LISP 1.5 程序员手册

J. McCarthy
P.W. Abrahams
D.J. Edwards　　原著
T.P. Hart
M.I. Levin

中国科学院理论物理研究所二室印

1980　北京

出 版 说 明

LISP是第一个、也是应用最广的表处理(List Processing)语言。自1960年提出以来，已经发展出一大批变种，如LISP 1.5，标准 LISP 以及功能更为完备的 INTERLISP 等等。由LISP提出者J.麦克阿瑟参加撰写的这本"LISP 1.5 程序员手册"事实上已经成为一种基础教科书，从1965年第二版定型以来，在美国已印刷多次。本书就是根据麻省理工学院出版社1979年7月第11次印刷本全文译出的。

LISP语言的基本精神是递归。用LISP写出的程序和程序处理的对象都具有类似的表结构，犹如机器语言的指令和数据都是一些保存在机器字中的数码。对于人工智能方面的许多应用，LISP更是起着一种机器语言的作用。例如，应用很广的代数运算语言 REDUCE 2 和 MAXSYMA，都是用 LISP 写的。在我国引入LISP语言和某些符号运算语言的工作已经开始。我们就是着眼于在理论物理研究工作中推广使用符号运算语言，组织翻译这本书的。

英文原版一直保留着早期在 IBM 7090 机器上实现 LISP 时与机器指令和控制台有关的一些说明（主要是在几个附录中）。这些说明对于现在的 IBM 360 和 370 系统虽无直接意义，但对于理解表结构和 LISP 本身的实现方式还是颇为有益的。很可能就是由于这种益处，原作者没有从新版中删去这些说明，现在也都全部照译。本书的翻译，涉及了一些我国尚未定型的名词译法。我们在正文中加了少量注释。本书由中国科学院物理研究所八室张淑誉同志翻译，[illegible]林同志校阅。

中国科学院理论物理研究所

1980年10月

前 言

LISP程序系统的整体设计是J. McCarthy的工作，并且基于他的文章“符号表达式的递归函数和它们的机器计算”，它发表在1960年4月份的 Communications of the ACM 上。

本手册是M.I. Levin写的。解释程序是S.B. Russell和D.L. Edwards设计的。打印和读入程序是J.McCarthy，D.J. Edwards和P.W. Abrahams写的。清理程序和算术程序是D.J.Edwards写的。编译程序和汇编程序是T.P. Hart和M.I.Levin写的。更早的编译程序是R.Brayton写的。

1960年3月1日的“LISP 1程序员手册”是P.A. Fox写的。

增补的程序和建议是麻省理工学院电子研究实验室人工智能小组下列成员的贡献：M.L. Minsky, B. Raphael, L.Hodes, D.M.R.Park, D.C.Luckham, D.G.Bobrow, J.R.Slagle和N.Rochester。

1962年8月17日

目　录

第一章 LISP语 言

LISP 语 言最初是为了符号数据处理而设计的。它已经用于微积分运算、电路理论、数理逻辑、博弈、以及其他人功智能领域的符号运算中。

LISP 是一种形式数学语言。所以有可能给它一个简明而完整的描述。这就是本手册第一章的目的。其他各章将进一步描述使用 LISP 的方法，并说明这个语言的各种扩充，它们使 LISP 成为方便的程序设计系统。

LISP在三个重要方面不同于多数程序设计语言。首先是数据的性质。在LISP 语言中，全部数据用符号表达式的形式，通常称为S表达式。S表达式是不定长度的，并具有分枝树型结构，因此有意义的子表达式很容易分离出来。在LISP程序设计系统中，大块可用的内存是以表结构的形式存放S表达式的。这种类型的内存组织，使程序员无须为他的各段程序分配存贮。

LISP语言的第二个重要部分是源语言本身规定了S表达式用何种方式处理。这是由S表达式的递归函数组成的。由于书写 S 表达式递归函数的记法本身不包含在S表达式记法规定内，它将被称为元（Meta）语言。因此，这些表达式将称为M表达式。

