函数程序设计语言

计算模型　编译技术　系统结构

郑纬民　周光明　田新民　　编著

清华大学出版社
广西科学技术出版社

(京)新登字 158 号

内 容 简 介

本书详细论述了函数程序设计语言的计算模型、编译技术以及面向函数程序设计语言的系统结构。其主要内容包括:函数程序设计语言,λ演算,函数语言的归约计算方法,函数语言的图重写计算模型,图重写计算模型描述语言 CIL,多态类型及类型检查,自由变量的清除与优化组合子,惰性模式匹配及编译方法,严格性分析方法,编译时粒度分析及估算方法,存储管理及垃圾单元回收,多机并行图重写执行模型 HPGREM,并行抽象机 PAM / TGR 及体系结构等。

本书适合于从事计算机系统结构、信息处理、并行处理技术研究的科技人员阅读,也可作为计算机等专业的研究生和高年级本科生的教材和教学参考书。

图书在版编目(CIP)数据

函数程序设计语言:计算模型、编译技术、系统结构/郑纬民等编著. —北京:清华大学出版社,1996
ISBN 7-302-02234-8

Ⅰ.函… Ⅱ.郑… Ⅲ.电子计算机-函数-程序设计语言 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(96)第 11670 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:北京丰华印刷厂
发行者:新华书店总店北京科技发行所
开 本:787×1092 1/16 印张:17.25 字数:405 千字
版 次:1997 年 2 月 第 1 版 1997 年 2 月 第 1 次印刷
书 号:ISBN 7-302-02234-8/TP·1084
印 数:0001—2000
定 价:20.00 元

清华大学出版社　广西科学技术出版社
计算机学术著作出版基金

评 审 委 员 会

主 任 委 员　张效祥

副主任委员　汪成为　唐泽圣

委　　员　王鼎兴　杨芙清　李三立　施伯乐　徐家福
夏培肃　董韫美　黄　健　焦金生

出 版 说 明

近年来,随着微电子和计算机技术渗透到各个技术领域,人类正在步入一个技术迅猛发展的新时期。这个新时期的主要标志是计算机和信息处理的广泛应用。计算机在改造传统产业,实现管理自动化,促进新兴产业的发展等方面都起着重要作用,它在现代化建设中的战略地位愈来愈明显。计算机科学与其它学科的交叉又产生了许多新学科,推动着科学技术向更广阔的领域发展,正在对人类社会产生深远的影响。

科学技术是第一生产力。计算机科学技术是我国高科技领域的一个重要方面。为了推动我国计算机科学及产业的发展,促进学术交流,使科研成果尽快转化为生产力,清华大学出版社与广西科学技术出版社联合设立了“计算机学术著作基金”,旨在支持和鼓励科技人员,撰写高水平的学术著作,以反映和推广我国在这一领域的最新成果。

计算机学术著作出版基金资助出版的著作范围包括:有重要理论价值或重要应用价值的学术专著;计算机学科前沿探索的论著;推动计算机技术及产业发展的专著;与计算机有关的交叉学科的论著;有较大应用价值的工具书;世界名著的优秀翻译作品。凡经作者本人申请,计算机学术著作出版基金评审委员会评审通过的著作,将由该基金资助出版,出版社将努力做好出版工作。

基金还支持两社列选的国家高科技重点图书和国家教委重点图书规划中计算机学科领域的学术著作的出版。为了做好选题工作,出版社特邀请“中国计算机学会”、“中国中文信息学会”帮助做好组织有关学术著作丛书的列选工作。

