1100 0000 0101
0010 1101 0001 0101 0000 0000
1110 1010 0000 0011
0000 0101 0001 0101 0000 0000
```

```
1010 0011 0001 1000 0000 0001
                ……
0000 0000 0000 0000
0000 0000 0000 0000
```

机器语言的特点是结构简单、功能强大、可构造性强，计算机可以直接地识别和执行。但是，用这种语言的程序设计比较困难且繁琐，程序设计生产率很低。特别是它随计算机技术的发展而发展，不断更新，令人学习和使用应接不暇。基于上述原因，人们引进了汇编语言。

1.1.2 汇编语言

为减轻人们在编程中的劳动强度，20 世纪 50 年代中期人们开始用一些“助记符号”来代替 0、1 码编程。这种用助记符号描述的指令系统，称为符号语言或汇编语言。

汇编语言是符号化了的机器语言，即引进一些助记符表示机器指令中的操作码、地址等等。完成例 1.1.1 中同样计算的 Pentium 汇编语言程序片段如下：

```
      MOV    AX  , X
      CMP    AX  , Y
      JL     S1
      SUB    AX  , 15
      JMP    S2
S1:   ADD    AX  , 15
S2:   MOV    Y   , AX
            ……
      X  DW  ?
      Y  DW  ?
```

汇编语言用助记符而不是 0 和 1 序列来表示指令，程序的生产效率和质量都有所提高，但是计算机不能直接识别、理解和执行。用它编写的程序必须先翻译成机器语言程序才能被机器理解、执行。汇编语言用来编制系统软件和过程控制软件，其目标程序占用内存空间少，运行速度快，有着高级语言不可替代的用途。

汇编语言和机器语言，都与具体的机器有关，它们都称为面向机器的语言。程序员用它们编程时，不仅要考虑解题思路，还要熟悉机器的内部结构，并且要“手工”地进行存储器分配，编程的劳动强度仍然很大，还阻碍着计算机的普及和推广。因此人们又进一步引进了高级语言。

1.1.3 高级语言

1954 年出现的 FORTRAN 语言以及随后出现的高级语言，不再是面向具体的机器，而是面向解题的过程，以较接近于自然语言或专业语言的方式描述操作。例如使用 C 语言完成例 1.1.1 中同样的计算，可用如下语句：

```
if (X < Y)
  Y = X + 15;
```

```
else
  Y = X - 15;
```

显然用高级语言编写的程序可读性好，几乎就类似于英语句子。这种程序编起来自然很轻松，而且它还具有通用性，可以在不同机器上运行，程序便于移植。

不论是机器语言还是汇编语言都是面向硬件的具体操作，语言对机器的过分依赖，要求使用者必须对硬件结构及其工作原理十分熟悉，这对非计算机专业人员是难以做到的，对于计算机的推广应用是不利的。计算机事业的发展，促使人们去寻求一些与人类自然语言相接近且能为计算机所接受的语意确定、规则明确、自然直观和通用易学的计算机语言。这种与自然语言相近并为计算机所接受和执行的计算机语言称为高级语言。高级语言是面向用户的语言。无论何种机型的计算机，只要配备上相应的高级语言的编译或解释程序，则用该高级语言编写的程序就可以通用。

目前被广泛使用的高级语言有 BASIC、PASCAL、C、COBOL、LISP 和 PROLOG 等。

1.1.4　第四代语言

第四代语言（Fourth-Generation Language，以下简称 4GL）的出现是出于商业需要。4GL 一词最早出现在 20 世纪 80 年代初期的软件厂商的广告和产品介绍中，4GL 语言由于具有“面向问题”、“非过程化程度高”等特点，可以成数量级地提高软件生产率，缩短软件开发周期，因此赢得了很多用户。20 世纪 80 年代中期，许多著名的计算机科学家对 4GL 展开了全面研究，从而使 4GL 进入了计算机科学的研究范畴。

