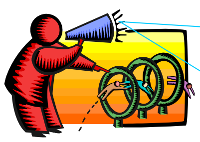
Programming Methodology
程序设计方法学

第5章 递归程序设计

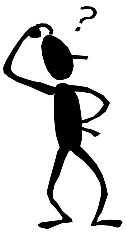
主讲：刘悦
yliu@staff.shu.edu.cn

回顾



- 类型
- 数据类型
- 抽象数据类型
- 数据抽象及其代数规范
- 大型程序设计与抽象数据类型

本章目标



?什么是递归定义，递归的优缺点
?什么是迭代，递归与迭代的区别
？什么是递归数据结构，
有哪些结构是递归数据结构
？什么是简化的函数型递归程序模型，
递归程序的计算规则
？简化的**LISP**程序
？递归程序的正确性证明

主要内容

- 递归的概念
- 递归与迭代程序
- 递归数据结构
- 递归程序
- 递归程序正确性证明

内容线索

- 递归的概念
  - 递归与迭代程序
  - 递归与递推
- 递归数据结构
- 递归程序
- 递归程序正确性证明

递归的概念

- 一个函数或者数据结构，如果在它们定义的内部有出现定义本身的应用，则称它们是递归的或者称为递归定义的。
  - **$P \equiv \beta[S_i, P]$**

Si是基语句，不包含P

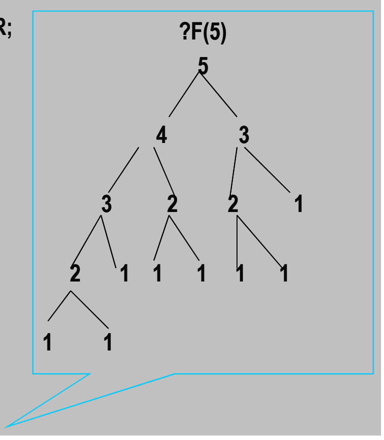
- 分为
  - 直接递归： **P**包含对自己的引用
  - 间接递归：**P**包含对另一程序**Q**的引用，而**Q**又直接或间接地引用**P**
- 递归算法的准确表示为： **$P \equiv \text{if } B \text{ then } \beta[S_i, P]$**
  - 其中， **B**是递归调用受限条件
  - 所以，递归程序依然要考虑终止问题
- 例题：**P150~152**
  - 计算阶乘的函数
  - 计算**Fibonacci**数列

计算阶乘的函数

```
Program A51(INPUT, OUTPUT);
  Var I: INTEGER;
  Function FACT(N:INTEGER): INTEHER;
  Begin
     if N=0 then FACT=1
     else FACT:=N*FACT(N-1);
  End;
  Begin
      READ(I);
     WRITELN(FACT(I));
  End.
```


计算Fibonacci数列

```
Program A53(INPUT, OUTPUT);
  Var I: INTEGER;
  Function F(N:INTEGER): INTEHER;
  Begin
     if N=1 then F:=0
     else if N=2 then F:=1
     else F:=F(N-1)+F(N-2);
  End;
  Begin
      READ(I);
     WRITELN(F(I));
  End.
```



递归的优缺点

- 优点：
  - 比较直观、精练，逻辑清楚，逼近数学公式的表示
  - 编程方便，易读
- 缺点
  - 效率较低
  - 每一次执行，都必须进行参数替换、环境保护等
  - 大量重复的计算

迭代的概念

- 迭代(iterate)
  - 重复地执行一系列步骤。
- 迭代法——数值计算中的一种重复方法。
- 例题：**P153~154**
  - 计算阶乘的函数
  - 计算Fibonacci数列

计算阶乘的函数

```
Program A54(INPUT, OUTPUT);
  Var I: INTEGER;
  Function FACT(P:INTEGER): INTEHER;
    Var I,F: INTEGER;
    Begin
       F:=1;
       for I:=1 to P do F:=F*I;
        FACT:=F
    end;
  Begin
      READ(I);
     WRITELN(FACT(I));
  End.
```


计算Fibonacci数列

```
Program A55(INPUT, OUTPUT);
 Var I: INTEGER;
  Function FACT(P:INTEGER): INTEHER;
    Var P,L,T,I: INTEGER;
    Begin
       if N=1 then F:=0
       else if N=2 then F:=1
       else begin
              L:=0; T:=1;
              for I:=3 to N do
                 begin
                    P:=L;L:=T; T:=P+L;
                 end;
              F:=T
            end;
    end;
Begin
     READ(I);
    WRITELN(F(I));
 End.
```

递归与迭代程序…

- 在程序设计中，为了处理重复性的计算，最常采用的方法是组织迭代循环，除此之外，往往还可采用递归计算的方法特别是在非数值领域中更是如此。
- 除了可调用的其他程序外，还可以直接或间接调用自身的程序称为递归程序。
- 实质上，递归也是一种循环结构，它把“较复杂”情形的计算归结为“较简单”情形的计算，一直归结到“最简单”情形的计算，并得到计算结果为止。就某种意义而言，递归是一种比迭代循环更强的循环结构。可以证明每个迭代程序原则上总可以转换成与它等价的递归程序，但是，反之不然，即并不是每个递归程序都可以转换成与它等价的迭代程序。

…递归与迭代程序

- 但就效率而言，递归程序的实现往往比迭代程序耗费更多的时间与存储空间。所以在具体实现时，又希望尽可能把递归程序转化成等价的迭代程序，从而提高程序的时空效率
- 但是，递归在解决实际问题时是十分有效的
  - **Hanoi**塔问题
  - **Hilbert**图案问题
  - 八皇后问题
  - 骑士游历问题

