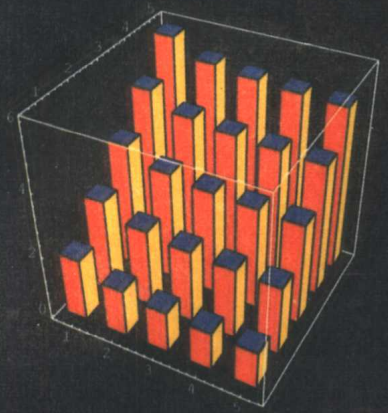
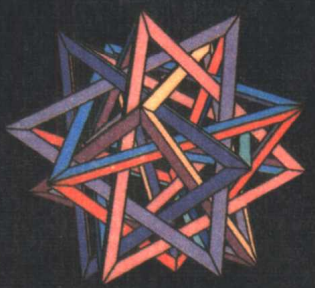
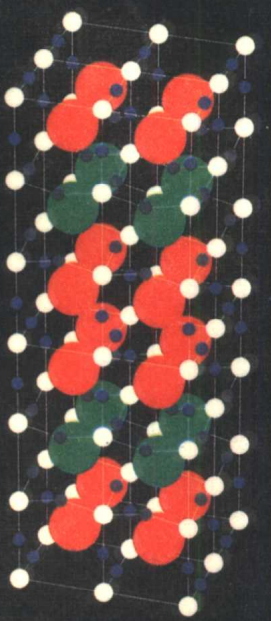
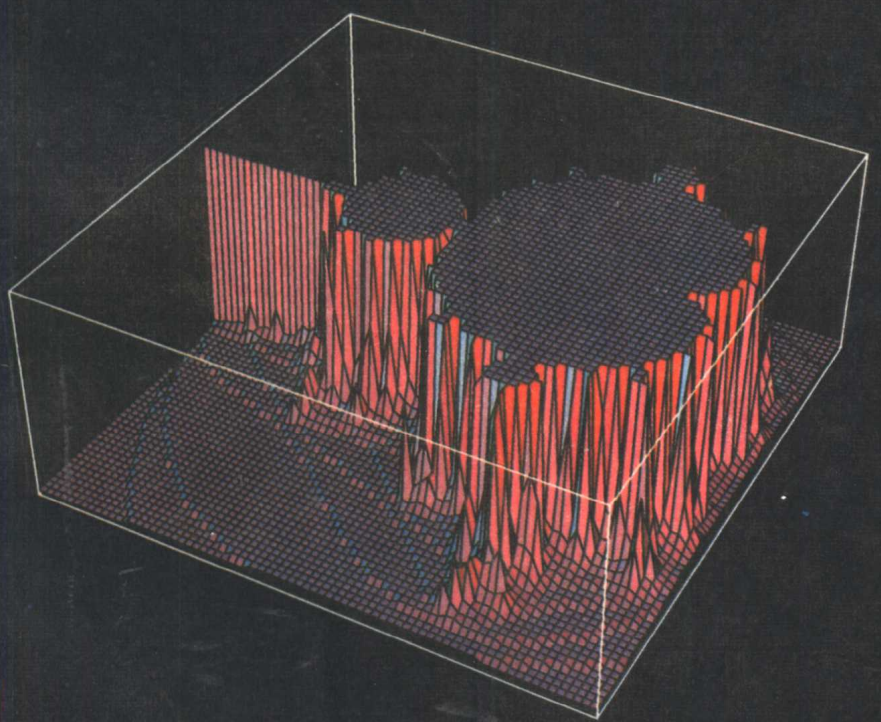
北京希望电脑公司高级程序设计语言丛书

微机高级程序设计语言的分类与剖析

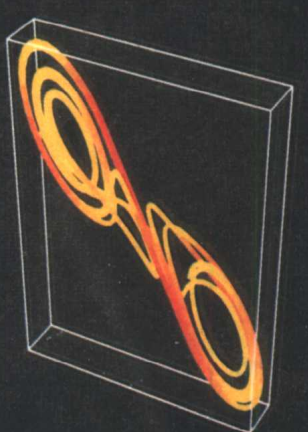








岳东　李南　编著
陈林　王坤　主校



海洋出版社

北京希望电脑公司高级程序设计语言丛书

微机高级程序设计语言的分类与剖析

岳东　李南　编著

陈林　玉坤　主校

- 程序设计最新动向
- 程序设计概念：值、存储、联结、抽象、并发性
- 程序设计分类剖析
- 命令程序设计 Pascal、C、Fortran
- 并发程序设计 Ada
- 面向对象程序设计 Smaltalk、C++
- 逻辑程序设计 Prolog

海　洋　出　版　社

1992.

