人工智能程序设计

李卫华　陈兆乾　潘金贵　编著

学出版社

人工智能程序设计

李卫华　陈兆乾　潘金贵　编著

科学出版社

1989

内 容 简 介

本书以 IBM-PC 机中英文编译型 LISP 语言 PC Scheme 为蓝本讲授人工智能程序设计。书中列有大量有实用价值的人工智能程序，内容涉及模式匹配、符号微分、推理技术、人机对弈、搜索方法、问题求解、机器学习、知识表示、专家系统、专家系统工具、离散事件仿真、自然语言分析、图示机器人规划等众多人工智能研究课题。概览全书，熟悉人工智能的学者能加深对人工智能理论方法的理解，提高动手研制人工智能程序的能力；不熟悉人工智能的读者能了解许多人工智能研究课题，激发学习、钻研人工智能的热情。

全书取材新颖、由浅入深、层次分明，一章讲一种程序设计方法，包括递归程序设计、分支程序设计、循环程序设计、转移程序设计、向量程序设计、符号程序设计、环境程序设计、表程序设计、数程序设计、串程序设计、宏程序设计、无穷概念程序设计、窗口菜单程序设计以及面向对象的程序设计等。各章后均附有习题，特别适于用作大学人工智能程序设计课程的教材，也可供计算机工作者参考。

人工智能程序设计

李卫华　陈兆乾　潘金贵　编著

责任编辑　刘晓融

科学出版社出版

北京东黄城根北街16号

武汉大学印刷厂印刷

新华书店北京发行所发行　各地新华书店经售

*

1989年8月第一版　开本：787×1092　1/32

1989年8月第一次印刷　印张：14.1/8

印数：00001—12,500　字数：324,000

ISBN 7-03-001271-2/TP·78

定价：4.95元

前　言

LISP 语言是一门历史悠久，用途广泛，功能极强的人工智能程序设计语言，自1960年由美国 John McCarthy 发明以来，国际上已有许多不同版本的 LISP 语言。在美国，LISP 语言有东岸 LISP 和西岸 LISP 之分，东岸 LISP的杰作是 MACLISP，SCHEME 和 COMMON LISP；西岸 LISP 的杰作是 INTERLISP。在 IBM-PC 机上，也有许多可运行的 LISP 语言，如：中英文兼用解释型宏 LISP 语言[6,7]，中英文兼用解释型 Golden Common LISP 语言，以及中英文兼用编译型 LISP 语言 PC Scheme等。本书将以后一种语言为蓝本讲授人工智能程序设计。

Scheme 是一种用于系统软件设计和开发的新型 LISP 语言。它由原美国麻省理工学院的 G.L.Steele Jr 和 G.J.Sussman 发明于1975年，由麻省理工学院、印第安纳大学、耶鲁大学以及德州仪器公司等15位来自大学和工业界的专家学者修订于1984年。自该语言问世以来，麻省理工学院一直用之作为计算机科学系和电子工程系的语言入门课程，耶鲁、印第安纳、德州奥斯丁等大学也用之作为计算机科学系大学生的语言必修课程。除用于教学外，该语言还广泛用于程序设计语义学、编译程序设计、程序证明、操作系统原理的研究以及人工智能领域。

PC Scheme 语言是美国德州仪器公司计算机科学实验室基于美国麻省理工学院 W. Clinger 1985年所编《关于 Scheme 修正报告的修正报告》在 IBM-PC 机上实现的编

译型 LISP 语言。1985年推出第一版，1986年推出第二版。本书作者随后开发出中英文兼用版本。中英文版本均可在512KB 以上的 IBM-PC/XT 机、IBM-PC/AT 机及其兼容机上运行。

