第7卷第3期
1996年3月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 3
Mar. 1996

# 归纳法模式的自动生成*

李卫华 张 黔 张 亮 刘 娟

(武汉大学计算机科学系 武汉 430072)

**摘要** 本文介绍归纳法推理系统的核心——归纳法模式的自动生成程序.该程序从递归函数定义出发,自动生成归纳法模板;从待证定理出发,借助归纳法模板,自动生成归纳法模式和归纳法公式.该系统已在微机上用编译LISP语言实现.

**关键词** 归纳法模板,归纳法模式,归纳法推理.

归纳法推理系统通过重写待证公式来证明定理,整个推理过程无需回溯.[1,2]系统先后运用了重写简化、分元符删除、交融、推广、无关式删除等推理策略[3],在这些策略均不奏效时,对待证公式采用归纳法,以进一步推理.由于最常用的数据结构(如整数、表、树、公式、堆栈等)均是递归定义的,这就要求证明程序属性的推理机具有处理归纳构造对象的能力.本系统具有这一能力,它从递归函数定义出发生成归纳法模板,从待证公式和归纳法模板出发生成归纳法公式.

## 1 归纳法模板的自动生成

### 1.1 归纳记录

**定义** 1. 递归函数 $fn$ 的归纳记录定义为:($tests\&cases$ ($tests$ $cases$) ()). 其中称 $tests\&cases$ 为记录名,称 $tests$ 和 $cases$ 为字段名.

一个递归函数的所有归纳记录由形式如下的表组成:

(($tests\&cases$ $test_1$ $case_1$) ... ($tests\&cases$ $test_n$ $case_n$))

其中 $test_i(i=1,\ldots,n)$ 形为($test$ ... $test$),是控制递归调用 $case_i$ 的条件合取式; $case_i$ 形为($case$ ... $case$),每一个 $case$ 是一次递归调用 $fn$ 的参量表.

例如,设 *Ackermann* 函数的定义如表1(a)所示,其归纳记录如表1(b)所示.

设 $fn$ 是递归函数名,$term$ 是 $fn$ 的定义体,$tests$ 是控制 $fn$ 递归调用的测试集(其初值为空),定义如表2中的 *induction-R* 可求出 $fn$ 的归纳记录表.

---

* 本研究得到国家863高科技项目和国家教委跨世纪优秀人才基金资助.**李卫华**,1952年生,教授,博士导师,主要研究方向为人工智能,知识工程,多媒体软件.**张黔**,1973年生,硕士研究生,主要研究方向为归纳法推理,多媒体软件.**张亮**,1963年生,博士生,主要研究方向为知识工程,多媒体软件.**刘娟**,女,1970年生,博士生,主要研究方向为知识工程,多媒体软件.

本文通讯联系人:李卫华,武汉 430072,武汉大学计算机科学系

本文1995-10-08收到

**表 1 函数定义及其归纳记录**

```
(defn (ack m n)
  (if (zero? m)
      (add1 n)
      (if (zero? n)
          (ack (sub1 m) 1)
          (ack (sub1 m) (ack m (sub1 n))))))
```

(a)

```
((tests&cases ((zero? m)) ())
 (tests&cases ((not (zero? m)) (zero? n))
              (((sub1 m) 1)))
 (tests&cases ((not (zero? m)) (not (zero? n)))
              (((sub1 m) (ack m (sub1 n)))
               (m (sub1 n)))))
```

(b)

**表2 *fn* 的归纳记录表**

```
(define (induction-R fn term tests)
  (if (or (variable? term) (fquote? term) (not (eq? (ffn-symb term) 'if)))
      (list (make tests&cases (append tests ()))
            (union-equal
              (do ((test tests (cdr test)) (ans () ans))
                  ((null? test) ans)
                  (set! ans (union-equal (all-arglists fn (car test)) ans)))
              (all-arglists fn term))));all-arglists 求 fn 的参量表
    (nconc (induction-R fn (fargn term 2)      ;(fargn term i)求 term 的第 i+1个项
                        (append tests (list (fargn term 1))))
           (induction-R fn (fargn term 3)
                        (append tests (list (negate-lit (fargn term 1)))))))  ;negate-lit 求 term 的否定
```

## 1.2 终止记录

**定义**2. 递归函数 *fn* 的终止记录定义为:(*tests&case* (*tests case*) ()),终止记录表形为:((*tests&case* $test_1$ $case_1$) ... (*tests&case* $test_n$ $case_n$)).