内容摘要

高级程序设计语言发展到目前为止，已有数百种之多，因此，对于如何设计高质量的高级程序设计语言的方法的探讨，就显得十分重要了。本书从程序设计的概念入手，对当今比较流行的几个概念，如：值、存储、联结、抽象、并发性等做了介绍。然后结合 C、Pascal、SmallTalk、Ada、Fortran、Prolog 等语言，分别介绍了命令程序、并发程序、面向对象程序、逻辑程序的设计方法。本书资料新颖、内容丰富、读来饶有趣味，不失为软件开发人员的有益参考书。

欲要本书的用户，请直接与北京 8721 信箱联系，电话 2562329，邮码 100080。

(京) 新登字 087 号

责任编辑 刘莉蕾

微机高级程序设计语言的分类与剖析

岳东 李南 编著

陈林 玉坤 主校

海洋出版社出版（北京市复兴门外大街 1 号）

海洋出版社发行 双青印刷厂印刷

开本：787×1092 毫米 1/16 印张：13.8125 字数：302 千字

1992 年 9 月第一版 1992 年 9 月第一次印刷

印数：1-3000 册

ISBN 7-5027-2655-1/TP·93 定价：11.00 元

前　言

高级程序设计语言发展到目前为止，已有数百种之多，因此，对于如何设计高质量的高级程序设计语言的方法的探讨，就显得十分重要了．第一批高级程序设计语言诞生于本世纪 50 年代，从此程序设计语言就成为令人瞩目的研究领域．若干年来，计算机科学家们一直在寻求一种设计程序设计语言的好方法，这种方法既要有丰富的表达能力、又十分简单高效．程序设计语言的研究也常常被称为程序设计语言学，这是仿效自然语言学而产生的．自然语言学是研究自然语言的，这两者都有语法语义概念，但程序设计语言与自然语言有着天然的区别．前者在语言范围，表达能力和微妙性等方面都无法与后者媲美．另外，自然语言只是人们说、写、用的语言，故自然语言学者仅限于研究已存在的语言；而程序设计语言则是人为创造的，可在计算机上实现的语言．目前，人们已经获得一些阶段性成果，当然，还远未解决问题．笔者积数年研究之结果，同时结合国内外一些最新研究资料，撰成此书，意欲起到抛砖引玉之效果，以期国内外同行对此问题多加关注．

编者　1992.9

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第一章　绪　论

1.1　关于程序设计语言

第一批高级程序设计语言是本世纪五十年代设计的．从此程序设计语言就成为令人瞩目的研究领域，而且成果累累．程序设计人员热衷于宣讲他们喜爱的程序设计语言的优点，有时还带有狂热的宗教色彩．然而，从更高一层的学术观点出发，计算机科学家们一直在寻求一种设计程序设计语言的好方法，这种方法既要有丰富的表达能力、又十分简单高效．

程序设计语言的研究也常常被称为程序设计语言学，这是仿效（自然）语言学而产生的．（自然）语言学是研究自然语言的，这两者都有语法形式和语义概念，但程序设计语言与自然语言有着天然的区别．前者在语言范围，表达能力和微妙性等方面都无法与后者媲美．另外，自然语言只是人们说、写、用的语言，故自然语言学者仅限于研究已存在的语言；而程序设计语言则是人为创造的，可在计算机上实现的语言．

下面将要讨论说明程序设计语言所具有的性能和特点．

1.1.1　概念和范例

每一种程序设计语言都是人造产品，都是人们蓄意设计的，一些程序设计语言是由一个人单独设计完成的，如 Pascal，还有一些语言是由一大批人共同设计完成的，如 PL／1 和 Ada．经验表明，由一个人或一小部分人设计出来的程序设计语言要比一大批人所设计的语言更紧凑更一致．