递推与递归

递推法是利用问题本身所具有的一种递推关系求问题解的一种方法。设要求问题规模为 N 的解，当 $N=1$ 时，解或为已知，或能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质，即当得到问题规模为 $i-1$ 的解后，由问题的递推性质，能从已求得的规模为 $1,2,\cdots,i-1$ 的一系列解，构造出问题规模为 i 的解。这样，程序可从 $i=0$ 或 $i=1$ 出发，重复地，由已知至 $i-1$ 规模的解，通过递推，获得规模为 i 的解，直至得到规模为 N 的解。

递归是设计和描述算法的一种有力的工具，由于它在复杂算法的描述中被经常采用，为此在进一步介绍其他算法设计方法之前先讨论它。

能采用递归描述的算法通常有这样的特征：为求解规模为 N 的问题，设法将它分解成一些规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模稍大问题的解。特别地，当规模 $N=1$ 时，能直接得到解。

递推的例子

【例 9.2】 编写程序，对给定的 $n(n\leqslant 100)$，计算并输出 k 的阶乘 $k!$ $(k=1,2,\cdots,n)$ 的全部有效数字。

由于要求的整数可能大大超出一般整数的位数，程序用一维数组存储长整数，存储长整数数组的每个元素只存储长整数的一位数字。如有 m 位长整数 N 用数组 $a[\,]$ 存储：

$$N = a[m]\times 10^{m-1}+a[m-1]\times 10^{m-2}+\cdots+a[2]\times 10^{1}+a[1]\times 10^{0}$$

并用 $a[0]$ 存储长整数 N 的位数 m，即 $a[0]=m$。按上述约定，数组的每个元素存储 k 的阶乘 $k!$ 的一位数字，并从低位到高位依次存于数组的第二个元素、第三个元素……例如，$5!=120$，在数组中的存储形式为：

3	0	2	1	……

首元素 3 表示长整数是一个 3 位数，接着是低位到高位依次是 0、2、1，表示长整数是 120。

计算阶乘 $k!$ 可采用对已求得的阶乘 $(k-1)!$ 连续累加 $k-1$ 次后求出。例如，已知 $4!=24$，计算 $5!$，可对原来的 24 再累加 4 次 24 后得到 120。细节见以下程序。

程序代码

```
# include <stdio.h>
# include <malloc.h>
# define  MAXN 1000
void  pnext(int a[ ],int k)
{
  int *b,m=a[0],i,j,r,carry;
  b=(int * ) malloc(sizeof(int)* (m+1));
  for ( i=1;i<=m;i++) b[i]=a[i];
    for ( j=1;j<=k;j++)
    {
      for ( carry=0,i=1;i<=m;i++)
      {
          r=(i<a[0]?a[i]+b[i]:a[i])+carry;
          a[i]=r%10;
          carry=r/10;
      }
      if (carry)  a[++m]=carry;
    }
  free(b);
  a[0]=m;
}
```

```
void  write(int *a,int k)
{
  int i;
  printf("%4d !  =",k);
  for (i=a[0];i>0;i--)
     printf("%d",a[i]);
  printf("\n\n");
}

void main()
{
  int a[MAXN],n,k;
  printf("Enter the number n:  ");
  scanf("%d",&n);
  a[0]=1;
  a[1]=1;
  write(a,1);
  for (k=2;k<=n;k++)
  {
    pnext(a,k);
    write(a,k);
    getchar();
  }
}
```


递推算法伪代码

```
if 求解初始条件Fi then
      begin{倒推}
        由题意确定最终结果Fa;
        求出倒推关系式Fi-1=g'(Fi);
        I:=n{从最后结果Fn出发进行倒推}
        While 当前结果Fi非初值F1 do
          由Fi-1=g'(Fi)倒推前项;
        输出倒推结果F1和倒推过程;
      end
else
   begin{顺推}
     由题意确定初值F1{边界条件}
     求出倒推关系式Fi=g'(Fi-1);
     I:=1{从边界F1出发进行顺推}
     While 当前结果Fi非终值Fn do
          由Fi=g'(Fi-1)顺推后项;
     输出顺推结果和顺推过程;
     end
```

递归的例子

fib(0) = 0

fib(1) = 1

fib(n) = fib(n−2) + fib(n−1)　　（当 n > 1 时）

写成递归函数有：

【例 9.3 函数】

```
int fib(int n)
{   if (n == 0) return 0;
    if (n == 1) return 1;
    if (n > 1) return fib(n-2) + fib(n-1);
}
```

递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题(规模为 n)的求解推到比原问题简单一些的问题(规模小于 n)的求解。例如上例中，求解 fib(n)，把它推到求解 fib(n−2)和 fib(n−1)。也就是说，为计算 fib(n)，推到计算 fib(n−2)和 fib(n−1)，而计算 fib(n−2)和 fib(n−1)，又把它们推到计算 fib(n−4)和 fib(n−3)。依此类推，直至计算 fib(0)和 fib(1)，分别能立即得到结果 0 和 1。在递推阶段，必须要有终止递归的情况，例如在函数 fib 中，当 n 为 0 和 1 的情况。

在回归阶段，当获得最简单情况的解后，逐级返回，依次获得稍复杂问题的解。例如得到 fib(0)为 0 和 fib(1)为 1 后，返回获得 fib(2)的结果……得到了 fib(n−2)和 fib(n−1)的结果后，返回获得 fib(n)的结果。

内容线索

- ✓ 递归的概念
- 递归数据结构
- 递归程序
- 递归程序正确性证明

递归数据结构

- 数据分为静态与动态的
- 动态数据结构的定义往往是递归地给定的，所以动态数据机构又称递归数据结构
- 递归定义的动态数据结构：序列、树和图
  - 基类型为T的序列或者是一个空序列，或者是类型为T的一个数据与基类型为T的序列的连接
    - 外存：顺序文件
    - 内存：链表、栈、队列

内容线索

- ✓ 递归的概念
- ✓ 递归数据结构
- 递归程序
  - 递归程序的一种模型
  - 递归程序的例子
  - 递归计算的计算规则
  - 简化的LISP程序
- 递归程序正确性证明

