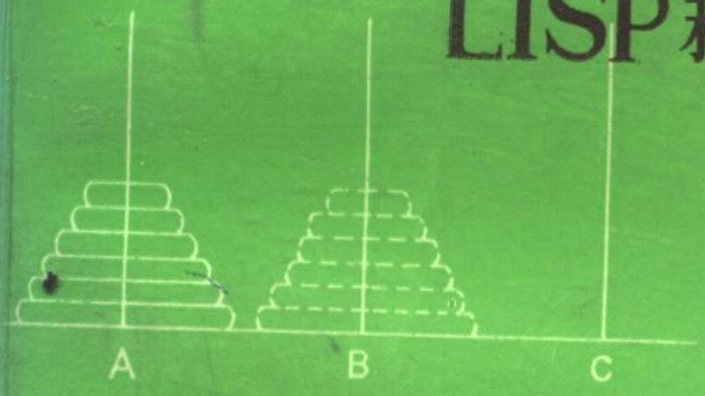
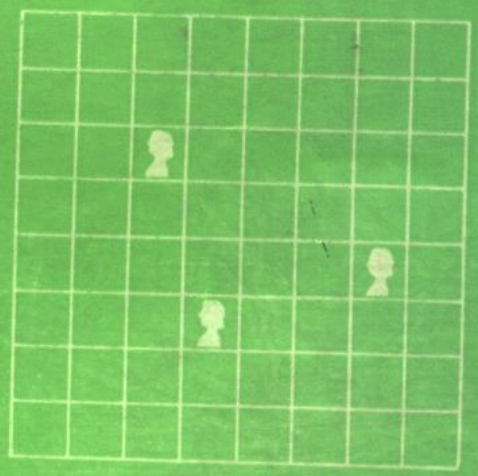
金 汉志 陈 珮珮 编写

人工智能程序设计

LISP和Prolog





南京大学出版社

人工智能程序设计
LISP和Prolog

金志权　陈珮珮　编写

南京大学出版社
1988·南京

内 容 简 介

本书深入浅出地介绍了两种最主要的人工智能程序设计语言 LISP 和 Prolog，通过各种例子说明如何使用这两种语言进行函数型和逻辑型程序设计。全书包括两篇，第一篇共十一章，介绍 LISP；第二篇分八章，介绍 Prolog。每一篇都是先描述语言，然后给出人工智能中的一些例子。

本书可作为大学计算机系有关专业大学生或研究生的教材或参考书，也可供从事计算机工作的科研人员、工程技术人员以及其他有关人员学习参考。

人工智能程序设计
——LISP 和 Prolog

金志权 陈珮珮 编写

责任编辑 丁 益

*

南京大学出版社出版

（南京大学校内）

江苏省国营练湖印刷厂印刷

江苏省新华书店发行 各地新华书店经销

*

开本：787×1092 1/16 印张：19 字数：473,000

1986 年 7 月第 1 版 1988 年 3 月第 3 次印刷

印数：10 001—20 000

ISBN 7-305-00060-4

TP.2 定价：4.20元

前　言

人工智能的研究工作，现已进入将其结果广泛应用于解决实际问题的阶段。作为该学科领域中应用得最为广泛的程序设计语言 LISP 和 Prolog，不仅对人工智能在计算机上实现具有重要意义，而且也是人工智能理论研究的重要工具。

本书包括 LISP 和 Prolog 两篇，彼此独立。每一篇都是首先描述语言的基本内容，然后通过若干中、小实例，说明如何应用它们进行人工智能程序设计。考虑到一些读者对人工智能可能不太熟悉，因此对每个例子，都是先简单介绍一下问题的本身，再阐明求解该问题的方法以及大致实现思想，最后给出相应的程序。目的在于使读者既能进一步掌握和巩固所学的内容，又可以了解人工智能的一个侧面。在前后两篇中，所选用的例子力求相同，以便于读者学习时比较对照，这样既可以熟悉函数型程序设计，又能掌握逻辑型程序设计。但为了保持各篇内容的独立性，在第二篇介绍 Prolog 的例子时，仍然概要地叙述一下问题和实现思想。