4GL 以数据库管理系统所提供的功能为核心，进一步构造了开发高层软件系统的开发环境，如报表生成、多窗口表格设计、菜单生成系统、图形图像处理系统和决策支持系统，为用户提供了一个良好的应用开发环境。它提供了功能强大的非过程化问题定义手段，用户只需告知系统做什么，而无需说明怎么做，因此可大大提高软件生产率。

进入 20 世纪 90 年代，随着计算机软硬件技术的发展和应用水平的提高，大量基于数据库管理系统的 4GL 商品化软件已在计算机应用开发领域中获得广泛应用，成为面向数据库应用开发的主流工具，如 Oracle 应用开发环境、Informix-4GL、SQL Windows、Power Builder 等。它们为缩短软件开发周期，提高软件质量发挥了巨大的作用，为软件开发注入了新的生机和活力。

虽然 4GL 具有很多优点和强大的优势，成为目前应用开发的主流工具，但也存在着以下严重不足：

(1) 4GL 虽然功能强大，但在其整体能力上却与高级语言有一定的差距。这一方面是语言抽象级别提高以后不可避免的（正如高级语言不能做某些汇编语言做的事情）；另一方面是人为带来的，许多 4GL 只面向专项应用。有的 4GL 为了提高对问题的表达能力，提供了同高级语言的接口，以弥补其能力上的不足，如 Oracle 提供了可将 SQL 语句嵌入 C 程序中的工具 PRO* C。

(2) 4GL 由于其抽象级别较高的原因，不可避免地带来系统开销庞大，运行效率低下（正如高级语言运行效率没有汇编语言高一样），对软硬件资源消耗严重，应用受硬件限制。

(3) 由于缺乏统一的工业标准，4GL 产品花样繁多，用户界面差异很大，与具体的机器联系紧密，语言的独立性较差（SQL 稍好），影响了应用软件的移植与推广。

(4) 目前 4GL 主要面向基于数据库应用的领域，不宜于科学计算、高速的实时系统和系统软件开发。

1.2 高级语言

高级程序设计语言是一种接近“人类语言”的语言，或用“类自然语言”(如类似英语的语言)，或用“数学语言”，或用两者结合的语言形式。高级语言可读性好，机器独立，具有程序库以及可以在运行时进行一致性检查从而检测程序中的错误，使得高级语言几乎在所有的编程领域中取代了机器语言和汇编语言。高级语言也随着计算机技术的发展而不断发展，据不完全统计，目前约有成百上千种用于不同目的的高级程序设计语言。

1.2.1 高级语言的分类

根据人们研究兴趣的不同，高级语言也有多种不同的分类方法。从程序设计方法学的角度来看，当今的大多数程序设计语言可以划分为四类。

1. 命令式语言 (Imperative Language)

命令式语言也称过程式语言。其特点是命令驱动，面向动作（语句），即把计算看作是动作（语句）的序列。一个命令式语言程序由一系列的语句组成，每个语句的执行引起若干存储单元中值的改变。Pascal、C 和 ADA 都是典型的命令式语言。

2. 函数式语言 (Functional Language)

函数式语言注重程序实现的功能，而不是像命令式语言那样一个语句接一个语句地执行。程序的编写过程是从已有函数出发构造出更复杂的函数，对初始数据集应用这些函数直至最终的函数可以从初始数据计算出最终的结果。因此，函数式语言也称应用式语言。LISP、ML 和 Haskell 都属于这种语言。

3. 面向对象语言 (Object-Oriented Language)

面向对象语言是当今最流行、最重要的语言。它主要的特点是支持封装性、继承性和多态性等。把复杂的数据和对这些数据的操作封装在一起，构成对象；对简单对象进行扩充、继承简单对象的特性，从而设计出复杂的对象。对对象的构造可以使面向对象程序具有命令式语言的有效性，通过作用于特定数据函数的构造可以具有应用式语言的灵活性和可靠性。SmallTalk 是当代面向对象语言的典范，此外，C + +，Java 也是面向对象语言。

4. 逻辑式语言 (Logical Language)