递归程序的一种模型

- 下面将讨论一种简化的函数型递归程序模型，其一般形式为

$$
\begin{aligned}
F(x_1,x_2,\ldots,x_n) \equiv\ & \text{if } p_1 \text{ then } E_1 \\
& \text{else if } p_2 \text{ then } E_2 \\
& \qquad \cdots\cdots\cdots\cdots \\
& \text{else if } p_m \text{ then } E_m \\
& \text{else } E_{m+1}
\end{aligned}
$$

  - 其中，$p_i(i=1,2,\ldots.m)$是测试谓词，$E_i(i=1,2,\ldots..m+1)$是表达式。

递归程序的一种模型

- 这种函数型递归程序的基本含义是
  - 为了计算自变元为$(x_1,x_2,\ldots.x_n)$的递归函数$F(x_1,x_2\ldots,x_n)$的值，先测试P_1的值，若P_1为真，则F的值为E_1的值；若P_1为假，则接下去测试P_2，若P_2为真，则F的值为E_2的值。
  - 按这种方式继续进行测试，直至找到第一个为真的P_i。这时F的值为相应的E_i的值。倘若没有一个$P_i(i=1,2,\ldots m)$为真，则F的值规定为E_{m+1}的值。
  - 这里所谓的递归，是指F可以在E_i和P_i中出现，这种出现称为F的递归调用。

递归程序的例子…

- 例 1 阶乘函数

  F(x)≡if x=1 then 1

  else x*F(x-1)

  其中，**x**为任意正整数。

  现在就自变元**x**的某个具体值**(**例如**x=4)**来考察一下**F(x)**的计算过程。

  F(4)=4*f(3)=4*3*f(2)=4*3*2*f(1)=4*3*2*1=4!=24

  可以证明，对任意正整数**x**有 **F(x)=x!**

递归程序的例子…

- 例 2 求非负整数x和y的最大公约数GCD(x, y)

  GCD(x,y)≡ if x=0 then y

  else if y<x then GCD(y,x)

  else GCD(x,y-x)

  其中，**x<0,y<0,**且**x,y**不同时为零。

  可以证明**GCD(x,y)**确实计算了**x**和**y**的最大公约数。

  例如**:**

  GCD(8, 4) = GCD(4, 8) = GCD(4, 4)

  = GCD(4, 0) = GCD(0, 4)=4

…递归程序的例子…

例 3 Fibonacci函数

$\varphi(x) \equiv$ if $x=0$ then 0
　　else if $x=1$ then 1
　　else $\varphi(x-1) + \varphi(x-2)$

其中，x为非负整数

我们有

$\varphi(0)=0$

$\varphi(1)=1$

$\varphi(2)= \varphi(1)+ \varphi(0)=0+1=1$

$\varphi(3)= \varphi(2)+ \varphi(1)=1+1=2$

$\varphi(4)= \varphi(3)+ \varphi(2)=2+1=3$

$\varphi(5)= \varphi(4)+ \varphi(3)=3+2=5$

…

…递归程序的例子…

例 4 计算x^y

利用下述公式不难编出相应的递归程序

$$F(x,y)= x^y= \begin{cases} F(x,y)=1 & \text{若}y=0 \\ F(x,y)=(x*x)^{y/2} & \text{若}y\text{为偶数} \\ F(x,y)=x^{y-1}*x & \text{若}y\text{为奇数} \end{cases}$$

$F(x,y) \equiv$ if $y=0$ then 1
　　else if even(y) then $F(x*x,y/2)$
　　else $F(x,y-1)*x$

其中，x 为正实数；y为非负整数

例如：$F(4,3)=F(4,2)*4=F(16,1)*4$
　　$=F(16,0)*64=64$

…递归程序的例子…

例 5 **McCarthy91函数**

M(x)≡ if x>100 then x-10

else M(M(x+11))

其中，**x**为任意整数。

对于**x=105,**有**M(105)=95,**

x=99,其计算过程为

M(99)=M(M(111))=M(101)=91

可以证明，对于任意整数**x**

$$M(x)=\begin{cases} x-10 & x>100 \\ 91 & x<=100 \end{cases}$$

值得注意的是，这是具有二重性递归的递归程序

…递归程序的例子

例 6 **Ackermann** 函数

A (x1,x2)≡if x1=0 then x2+1

else if x2=0 then A(x1-1,1)

else A(x1-1,A(x1,x2-1))

其中，**(x,x)**为任意非负整数时。

对**(x1,x2)=(1,2)**，其计算过程为

A(1,2)

=A(0,A(1,1))

…递归程序的例子

A (x1,x2)≡if x1=0 then x2+1
else if x2=0 then A(x1-1,1)
else A(x1-1,A(x1,x2-1))

A(1,2)
=A(0,A(1,1))

(1) 假定应先计算A(1,1)，然后再计算外层的递归调用
=A(0,A(0,A(1,0)))
=A(0,A(0,A(0,1)))
= A(0,A(0,2))
= A(0,3)
=4

(2) 然而，若先规定先计算A的外层调用，则计算过程变为：
=A(1,1)+1
=A(0,A(1,0))+1
=(A(1,0)+1)+1
= (A(0,1)+1)+1
=(2+1)+1
=4

递归计算的计算规则

- 计算**A(1,2)**的上述两种计算过程虽然不同，但却导致相同的结果，即**A(1,2)＝4.**
- 可以证明：就同一个递归程序而言，如果对相同的自变元采用不同的计算过程，只要计算过程都终止，则所得结果将是相同的。
- 但是，可能出现这样的情况，一个计算过程不终止，而另一个计算过程却将终止。下面的例子就能说明这点。

例 7 F(x1,x2) ≡ if x1=0 then 0

else F(0,F(x1,x2))

其中，(x1,x2) 是任意非负整数对

我们采用两种不同的计算规则来计算F(1,2)

1：规定先计算F的最外层调用

F(1,2)=F(0,F(1,2))=0 (因x1≠0)

2：规定先计算F的最内层调用

F(1,2)= F(0,F(1,2)) (因x1≠0)

= F(0,F(0,F(1,2)))= (因对 F 的最内层调用有x1≠0)

=F(0,F(0,F(0,F(1,2))))

=.....