近年来，我国人工智能的研究工作发展得较快，从事这方面工作的人越来越多，而国内有关人工智能程序设计方面的书籍较少。为适应教学与科研工作的需要，根据有关参考资料，以及我们教学工作的一些经验，编写了本书。限于编者水平，书中一定会存在不少错误和不足之处，希望读者批评指出。

本书承南京航空学院夏振华教授和舒正忠同志审阅，并提出了许多宝贵的意见。在编写过程中，得到了我系陈世福副教授的热情帮助，他为本书的纲目和内容提出了有益的意见。徐洁磐副教授也对本书的内容提出了十分宝贵的意见。在此谨向他们表示衷心的感谢。还要感谢本系 84 届的研究生，他们对本课程的热心以及出色地完成所指定的作业，给予我们很大的帮助。其中陈未竞、翟成祥、周小方、姜馨杰、陈军、魏红(83届)、张豪煜、叶晓峰等同志协助编写了 §11.6，§17.5，§17.8，§18.2，§18.3，§18.4，§18.5 中的七个例子，并在 IBM PC 机上调试通过。

需要说明的是，书中的 Prolog 程序多数已用IBM PC 机上的 Micro Prolog 3.1(1984 版）编写通过；不少 LISP 程序也用 PDP-11 机上的 LISP-11 编写通过。但为了标准起见，编入本书时略作了些修改。例如，DEFINE 函数已把它改为通常的形式，而不用 LAMBDA 形式。

编　者　1985年6月
于南京大学计算机科学系

目　　录

第一篇　LISP 程序设计

第二篇 Prolog 程序设计

第一篇　LISP程序设计

第一章　引　　言

§1.1　LISP语言的历史及其重要性

LISP语言是迄今在人工智能学科领域中应用最广泛的一种程序设计语言，其名字取自于英文的表处理语言——LISt Processing Language。

LISP语言是麻省理工学院的John McCarthy和他的研究小组在1960年首先设计实现的，在LISP1.5程序员手册中描述了这个LISP系统。此后，在各种各样的计算机上都实现了LISP。LISP语言是由最初的LISP1发展到LISP1.5以及后来的LISP2，但LISP2并没有广泛地被接受。目前使用最广泛的版本仍然是LISP1.5。因此，LISP教科书都是以LISP1.5为基础的。

自从1960年J.McCarthy提出LISP以来，虽然其间也提出过一些新的人工智能语言，但就应用的广泛性和普遍性来说，LISP语言仍然是最主要的人工智能程序设计语言之一。由于现有的大量人工智能程序以及一些著名的人工智能系统都是用LISP写成的，因此从这种意义上说，LISP语言与FORTRAN程序设计语言具有相似之处。

在美国，一些大学开发了他们自己的LISP版本，目前应用得比较普遍的有：

MAC LISP：M.I.T.实现的LISP，其中运行在VAX计算机上的LISP称作Franz LISP。

Inter LISP：最初是在麻萨诸塞的坎布里奇的BBN实现的，后来由Xerox PARC（Palo Alto Research Center）维护。

UCI LISP：开始于M.I.T.，第一个文本在Stanford大学完成，以后由U.C. Irvine扩充。

Portable Standard LISP：由Utah大学开发的LISP。

前几年，美国的M.I.T.，Stanford，Berkeley等著名大学和DEC，Xerox等公司在1983年共同完成了Common LISP语言，它是在MAC LISP基础上发展起来的，并参考了Inter LISP语言，目前已被广泛使用。

LISP语言不仅对人工智能在计算机上实现有着重要意义，而且也是人工智能理论研究的重要工具。事实上，人工智能的研究与程序设计是无法分开的，因为它是把关于人类智能的假说通过程序体现出来，然后再将程序的工作结果与实际人的思考加以比较，进一步验证这些假设。