逻辑式语言也叫基于规则的语言 (Rule-based Language)。逻辑式程序设计以“项”之间“关系”的定义、应用这些关系的事实以及从现存的事实中“推理”出新的事实的规则为基础。项可能是逻辑变量或者包含逻辑变量。事实和规则称为“子句”。逻辑式语言的程序由“子句列表”组成。最有代表性的逻辑式语言是 PROLOG。PROLOG 以逻辑程序设计为基础，以处理一阶谓词演算为背景。它文法简洁，表达力丰富，具有独特的非过程型语言（一个语句就相当于过程语言的一个子程序而并非算法的一步)，是一种具有推理功能的逻辑型语言。PROLOG 语言已被广泛地应用于关系数据库、抽象问题求解、数理逻辑、公式处理、自然语言理解、专家系统以及人工智能等许多领域。

1.2.2　高级语言的实现

计算机的指令系统只能执行自己的指令程序，而不能执行其他语言的程序。因此，想用高级语言，则必须有这样一种程序，它能把用汇编语言或高级语言写的程序（称源程序）翻译成等价的机器语言程序（称目标程序），我们称这种翻译程序为翻译器。汇编语言的翻译器为汇编程序，高级语言的翻译器为编译程序。

翻译器的“翻译”通常有两种方式，即编译方式和解释方式。编译方式是：事先编好一个称为编译程序的机器语言程序，作为系统软件存放在计算机内，当用户把由高级语言编写的源程序输入计算机后，编译程序便把源程序翻译成用机器语言表示的与之等价的目标程序，然后计算机再执行该目标程序，以完成源程序要处理的运算并取得结果。编译程序将源程序翻译成目标程序发生在编译时间，翻译成的目标代码随后运行的时间，称为运行时间。解释方式是：源程序进入计算机时，解释程序边扫描边解释作逐句输入逐句翻译，计算机一句句执行，并不产生目标程序。

高级程序设计语言的编译方式和解释方式如图 1-1 所示。

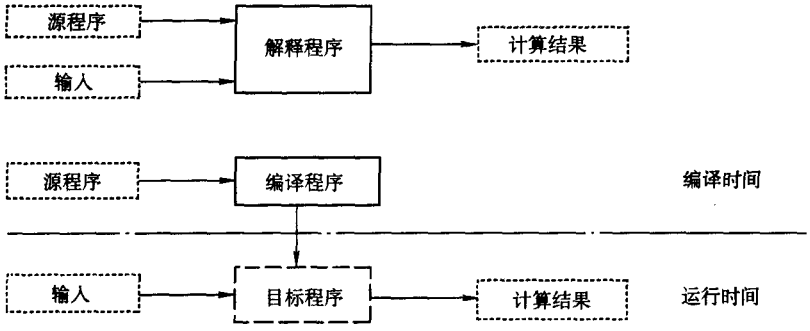


图 1-1　编译程序和解释程序

解释器是源程序的一个执行系统，而编译程序是源程序的一个转换系统；换句话说，解释程序的工作结果是得到源程序的执行结果，因此解释程序相当于执行程序的抽象机；而编译程序的工作结果是得到等价于源程序的某种目标程序，因此编译程序是高级语言程序到某种低级语言程序的转换器。

C、C++、VB、VC++等高级语言执行编译方式；Java 语言则以执行解释方式为主；而 C、C++等语言是能书写编译程序的高级程序设计语言。

1.3　未来的语言

程序设计语言的进一步发展是自然语言。要采用的那种自然语言对程序员只需要很少的（甚至不需要）程序设计训练。程序员将直接写或口述程序功能说明书，而与程序设计的结构和语法（产生程序指令的规则）无关。

目前研究人员正在致力于开发自然语言。在开始时自然语言将带有某些语法限制。虽然很难用几句话来概括未来的程序设计语言，但是可以预见未来的语言将是一种可以不受限制地在个人与计算机之间会话的语言。

第2章　元程序设计

2.1　元程序介绍

1. 元程序概念

在现代程序设计中，程序已经取代数据作为操作对象，变得越来越重要。元级程序设计系统是一个实现操作程序的有效工具，它可用于各种元程序设计。其中，需要一些对程序进行处理的基本操作，我们称之为元级操作。提供元级操作的实现语言，叫做元语言，所处理的语言，叫做目标语言。

元程序(Meta Program)是可以操作目标程序(Object Program)的程序，它可以构造目标程序，也可以将目标程序段组合成更大的目标程序，还可以观察目标程序的结构和其他特性。目标程序是以形式语言书写的一些句子，如常见的高级语言程序。

元程序是处理程序的程序，如大家所熟悉的编译器、解释器、类型检查器、定理证明器、程序生成器、转换系统和程序分析器等等。

2. 元程序和元程序设计系统的分类

从功能上看，元程序可分为两大类：程序分析器和程序生成器。程序分析器关注目标程序的结构和环境，并计算出一些值作为结果。这些结果可以是数据流或控制流图，或者是带有基于源目标程序的属性的另一个目标程序。该类型元级系统的例子如程序转换、优化和部分求值系统。而程序生成器则为了解决某一类具有相似解的问题，构造另一个可解决该类问题的程序(目标程序)。通常，生成的目标程序专用于一个特定的实例，这样比使用一个一般目的的、非生成的解方案所使用的资源少。通常，元程序还可以是程序分析器和程序生成器二者的混合体。