热诚希望得到广大计算机界同仁的支持和帮助。

清华大学出版社
广西科学技术出版社
计算机学术著作出版基金办公室

1992 年 4 月

序　　言

计算机是当代发展最为迅猛的科学技术，其应用几乎已深入到人类社会活动和生活的一切领域，大大提高了社会生产力，引起了经济结构、社会结构和生活方式的深刻变化和变革，是最为活跃的生产力之一。计算机本身在国际范围内已成为年产值达 2500 亿美元的巨大产业，国际竞争异常剧烈，预计到本世纪末将发展为世界第一大产业。计算机科学技术具有极大的综合性质，与众多科学技术相交叉而反过来又渗入更多的科学技术，促进它们的发展。计算机科学技术内容十分丰富，学科分支生长尤为迅速，日新月异，层出不穷。因此在我国计算机科学技术尚比较落后的情况下，加强计算机科学技术的传播实为当务之急。

中国计算机学会一直把出版图书刊物作为学术活动的重要内容之一。我国计算机专家学者通过科学实践，做出了大量成果，积累了丰富经验与学识。他们有撰写著作的很大积极性，但相当时期以来计算机学术著作由于印数不多，出版往往遇到不少困难，专业性越强越有深度的著作，出版难度越大。最近清华大学出版社与广西科学技术出版社为促进我国计算机科学技术及产业的发展，推动计算机科技著作的出版工作，特设立“计算机学术著作出版基金”，以支持我国计算机科技工作者撰写高水平的学术著作，并将资助出版的著作列为中国计算机学会的学术著作丛书。我们十分重视这件事，并已把它列为学会本届理事会的工作要点之一。我们希望这一丛书能对传播学术成果、交流学术思想、促进科学技术转化为生产力起到良好作用，能对我国计算机科技发展具有有益的导向意义，也希望我国广大学会会员和计算机科技工作者，包括海外工作和学习的神州学人们能积极投稿，出好这一丛书。

中国计算机学会

1992 年 4 月 20 日

前　言

目的

函数程序设计语言具有良好的数学基础，程序简洁、易于进行程序推理和正确性证明，与软件工程、人工智能、并行处理、新型计算机系统等领域密切相关。函数程序设计语言受到有关研究人员的普遍重视。当前制约着函数语言进一步广泛应用的关键因素是其执行效率，因此本书将全面讨论函数语言的计算模型、编译技术以及面向函数程序设计语言的系统结构。

内容

本书共 14 章，分成技术基础、计算模型、编译技术和系统结构 4 大部分。

第 2, 3, 4 章介绍函数程序设计语言实现的技术基础。它将从函数语言的描述形式、函数求值的语义和语用等方面对函数程序实现的有关问题进行讨论。这几章将为读者学习后续章节打下必要的基础。

第 5, 6 章介绍函数程序设计语言的计算模型。它对函数程序的组合子图归约计算模型进行了深入的分析，这种计算模型是目前已知的最高效的函数程序计算模型，是国内外研究的重点。

第 7, 8, 9, 10 章介绍函数程序设计语言的编译技术。它对函数程序中的多态数据类型、复杂的模式匹配、函数计算表达式中的严格性和自由变量等描述成分的编译转换算法进行了讨论。这些编译技术对函数程序的执行效率有直接的关系。

第 11, 12, 13, 14 章讨论面向函数程序设计语言的系统结构。它从更具体的层次上，如系统资源组织、存储管理、执行模型、并行抽象机等，考察如何提高函数程序的执行效率。

致谢

本书中的许多内容是清华大学计算机科学与技术系近几年来参加有关国家 863 高科技项目和国家自然科学基金资助项目所取得的成果。作者特别感谢王鼎兴教授、沈美明教授、汤志忠教授、温冬婵副教授、鞠大鹏讲师、杜晓黎博士、刘德才博士和“并行图归约智能工作站”课题组的其他同志。还要感谢博士生傅强和硕士生胡孺蔚为本书的第 2 和第 3 章提供了初稿。

作者感谢清华大学出版社、广西科学技术出版社计算机学术著作出版基金评审委员会对出版本书的支持。作者自知专业水平有限，其中定有不当或欠妥之处，务望读者不吝给以指正。

编　者

1995 年 9 月于清华园

目　　录

第 1 章　引　　论

函数式程序设计语言具有良好的数学基础、程序简洁、易于进行程序推理、程序变换和正确性证明,与软件工程、并行处理、人工智能、以及近年来蓬勃发展的具有智能处理机制的计算机系统等研究领域密切相关,因而受到许多研究人员的重视。本章首先介绍函数式程序语言,其次阐述实现函数式程序设计语言的基本方法和技术。