LISP现已用于：

- 符号代数处理
- 自然语言理解

* 机器翻译
* 形式逻辑推论
* 专家系统
* 自动定理证明
* 自动程序设计
* 机器人

目前世界上已有不少 LISP 机器。例如，Symbolics 3600, Xerox LISP, MIT机器。这些 LISP 计算机能够高效地运行 LISP 程序，具有很强的编辑、作图等功能。LISP 是这些计算机上唯一可用的语言。不少计算机科学家和工作者正在使用 LISP 机器从事人工智能领域中的各种研究工作。

§1.2 LISP 语言的特性

1973 年，Jean Sammet 曾说过："程序设计语言能够分成两类：第一类中有 LISP；第二类包括所有的其它程序设计语言。" 事实上，LISP 语言具有许多与其它程序设计语言比如 FORTRAN, ALGOL, COBOL, PASCAL 不同的特性。

现在人们把语言分为命令式语言（Imperative Language）和应用式语言（Applicative Language）两大类，后者常被称为函数式语言（Functional Language），这是因为应用式语言具有数学函数性质。命令式语言是在冯诺依曼型计算机体系结构基础上发展起来的，这种语言有其优点和固有的缺点，这里不作讨论。

本节将简单地介绍 LISP 语言的特性。LISP 语言是过去所有现存语言中最接近函数式语言的一种语言。J.McCarthy 在 1960 年提出的最初的 LISP 语言完全是函数型的，后来为改善在传统计算机上的执行效率，就在流行的 LISP 版本中，把非应用式特性加入了语言中。LISP 语言具有如下特性：

（1）LISP 程序的通常形式是一串函数定义，其后跟着一串带有参数的函数调用，函数之间的关系只是在调用执行时才体现出来。LISP 中没有语句概念，也没有分程序结构或其它语法结构。语言中的一切成分都是以函数的形式给出。

（2）在纯 LISP 中只有很少几个原始函数，虽然现有的 LISP 系统已增加了大量的内部函数，但这些新增加的函数都可以用最初的原始函数来表示。

（3）在 LISP 中，程序和数据在形式上是等价的。LISP 的唯一数据结构是 S-表达式（表），而程序本身也是用 S-表达式写的，因此可以把程序当作数据来处理，也可以把数据当作程序来执行。

（4）递归是 LISP 的基础，是语言的主要控制结构，它不象大多数程序设计语言那样以迭代（循环）作为主要控制结构。LISP 的递归处理是基于递归定义的数据结构。

第二章　LISP的数据结构——S-表达式

LISP是一种适合于符号处理的语言，它与一般高级语言如FORTRAN，ALGOL，PASCAL等有着很大的不同。LISP处理的唯一对象是符号表达式。这种符号表达式又称S-表达式，这里S代表符号（Symbol）。因此，LISP程序就是对符号表达式进行加工和处理的。

§2.1　原　　子

原子是S-表达式的最简单情况，它可分为符号原子和数原子。符号原子是以字母（A—Z）开头的字母数字串，可用来表示变量、常量和函数的名字等。数原子是一串数字，在其前面可冠以符号'－'或'＋'（＋号可缺省），分别表示负数原子和正数原子。为了严格起见，下面以BNF[①]形式来表示。

〈原子〉::＝〈符号原子〉|〔＋〕[②]〈数原子〉|－〈数原子〉

〈符号原子〉::＝〈字母〉|〈符号原子〉〈字母〉|〈符号原子〉〈数字〉

〈数原子〉::＝〈数字〉|〈数原子〉〈数字〉

〈字母〉::＝A|B|C|……|Z

〈数字〉::＝0|1|2|……|9

例	正确的原子	不正确的原子
	ABC123	2A
	12	a
	A4D6	$$g
	NIL	ABD.
	T	(A·B)

说明：（1）以上定义的数原子只限于整数原子。实际上，在一些LISP系统中，数原子又分为定点数原子与浮点数原子。定点数原子表示整数，浮点数原子表示实数。此外，除了符号原子、数原子外，有的系统还引入了串原子，通常表示成'字符串'。