第三，LISP 能解释并执行用 S 表达式写出的程序。这样，它就更像机器语言，而不像多数其他高级语言，能用来生成可以继续执行的程序。

1.1 符号表达式

最基本的一类S表达式是原子（Atom）符号。

定义：原子符号是一个不多于30个数字和大写字母的字符串，第一个字符必须是字母。

例子

A

APPLE

EXTRALONGSTRINGOFLETTERS

A4B66XYZ2

这些字符称作原子，是由于他们作为整体取来，在LISP内不能再分割成单个的字符。这样A，B，以及AB互相没有关系，它们只是三个不同的原子符号。

所有的S表达式都是由原子符号和标点符号“(”,“)”和“.”构成的。形成S表达式的基本操作，是把两个S表达式组合成更大的一个。用两个原子符号A1和A2，能形成S表达式（A1.A2）。

定义：一个S表达式或是一个原子符号，或是由这些元素按以下顺序组成：一个左括号，一个S表达式，一个圆点，一个S表达式和一个右括号。

注意：这个定义是递归的。

例子

ATOM

(A.B)

(A.(B.C))

((A1.A2).B)

((U.V).(X.(Y.Z)))

1.2 基本函数

我们将引入S表达式的一些基本函数。为了把函数和S表达式本身区别开。我们用小写字母写函数名字，因为原子符号总是用大写字母。此外，函数的自变量将用方括号，而不是圆括号括在一起，我们用分号作隔离符或标点符号。

我们引入的第一个函数是cons（组合）。它有两个自变量，实际上就是用这个函数从更小的S表达式构成S表达式。

例子

cons[A;B] = (A.B)

cons[(A.B);C] = ((A.B).C)

cons[cons[A;B];C] = ((A.B).C)

最后一个例子是函数的复合。这就有可能用复合cons函数的办法，从原子成分构造任意的S表达式。

下一对函数的作用与cons相反。它们从给定的表达式中取出子表达式。

car（取首）函数有一个自变量。它的值是其复合自变量的第一部分。原子符号的car是未定义的。

例子

car[(A.B)] = A

car[(A.(B1.B2))] = A

car[((A1.A2).B)] = (A1.A2)

car[A]未定义

cdr（取尾）函数有一个自变量。它的值是其复合自变量的第二部分。如果cdr的自变量是原子，它也是未定义的。

例子

cdr[(A.B)] = B

cdr〔(A.(B1.B2))〕=(B1.B2)

cdr〔((A1.A2).B)〕=B

cdr〔A〕未定义

car〔cdr〔(A.(B1.B2))〕〕=B1

car〔cdr〔(A.B)〕〕未定义

car〔cons〔A;B〕〕=A

给定任意的S表达式，就有可能用car和cdr的适当组合产生它的任何一个子表达式。如果x和y代表任意两个S表达式，下列恒等式成立。

car〔cons〔x;y〕〕=x

cdr〔cons〔x;y〕〕=y

对任意S表达式x，只要x是复合的（即非原子的），以下恒等式也成立：

cons〔car〔x〕〕; cdr〔x〕〕=x

在这些恒等式中用到的符号x和y称为变量。在LISP中，变量用来代表S表达式。在为变量和函数取名字时，我们使用形成原子符号时用到的同类型字符串，只是以小写字母书写。

如果一个函数的值或是真，或是假，就称为谓词（predicate）。在LISP中，真值和假值分别用原子符号T和F表示。所以LISP谓词是一个函数，它的值是T或F。

谓词eq检查原子符号是否等同。它对非原子自变量没有定义。

例子

eq〔A;A〕=T

eq〔A;B〕=F

eq〔A;(A.B)〕未定义

eq〔(A.B);(A.B)〕未定义

谓词 atom 是真，如果它的自变量是一个原子符号，如果它的自变量是复合的，则是假。

例子

atom〔EXTRALONGSTRINGOFLETTERS〕= T

atom〔(U.V)〕= F

atom〔car〔(U.V)〕〕= T

1.3 表的记法

至今用到的S表达式是用圆点记法写的。如果能够书写不定长度的表达式的表，如(A B C D E)，往往就更为方便。

用圆点记法能够表示任意S表达式。然而，LISP有另一种S表达式的替代形式，称为表记法。表($m_1 m_2 \cdots m_n$)可以用圆点记法定义。它等同于$(m_1 \cdot (m_2 \cdot (\cdots \cdot (m_n \cdot NIL)\cdots))$。