1.1　函数式程序设计语言

函数语言的诞生以 McCarthy 等人在 60 年代中设计和实现的表处理语言 Lisp 为标志,Lisp 的核心部分采用了 Church 在 30 年代提出的 λ演算(λ calculus)作为基本计算方法,并被公认为是第一个函数式程序设计语言。J Backus 在 1977 年的图灵奖获奖演说中对函数式程序设计语言的研究给予了很高的评价,他认为函数语言的诞生是解放软件生产力的一个里程碑。函数语言可以使软件设计者从研究"怎么干(how to do)"转为研究"干什么(what to do)",从而摆脱采用传统语言的"逐字工作方式",大大提高软件生产率。

函数式程序设计语言是面向问题的,它以某种形式化的方法描述问题求解的行为,主要描述需要干什么(what to do)。程序执行的重要特征是具有引用透明性(referential transparency),即变量只与值相关,与物理存储地址无关,变量与值环境一一对应。这一性质消除了程序与机器结构的相关性,使得程序员在进行编程时无需考虑机器的操作行为和状态变化,只要按有关语言的文法要求用数学和逻辑的方法给出问题的描述和目标即可求解。从执行控制的角度看,函数式程序的计算行为蕴含在其形式化描述中,而非显式表示[Backus 1978]。另外,由于函数式程序的执行不存在赋值操作所引起的副作用(side-effect),因此为开发和管理函数式程序中潜在的计算并行性提供了有力的支持。

传统的程序设计语言是面向机器状态的,它以变量的赋值操作为核心描述怎样进行问题求解(how to do)。程序员在编程过程中需要把问题求解转化为具体的面向机器的操作,从而极大地增加了程序员的负担。Backus 和 Darlington 等人认为传统的程序设计语言是一种命令式程序设计语言(imperative programming languages),其操作语义是对复杂机器状态变化的描述,这给程序的正确性验证和软件维护带来了许多困难。此外,由于命令式程序的执行中赋值操作所产生的副作用,使开发和管理程序中的计算并行性变得复杂和困难[Backus 1978][Peyton 1990]。表 1.1 从两类程序设计语言的语用角度给出了两类语言的定性对比。

表 1.1　函数语言与命令式语言的语用对比表

特性 / 语言	程序设计	描述能力	执行控制方式	程序正确性验证	软件维护
函数式语言	容易	较强	自然地蕴含在程序中	容易	比较容易
命令式语言	较难	较弱	需在程序中显式描述	较难	比较困难

函数程序设计语言(functional programming languages)是作为说明性程序设计语言有

代表性的一类语言,自 McCarchy 提出 Lisp 语言后,函数语言的发展一直为众多的研究人员所关注。函数语言以 Church 等人提出的 λ演算为理论基础,其核心是函数表达式的作用[Turner 1979]。目前主要的函数语言有 APL, FP, ML, Lisp, SASL, KRC, Miranda, Haskell。下面分别扼要介绍这些函数式程序设计语言。

1.1.1 APL 语言

Iverson 提出的 APL 语言虽然不是纯函数程序设计语言,但是它的函数子集却导致许多人很自然地以函数方式进行程序设计,而无需依赖 λ表达式。这一结果超出了 Iverson 预先所设想的设计一种用于数组运算的代数程序设计语言(Algebraic Programming Language)。尽管 APL 在后来的发展过程中引入了许多命令式特征,但不可忽视 APL 固有的函数性质。此外 APL 得以流行的另一个很有趣的原因是,语言中定义了一个特殊的字母集,每个字母对应一个操作符,而这些字母在当时各种计算机终端的键盘上都有。Backus 所提出的 FP 语言可以说在很大程度上受了 APF 设计思想的影响,两者有很多共同之处,所不同的是 FP 的基本数据结构是 sequence,而 APL 的基本数据结构是 array。值得关注的是,近年来不少研究人员通过在 APL 中引入纯函数成份,而提高 APL 的描述能力。比如 Turner 采用了绝大部分的 APL 文法和程序设计原理,通过引入无穷数组支持惰性计算语义。有关 APL 更详细的介绍请参阅文献[Iverson 1962]。