一个程序设计语言如能名副其实，则必须满足一些要求：

首先，程序设计语言必须是通用的，这即是说如果一个问题能由计算机解决，则此问题一定能用该语言编写出程序．这似乎是条很苛刻的要求，但实际上即使是一个小小的程序设计语言也能满足这项要求；任何可以定义递归的语言都是通用的；从另一方面讲，如果一个语言不能具备递归也没有迭代，则这个语言一定不是通用语言．确有一些应用语言是不通用的，但把它们称作程序设计语言是不太合适的．

实际上，程序设计语言也应该是解决问题的"自然"语言，至少在该语言的应用领域中必须如此．例如，某语言的数据类型仅是数字型和数组型，则此语言自然而然地适于解决数字问题，而用此语言来解决商业数据处理或人工智能问题则是强它所难了．对程序设计语言的一个更基本的要求是该语言必须能在计算机上实现，即可以在计算机上执行每一个格式齐整的程序，而数学表达方式却不是"可实现"的，因为有些数学问题无法用计算机解决，同样，自然语言也不是可实现的，因为受许多因素阻挠，如：自然语言太不精确，而且具有二义性．

程序设计语言还必须能接受高效的实现方式，对于什么是可接受的高效，有许多不同

的意见，尤其语言实现的效率问题很大程度上受到计算机体系结构的影响，Fortran，C 和 Pascal 的编程人员希望他们编写的程序具有汇编语言程序效率的 1／5 到 1／2；Prolog 语言的效率略低，而它在其应用领域的性能良好，仍为人们称赞；人们希望将来能出现一种新的计算机体系结构，它将比传统体系结构更适用于处理 Prolog 程序，在本书中，我们将要研究设计程序设计语言的各种概念：值，存储、联结、抽象、封装、类型系统、序列和并发性。设计程序设计语言本身是一件很难做好的事情，即使有经验的计算机科学工作者也要经过慎重考虑才能下决心，但是，编程人员却能从学习研究这些概念中获益匪浅。程序设计语言是我们最基本的工具，所以我们一定要掌握好。当我们学习了一种新语言并发现它能有效地建造可靠性高、结构化程序好的程序时，以及当我们必须为给定的问题选择一个最适当的程序设计语言时，我们会认识到必须对语言概念有一个基本的了解，如果新语言与我们已掌握的语言有许多共同之处，则我们就能高效地学习并掌握该新的语言。

与单个概念同样重要的是如何把概念融合起来形成一个完整的程序设计语言，以及语言所支持的编程风格，如：命令编程，函数编程，逻辑编程等。在本书中我们也将研究这些范例。

1.1.2 语法和语义

每种语言都有语法和语义：

- 程序设计语言的语法说的是程序的形式，即：表达式、命令、说明等是什么样的，以及如何把它们组合起来成为一段程序。
- 程序设计语言的语义是说明程序的含义，即程序在计算机上运行的情况。

程序设计语言的语法影响着程序员书写程序的方式，供他人阅读程序和计算机对程序进行语法分析的方法，而语义决定了程序的组成，对程序的理解和计算机对程序的翻译。语法是重要的但语义更为重要。

在本书中，我们的注意力不放在语法问题上，一种给定的构造方式可能在整个语言中体现，但可能有一些表面上的变化。语义是极其重要的，我们要好好体味语义上的微妙之处，哪怕是几乎完全相同的构造方式的语义差别，我们必须认清一种给出的语言有没有混淆有不相同的概念，是否正确支持重要的概念或者根本就不支持。

在本书中我们把注意力集中在语义概念上，学习研究那些几乎对每一种程序语言来说都必须支持（或应该支持）的极为重要的概念。

为了避免纠缠语法上的变形，只要有可能我们就用 Pascal，ML 和 Ada 语言讲解每一个概念。对于 Pascal 语言我们要有一个正确的评价，此语言已届中年，且有许多设计不当之处，但它毕竟广为人知，而且它的缺点本身很有指点作用——本书中有很多例子和练习说到如何改进 Pascal 语言，Aad 是 Pascal 的后代，但 ML 却截然不同，它们都比 Pascal 摩登，但也均有瑕庇，理想的程序设计语言尚没有能设计出来，而且似乎永远不可能出现。