原子符号NIL用作表的结束符。空表( )等同于NIL。表可以含有子表。圆点记法和表的记法可以用在同一个S表达式中。

历史上，曾经用逗点(,)作元素的隔离符；然而，现在通常用空格。在LISP中这两者是完全等价的。(A, B, C)等同于(A B C)。

例子

(A B C) = (A.(B.(C.NIL)))

((A B) C) = ((A.(B.NIL)).(C.NIL))

(A B (C D)) = (A.(B.(C.(D.NIL)).NIL))

(A) = (A.NIL)

((A)) = ((A.NIL).NIL)

(A (B.C)) = (A.((B.C).NIL))

重要的是熟悉基本函数作用在用表记法写的S表达式上的结果。转换到圆点记法，这些结果总是能够定出来的。

例子

car〔(A B C)〕= A

cdr〔(A B C)〕= (B C)

cons〔A；(B C)〕= (A B C)

car〔((A B) C)〕= (A B)

cdr〔(A)〕= NIL

car〔cdr〔(A B C)〕〕= B

为了方便起见，可把多个car和cdr统写。这就是用c开头，以r结尾，中间写多个a和d，来形成函数名字。

例子

cadr〔(A B C)〕= car〔cdr〔(A B C)〕〕= B

caddr〔(A B C)〕= C

cadadr〔(A (B C) D)〕= C

名字中最后的a和d实际上是最先被执行的操作，因为它最靠近自变量。

1.4 LISP元（Meta）语言

我们已经引入了叫作S表达式的一类数据，和五个基本函数。我们也讨论了元语言的下列特性。

1.函数名字和变量名字都像原子符号，只是用小写字母。

2.函数的自变量用方括号组合起来，以分号互相隔离。

3.复合函数可用多重嵌套的括号写出来。

这些规则允许我们写出函数的定义，如像

third〔x〕= car〔cdr〔cdr〔x〕〕〕

这个函数选择一个表中的第三项，例如

third〔(A B C D)〕=C

实际上 third 是同 caddr 一样的函数。

能用这种办法形成的函数类十分有限，而且不是很有意义的。借助条件表达式可以定义大得多的函数类，这是在函数定义中实现分枝的手段。

条件表达式具有以下形式：

〔$p_1 \rightarrow e_1; p_2 \rightarrow e_2; \cdots; p_n \rightarrow e_n$〕

其中每个 p_i 是一个表达式，它的值可以是真或假，而每个 e_i 是任意的表达式。条件表达式的意思是：如果 p_1 是真，那什 e_1 的值就是整个表达式的值。如果 p_1 是假，而 p_2 是真，e_2 的值就是整个表达式的值。由左向右检查 p_i，直到找到第一个真值。然后相应的 e_i 被选中。如果 p_i 中没有一个真，那么整个表达式的值未定义。

每个 p_i 或 e_i 本身可以是一个 S 表达式，或一个函数，或几个函数的复合，或者是另一个条件表达式。

例子

〔eq〔car〔x〕;A〕→cons〔B;cdr〔x〕〕;T→x〕

原子符号T代表真。这个表达式的值是这样得到的：如果 x 的 car 恰好是A，就把它换成B、如果 x 的 car 不是A，就保持 x 不变。

条件表达式的主要用途在于递归地定义函数。

例子

ff〔x〕=〔atom〔x〕→x;T→ff〔car〔x〕〕〕

这个例子定义函数 ff，它从任意的给定表达式中选定第一个原子符号。这个表达式可以读作：如果 x 是一个原子符号，那么x本身就是答案。否则要把函数ff用到x的car上。如果 x 是原子，那么第一个分枝即“x”被选出。否则就选