1.1.2 FP 语言

Backus 提出的 FP 语言是最早引起人们广泛关注的一种函数语言,尽管 FP 的许多语言特征在许多现代函数语言中未被采用,但 Backus 在 1978 年的图灵奖演说仍是迄今为止在函数语言研究领域中的一篇最有影响和引用最为广泛的论文。Backus 将 FP 系统看成是用函数形式表示的操作集合,例如一个求内积的 FP 程序定义为:

Def IP = (/+) (α×) Trans

其中符号‘α表示将操作‘×’作用于 Trans 中所有的元素对,符号‘/’表示在元素之间插入操作‘+’,符号‘ ’表示复合操作,即(f g)xy = f(g(x y))。FP 语言现在已经有了很大的发展,如引入了强类型、抽象数据类型、高阶函数和允许用户自定义数据类型等,但 FP 的研究者们仍然强调它固有的代数性质。

1.1.3 ML 语言

70 年代中期,当 Backus 在 IBM 研究 FP 语言的同时,Edinburgh 大学的研究人员设计出了一种用于递归函数的自动证明系统 LCF。LCF 的程序设计采用了多态谓词演算 PP-λ(Polymorphic Predicate Calculus)和一种交互式元语言(Meta Language)[Gordon 1978]。LCF 的设计者们很快发现 ML 可以独立地作为一种函数语言使用。尽管 ML 的 I/O 子系统具有副作用,而且不具有引用透明性,但从 ML 的程序设计风格这个角度看,ML 仍不失为一种具有实用价值的函数语言。Milner 和 Wikstrom 等人于 1984 年在 ML 的基础上定义包括高阶函数、模式匹配、简单 I/O 功能、模块系统和例外处理的 ML 标准文本 SML(Standard ML),从而进一步提高了 ML 语言的实用性。

1.1.4 Lisp 语言

McCarthy 在 50 年代后期提出 Lisp 语言的主要动机是设计一种用于人工智能研究的代数表处理语言,虽然当时有关符号处理的思想还相当原始,但 McCarthy 的目标很具有语用性。McCarthy 在 Lisp 语言设计中的主要贡献是:(1) 在递归函数定义中引入了条件表达式的表示方式;(2) 提出了一类诸如 mapcar 的作用于表的高阶操作子;(3) 提出了构造子 cons 和无用单元回收(garbage collection)技术;(4) 采用 S-expression(S-表达式)表示程序和数据。迄今为止,在任何一个已实现的 Lisp 系统中,这些特色都必包含在内。下例是一个含有上述特色的 Lisp 程序。

```
(define mapcar (fun list)
    (if (null list) nil
        (cons (fun (car list)) (mapcar fun(cdr list))) ))
```

值得注意的是函数 fun 是作为参量传递给高阶函数 mapcar 的,例如对于一个给定的函数调用(mapcar fun (cons a (cons b nil))),其结果为(cons(fun(a) (cons fun(b) nil))),并且在计算完成后表 list 将作为垃圾被自动回收。在现代函数语言如 Haskell 语言中,函数 mapcar 也采用类似的方式定义,定义如下:

```
mapcar fun[ ] = [ ]
mapcar fun(head: tail) = fun head: mapcar fun tail
```

在上面的函数定义中,符号‘: ’是构造子 cons 的中缀表示,‘[ ]’表示是空表。此外,由于 McCarthy 的主要兴趣在于设计实用型语言,因此 Lisp 具有许多语用特色,特别是引入了具有副作用的赋值语句和一些原语。尽管如此,Lisp 语言对函数语言的发展具有不容置疑的巨大影响,而且 Lisp 语言也在不断地向纯函数方向发展,比如 Henderson 于 1980 年提出的 LispkitLisp 就是 Lisp 的一个纯函数版本。