1.1.3 语言处理器

本书只讨论高级程序设计语言，即（多多少少）独立于机器的语言．这些语言由编译程序、翻译程序、或编译程序和翻译程序的某种组合方式译成机器语言来实现．

任何处理程序——执行或执行前的准备——的系统都叫语言处理器，语言处理器包括编译器、解释器以及一些辅助工具，如语法制导编辑器等．

程序设计语言必须是可执行的，但编程人员并没有必要深入了解它究竟是如何实现的．因此本书中忽略了语言的实现部分，除非实现方式对设计语言影响很大．

1.2 程序设计语言的发展

今天的程序设计语言是始于本世纪五十年代的研究产物，许许多多的语言概念被其后接二连三的语言所吸收，从而得到创新，测试和改进．除了极个别的情况，每一种语言的设计都要深受已存在语言的影响．对历史的简短回顾能使我们对本书中介绍的概念的演变发展史，以及主要程序设计语言的来龙去脉有一个整体印象，从中我们还可以认识到今天的语言并不是设计程序语言进程的最终产品，相反，有许多令人激动的新的概念还在不断产生，十年后的程序语言设计的面貌可能与今日的面貌大为不同．

图 1.1 总结了几个重要语言的出现时期和它们的先辈．因为不是对语言家族的综合考查，所以图中仅列举了主要的几个语言．部分影响这些重要语言的小的先导语言因空间关系就没有列举．

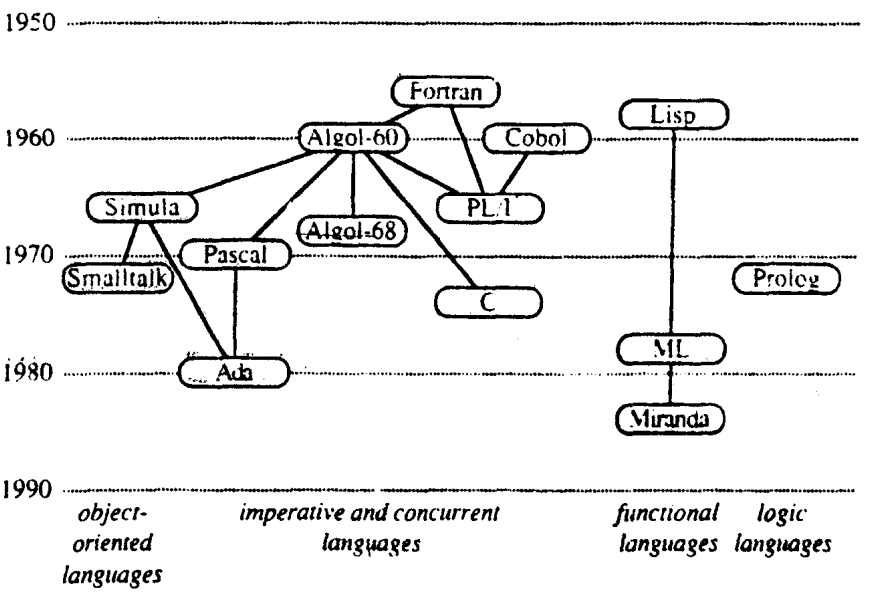


图 1.1 主要程序设计语言的产生年代及其相互关系

最早的主要高级语言是 Fortran 和 Cobol，Fortran 引入了符号表达式和带参数

的子程序的概念，而 Cobol 则引入了数据描述体的概念．这些语言的其它方面（在其原形式中）的层次都相对地低了些，如控制流在很大程序上受到跳转语句的控制．（ Fortran 和 Cobol 以及其它许多语言都经历了很长时间才从原版发展起来）．

第一个为通讯算法而不仅仅局限在一台计算机上编程的主要程序设计语言是 Algol-60．它引入了块结构概念，从而变量、过程等均可以由程序在需要的地方说明．Algol-60 极大地影响了后来的语言，以至后来的语言常被称作为类 Algol 语言．

Fortran 和 Algol-60 适于数字计算，Cobol 则适用于商务数据处理，PL／1 语言的设计目标是想成为一种通用语言，兼有上述几种语言的特征，同时引入了低级异常处理和开发机制以及其它一些特性，结果，PL／1 语言成为庞大、复杂不紧凑、难以实现的语言．PL／1 的经验告诉我们，一味地堆积特性不是一种创建高性能通用语言的好方法．