第二个分支“ff〔car〔x〕〕”，因为T总是真。

ff的定义是递归的，因为它实际上是用函数ff本身定义的。如果坚持对任意S表达式不断取car，最终它会产生一个原子符号，所以这个过程是很好地定义了的。

有些递归函数只能对一定的自变量很好地定义，但对另一些自变量是无穷递归的。当LISP程序设计系统中解释这样的函数时，它或者会耗尽可用的内存，或者一直循环到程序被人为终止。

现在我们看一下函数ff〔((A.B).C)〕是怎样计算的。首先，我们用自变量代替定义中的x变量，得到

ff〔((A.B).C)〕=〔atom〔((A.B).C)〕→((A.B).C);
T→ff〔car〔((A.B).C)〕〕〕

但((A.B).C)不是原子，因此我们有

=〔T→ff〔car〔((A.B).C)〕〕〕
=ff〔car〔(A.B).C)〕〕
=ff〔(A.B)〕

此刻，必须递归地使用ff的定义。用(A.B)代替x给出

=〔atom〔(A.B)〕→(A.B);T→ff〔car〔(A·B)〕〕〕
=〔T→ff〔car〔(A.B)〕〕〕
=ff〔car〔(A.B)〕〕
=ff〔A〕
=〔atom〔A〕→A;T→ff〔car〔A〕〕
=A

条件表达式对于数值计算和S表达式的计算都是有用的。一个数的绝对值可定义为

|x|=〔x<0→-x;T→x〕

一个非负整数的阶乘可定义为

$n! = (n = 0 \rightarrow 1; T \rightarrow n \cdot (n-1)!)$

这一递归定义对负自变量不能终止。只对一定的自变量定义的函数叫作部分函数。

求两个正整数的最大公因子的欧几里得算法可用条件表达式定义如下：

$$gcd(x;y) = (x > y \rightarrow gcd(y;x);$$
$$rem(y;x) = 0 \rightarrow x;$$
$$T \rightarrow gcd(rem(y;x);x))$$

$rem(u;v)$是u被v除的余数。

用条件表达式递归地定义函数的理论的详细讨论，可在1961年5月“The western Joint computer Conference”会议录中（无线电工程师协会IRE出版），J.McCarthy的“计算理论的数学基础”一文中找到。

对大多数数学家，除了那些从事逻辑的人外，常常不严格地使用“函数”一词，也把它用于诸如y^2+x这样的型。由于以后要计算作为函数的表达式，我们就需要一种记法来表明函数和型的差别。我们将采用A.Church的λ记法[1]。

设f是一个表达式，代表两个整变量的函数。写成$f(3;4)$应具有意义，并有可能决定这个表达式的值。例如，$sum(3;4)=7$。表达式y^2+x不满足这个要求。一点也不清楚$y^2+x(3;4)$的值是13还是19。像这样一个表达式y^2+x，最好称为型，而不叫作函数。一个型能够转换成一个函数，办法是规定型中的变量和所希望的函数的自变量之间的对应关系。

如果e是变量$x_1;\cdots;x_n$的一个型，那么表达式$\lambda((x_1;$

---

1. A. Church, The Calculi of Lambda-Conversion Princeton University press, Princeton, New Jersey, 1941。

…；x_n〕；ε〕分别表示对变量x_1；…；x_n依次代以n个自变量所获得的n元函数。

例如，函数 λ〔〔x；y〕；y^2+x〕是一个二元函数，因而 λ〔〔x；y〕；y^2+x〕〔3；4〕$=4^2+3=19$。λ〔〔y；x〕；y^2+x〕〔3；4〕$=3^2+4=13$。

在一个λ-表达式中的各个变量是虚的或限定的变量，因为当系统地改变它们时，表达式的意义并不变。这样 λ〔〔u；v〕；v^2+u〕的意思，与λ〔〔x；y〕；y^2+x〕相同。