在数据类型方面，PC Scheme 除具有传统的数、表、串、字符、符号、向量、记录、端口外，还增有流、过程、继续、环境和引擎。

在控制结构方面，PC Scheme 除提供了传统的顺序程序设计、分支程序设计、循环程序设计以及宏程序设计外，还提供了类似于 ALGOL 语言的块结构程序设计。

在交互环境方面，PC Scheme 提供了用于编辑程序、数据和文本的类似于 EMACS(Editing MACros) 的实时编辑环境，用于查错改错的调试追踪程序，以及用于在同一时刻将不同运行信息显示在屏幕不同区域上的窗口过程。特别是，在 PC Scheme 内，任何用户定义过程都可访问其它在 PC 机上运行的可执行程序，从而解除了用 LISP 语言会丢弃其它 PC 机软件的后顾之忧。

在时间空间方面，由于 PC Scheme 采用了优化编译、过程覆盖、程序压缩、快速装入等技术，从而比 PC 机上的宏 LISP 语言和 GC LISP 语言少占内存，其运行速度视不同应用较宏 LISP 语言和 GC LISP 语言快一至三倍。

在学习使用方面，PC Scheme 提供了大量关于符号处理、数值计算、图形显示、输入输出、窗口菜单等方面的标准过程。作者在本书中阐述了这些标准过程的形式和意义，编制了涉及面广、有实用价值的程序实例，配备了可直接在 PC 机上运行的系统与实例软盘，适于新手学习掌握，老手借鉴比较。例如，利用我们编制的窗口菜单程序，学生很好理解并实现类似于 Turbo PROLOG 语言的高级菜单程序。

在推广应用方面，国内外不少学者已用 PC Scheme 设计了一些有关机器翻译、符号代数、演绎推理、归纳推理、专家系统、决策支持系统、离散事件仿真系统、面向对象的程序设计系统、专家系统工具，以及自然语言理解等方面的应用程序。武汉大学计算机科学系人工智能研究室已用中英文 PC Scheme 语言研制出多知识表示、多推理方式、中英文兼用的通用型专家系统工具 PCEST 和编译 LISP 语言工具箱，这些人工智能软件已在国内外广为应用。

本书是我们给计算机系大学生讲授72学时“人工智能程序设计”课程的教材之一，它可在52学时内讲完，另20学时讲授《IBM-PC 机编译型 PROLOG语言》一书[8]。学生除应理解书中的程序实例、完成书中的多数习题外，还应能比较 LISP 语言和 PROLOG 语言各自的优缺点，并能上机用这两种语言分别编制求解同类问题的不同程序。

本书还是读者学习《专家系统设计》和《知识工程软件》等书籍的必备资料。在这两本书中，我们以中英文 PC Scheme 为蓝本介绍了使用专家系统工具设计专家系统的方法、专家系统实例以及专家系统工具的 LISP 实现。

编著本书期间，得到美国德州仪器公司 D. Bartley 先生、美国硅谷地区环太平洋国际电子公司 Joe J. Zuh 先生以及何学平、翁寄遥、张治萍、李志超、肖笑、杜军等同事的多方帮助，在此谨对他们表示衷心的感谢。

李卫华　陈兆乾　潘金贵

1988 年 7 月

目　录

第一章　递归程序设计

开始学 LISP，学生常有深不可测、望而生畏的感觉，似乎只有 BASIC，PASCAL 最易学习。其实，LISP 也不难掌握。它的结构清晰、层次分明，是一门更接近于数学语言的语言。本章第一节试就几个简单的数学问题讲述其 LISP 程序设计，第二节介绍如何上机运行这些已设计好的 LISP 程序。

§1.1　简单数学问题的程序设计

例1. 求正数 m 的整数 n 次幂。

用数学语言，此问题可描述为：当 $n=0$ 时，m 的 n 次幂为1；当 $n>0$ 时，m 的 n 次幂为 m 乘以 m 的 $n-1$ 次幂。用 LISP 语言，可把上段用数学语言描述的求幂公式描述为下面的程序：

```
(define power
  (lambda (m n)
    (if (=? n 0) 1 (* m (power m (- n 1))))))
```

程序中，lambda 用于定义新过程，power 是新过程名，define 把由 lambda 定义的新过程约束给过程名 power。该过程带两个形参，一个 m，一个 n。过程体为 if 引导的条件表达式，其中 =? 为判断两个数是否相等的过程，* 和 - 分别为计算乘和减的过程。