从元语言和目标语言的异同来看，元程序设计系统有两种截然不同的种类：元语言和目标语言相同的同类系统，元语言和目标语言不同的异类系统。两种系统对描述自动的程序分析和操作都是非常有用的，但相对于异类系统而言，同类系统具有更好的教学性和实用性，因为用户只需要学习一种语言就可掌握并操作整个系统。而且它还支持反射，能够提供 n-level 程序概念——一个 n-level 目标程序自身能够成为一个操作(n + 1)-level 的目标程序的元程序。而异类系统通常扮演更为重要的角色，带有固定的元语言的元级系统，无论其构造的是每个系统都具有不同目标语言的多重系统，还是一个带有多重目标语言的单一系统，都是非常具有实用价值的。

3. 元程序的使用

元程序设计为用户提供了很多便利：

(1)性能　大多数元程序设计系统中的一个普遍的目标就是性能。元程序提供了这样一种机制，它可以将源程序转换成一种易操作的中间表示形式，并提供多种针对这种表示形式的

操作，操作的不同组合形式就形成了源程序的不同解释执行。显然，比起写一个多用途的但却低效的程序，写一个能够从一个规范生成一个高效结果的程序生成器要好得多。

(2)部分求值　部分求值是用于提高性能的另一种元级程序设计技术，通过对程序输入的部分信息的例化，优化该程序，目的是在程序运行之前，识别并执行尽可能多的计算。

(3)解释　元程序最普遍的用途就是从一个目标程序到另一个目标程序的解释。源语言(Source Language)和目标语言(Target Language)可以相同也可以不相同。例如编译器和程序转换系统。

(4)推理　元程序的另外一个重要用途就是关于目标程序的推理。由于目标程序是以形式语言书写的句子，因此通过分析能够发现该程序的一些特性。利用这些特性，可以提高目标程序性能，为目标程序的行为提供保障。推理元程序的例子如各种流分析和类型检查的程序分析。此外，推理元程序还被用于构造定理证明系统，如 LEGO，HOL，Coq 和 Isabelle；逻辑框架的研究与实现，如 Elf，Twelf，LF。

(5)移动代码　最近，元程序设计逐渐作为程序传送的一种手段。不同于以前使用网络传送数据给程序，程序已经成为目前网络的传输对象。由于安全的原因，网络传输的内容是程序的内在表示，为了其安全性和保险性，可分析这些表示以确认它们没有危及所运行主机的完整性。这些被传送的程序是目标程序，分析程序是元程序。

4. 元程序的应用领域

作为一个领域，元程序设计已经有一段时间了，但作为一个正式的研究领域，它还仅仅在最近十年才成为活跃的前沿。下面是一些领域的概述：

(1)表示程序　源程序是数据，但它们是个复杂的实体。我们如何在表示它们的同时，保证隐藏一些不必要的细节，而使重要的机构更加明显？并且如何让它们的一般操作容易被表达和有效实现？

(2)表示形式　表示形式是元程序设计系统提供给程序员的到目标语言的接口。表示形式对于一个系统的可用性有着极其重大的影响。后面讲到的过程式的树型表示和对象式的抽象语法树表示分别为过程式和对象式的元程序设计系统提供了有效的表示形式。

(3)代码结构的观察　将代码表示为数据有很多种技术，其中大多数是将代码构造成一个抽象类型，目的是为了支持代码的双重性或为其提供一个更为有用的表现形式。由于这样的表现形式隐藏了代码的内部结构，因此如果这些代码需要被拆析或观察的话，必须另外提供某个可对该代码内部结构进行访问的接口。因此如何得到这样一个既易于使用，又能够反映用户对于目标代码结构的逻辑观点(重于如何实现的观点)的接口，是元程序设计在这个领域的研究要点。

(4)操作绑定结构　许多目标语言包含绑定结构——用于引入并描述局部变量的作用域。就元程序而言，这些变量的实际名字是非实质的，即不重要的。因此任何一个合法的名字，只要保持其使用方式一致，都可以被使用。但在元程序设计中，我们通常会遇到这样的问题：生成程序时需要创造一些新的名字，这些名字必须要保证与已经存在的名字不同，否则将会覆盖原有变量的作用域。可见，在程序分析中，必须要注意并处理某些发生概率极小的情况。因此，在处理带有绑定结构的目标语言时，我们面临的问题是：当拆析某一程序时，如何才能确保局部变量不会脱离其作用域而变得没有约束？