设 *fn* 是递归函数名,*term* 是 *fn* 的定义体,*tests* 是控制 *fn* 递归调用的测试集(其初值为空),定义如下的 *terminat-R* 求 *fn* 的终止记录表:

(*define* (*terminat-R fn term tests*) ...)　(略)

设函数 *fn* 的终止记录为 *terminat-R*,形参表为 *formals*,*guess-RMs* 返回一张 *RM* 表:

((*LESS*? (*COUNT* $x'_1$)) ... (*LESS*? (*COUNT* $x'_n$))),

其中 $x'_i$ 为($x_1$ ... $x_n$)中的某个变量:

```
(define (guess-RMs formals terminatR)
  (do ((var formals (cdr var))
       (i 0 (add1 i)) (ans () ans))
      ((null? var) (reverse ans))
      (when (for x in terminatR always      ;access 把记录中字段值存入变量 x
                 (and (occur-lst (car var) (access tests&case tests x))
                      (not (equal (car var) (nth i (access tests&case case x))))))
            (set! ans (cons (list 'less? (list 'count (car var))) ans)))))
```

当系统无法自动证明递归函数 *fn* 满足定义条件时,就借助用户提示的归纳法引理来证明 *fn* 满足定义条件. 例如:

```
(defn (gcd x y)
  (if (zero? x) (fix x)
      (if (zero? y) x
```

```
(defn (lex2 l1 l2)
  (or (less? (car l1) (car l2))
      (and (equal (car l1) (car l2))
```

```
    (if (less? x y)(gcd x (diff y x))          (less? (cadr l1) (cadr l2))))
      (gcd (diff x y) y))))          ())
  (lex2 (list (count x) (count y))))
```

有了函数 *fn* 的终止记录、形参表和 *RMs* 后，就可用下一程序证明 *fn* 满足定义条件：

```
(define (prove-terminat formals RMs terminat-R) ...)   (略)
```

### 1.3 证实记录

**定义**3. 在检验函数 $f$ 是否满足定义条件时，常常存在一个测度 $m$ 和形参$(x_1 \ \ldots \ x_n)$的一个子集 $s=(x_1 \ \ldots \ x_j)$，使得把$(m \ x_1 \ \ldots \ x_n)$ 定义为 $(m \ x_1 \ \ldots \ x_j)$时，就能满足定义条件. 若这个 $m$ 和 $s$ 存在，就称 $s$ 为 $f$ 的一个被测度子集.

**定义**4. 函数 *fn* 的递归调用提示的归纳法证实记录为：

```
(justification (subset measure-term R lemmas) ())
```

其中 *subset* 为 *fn* 之形参的被测度子集，*measure-term* 是作用于 *fn* 形参子集的项，$R$ 是一良基关系，*lemmas* 是用于证明项 *measure-term* 按良基关系 $R$ 下降要用到的引理. 生成证实记录的程序为：

```
(define (put-induct-info fn formals body RMs tako)
  (define terminatR (terminat-R (or tako fn) body ()))   (define ind-R ())
  (cond ((null? terminatR) (set! all-lemmas-used ()) ())
        (t (or RMs (set! RMs (guess-relat-measure-l formals terminatR)))
           (add-fact fn 'justificats
              (or (for RM in RMs when (prove-terminat formals RM terminatR)
                       save (make justification (all-vars (cadr RM)) (cadr RM) (car RM)
                                  prove-terminat-lemmas-used))
                  (list (make justification formals () () ()))))
           (set! all-lemmas-used ())
           (set! ind-R (induct-R fn (if tako (subst-fn fn tako body) body) ()))
           (add-fact fn 'induct-R ind-R)
           (add-fact fn 'quick-block-info (quick-block-info formals ind-R))
           ())))
```