对于第一种计算规则，F(1,2)的计算过程终止，且F(1,2)=0

但对第二种计算规则，

F(1,2)的计算过程却永不终止，因而F(1,2)无定义。

结论…

- 上述讨论表明：
- (1) 递归程序可以采用不同的计算规则来进行计算；
- (2) 采用不同的计算规则来计算递归程序时，对相同的变元，计算过程可能终止，也可能不终止；
- (3) 如果对于不同的计算规则，相应的递归程序(对相同的自变元)的计算过程都终止，则它们所得的结果一定相同；
- (4) 在(3)的情况下，因为计算过程不同，所以虽然得到的结果相同，但其效率(计算时间和存储量)却可能差别大。

…结论

- 总之，在递归程序的执行过程中，计算规则的选取是很重要的。本章及后面的章节中，将统一规定：
  - 采用”最左，最内”的计算规则，即在计算过程中，总是先计算最内层的**F**中最左的一个。
  - 例如，在例**6**中，计算**A(1,2)**的第一种计算顺序就是按”最左，最内”的计算规则进行的。但在例**7**中，按”最左，最内”的计算规则去计算**F(1,2)**却是不终止的，故不能认为**F(1,2)＝0.**
- 虽然”最左，最内”的规则未必是最佳的，但现今具有处理递归调用功能 的程序设计语言大都采用这种计算规则。

简化的LISP程序

- 在非数值计算中(如描述表格及树型结构等)经常采用递归结构
- 在简化的递归程序模型中，对表格及在其上的一些运算做些规定，这些规定和人工智能语言**LISP**中的有关规定是相似的。不过为了讨论方便起见，我们对**LISP**语言做了很大的简化，即下面所讨论的只是一种简化的**LISP**程序。

原子和表

- 简化的**LISP**程序中，基本的数据结构是原子和表
  - 原子是指字母和数字组成的字符串，且字符之间不允许出现空格并约定原子必须以字母**A,B,C,D,E**之一打头，而以其他字母打头的均指变量。
  - 表是指由表元素组成的集合
    - 表元素之间用空格隔开
    - 整个集合用一对方括号［］括起来
    - 表元素可以是原子或其他表(在后面的一些例子中，有时也用数字作为表元素)
    - **NIL**表示空表，即不包含任何表元素的表。

同步练习

- （**1**）［**A B C**］
  - 是一个具有 3 个表元素的表
  - 每个表元素是一个原子
- （**2**）**[A [B A1 [C]] D]**
  - 是一个具有 3 个(顶层)表元素的表
  - 其中第一个表元素**A**和第 3 个表元素**D** 均为原子，而第二个表元素［**B A1 [C]**］是一个表，且这个表的第 3 个表元素［**C**］本身又是一个表

基本函数…

- 在简化的LISP程序中，允许采用以下 5 种基本函数：
- (1)　测试函数”=”
  - 功能：检查两个原子或表是否相同
  - 值：若相同，其值为true；否则其值为false
  - 例如
    - A=A 为true
    - A=B 为false
    - [A B]=[A B]为true
    - [A B]=[B A]为 false
    - [A B]=[A [B]]？
    - A=NIL？

false

false

…基本函数…

- （2）测试函数ATOM (x)
  - 功能：检查自变元x是否为原子或空表
  - 值：若是，其值为true；否则，取值为false
  - 例如：
    - ATOM (A)为true
    - ATOM (NIL)为true
    - ATOM ([A B])为？
    - ATOM ([A])为？

false

false

…基本函数…

- (3) 函数CAR(L)
  - 功能：提取表L中的第一个(顶层)元素
  - 例如：
    - CAR([A B C])=A
    - CAR([[A B] C])=[A B]
  - 注意，CAR(A)及CAR(NIL)均无定义

…基本函数…

- (4) 函数CDR(L)
  - 功能：提取表L中删去其第一(顶层)元素后余下的元素所组成的表
  - 例如：
    - CDR([A B C])=[B C]
    - CDR([A [B C]])=[[B C]]
    - CDR([[A B] C])=？
    - CDR([A])=？
  - 注意，CDR(A)与CDR(NIL)均无定义

[c]
NIL

…基本函数

- (5) 函数CONS(x,L)
  - 功能：将x(原子或表)加到表L中，得到一个新表，x成为新表的第一(顶层)元素
  - 例如：
    - CONS(A, [B C])=[A B C]
    - CONS([A], [B C])=?
    - CONS(A, NIL)=?

[[A] B C]

[A]

  - 注意，CONS(A,B) 无定义

5 - 43

例子…

- 由简化的LISP程序中规定的5种基本函数出发，通过上述IF-THEN-ELSE型的递归程序模型可以构造出许多功能更强的递归程序。
- 例 8 函数MAX(L)

求表的最大元素

表仅含一个元素，最大值就是该元素

$$\text{MAX(L)} \equiv \text{if CDR(L)=NIL then CAR(L)}$$
$$\text{else if CAR(L)>MAX(CDR(L)) then CAR(L)}$$
$$\text{else MAX(CDR(L))}$$

最大值为第一个元素

其它情况

其中，L为非空实数表。

让我们考察以下的计算过程：

MAX([2 5 6 4])=MAX([5 6 4])=MAX([6 4])=6

可以证明，MAX(L)将产生L表中的最大元素，即

$$\text{MAX(L)}=\max_{x\in L}(x)$$

5 - 44

...例子...