要想获得表达能力，最好方法是选择一组恰当的概念并把它们系统地组合起来，这就是 Algol-68 设计原则，程序员可以定义整数型数组，数组的数组，过程数组等，同样可定义过程，能调用或返回整数，数组过程等．Pascal 语言可以说是最流行的类 Alogol 语言，因为 Pascal 简单，系统化并能有效地实现． Pascal 和 Algol-68 是第一批语言中具有丰富的控制结构，丰富的数据类型和类型定义的语言．

Pascal 语言的优秀继承者 Ada，引入了包和类属概念（为了便于构造大的模块程序）以及高级异常处理和开发机制．和 PL／1 相同， Ada 的设计师们也想把 Ada 设计成为一种标准的通用语言，这种想法或许太鲁莽，Ada 也遭到许多批评，如 Tony Hoarer 曾称赞 Pascal 和更早期的 Algol-60 一样成功地吸取了先辈语言的优点，然而 Ada 只是比 PL／1 好得多，这对 Ada 的研究是有指导作用的．

程序设计语言的发展历史有某些倾向，其中之一是倾向于设计高抽象的语言．汇编语言的辅助记忆法和符号标识符是从操作码和机器地址中抽象而成；变量和赋值是从存储器存取和刷新抽象而成的；数据结构是从存储结构抽象而来的．控制结构是由跳转抽象来的；过程是由例程抽象出来的；块结构和模块是由封装抽象出来的，并有助于提高程序的模块化；类属把程序的某些部分从值的类型中抽象出来，以便程序具有重用性．

另一个倾向是程序设计范例的激烈增长．迄今我们提及的语言还都是命令语言，其特点是通过使用命令来刷新变量．今天命令语言仍然占统治地位，但其他范例正在快速流行起来．

Smalltalk 基于对象类型，作为变量的对象只能由与它关联的操作来存取．Smalltalk 是一种面向对象语言的例子，它是高结构层次的命令语言，它的全部程序都由这种对象类构造而成．面向对象的编程方式最初是在 Siuulla 语言中出现的，它也是一种类 Algol 语言．

无需求助变量就能解决大型问题是由 Lisp 语言体现出来的，Lisp 的出现是令人瞩目的，该语言（其纯粹的形式）整个基于表和树的函数之上，它是函数式程序设计语言的祖先，ML 和 Miranda 是摩登的函数语言，它们把函数当作第一类值处理，同时也包括高级类型系统．

综上所述，数学表达方式其本身在计算机上是不能实现的，但许多语言设计者都想方设法地在程序设计语言中使用数学表达方式的一个子集．逻辑程序设计语言是一个基于数学逻辑的子集的一种语言．计算机处理程序只关心值与值之间的关系，而不是从输入值中

计算出输出值来．Prolog 使逻辑程序大众化，Prolog （其纯形式）有弱点且并不高效，但是 Prolog 也在不断改善，在增添了非逻辑的特征之后，Prolog 作为一种编辑语言就更有用了．

此为试读，需要完整PDF请访问：www.ertongbook.com

第二章　值

数据是计算机的原料，就实际意义而言，数据至少和程序一样重要．大量的数据，如电话号码薄、字典、人口普查数据、卫星采集的地球数据，比起处理它们的、相对简单的程序其经济价值要高得多．因此，在计算机科学中把数据研究看作是有其自身法则的一项重要课题就不足为奇了．

本章我们研究值的类型、程序语言中的值是作为数据操纵的．类型包括基本类型、复合类型和递归类型．类型系统包括实施于值上的操作和可在程序中把老值算成新值的各种表达式．第三章我们还将研究怎样存储值，第四章研究值如何和标识符结合．

2.1　值和类型

术语值（value）的使用在计算机科学中是很广泛的．本书把值归为有如下性态的那一类：它可以求值、可存储、可与数据结构结合、可作参数传递到过程或函数、可作为函数返回值等等．换句话说，值可定义为计算期间存在的某个实体．