(5)异类元程序设计系统　在目标语言和元语言不同的异类系统中，我们能做些什么呢？

可以说,异类系统给整个元程序的研究带来了新的挑战。其研究的难点在于,在异类系统中,对于每一个目标语言,都需定义一套与之对应的新的不同的类型系统,如果元程序的类型要涉及一些它所产生的目标程序的类型,那么必须将目标语言的类型系统嵌入到元语言的类型系统中。而目标语言的类型系统以何种方式相容或相似于元语言的类型系统是无法保证的,因此,两种类型系统可能是不相适应的。例如元语言可以是多态的 lambda 演算,而目标语言是 Cardelli 的对象演算,一个简单的嵌入通常是行不通的。

(6)元程序理论 只有当我们掌握了我们希望构建的系统背后的理论后,才有可能构造一个可靠的系统。由于元程序中包含了很多在一般程序设计中不可能发生的复杂的语义上的难点,因此一套好的元程序设计系统理论必须要足够圆满来解决这些问题并且理解这些语义难点与其他系统特性之间的交互。

在后面的章节中,我们将介绍上述领域中较为基本的表示形式在元程序设计系统中的重要应用及其两种不同的实现手段。

2.2 元程序设计系统

2.2.1 元程序系统的组成

元程序的处理对象是程序而不是通常的数据,因此其处理过程要相对复杂得多。通常一个元程序设计系统由下面两个部分构成:

- 中间表示构造——将目标语言程序转换成内部表示;
- 元级操作部分——提供对于内部表示能够直接进行处理的元级操作。

其中,中间表示构造决定了该系统中目标程序到元级用户的接口特性,对于整个系统的可用性起到重要作用;而元级操作部分是系统的另一重要部分,用户主要使用系统所提供的这些元级操作对目标程序进行各种元程序设计(注意此时的操作对象已经不是目标程序,而是目标程序的内部表示结构)。

2.2.2 中间表示

正如前之所说,中间表示是元程序设计系统提供给程序员的到目标语言的接口,它的结构设计的好坏在元程序设计系统中起着极其重要的作用。在元程序设计中,中间表示可以有很多种形式,常见的有四元式、逆波兰式、树形结构和抽象语法树结构。

1. 四元式

一般结构为:(operator,operand1,operand2,result)

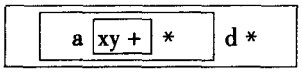
其中,operator 表示操作符;operand1,operand2 分别表示运算分量;result 表示运算结果。

对于运算型四元式,如(+,a,1,t1),其含义是将 operand1 和 operand2 在 operator 作用下得到的结果送入 result 中,此例中即为变量 a 和常量 1 的 + 操作结果存入临时变量 t1 中。而对于控制型四元式,是对于程序的执行起标志或控制转移的作用,此时 operator 位置已不再表示操作符,而为标志字,其他部分为空或者为控制转移条件表达式的结果。如(WHILE,_,_,_)则表示 while 循环的入口。

2. 逆波兰式

逆波兰式，即后缀式。表示方式为：表达式中各运算分量顺序不变且计算顺序排定；计算方式为：将表达式从左至右依次扫描，扫描到一个计算符后，向前取两个运算分量，计算出结果即可。

例如，算术表达式 a * (x + y) * d 的逆波兰式表示为：axy + * d *，按照下图中的方块，从内到外为计算顺序：



从中缀表达式到后缀表达式转换的实现可以通过 T 形的栈式结构(图 2-1)实现。

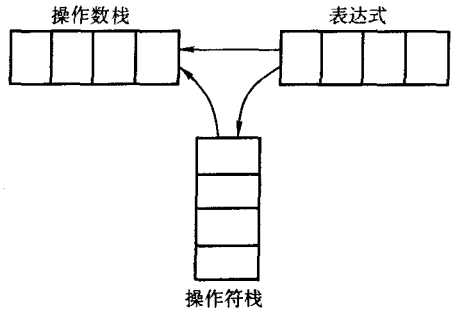


图 2-1　中缀表达式到后缀表达式转换时使用的 T 形结构

3. 树形结构

树形是按分支关系把信息项联系起来的数据结构。这种结构有一个称为根的节点没有前驱节点；除根以外的其余节点有且仅有一个前驱节点，并且对于非根节点都存在一条从根到该节点的路径。对于上面所举的例子，表达式 a * (x + y) * d 的树形结构表示如图 2-2、图 2-3 所示。