例９　用于判断x(原子或表)是否是表L的(顶层)元素的递归程序MEMBER(x,L)。

MEMBER (x,L) ≡ if L=NIL then false
　　else if x=CAR(L) then true
　　else MEMBER(x,CDR(L))

例如：

MEMBER(C,[A B C D])
=MEMBER(C,[B C D])
=MEMBER(C,[C D])
=true

MEMBER(C,[A B [D]]
= MEMBER(C,[B [D]])
= MEMBER(C,[[D]])
= MEMBER(C,NIL)
=false

...例子...

例１０　产生表L1和表L2的交集的递归程序INT(L1,L2)。

INT(L1,L2)≡if L1= NIL then NIL
　　else if MEMBER(CAR(L1),L2) then
　　　CONS(CAR(L1),INT(CDR(L1),L2))
　　else INT(CDR(L1),L2)

例如：

INT([A B C],[B C D])
=INT([B C],[B C D])
=CONS(B,INT([C],[B C D]))
=CONS(B,CONS(C,INT(NIL,[B C D])))
=CONS(B,CONS(C,NIL))
=[B C]

…例子…

例11 产生表L1和表L2的并集的递归程序UNION(L1,L2)。

UNION(L1,L2) ≡ if L1=NIL then L2
else if MEMBER(CAR(L1),L2) then
UNION(CDR(L1),L2))
else CONS(CAR(L1),UNION(CDR(L1),L2))

例如： UNION([A B C],[B C D])
=CONS(A,UNION([B C],[B C D]))
=CONS(A,UNION([C],[B C D]))
=CONS(A,UNION (NIL,[B C D]))
=CONS(A,[B C D])
=[A B C D]

5 - 47

…例子…

例１2　将表Ｌ１和表Ｌ２合并成一个新表，且Ｌ１的元素置Ｌ２之前的递归程序APPEND(L1,L2)

APPEND(L1,L2) ≡ if L1=NIL then L2
else CONS(CAR(L1),APPEND
(CDR(L1),(L2))

例如：
APPEND([A B],[C D])
=CONS(A,APPEND([B],[C D])
=CONS(A,CONS(B,[C D]))
=CONS(A,[B C D])
=[A B C D]

请注意APPEND和CONS的区别：
CONS([A B],[C D])
=[[A B] C D]
APPEND([A B],[C D])
=[A B C D]

5 - 48

…例子…

例１ ３将表Ｌ的元素倒排的递归程序**REVERSE(L)**

REVERSE(L) ≡ if L=NIL then NIL
else APPEND (REVERSE(CDR(L)),
CONS (CAR(L), NIL))

例如：

REVERSE ([A B C])
=APPEND (REVERSE([B C]), [A])
=APPEND (APPEND (REVERSE([C]), [B]), [A])
=APPEND (APPEND (APPNED (REVERSE (NIL),
[C], [B]), [A])
=APPEND (APPEND (APPEND (NIL, [C]), [B]), [A])
=APPEND ([C B], [A])=[C B A]

…例子

例**14** 求自然数**x**的质因子分解表的递归程序**PER(x)**。

PER(x)≡if R(x,2)=0 then CONS(2,PER(x/2))
else PEI(x,3)

求x除以y的余数

其中，**R(x,y)≡if x<y then x**
else R(x-y,y)

求x能否被3,5,7,9,11,13,…整除

PEI(x,y)≡if x=1 then NIL
else if x<y*y then CONS(x,NIL)
else if R(x,y)=0 then CONS(y,PEI(x/y,y))
else PEI(x,y+2)

同步练习

- PER(5)=PEI(5,3)= CON(5,NIL)=[5]

- PER(30)=CONS(2,PER(15))
  =CONS(2,PEI(15,3))
  =CONS(2,CONS(3,PEI(5,3)))
  =CONS(2,CONS(3,CONS(5,NIL)))=[2 3 5]

- PER(35)=PEI(35,3)
  =PER(35,5)
  =CONS(5,PEI(7,5))
  =CONS(5,PEI(7,7))
  =CONS(5, CONS(7,PEI(1,7)))
  = CONS(5, CONS(7,NIL))
  = CONS(5,[7])
  =[5,7]

内容线索

- ✓ 递归的概念
- ✓ 递归数据结构
- ✓ 递归程序
- 递归程序正确性证明
  - 偏序集和良序集
  - 结构归纳法
  - 良序归纳法

偏序集的概念…

- 设有一个非空集合W和一个定义在W上的二元关系，且这个关系满足下列性质：
  - (1)传递性，即对于一切a,b,c∈W,如果a ≺ b,b ≺ c 则有a ≺ c；
  - (2)反对称性，即对于一切a,b∈W,如果a ≺ b则有b ⊀ a；
  - (3)反自反性，即对于一切a∈W,a ⊀ a。

    称W为具有关系 ≺ 的偏序集，记为(W, ≺)。

…偏序集的概念

- 例如：
  - (1) 具有小于关系(<)的0和1之间的实数集合A1是偏序集；
  - (2) 具有关系<的全体整数的集合B1是偏序集。
  - 但是，在上面的例子中若将关系<换为≤，集合A1和B1将不再是偏序集
    - 反对称性和反自反性不满足

良序集的概念…

- 设(W,≺)是一个偏序集，如果不存在由W中的元素构成的无限递减序列$a_0>a_1>a_2\ldots$，则称(W, ≺)为良序集。
- 任一偏序集，假如它的每一个非空子集存在最小元素，这种偏序集称为良序集。

…良序集的概念

- 例如：
  - (1) 若N是自然数的集合，那么(N , ≺)是良序集
  - 而上面的偏序集(A1, ≺), (B1,≺)不是良序集。
  - (2) 若W是非负整数对的集合，在其上定义字典顺序--<，
    - 即如果 $x_i<x_j$
      或者$x_i=x_j$，且$y_i<y_j$
      则说(x_i, y_i) --<(x_j, y_j)