例如，在 Pascal 中我们可以找到下述几类值：

- 初等量（真值、字符、枚举值、整数、实数）
- 复合量（记录、数组、集合、文件）
- 指针
- 对变量的引用
- 过程抽象和函数抽象

前面三种很明显，初等量、复合量、指针值对上面提及的所有行为差不多都行，但要包括变量引用、过程抽象和函数抽象似乎令人吃惊，因为这些实体不能存储，也不能与数据结构结合．我们把它们算作值是因它们可以作为参数传递（我们把抽象作为过程、函数的集合术语，类似于其他语言的实体）．

在 ML 中，我们可以找到以下几类值：

- 初等量（真值、整数、实数、字符串）
- 复合量（元组、记录、结构、表、数组）
- 函数抽象
- 对变量的引用

ML 的函数抽象和对变量的引用完全可以象其他值一样使用．

我们发现把值象上述列表那样组成为类型是很有用的．举个例说，真值和整数就有明显的区别，适用于整数的加法和乘法操作就不适合于真值．

类型到底是什么呢？最显而易见的回答或许是：类型是值的集合，我们说 V 是类型 T 的值，意思仅仅是 V∈T，我们说表达式 E 是 T 类型，就等于断言 E 求值的结果将是类型 T 的值．这样，集合 {13,　true,　Monday} 就不是类型，而 {false,　true}

是类型，因为后者在逻辑'非'、'与'、'或'操作下展示出一致的行为．{…−2，−1，0，+1，+2=…} 也是类型，因为在加法，乘法等操作时，它们所有的值也展示了一致的行为．于是我们看到，类型不仅以其值的集合表征，还要由在这些值上的操作来表征．第六章和第七章我们要进一步研究类型的性质．

每种程序设计语言都有基本类型和复合类型，前者的值是原子的，后者的值由较简单的值组合成本，某些语言有递归类型，递归类型值是由该类型的其他值复合而成，后面两节我们将考察基本类型、复合类型和递归类型．

2.2 基本类型

基本类型的值是原子的，因而不能够分解为更简单的值．

程序设计语言选择了哪些基本类型就等于告诉了我们该语言的应用领域，商用数据处理语言（如 Cobol）是值为定长字符串、定点数的基本类型，数值计算语言（如 Fortran）是值为可选精度的实数．或许还有复数的基本类型．用于串处理的语言（如 Snobol）是值为变长字符串的基本类型．

相似的基本类型在不同语言中常以不同的名字出现，例如，Pascal 中有 Boolean Integer 和 Real；ML 中有 bool，int 和 Real，但这些名字的差别并不重要．为一致起见，我们把比较常见的基本类型用下述表示法表示：

$$Truth-Value = \{false,\ true\} \tag{2.1}$$

$$Integer = \{\cdots -2,\ -1,\ 0,\ +1,\ +2,\ \cdots\} \tag{2.2}$$

$$Real = \{\cdots -1.0,\ \cdots,\ 0,\ \cdots +1.0\cdots\} \tag{2.3}$$

$$Character = \{\cdots 'a',\ 'b',\ \cdots 'z'.\cdots\} \tag{2.4}$$

我们把真值统一地写为斜体．有些语言用黑体的字面量 False 和 True 表示．有的语言用预定义常量标识符（小写体）表示．

要注意，类型 Integer，Real，Character 都要由实现定义，也就是说，同一程序设计语言各不同的实现所定义的值的集合可以不同．Integer 是实现定义的整数域．Real 是实现定义的（有理）实数的子集．Character 是实现定义的字符集．从中可以看出硬件的限制及差异对高级语言的某些影响，这对想要写出移植程序的程序员会带来许多问题．

在 pascal 和 Ada 中，我们还可以定义枚举值的全新基本类型（更精确地说就是通过枚举标识符表示其值）．这种类型称之为枚举类型，其值叫做枚举值．

例 2.1

考虑下述 Pascal 类型定义：

```
type Month = (jan,  feb,  mar,  apr,  may,  jun,
              jul,  aug,  sep,  oct,  nov,  dec)
```

它定义了一个新类型，其值是 12 个枚举值：

```
month = {  jan,  feb,  mar,  apr,  may,  jun,
           jul,  aug,  sep,  oct,  nov,  dec}
```

这些枚举值和其他类型同名值是不同的.