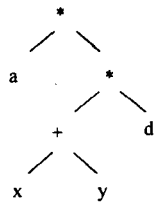


图 2-2　表达式 a * (x + y) * d 的右结合树

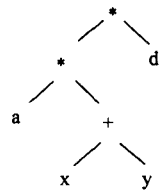


图 2-3　表达式 a * (x + y) * d 的左结合树

由于树结构便于进行元级操作并且比较直观，所以做程序分析和程序变换时比较适合使用这种结构来作为内部表示结构。

4. 抽象语法树结构

当元语言是面向对象的语言时，对象结构作为中间表示的基本数据结构是更为合适的。不同于之前提到的利用指针构造的树形结构，这种方法利用类，建立一种抽象语法树作为目标语言的内部表示。这种利用对象及其成员变量之间的关系建立起的树形结构与普通的树形结构相比，对于目标语言的描述更加清晰、细致，并易于施加各种操作。

下面关于对应于规则分类的结构介绍中，我们将主要以树形结构和抽象语法树结构为例进行描述。

2.2.3　规则分类和对应的结构

由于元程序设计是基于给定目标语言文法的，故先对文法规则进行分类，针对每种不同的规则来构造不同的内部表示结构。此处，我们使用类似语法树的树形结构来表示规则对应的内部结构。

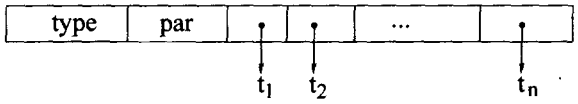
下面是常用的五种文法规则：

1. 结构规则(Constructor Rule)

$$< A_0 > \rightarrow \{N_0\} w_1 < t_1 : A_1 > w_2 \cdots w_n < t_n : A_n >$$

该规则描述了从非终极符 $< A_0 >$ 可以导出串 $w_1 < A_1 > w_2 \cdots < A_n >$。其中，$< A_0 >$，$< t_1 : A_1 >$，…，$< t_n : A_n >$ 是非终极符，$< t_i : A_i >$ 中 t_i 是标签名，A_i 是语法范畴；$w_1, w_2, \cdots, w_n$ 是可能为空串的终极符；N_0 是由大括号包围着的该规则的名字，可以省略。标签名用于区别属于同一个语法范畴的非终极符，因此一条规则中的所有标签名都必须是不同的。如果没有提供标签名，那么语法范畴的名字就会被当作标签名使用。

此类规则对应的树形结构的节点结构为：



- type——节点的种类(由此可以判定右侧的语法成分)；
- par——父节点的指针；
- t_i——指向儿子节点的指针($1 \leqslant i \leqslant n$)。

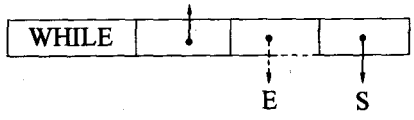
抽象语法树的节点类为：

```
● class  A0{};                    //结构规则左侧的非终极符对应的抽象类
● class  N0A0:A0{                 //以结构规则右侧所有成分为成员变量组成
         W1   w1;…                的类，是抽象类 A0 的子类
         Wn   wn;
         A1   t1;…          //其中 Wi 和 Ai 分别为右侧终极符 wi 和非终极
         An   tn;   };      符 < ti:Ai > 对应的各自的类
```

假设有 while 语句的产生式如下：

< while_stmt > →{all} while(< predicate:exp >)do < repetition:stmt >

对应该规则的树形结构的节点结构为：



对应的抽象语法树类结构为：

```
class WHILESTMT{};
class allwhilestmt:WHILESTMT{
TOKEN   while;
TOKEN   leftbracket;
EXP     predicate;
TOKEN   rightbracket;
TOKEN   do;
STMT    repetition;}
```

2. 分支规则(Alternation Rule)