例2. 求表中数之和。

在 LISP 中，由 n 个元素 $x_1, x_2, \cdots, x_n$ 组成的表通常表示为：$(x_1\ x_2 \cdots x_n)$；表中第一元素由过程 car 获取，由过程 cdr 丢掉。表是否为空用过程 null? 判断。非表元素称为原子，用过程 atom? 判断。例如，由过程 number? 判断为真的对象是数原子，但不是表。

下面是求表中数之和的 LISP 程序，它用 define 的另一种方式定义了过程 addnumber，其形参为 lyst，其过程体为由 cond 引导的条件句。

```
(define (addnumber lyst)
  (cond ((null? lyst) 0)
        ((number? lyst) lyst)
        ((atom? lyst) 0)
        (t (+ (addnumber (car lyst))
              (addnumber (cdr lyst))))))
```

从上面两个程序例子的结构来看，每一过程定义体中都含有对其自身的过程调用。此类过程称为直接递归过程。一个过程 p_1，如果在其过程体中引用了另一过程 p_2，过程 p_2 在其过程体中引用了另一过程 p_3，…，过程 p_{n-1} 在其过程体中引用了另一过程 p_n，过程 p_n 在其过程体中又引用了过程 p_1，则称过程 p_1 为间接递归过程。直接和间接递归过程简称为递归过程。§9.3 中的 minimax-A-B 是间接递归过程的例子，该例中，过程 minimax-A-B 调用了过程 search1，过程 search1 调用了过程 search2，过程 search2 调用了过程 minimax-A-B。

递归是一种将复杂问题化为简单问题直至解决的问题求解方法，特别适于描述数学和符号处理问题。编写递归过程时，要注意构成递归过程的两个主要部分。一部分即基本情况，应能直接给出最简单问题的答案；另一部分即递归部

分，应把复杂问题化为较简单的问题。

在 例1 中，基本情况为：当 $n=0$ 时给结果 1。递归部分为：当 $n>0$ 时，把求解 m 的 n 次幂问题化为求解 m 的 $n-1$ 次幂问题。因为正数 n 每次少1，最终定能到达数0，从而得到问题的解。

在 例2 中，基本情况为：当 lyst 为空或为非数原子时给结果0，当 lyst 为数原子时给数本身，当 lyst 为非数原子时，给结果 0。递归部分为：当 lyst 为表时，把求解表 lyst 中数之和的问题化为求子表(car lyst)中数之和的问题和求子表(cdr lyst)中数之和的问题。因为每递归一次，实参表中的元素个数减少一次，所以最终总可到达基本情况直接得解。

例3. 求一组数中的最大数。

把这组数表示为表之后，求解此问题的 LISP 程序为：

```
(define (largest l)
  (define (largest-so-far max l)
    (cond ((null? l) max)
          ((>? max (car l))
           (largest-so-far max (cdr l)))
          (t (largest-so-far (car l) (cdr l)))))
  (largest-so-far (car l) (cdr l)))
```

此例是 块-子块 或 程序-子程序结构的例子。过程(largest l)是主程序，它返回数表 l 中的最大数。过程(largest-so-far max l)是子程序，它使参量 max 始终存放数表 l 中至今遇到的最大数。这种积木式程序有许多优点。其一，层次分明，结构清晰，便于调试。各子程序完成各自的任务，调试完后，将之组合、集成便可成为完成总目标的大程序。其

二，全局变量可以共享，局部变量可以同名。前者使过程执行时由形实替换耗费的时间得以节省，后者使程序的编制和阅读更加容易。特别是，使变量的动态约束概念在变量的静态约束语言 Scheme 中得以实现。其三，需要时调用方便，不要时删除容易。将主程序、子程序及各程序共享的变量约束放在一个以主程序名命名的文件内，用(load "主程序名")便可装入源程序，用 (define 主程序名 #!unassigned)便可把由主程序及其子程序占用的内存空间节省下来。