上面的(*quick-block-info formals ind-R*)用于求 *fn* 的快锁信息：

```
(... self-reflexive|unchanging|questable ...)
```

第 $i$ 个形参位置出现的 *self-reflexive* 表示第 $i$ 个参量在递归调用中测度$(M \ x_i)$按良基关系 $R$ 下降；出现 *unchanging* 表示测度$(M \ x_i)$保持不变；出现 *questable* 表示测度$(M \ x_i)$变化不能判断. 例如，Ackermann 函数的快锁信息为：(*questable self-reflexive*).

**定义**5. 对于与递归函数 *fn* 相关的每一个被测度子集，若找到一个与之相关的测度、良基关系，则称该被测度子集、测度、良基关系以及归纳记录、终止记录、证实记录等信息构成了一个归纳法模板，即提示了一个与 *fn* 相关的归纳法.

例如，*Ackermann* 函数的归纳法模板是：被测度子集$(m \ n)$，测度(*list* (*count m*) (*count n*))，良基关系是 *lex2*. 为证明含有 *ack* 的公式$(p \ m \ n)$，可证明：

```
(and (implies (zero? m)) (p m n))
     (implies (and (not (zero? m)) (zero? n) (p (sub1 m) 1)) (p m n))
     (implies (and (not (zreo? m)) (not (zero? n)) (p (sub1 m) (sub1 n))) (p m n)))
```

## 2　归纳法模式的自动生成

从待证公式($p\ x_1\ \ldots\ x_n$)出发,查看($p\ x_1\ \ldots\ x_n$)中的每一个项,分析由每个项提示的归纳法模板,判断该模板可否成为归纳法模式,进而采用各种启发式法挑选最好的归纳法模式,这是归纳法模式自动生成程序 *induct* 应完成的主要工作:

(*define* (*induct cl-set*) ...)　(略)

### 2.1　归纳法模式的选择

**定义**6. 出现在被测度位置中,有时又在递归中改变的变量称为可变元.

**定义**7. 出现在那些占有被测度位置的项中,且绝不在递归中改变的变量被称为不可变元.

**定义**8. 若一被测度项不是变量,或是与另一测度相同的变量,称该模板为不可用模板.

**定义**9. 当用实参例示归纳法模板时,下述3种代换偶对称为应删除代换:代换非变量的偶对;代换不可变元的偶对;多重性代换的偶对,即要对同一变量作二次代换.

设项($fn\ t_1\ \ldots\ t_n$)出现在待证公式中,设已知递归函数的一个归纳法模板,则可通过删除不可用模板,并去掉应删除代换选择归纳法模式:

```
(define (get-cands term)
  (define (possible-ind term) ...) ;略
  (cond ((variable? term) ())
        ((quote? term) ())
        (t (nconc (possible-ind-principles term)
                  (for arg in (fargs term) splice (get-cands arg))))))
```

### 2.2　归纳法模式的包括检查

归纳法模式的包括检查用于抛弃被别的归纳法模式"包括"的任何归纳法模式.

**定义**10. 被代换的(不管是被测度的还是未被测度的)所有变量集称为更换变量.

**定义**11. 由所有不可变元和所有变量组成的集称为不更换变量.

**定义**12. 若 $s_1$、$s_2$ 满足以下3个条件,则称 $s_2$ 包括 $s_1$:

(1)$s_1$ 的更换变量必为 $s_2$ 中更换变量的一个子集;

(2)$s_1$ 的不更换变量必为 $s_2$ 中不更换变量的一个子集;

(3)$s_1$ 的每种情况被 $s_2$ 的一种情况包括,即 $s_1$ 情况的测试都是 $s_2$ 情况中测试的一个子集,且对于 $s_1$ 情况中的每一代换,存在着 $s_2$ 情况中的一个代换,使得对于 $s_1$ 代换中的每一组分,存在 $s_2$ 中带有同一变量的一个组分,并存在一个项,它提到 $s_1$ 组分中的项.