要注意: 枚举值（按约定我们写为小写斜体）和在程序中表示它们的标识符之间的区别（程序中用小写）. 这种区别是必要的, 例如, dec 可在程序下文中重新定义表示其他实体（或许是整数递减的过程), 但枚举值仍然存在并能运算. 例如, 用'succ (nov) '即可得到它.

在 Pascal 中, 我们还可以用子界类型定义已有类型的子集. 例如, 子界类型'28..31'的值的集合 {28, 29, 30, 31} 是 Integer 的子集. 子集必须是连续值域. 7.6 节我们还要考察子类型更为一般的表示法.

一个离散基本类型是其值和整数（域）有一对一关系的基本类型. 在 Pascal 和 Ada 中这是很重要的概念, 任何离散基本类型的值就可以用于各种各样的操作, 为记数、情况选择、数组索引, 在大多数其他语言中, 只有整数可用于这些操作.

有时我们会对集合（或某类型）的基数感兴趣. 我们用#S 代表 S 中不同值的个数, 例如:

#Truth-Value = 2	(如上述)
#Month = 12	(在Pascal中)
#Integer = 2 * maxint+1	(在Pascal中)

2.3 复合类型

复合类型（或称结构数据类型）是一种类型, 其值由更简单的值复合构造而成, 程序设计语言支持各种各样的结构数据: 元组、记录、变体记录、联合、数组、集合、串、表、树、顺序文件、直接查找文件、关系等等. 这么多结构令人眼花乱, 实际上只要很少几个结构概念就可以理解这些类型, 这些概念是:

- 笛卡儿积 (元组和记录)
- 不相交的联合 (记录和联合)
- 映射 (数组和函数)
- 幂集 (集合)
- 递归类型 (动态数据结构)

本节讨论前面四个复合类型, 递归类型在第 2.4 节中讨论.

每个程序设计语言都提供自己的表示法来描述复合类型, 我们下文中介绍的数学表示法是按上述方式定义复合性的集合的最简单、标准、适用的表示法, 该表示法的表达能力是以描述数据结构的各种变体为基础.

2.3.1 笛卡儿积

最简单的一类值的复合是笛卡儿积, 其中两个（可以不同的）类型的值组成对偶. 我们用表示法 SXT 代表所有值的对偶集合, 其中每一对偶的第一个值取自集合 S, 第二个值取自集合, 形式地:

$$S \times T = \{ \quad (x,\ y | x \in S;\ y \in T\} \tag{2.5}$$

这可在图 2.1 中说明

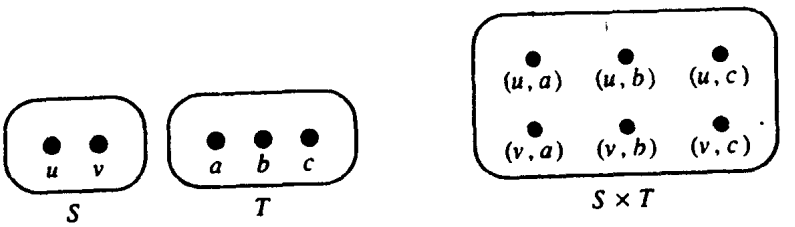


图 2.1　两集合的笛卡儿积

对偶上的基本操作只有选择它的第一成分和第二成分.

笛卡儿积的基数极易推断:

(S × T) = #S × #T

这就是我们把 × 记号用于笛卡儿积的原因.

可以把笛卡儿积的概念从对偶扩充到三元组，四元组等等. 一般地，表示法 S1 × S2 ··· × Sn 代表所有 n 元组的集合，其中每个 n 元组的第一成分选自 S1，第二个选自 S2，如此等等.

ML 中的元组，Cobol、Pascal、Ada、ML 中的记录，Algol-68 和 C 中所谓的结构都可以理解为笛卡儿积，形如:

```
record
        I1: T1;
            ···
        In: Tn
end
```

的 Pascal 记录类型，其值是集合 T1 × ··· × Tn 的 n 元值.