有时候我们用到一些表达式，其中一个变量并没有被λ限定。例如，在二元函数 λ〔〔x；y〕；x^n+y^n〕中，变量n没有限定。这叫作自由变量。它可以当作一个参量对待。除非在试图计算这个函数前给了 n 一个值，否则函数的值是未定义的。

仅仅用λ记法不足以定义递归函数。不仅变量必须限定，而且函数的名字也必须限定，因为它被用在一个表达式中来代表整个表达式。函数 ff 在前面是用等式定义的：

ff〔x〕=〔atom〔x〕→x；T→ff〔car〔x〕〕〕

用λ记法，我们可以写成

ff=λ〔〔x〕；〔atom〔x〕→x；T→ff〔car〔x〕〕〕〕

这些等式中的等号实际上不是 LISP 元语言的一部分，而仅仅是我们发展正确记法之前的一种拐仗。最后一个方程式的的右端不能作为函数 ff 的表达式，因为没有任何东西指明其中出现的 ff 代表要定义的函数。

为了可以书写带有自己的名字的表达式，我们引入label记法。如果 ε 是一个表达式，而 α 是它的名字，就写成label〔α；ε〕。

label〔ff；λ〔〔x〕；〔atom〔x〕→x；T→ff〔car〔x〕〕〕〕〕

在这个表达式中，x 是限定的变量，ff是限定的函数名字。

1.5 句法小结

LISP 语言的各个部分到此解释完了。下面是LISP语言的完备的句法定义和一些语义注解。定义是用巴科斯(Backus)记法给出的，再加上用三个圆点(…)，省去不必要的重复列举句法成份。

在巴科斯记法中使用符号“::=”，“<”，“>”，和“|”。像〈S表达式〉::=〈原子符号〉|(〈S表达式〉·〈S表达式〉)这样一条规定，其意义是：S表达式或是一个原子符号，或是一个左括号随以S表达式，随以圆点、随以S表达式、再随以右括号。竖线的意思是“或”，尖括号总是用来括住要定义的句法元素。

数据语言：

〈大写字母〉::=A|B|C|…|Z

〈数字〉::=0|1|2|…|9

〈原子符号〉::=〈大写字母〉〈原子部份〉

〈原子部份〉::=〈空〉|〈大写字母〉
〈原子部份〉|〈数字〉
〈原子部份〉

原子符号是LISP中最小的对像。他们分解成字符是没有意义的。

〈S表达式〉::=〈原子符号〉|
(〈S表达式〉·〈S表达式〉)|
(〈S表达式〉…〈S表达式〉)

以这种方式用到三个圆点时，它们的意思是给定类型的符号可以出现任意次，包括根本不出现。根据这个规则，()是有效的S表达式(它等价于NIL)。

圆点记法是S表达式的基本记法，然而表记法常常更方便。用圆点记法可以写出任意的S表达式。

元语言：

〈小写字母〉::=a | b | c | … | z

〈标识符〉::=〈小写字母〉〈标识符部份〉 |
〈数字〉〈标识符部份〉

〈标识符部份〉::=〈空〉 | 〈小写字母〉
〈标识符部份〉 | 〈数字〉
〈标识符部份〉

函数和变量的名字与原子符号的构成一样，但用小写字母。

〈型〉::=〈常数〉 |
〈变量〉 |
〈函数〉〔〈自变量〉；…；〈自变量〉〕 |
〔〈型〉→〈型〉；…；〈型〉→〈型〉〕

〈常数〉::=〈S表达式〉

〈变量〉::=〈标识符〉

〈自变量〉::=〈型〉

型是一个可以计算的表达式。仅仅是一个常数的型，其值就是这个常数。如果型是一个变量，则其值就是我们计算这个型时，限定到这个变量上的那个表达式。

型的规定的第三部份指出，我们可以写一个函数随以放在方括号中、用分号隔离的自变量表。自变量的表达式本身是型；这说明允许有复合函数。

这个规定的最后部份给出条件表达式的格式。它是这样计算的：顺序计算命题位置上的型，直到找到一个值为T；然后计算箭头后面的型，它给出整个表达式的值。