1.1.5 SASL,KRC 和 Miranda 语言

在 Backus 和 Milner 等人开展 FP 和 ML 研究的同时,David Turner 教授领导的研究小组提出了三种新的函数语言 SASL(St Andrews Static Language),KRC(Kent Recursive Calculator)和 Miranda。Turner 等人的研究重点为如何使程序员能更方便自然地使用函数语言,因而在语言的文法定义中采用了类似数学函数定义的方式。例如:求阶乘(factorial)的数学函数定义(1)和 SASL 的文法定义(2)分别为:

(1) $fac\ n = 1 \quad if\ \ n = 0$
$\qquad = n * fac(n-1) \quad if\ \ n > 0$

(2)
```
fac n ⊙| = = 0 = 1
      fac n ⊙| > 1   = n* fac(n- 1)
```

考察(1)式和(2)式可以发现两种定义方式十分相似,在(2)式中通过在每个等式左部引入一个卫士(guard)建立了函数的条件定义,定义方式十分自然。此外 Turner 等人在 KRC 语言中还采用了 Darlington 提出的集合抽象,并允许程序员用[a. .]的方式表示无穷序列。Miranda 语言是 KRC 语言的进一步完善。特别是在类型处理方面 Miranda 语言采用了 HindleyMilner 类型系统,是强类型语言,支持高阶函数(high-order functions)和惰

性计算(lazy evaluation),允许用户自定义抽象数据类型(abstract datatype)。

1.1.6 Haskell 语言

Haskell 语言是一种通用的纯函数语言,它吸收了现有函数语言以及其它一些程序设计语言的最新研究成果,如高阶函数、惰性计算、静态多类型、用户自定义数据类型、模式匹配、表处理功能、纯函数型 I/O 系统、模块系统、丰富的数据类型等。迄今为止,Haskell 可以说是定义最为完整的一种函数语言,由于很多用户不习惯 λ演算的表达方式,因此 Haskell 提供了比 λ演算更为复杂的表示语言静态和动态行为的指称语义。目前,人们所关心的是 Haskell 语言是否能成为现代函数语言的一个标准和被程序员广泛采用。

1.1.7 其它的函数语言

前几小节介绍了几种较为典型的函数语言,实际上从 70 年代后期到 80 年代初期人们还提出了许多其它的函数语言,如英国 Edinburgh 大学提出的 Hope 语言、美国 Utah 大学提出的 FEL 语言、美国 Yale 大学提出的 ALFL 语言、英国剑桥大学提出的 Ponder 语言、英国牛津大学提出的 Orwell 语言,其中最为著名的语言大概可以认为是由 Ediburgh 大学的 Rod Burstall, DavidMacQueen 和 Ron Sannella 等人设计和实现的 Hope 语言。他们的目标是设计和实现一种十分简单的函数语言,使程序员可以方便地编写清晰的可操作的程序。Hope 也是一种强类型语言,允许采用多态类型,但是所有的函数定义必须带有类型说明。除此之外,Hope 语言还具有模块处理、用户自定义混合数据类型的模式匹配、惰性表处理等功能。事实上,Hope 的许多语言特色来自于 SML 语言。

1.2 函数语言的基本特征和实现论题

函数语言与传统语言相比消除了状态(state)、程序计数器(program counter)、存储器(storage)等概念,并把"程序"视为一个作用于输入数据集的函数,函数的计算结果就是程序的输出结果,因此也称之为作用式语言(applicative languages)。下面讨论函数语言的基本特征和主要实现论题。

1.2.1 函数语言的基本特征

Backus 指出用传统语言编写的程序是语句的集合,而用函数语言编写的程序则是一系列函数定义的集合,即一些简单函数组成复杂函数,再由复杂函数组成程序。下面介绍函数程序的基本特征:

(1) 程序设计可以在一个比较高的层次上进行,从而使软件开发时间主要花费在算法设计而不是实现细节上,以提高软件生产率。

(2) 函数语言中无副作用(side-effect free),即程序中不包括采用别名(aliasing)和修改变元的成份。

(3) 函数语言具有丰富的数据类型(包括各种结构型数据类型),描述能力强,易于进行正确性验证。

(4) 函数程序中通常含有丰富的隐式并行性,易于检测,并适于在并行系统上执行。

(5) 函数语言的 I/O 功能和文件处理参力较弱，不利于解决“Real Applications”问题。

(6) 理论上通过跟踪函数程序的执行树，使函数程序的调试比较易于实现，但对于惰性计算，若采用传统的断点方式，由于计算序本身不直观，则容易造成混淆。

1.2.2 程序的执行效率

函数程序的执行速度给人的感觉一般说是比较慢，尽管函数语言的推崇者认为，由于函数程序内部的固有数据相关性具有局部性，并且函数的计算仅依赖于它的参量，因此便于有效地安放函数和它的参量使通讯开销最小，从而开发有效的并行性。但这并不能真正消除影响函数程序执行效率的原因，这些原因概括如下：

(1) 用链表代替数组后，原来对一个数组元素随机访问的时间开销由常量转为线性增长，即随机访问时间随链表的长度增加而线性增长。

(2) 高频度的函数调用和参量传递所引起的额外开销。

(3) 参量(尤其是结构型参量)的拷贝开销，垃圾回收开销和通讯同步开销。

减小上述各种额外开销、提高函数程序执行效率的有效途径是采用优化编译技术，如编译时的类型推导、共享参量分析和标记、严格性分析、程序变换等。目前编译优化是提高函数程序执行速度所普遍采用的技术途径，这种方法远比设计专用硬件来提高函数程序执行速度的方法有效而且经济。

1.2.3 主要实现论题

函数语言有效实现所涉及的问题较多，如处理机的互连结构、计算的驱动方式、程序和数据的表示方式等等，下面分别给予扼要的阐述。

(1) 处理机的互连结构：虽然函数程序对数据的引用局部性较强，但是处理好一个处理器对其它处理器的访问仍然十分重要，尤其是当程序执行过程中需要共享参量时，处理机间的互连结构就显得更为重要。比如采用共享总线，环型结构往往导致系统瓶颈而不能有效地支持函数计算具有的局部特性，相比之下采用 N-cube, Mesh 结构则效果较好。

(2) 计算的驱动方式：函数的计算序一般有 Top-down 和 Bottom-up 两种，Top-down 方式对应需求驱动(Demand-driven)方式，Bottom-up 对应于数据驱动(Data-driven)方式。需求驱动方式的优点是能避免大量的无用计算，能自然地支持无穷数据结构和函数程序惰性语义的实现，但是用于请求计算和同步的开销较大。数据驱动方式的优点是用于请求计算和同步的额外开销小，但是不能支持无穷数据结构和函数程序惰性语义的实现，而且存储开销大。顺序计算是数据驱动方式的一种特殊情况，但不开发计算并行性。

(3) 程序和数据的表示方式：函数程序与数据的表示与它的执行方式密切相关。若执行方式为传统的顺序方式，则程序和数据可分开，程序以传统的机器指令的方式表示。若执行方式为并行方式，则程序和数据可以统一地用数据流图，或者用组合子图表。目前比较有效和被普遍采用的一种方法是将程序和数据分开，程序以图结构机器码表示，数据用带标记的方式表示，这是一种具有实用性的综合性方法，如并行抽象机< v, G> -machine, PAM/TGR 都采用了这种表示方法。

(4) 并行性的开发和控制：为平衡函数程序的并行执行开销、通讯/同步开销和压缩各种额外开销，任务粒度的分析在进程(或任务)的创建和调度中具有十分重要的作用。有效