例4. 求下面各式的值：

a. $\sum_{x=1}^{10} x^3$

b. π

c. $\int_0^1 x^3 dx$, 其中 $dx=0.01$

求解问题 a, b, c 的数学公式分别为：

$$\sum_{x=1}^{10} x^3 = 1^3+2^3+3^3+\cdots+10^3$$

$$\pi = 8\times\frac{\pi}{8} = 8\times\left(\frac{1}{1\times 3}+\frac{1}{5\times 7}+\frac{1}{9\times 11}+\cdots\right)$$

$$\int_a^b f(x)dx = \left[f\left(a+\frac{dx}{2}\right)+f\left(a+dx+\frac{dx}{2}\right)+\cdots\right]dx$$

不难看出，上面各式可抽象地描述为算式：

$$\sum_a^b f(n) = f(a)+\cdots+f(\text{next})+\cdots+f(b)$$

其中第 i 项 f(next)中的 next 是一个与第 $i-1$ 项有关的函数。把通用算式定义为过程

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a) (sum term (next a) next b))))
```

之后，问题 a 可用表达式

```
(sum (lambda (x) (* x x x)) 1 1+ 10)
```

求解．问题 b 可用表达式

```
(* 8 (pi-sum 1 1000))
```

求解，其中 pi-sum 定义为：

```
(define (pi-sum a b)
  (define (pi-term x) (/ 1 (* x (+ x 2))))
  (define (pi-next x) (+ x 4))
  (sum pi-term a pi-next b))
```

问题 c 可用表达式：

```
(integral (lambda (x) (* x x x)) 0 1 0.01)
```

求解，其中 integral 定义为：

```
(define (integral f a b dx)
  (define (add-dx x) (+ x dx))
  (* (sum f (+ a (/ dx 2)) add-dx b) dx))
```

例 4 是一个高阶过程（即处理过程的过程）的例子．PC Scheme 允许过程作为实参传递，作为过程值返回，作为数据存入变量或某个数据结构之中．这些特征是构造高阶过程的基础，也是作面向对象程序设计的基础．

§1.2 程序的阅读、求值与打印

与本书相配的软件有两张软盘，一张系统盘，一张例子盘．有双软盘驱动器时，应把系统盘置入 A 驱动器，把例

子盘置入 B 驱动器。仅有单软盘驱动器时，要把系统盘置入 A 驱动器。有硬盘驱动器时，最好先用 COPY 命令依次把系统盘和例子盘的全部内容拷入 C 驱动器，并选用 C 驱动器为默认磁盘驱动器。

在 DOS 提示符下键入 PCS 并按回车键后，便进入英文版 PC Scheme。在 CCDOS 提示符下键入 LISP 并按回车键后，便进入中英文 PC Scheme。这时，屏幕上出现 PC Scheme 的版本信息和顶层提示符[1]。

之后，PC Scheme 便执行顶层 阅读-求值-打印 循环。即显示顶层提示符[i](这里的 i 为整数，初值为1)，从键盘中阅读一个表达式 exp_i，对 exp_i 求值，打印对 exp_i 的求值结果，i 值加 1，重复上述过程直至所阅读的表达式 exp_i 为过程

(EXIT)

时退出 PC Scheme 系统，回到 DOS 操作系统。

在循环时，可用过程

(%C i)

查出第 i 步所键入的待求值表达式 exp_i。可用过程

(%D i)

查出 PC Scheme 对表达式 exp_i 的求值结果。可用过程

(TRANSCRIPT-ON file)

把在此之后从键盘上阅读的表达式 exp_i 及其求值结果记录在由串 file 规定的磁盘文件中。若 file 为 "prn"，则把 exp_i 及其求值结果打印在打印机上。不再需要记录时，或需与 DOS 交互时，要用过程

(TRANSCRIPT-OFF)

关闭由 file 规定的磁盘文件或打印机。

在键入待求值表达式时，可使用下述功能键和控制键。