（2）通常LISP规定原子中出现的字母一定是大写字母。

（3）LISP把原子NIL和T作为两个特殊原子，T表示逻辑真，NIL表示逻辑假，NIL也表示空表，即（）。

---

① BNF指Backus-Naur Form。

② 〔〕是任选符号，说明里面的符号可有，也可缺省。

§2.2 S-表达式的定义

S-表达式定义如下：

（1）原子是S-表达式。

（2）如果S_1和S_2是S-表达式，则（$S_1 \cdot S_2$）也是S-表达式。

我们把（$S_1 \cdot S_2$）称为S-表达式的点对表示，S_1、S_2分别称为S-表达式点对表示的头部（左部）和尾部（右部）。

应该注意，这个定义是一个递归定义。下面给出S-表达式的BNF表示：

〈S-表达式〉::=〈原子〉|（〈S-表达式〉·〈S-表达式〉）

例	正确的S-表达式	不正确的S-表达式
	A	A·B
	(A·B)	(A·B·C)
	(((A·B)·C)·(A·B))	((A·B))

说明：（1）S-表达式的点对表示，首先要有一个左括号，紧接着一个S-表达式，再跟一个圆点，一个S-表达式以及右括号。在上面最后一个错误的例子((A·B))中，（A·B）虽是S-表达式，但对于外层括号来讲少了一个圆点及一个S-表达式。这种错误可用原子NIL来改正，即改为((A·B)·NIL)。

（2）如果在LISP系统中允许有定点与浮点数原子，那么在写点对形式的S-表达式时需要注意。例如（3·1·2）可理解为（3·1 · 2）或（3 · 1·2），这种二义性往往依赖于具体系统的实现。

§2.3 S-表达式的表表示法

上节已经介绍过用点对表示法定义S-表达式。用点对表示法定义S-表达式显然是方便的，但当用它来表示一个长的S-表达式时，就会产生书写不简便，形式难看的问题。例如下面的S-表达式，不仅括号很多，而且有许多圆点。

(A·(B·(C·(D·NIL))))　　(1)

为了使S-表达式表示简洁明了，在LISP中S-表达式往往采用另一种表示法，即表表示法。表表示法是人们实际使用的一种表示法。式(1)的S-表达式若采用表表示法则可写成

(A B C D)　　(2)

我们称式(2)为一个表。由此可见，表是由左括号开始，后面跟着任意多个元素（原子或表），元素之间用一空格符号分开，最后以一右括号结束。

在点对表示(1)中，我们称A为头部，(B·(C·(D·NIL)))为尾部。在表表示(2)中，我们同样称A为头部，但尾部则为(B C D)。

表表示法的一般形式为：

（〈S-表达式〉〈S-表达式〉…〈S-表达式〉）

其中，每个〈S-表达式〉可以为原子，也可以为表。例如，(A (B C) (D))是表。

表中有三个元素，即一个是原子 A，另外两个是表（B C）和（D），叫做子表。不难看出，表的结构是嵌套的，定义是递归的。我们把最外层表中元素的个数定义为该表的长度。例如，表（A （B C）（D））的长度为 3。表元素是按次序排的，所以表是有序的，比如（A B C）不同于（B C A）。

在表表示法中有一种特殊情况。若表的长度为 0，亦即表中没有任何元素时，此表称空表，可写作（ ）或 NIL。所以，NIL 既可代表原子又可代表空表，特别要注意（ ）不是空表，而是以空表（ ）为元素的表。当用表表示法书写S-表达式时，要留心右括号与左括号的匹配，即任何一个左括号应该有一个右括号与之对应。但在某些实现系统中，有时可允许以一个特殊符号，例如右方括号 '〕' 来代替结尾处的任意多个右括号。

综上所述，S-表达式有两种书写表示方法，一种是点对表示法，另一种是表表示法。它们可用来表示同一对象，但形式不同，因此在这二者之间必定存在着某种转换关系。一般来说，点对表示法转换成表表示法有如下两条规则：