  容易证明，这样规定的(W, ≺)是良序集。

证明

- (1)传递性
  - $\because (x_i，y_i) \mathrel{-\!\!-\!\!<} (x_j，y_j)$, $\therefore x_i \le x_j，y_i < y_j$
  - $\because (x_j，y_j) \mathrel{-\!\!-\!\!<} (x_k，y_k)$, $\therefore x_j \le x_k，y_j < y_k$
  - $\therefore x_i \le x_k，y_i < y_k$
  - $\therefore (x_i，y_i) \mathrel{-\!\!-\!\!<} (x_k，y_k)$
- (2)反对称性
  - $\because (x_i，y_i) \mathrel{-\!\!-\!\!<} (x_j，y_j)$， $\therefore x_i \le x_j，y_i < y_j$
  - $\therefore (x_j，y_j) \not\mathrel{-\!\!-\!\!<} (x_i，y_i)$
- (3)反自反性
  - $X_i，y_i \in W$, $\therefore (x_i，y_i) \not\mathrel{-\!\!-\!\!<} (x_i，y_i)$
- (4)最小元素(0,0)

结构归纳法…

- 递归程序具有如下结构：
  - 首先指出对于自变元的某些“最简单”的数据如何进行计算
  - 然后指出如何将对于“较复杂”的数据的计算通过递归方法归结为“较简单”的数据的计算

…结构归纳法…

- 因此，为了证明递归程序的正确性，我们可以采用如下的步骤：
  - **(1)** 证明对于“最简单”的数据，程序运行正确；
  - **(2)** 假设对于“较简单”的数据，程序运行正确[归纳假设]，在此基础上证明对于“较复杂”的数据，程序亦运行正确
    - 这里所说的数据，是指自变元所允许取的值。它可以是一般的数值，也可以是由数值、原子或者表等组成的某种数据结构

…结构归纳法

- 容易看出，这种证明递归程序正确性的基本思想与通常的数学归纳法是很类似的
  - 两者的区别：
    - 归纳过程不一定是对简单的变量N进行的，而可以是对程序所操作的数据的结构进行的
      - 当然也包括对程序所操作的数据本身进行归纳
      - 作为其特殊情形，实质上就是普通的数学归纳法
- 通常把这种方法称为结构归纳法。

例子…

例15 证明例1中给出的计算阶乘的递归程序。

$F(x)\equiv$ if x=1 then 1
else x*F(x-1)

的正确性。

我们要证明对于所有正整数x， F(x)=x!。因而，这个程序的输入、输出断言为

φ(x):x>0(x为整数)

ψ(x,z):z=x!

…例子…

由于这个程序的自变元是整型变量x，所以直接对正整数x进行归纳：

(1) 证明x=1时，F(1)=1=1!

(2) 归纳假设。设对任意正整数x，F(x)=x!。
在此基础上证明，对正整数x+1， F(x+1)=(x+1)!
事实上，因为x是正整数，所以x+1=1为假，
于是根据程序有
F(x+1)=(x+1)*F((x+1)-1)=(x+1)*F(x)
=(x+1)*x! (根据归纳假设)
=(x+1)!

$F(x)\equiv$ if x=1 then 1
else x*F(x-1)

…例子…

- 这样，我们就证明了对于任何满足输入断言φ(x)的x，程序执行结果将得到z=F(x)=x!，即程序是部分正确的。
- 这个程序的终止性是显然的。事实上，由于(N,<)是一个良序集，所以对于任何x∈N,归纳总在有限步结束(最终都归纳到x=1)。不然将产生一个无限递减序列，这与(N,<)是良序集矛盾。
- 在这个归纳证明中，我们是对程序所操作的数据本身进行归纳。一般说来，只有在相当简单的情形下，才有可能直接对数据本身进行归纳，而在较一般的情形下，必须对程序所操作的数据的结构进行归纳。

…例子…

- 例16 证明例9中给出的递归程序。

```
MEMBER (x,L)≡if L=NIL then false
             else if x=CAR(L) then true
             else MEMBER(x,CDR(L))
```

的正确性(对于任意原子(或表)x和表L)。

我们要证明

	MEMBER(x,L)=true	若x是L的(顶层)元素
或者	MEMBER(x,L)=false	否则

考察这个程序时，我们注意到当对MEMBER进行递归调用时，即MEMBER(x,CDR(L))，其第一个自变元x保持不变，而第二个自变元CDR(L)比调用前的L更简单(指CDR(L)所包含的顶层元素个数比L所包含的顶层元素个数少一个)，于是我们可以对第二个自变元L所包含的顶层的元素的个数进行归纳(而不是数据本身)。

…例子…

(1) 证明对于任何含有0个元素的表，程序运行正确。由于唯一含有0个元素的表是空表NIL，MEMBER(x, NIL)=false，这表明程序运行是正确的，因为x不是空表NIL的顶层元素。

(2) 归纳假设：假设对于一切含有N个(N为非负整数)顶层元素的表L’，MEMBER程序运行正确，即

MEMBER (x, L’)=true 若x是L的(顶层)元素

或 MEMBER (x, L’)=false 否则

在此基础上证明，对于含有N+1个顶层元素的表L，程序也运行正确。

…例子…

事实上，由于N+1>1，故L=NIL为假，根据程序有

MEMBER (x, L)=true 若x=CAR(L)