例 2.2

以下 Pascal 记录类型:

```
type  Date = record
                m:   Month;
                d:   1···31
            end
```

其值的集合为 Date = {jan, feb; ···, dec} × {1, ···, 31}. 这些值有下述的 372 对:

```
(jan, 1)        (jan, 2)        (jan, 3) ··· (jan, 31)
(feb, 1)        (feb, 2)        (feb, 3) ··· (feb, 31)
        ···                             ···
(dec, 1)        (dec, 2)        (dec, 3) ··· (dec, 31)
```

请注意，Date 中的某些值和真实的日期不对应，它只是真实世界写照的近似模型，是一种普遍现象.

标识符 m 和 d 用来访问记录类型 date 的单个成分，如:

```
var  someday:   Date;
···
someday.d:  =29;    someday.on:  =feb
```

这两个命令分别访问了 someday 的第二和第一成分. 象 m 和 d 这样带点的标识符可减轻程序员必须记住各成分次序的负担.

ML 有直接和笛卡儿积对应的表示法，元组类型'$T_1 \times T_2 \cdots \times T_n$'有值 $T_1 \times T_2 \cdots \times T_n$ 的集合.

例 2.3

以下是 ML 的元组类型

```
type  person = string * string * int * real
```

其值的集合为 person = string × String × string × integer × Real. 类型 person 的 someone 其值可分解如下:

```
val (surname, forename, age, height)  =someone
···
if  age>18  then···else···
```

这样，若要访问元组内某个成分，程序员就必须记住它在元组类型中的位置. 类型定义并未为使用单个成分提供什么线索. 如，我们往往一时记不清 person 的第一成分是名还是姓.

为此，ML 又提供了记录类型

```
type  person = {surname   : string,
                forename  : string,
                age   : int,
                height   : real}
```

实际上，这就和 Pascal 的表示法一样了.

笛卡儿积的一种特殊情况是所有元组成分均取自同一集合．这种情况的元组我们称它为同构的．写为:

$$S^n = S \times S \times \cdots \times S \tag{2.7}$$

就是所有成分均取自集合 S 的同构 n-元组集合，同构 n-元组的基本数是:

$$\#(S^n) = (\#S^n) \tag{2.8}$$

这就是采用上标表示法的原因．

最后再分析一个非常特殊的情况，n＝0．由等式（2.8）知 S0 的值为 1．该值是 O-元组（ ），即什么成分都没有的元组．我们以后会发觉定义一个类型用以下单值组合表示是很有用的:

Unit＝{ ( )}

Unit 在 ML 中对应为 unit，在 Algol-68 和 C 中为 void．请注意 Unit 并非空集，它是单一元组，只是它没有成分．

2.3.2 不相交的联合

另一类复合值是不相交的联合，其值取自两个（通常是不同的）类型．用表示法 S+T 表示这种值的集合，其中每个值都取两个，来自集合 S 和集合 T．为了标明值的来源，我们给各个值加上标签，形式地:

$$S + T = |left\ x|x \in S|U|right\ y|y \in t \tag{2.10}$$

其中取自 S 的值标以 left，取自 T 的值标以 right．加上标签只是为了区分各值的来源，因为它们要有区分，要不然就可任选了．

不相交的联合可由图 2.2 说明．

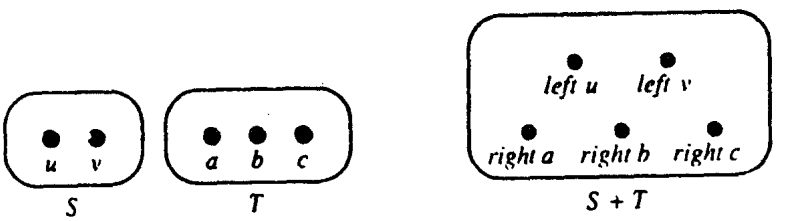


图 2.2 两集合的不相交的联合

对 S+T 中值的基本操作是:（a）测试它的标签以判明它是取自 S 还是 T．（b）投影还原为原有集合值，根据具体情况还原为 S 或 T．例如，若测得值为 right b 我们就可知其来自 T，因而投影还原为 T 的 b 值．

不相交的联合的基数也容易推断

$$\#(S + T) = \#S + \#T \tag{2.11}$$

这就是采用记号⊕+⊕来表示不相交的联合的原因．

可以把不相交的联合扩充到任意多个集合上．一般地，表示法 S1+S2+…+Sn 代表一

此为试读，需要完整PDF请访问：www.ertongbook.com