（1）（A · NIL）⇒（A） 也即（A · ( )）⇒（A）

（2）（A · (B C D … H)）⇒（A B C D … H）

这表明若点对的右部是 NIL 时，可把圆点与 NIL 去掉，剩下的部分构成一新表。若点对的右部是一张表时，此时称混合表，则可把点与右部表的括号去掉，剩下的部分组成一张新表。在转换中，原子仍保持原来形式。（A · B）这类S-表达式也保持不变，这是因为上面两个规则不能用于这种情况。

根据以上两条规则，给出若干例子来表明点对表示和表表示之间的关系。

	点对表示	表表示
（1）	A	A
（2）	（A · B）	（A · B）
（3）	（A · (B · NIL)）	（A B）
（4）	((A · (B · NIL)) · (C · NIL))	((A B) C)
（5）	((A · (B · NIL)) · (C · (D · NIL)))	((A B) C D)
（6）	((A · NIL) · (B · NIL))	((A) B)

下面是点对表示到表表示的转换过程：

上例3（A · (B · NIL)）⇒（A · (B)）⇒（A B）①

上例4（(A · (B · NIL)) · (C · NIL)）⇒（(A · (B)) · (C)）

⇒（(A B) · (C)）⇒（(A B) C）

上例5（(A · (B · NIL)) · (C · (D · NIL)))

⇒（(A · (B)) · (C · (D)）⇒（(A B) · (C D)）

⇒（(A B) C D）

由此可见，以 NIL 结尾的点对表示以及当点对表示中尾部为非空表时可转化为等价的表表示。反之，也可以把表表示转换成点对表示，只要逆过来应用上面的两条规则。从外层到里层逐级把表的尾写成 · ( ) 形式，把每个隐含的 NIL 都变成显式，即（A）变为（A · NIL）。例如，表（A B）⇒（A · (B)）⇒（A · (B · NIL)）。

① 为了便于理解，这里用符号⇒表示中间解释步骤。

表（A （B C） D）的点对形式为（A·((B·(C·NIL))·(D·NIL))），其转换过程如下：

（A （B C） D）⇒（A·((B C) D)）⇒（A·((B C)·(D)))

⇒（A·((B C)·(D·NIL)))　⇒（A·((B·(C·NIL))·(D·NIL)))

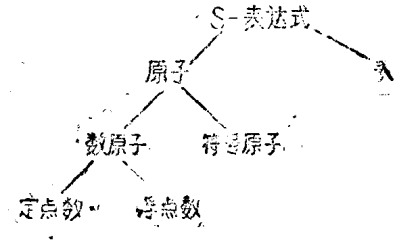
下面给出另一些例子。

表表示	点对表示
（A）	（A·NIL）
（A B）	（A·(B·NIL)）
（A B （C D））	（A·(B·((C·(D·NIL))·NIL))）
（A B C D）	（A·(B·(C·(D·NIL)))）
（（A））	（(A·NIL)·NIL）
（）	NIL

读者自己可以写出它们之间的转换过程。

在一般情况下，人们常使用表表示法。但在有些场合，必须用点对表示法或者只有用点对表示法才能更清楚解释时，就用点对表示或采用两种写法的混合，例如（A B C D …X·Y）。

右图是S-表达式的语法图。



习　　题

1. 判别下列各式子，哪些是正确的S-表达式(并区分原子、点对表示、表表示)？哪些是错误的S-表达式？

ATOM

(THIS IS A BOOK)

(A·B C)

(PLUS 3A X)

(A·(B·(C·D)))

3

(3)

(LIST 3)

(QUOTIENT (ADD1 3) (SUB1 3))

((( )))

(( ) ( ))

((( ))

( )X

((A B C

2. 进行点对表示与表表示之间的互相转换(如果原式是点对表示则转换成表表示，如果原式是表表示则转换为点对表示)。

(1) (A . ((B . (C . NIL)) . ((D . (E . NIL)) . (F . NIL))))

(2) ((A . B) . (C . (D . E)))

(3) (A (B C) (D (E F)))

(4) (((A B)))