或 MEMBER (x, L)= MEMBER(x, CDR(L)) 否则

我们分x=CAR(L)和x≠CAR(L)两种情形来讨论。

(1)若x=CAR(L)，这表明x是L的一个(顶层)元素，故MEMBER(x, L)的值为true，程序运行正确。

…例子…

(2)若x≠CAR(L)，因为L≠NIL，CDR(L)有定义，这时MEMBER(x,L)=MEMBER(x,CDR(L))，这是正确的。

因为我们知道在这种情况下，x是L的元素的充分必要条件为x是CDR(L)的元素。而CDR(L)是含有N个元素的表，根据归纳假设有

MEMBER(x,CDR(L))=true　若x是CDR(L)的元素

或 MEMBER(x,CDR(L))=false 否则

故根据x是否为CDR(L)的元素，MEMBER(x,L)正确的取值true或false。

这样就证明了程序的部分正确性，而程序的终止性可根据(N,<)是良序集而推出。

注意，在这个程序中，我们是对程序所操作的数据的结构，而不是对数据本身(即表L本身)进行归纳论证的。

…例子…

- 例17 证明例12中的递归程序

```
APPEND (L1,L2)≡if L1=NIL then L2
        else CONS(CAR(L1),APPEND(CDR(L1),L2))
```

的正确性。

这一程序的功能是将表L1和表L2合并成一个新表，且L1的元素置于L2之前。

注意到当APPEND被递归调用时，第一个自变元变得简单了一些(由L1变为CDR(L1)，而第二个自变元保持不变(均为L2))。因而，和例16类似，可对第一个自变元所包含之元素的个数进行归纳。

…例子…

- 证明：

(1)证明对于含有0个元素的表L1，程序运行正确。

由于此时L1=NIL，执行程序得APPEND (NIL,L2)=L2，而L2可看作是NIL和L2合并，且NIL的元素(实际上没有)置于L2之前的一个表，即程序运行正确。

(2)归纳假设：假设对于任何含有N个元素的表L1'，程序运行正确，即

APPEND(L1',L2)是由L1'的元素后跟L2的元素组成的表

在此基础上证明，对于含有N+1个元素L1，程序运行正确。

5 - 69

…例子

设**L1=[l_1 l_2 … l_{N+1}]**，**L2=[l'_1 l'_2 … l'_M]**，这里**l_i**，**l'_i**分别为**L1**和**L2**的元素。

由于**N+1>1**，故**L1=NIL**为假，执行程序得

APPEND(L1,L2)=CONS(CAR(L1), APPEND(CDR(L1), L2))

=CONS(l_1, APPEND(CDR(L1), L2))

进一步，由于**CDR(L1)**是由**N**个元素组成的表，根据归纳假设，有

APPEND(CDR(L1),L2)=[l_2…l_{N+1} $l'_1$$l'_2$…$l'_M$]

再根据**CONS(x,L)**的定义可知

APPEND(L1,L2)=[l_1 l_2…l_{N+1} $l'_1$$l'_2$…$l'_M$]

即程序运行正确。

这样，就证明了程序的部分正确性。

程序的终止性可根据**(N,<)**是良序集得出。

5 - 70

良序归纳法

- 例16和例17虽然比例15要复杂一些，但仍然是比较简单的，因为它们均可直接对表中所含的元素个数进行归纳。
- 对于更为复杂一些的情形，为了证明程序的正确性，需要采用较强形式的结构归纳法——良序归纳法。
- 设(W, $\prec$)是一个良序集，P(x)是一个命题，为了证明对于所有的x∈W，P(x)为真，只要

(1) 证明$P(x_0)$为真(x_0是W中“最小”元素)

(2) 归纳假设：假设对于所有的x’ $\prec$ x，P(x’)都为真。在此基础上证明P(x)为真。

例子…

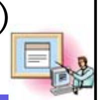
- 例18 证明例6中的递归程序(Ackermann函数)

$$
\begin{aligned}
A(x_1, x_2) \equiv\ & \text{if } x_1=0 \text{ then } x_2+1 \\
& \text{else if } x_2=0 \text{ then } A(x_1-1,1) \\
& \text{else } A(x_1-1,A(x_1,x_2-1))
\end{aligned}
$$

对任何非负整数对(x_1, x_2)计算终止。

- 考察这个程序，我们注意到：

在第一个A被调用的地方(即$A(x_1-1,1)$)，其第一个自变元(即x_1-1)比调用前(即x_1)简单(指$(x_1-1)<x_1$)，在第二个A被调用的地方(即$A(x_1-1, A(x_1, x_2-1))$)，外层调用的第一个自变元(即x_1-1)比调用前(即x_1)简单，内层调用的第二个自变元(即x_2-1)也比调用前(即x_2)简单。



例子…

- 这启示我们选取良序集为$(\mathbf{W}, \prec)$，其中，$\mathbf{W}=\{(x_1, x_2) \mid x_1 \geq 0 \wedge x_2 \geqslant 0$，$x_1, x_2$是整数$\}$；$\prec$为字典顺序。
- 设命题P(x1, x2)为A(x1, x2)计算终止。为了证明P(x1, x2)对所有的非负整数对成立，分以下两步进行。

(1)证明对于W中的最小元素(0，0)，P成立。
由于A(0, 0)=0+1=1，计算显然终止，
即P(0, 0)为真。

(2)归纳假设：假设对于所有的(x1’, x2’) ≺ (x1, x2)，P(x1’, x2’)为真。
在此基础上证明P(x1, x2)为真。
事实上，可分以下3种情形来论证。

…例子…

- (1)若x1=0，这时 A(x1, x2)= A(0, x2)=x2+1 计算显然终止。
- (2)若x1≠0，x2=0，这时 A(x1,x2)=A(x1,0)= A(x1-1,1)，
- 由于(x1-1,1) ≺ (x1, x2), 根据归纳假设A(x1-1,1)计算终止，
- ∴ A(x1,x2)计算终止
- (3)若x1≠0，x2 ≠ 0，这时 A(x1, x2)= A(x1-1, A(x1, x2-1))
- 根据“最左，最内”计算规则，接下去应先计算A(x1, x2-1)。
- 由于(x1, x2-1) ≺ (x1, x2)，根据归纳假设，可知A(x1, x2-1)计算终止。
- 假设A(x1, x2-1)=z，接下去将计算A(x1-1,z)，
- 由于对任何z，(x1-1,z) ≺ (x1, x2)，
- 再次根据归纳假设，可知A(x1-1,z)计算终止，
- 因此A(x1, x2)的计算也将终止。

…例子…

- 例19 证明例5中的递归程序(McCarthy91函数)
  M(x) ≡ if x>100 then x-10
  else M(M(x+11))
  的正确性(对所有整数x)。
  更确切的说，要证明这一递归程序的计算结果是：

$$M(x)=\begin{cases} x-10 & 若x>100 \\ 91 & 若x\leqslant 100 \end{cases}$$

显然，根据递归程序的定义，

当x>100时， M(x)=x-10。

因此只要证明，当x<=100时，M(x)=91。

为此，选取良序集(W, ≺)，

其中，W={x ｜ x是整数，且x<=100}

选为一般的<关系的逆次序，即x1≺x2当且仅当x2<x1。

…例子…

(1) 证明对于W中的“最小”元素x_0，$M(x_0)$=91。

事实上，根据我们选取的≺，W中的“最小”元素是100。按照递归程序，有

M(100)=M(M(100+11))=M(M(111))

= M(111-10)= M(101)

=101-10=91

(2) 归纳假设：对于一切x’≺x, M(x’)=91。

在此基础上证明M(x)=91。

由于x’<=100而x’≺x，即x<x’，故x<100，分以下两种情况讨论。

…例子…

①若x+11<100，这时x<90,

根据递归程序M(x)=M(M(x+11))

由于x+11<=100,故x+11∈W

由于x+11>x,故x+11 $\prec$ x

根据归纳假设，M(x+11)=91，从而M(x)=M(91)

又由于x<90<91,故91 $\prec$ x,再次根据归纳假设

M(91)=91

于是 M(x)=91

②若x+11>100，这时90<=x<100，

根据递归程序定义M(x)=M(M(x+11))=M(x+11-10)=M(x+1)

由于x+1<=100,又x+1>x,故x+1 $\prec$ x,

根据归纳假设有M(x+1)=91，从而

M(x)=91

…例子…

- 例20证明例2中的递归程序

  GCD(x,y) ≡ if x=0 then y

  else if y<x then GCD(y,x)

  else GCD(x,y-x)

的正确性(对于满足x>0∧y>0∧(x≠0∨y≠0)的任意整数x和y)。

在证明过程，我们将用到GCD函数的下述基本性质：

(1) GCD(0，y)=y

(2) GCD(x，y)=GCD(y，x)

(3) GCD（x，y）=GCD（x，y-x）（当y>x时）

…例子…

- 考察这个程序，我们发现第一个递归调用GCD的地方(即GCD(y,x)),其第一个自变元(即y)比调用前(即x)简单(指y<x)。第二个递归调用 GCD(x,y-x)，其第一个自变元(即x)保持不变，而第二个自变元(即y-x)比调用前(即y)简单(指y-x<y),

  这就启示所选取良序集(W, $\prec$)为
  $$W=\{(x,y)|x>0\wedge y>0\wedge(x\neq 0\wedge y\neq 0)$$
  为字典顺序。

…例子…

- 为证明GCD(x,y)正确地计算x和y的最大公约数，则需要：
  - (1) 证明对于W中的最小元素(0，1)，程序能正确地运行。
    事实上，这时有GCD(0,1)=1，这显然是正确的。
  - (2) 归纳假设：假设对于一切(x’,y’) < (x,y)，程序正常地运行，在此基础上证明，对(x,y)程序也正确地运行。
    事实上，可就以下3种情形分别论证。

…例子…

①x=0,这时GCD(x,y)=GCD(0,y)=y程序运行正确。这是因为由GCD函数的性质(1)可知，程序运行正确。

②x≠0∧y>x,这时GCD(x,y)=GCD(y,x)程序运行正确，这是因为：

一方面有GCD函数的性质(2)；

另一方面，由于x≠0∧y>x，故(y,x)< (x,y),根据归纳假设，可知GCD(y,x)正确地工作，从而GCD(x,y)也正确地运行。

③ x≠0 Λ y>=x，这时GCD(x,y)=GCD(x,y-x)，程序运行正确。这是因为：

一方面有GCD函数的性质(3)；

另一方面，由于x≠0 Λ y >= x，故y-x<y，因而(x,y-x) ＜ (x,y)，根据归纳假设，GCD(x,y-x)正确的工作。从而GCD(x,y)也正确的运行。

…例子…

顺便指出，若将这个递归程序稍加改动，即改成如下的递归程序

GCD1(x,y)≡if x=0 then y
else if y<x then GCD1(y,x)
else GCD1(x,y-x)

那么，不难发现对许多输入(例如对x=y的输入(x,y))，GCD1 的程序将不终止。

…例子

例如，GCD1(4,4)=GCD1(4,0)=GCD1(4,0)=…..。
问题发生在哪里？

注意到在第一个调用GCD1的地方(即GCD1(x-y,y)，根据$x\neq 0 \land y<x$，例如y=0时x-y=x)，从而不能使用归纳假设来论证。

GCD程序和GCD1程序在形式上十分相似，但其在终止性方面却有很大差别，这从一个侧面说明对递归程序(乃至一般循环程序)进行验证的必要性。

作业4

- P205
- 5．设L_1和L_2是含有相同分量个数之数值向量，若把它们当作表，试编写计算L_1和L_2内积的递归程序$IP(L_1, L_2)$，并证明它的正确性。
  - 内积：
- *7．设N为自然数，考察下面的递归程序

  F(N)≡if N>202 then N-3
  else F(F(N+4))

  试证明对一切自然数N

$$\begin{cases} F(N)=N-3 & N>202 \\ F(N)=200 & N\leq 202 \end{cases}$$