第三章　基本函数

如前所述，LISP 语言没有语句概念，LISP 程序是一串函数定义，其后跟着一串带有参数的函数调用组成。LISP 语言最初只提供了少数基本函数，后来增加了许多内部函数。用户可以定义自己的函数。大的 LISP 函数可以由一些基本函数与内部函数通过复合及递归等手段来构成。本章主要介绍 LISP 中的基本函数。在介绍之前，首先给出一些符号约定。

LISP 中函数调用的一般形式为：

（函数名　自变量1　自变量2　……　自变量n）

其中函数名为符号原子，每个自变量可为下列六种形式之一。

我们约定：S　代表 S-表达式

L　代表表

F　代表函数

A　代表原子

NA　代表非原子

N　代表数原子

§3.1　基本函数

（1）（CAR　NA）

一元选择函数。自变量为非空表。

函数 CAR 取自变量的第一个元素（即表的头部）。

例 1　（CAR　'(A　B　C　D)）

A

其中符号 ' 表示后面的 S-表达式（A　B　C　D）不再被计算①，或者说 '（A　B　C　D）的计算结果仍为（A　B　C　D）。这里 CAR 的自变量是表（A　B　C　D），A是其第一个元素。

A 为函数调用的结果值。下面我们均以加下横线来表示机器回送的函数值。

例 2　（CAR　'((A　B　C)　X　Y　Z)）

(A　B　C)

自变量是表（(A　B　C)　X　Y　Z），（A　B　C）是其第一个元素。

① 在S-表达式(A　B　C　D)之前加符号 ' 是因为 LISP 中任一函数的调用形式为（函数名自变量表）。若(A B C D)之前不加 '，则(A　B　C　D)正好就是函数形式，A被理解为函数名，B，C，D被理解为函数A的自变量，因此(A　B　C　D)将作为函数调用而被求值。而这里(A　B　C　D)表示一张表，不是一个函数调用，故以 '(A　B　C　D)区别。需指出，符号 "'" 在有些实现系统中用 "'" 。

例 3 （CAR 'M）

出错

自变量为原子，函数无意义。

例 4 （CAR '( )）

出错（有些系统回答NIL）。

自变量为空表，函数无意义。

从例 3 和例 4 可以看出，CAR 对原子和空表求值无意义。如果 CAR 作用于点对形式的自变量，则取其左部，例如（CAR '(A · B)）⇒A。

（2）（CDR NA）

一元选择函数。自变量为非空表。

函数 CDR 回送一张表，这张表包含了自变量中除第一元素以外的所有元素，即表的尾部。所以它是 CAR 的补函数。

例 1 （CDR '(A B C D)）

(B C D)

例 2 （CDR '((A B C) X Y Z)）

(X Y Z)

例 2 中，自变量为表（(A B C) X Y Z）。X，Y，Z 为除第一元素（A B C）以外剩下的元素，由它们构成一张表（X Y Z）作为结果值。

例 3 （CDR 'M）

出错

自变量为原子，函数无意义。

例 4 （CDR '( )）

出错（有些系统回答NIL），

自变量为空表，函数无意义。

当 CDR 作用于点对形式的自变量时，则取其右部，例如，（CDR '(A · B)）⇒B。当 CDR 作用于只含一个元素的表时，则 CDR 的结果为空表，即（CDR '(A)）⇒NIL。同样（CDR '(( ))）和（CDR '(((A))）都等于NIL.

（3）（CONS S_1 S_2）

二元构造函数，第一自变量与第二自变量分别是 S- 表达式。当第二自变量为一张表时，则函数 CONS 回送一张表，该表的 CAR 是第一个自变量，它的 CDR 是第二个自变量.

例 1 （CONS 'A '(B C)）

(A B C)

例 2 （CONS '(A B) '(C D)）

((A B) C D)

所以，当 CONS 的第二自变量为非空的表时，CONS 函数是将第一自变量作为元素加入到第二自变量的表中，作为头部。

例 3 （CONS 'A '( )）

(A)

原子和空表通过 CONS 作用构成了表。