如果因为 $s_2$ 包括 $s_1$ 而抛弃 $s_1$ 的话,就把 $s_1$ 的分数加到 $s_2$ 的分数上,并在 $s_2$ 计算的所有项中加上 $s_1$ 计算的所有项.

**定义**13. 候选归纳法记录定义为:

```
(candidate (score controllers changed-vars unchangeable-vars tests & alists-lst
                  justfication induct-term other-term)  ())
```

其中 *score* 为分数,*changed-vars* 和 *unchangeable-vars* 为更换变量、不更换变量,*induct-term* 用于提示归纳法.

### 2.3 归纳法模式的合并

若模式 $s_1$,$s_2$满足以下条件,则可将模式 $s_1$合并成 $s_2$:模式 $s_1$,$s_2$的更换变量有一个非空交集,它们的不更换变量交集为空,且对于 $s_1$情况中的每一代换,可找到 $s_2$情况中的一个代换,使得2个代换替代某个公共变量,2个代换对所有公共变量进行恒等替代,并且至少存在一个公共变量 $v$,使得替代 $v$ 的项不是 $v$ 本身.

$s_1$,$s_2$合并后的结果很象 $s_2$. 不同的是,对于 $s_2$中的每一情况,若它曾吸收了一个 $s_1$情况,则把那个 $s_1$情况中的测试加到这个 $s_2$情况的测试中,并把它所相交的一个 $s_1$代换中的任何新偶对加到这个 $s_2$情况中的每一代换里. 把 $s_1$的更换变量加到 $s_2$的更换变量中,并把 $s_1$的不更换变量加到 $s_2$的不更换变量中,新模式的分数为 $s_1$与 $s_2$的分数之和,且新模式计算2个旧模式的所有项. 最后,抛弃 $s_1$并保留修改过的 $s_2$.

(*define* (*merge-cand1-into-cand2 cand1 cand2*) ...) (略)

### 2.4 缺陷归纳法模式的清除

**定义**14. 若存在一个 $s_1$计算的项 $t$ 和关于 $t$ 之函数符的一个模板 *template*,使得 $v$ 相对于 $t$ 和 *template* 而言是一个可变元,则称 $v$ 是 $s_1$的一个归纳法变量.

**定义**15. 对于任一模式 $s_1$,若存在另一模式 $s_2$,使得 $s_2$的某个更换变量或不更换变量是 $s_1$的一个归纳法变量,则把 $s_1$当做是有缺陷模式.

若某一模式是无缺陷的,则抛弃全部有缺陷模式;若所有模式均有缺陷,则一个模式也不抛弃,并继续其它启发式法.

(*define* (*compute-vetoes candl*) ...) (略)

### 2.5 归纳法模式的优选

如果在运用了以上方法之后还有多个候选模式未处理,就把具有最高分数的模式看作是最可能成功的. 这时,两个或多个模式若因为具有同样的高分而打起结来,系统将运用如下解结规则:选择非原始递归项个数最多的归纳法模式.

```
(define (high-scores candl)
        (max-elems candl (lambda (cand) (access1 candidate score cand))))
```

### 2.6 归纳法公式的生成

若从所选择的归纳法模式中得到了 $x_i$,$q_i$ 和 $s_{ij}$,就把归纳法原理应用到相应于其文字析取式的那个项中,产生 $k+1$个项. 然后,在展开所有的 and,or,not,implies 后,把这些项转换成与命题等价的一个无 if 子句集,供系统证明.

(*define* (*ind-formula tests&alists-l terms cl-set*) ...) (略)

## 3 归纳法模式生成举例

为增加推理能力,系统采用了元函数策略,即允许把解决特定问题的元函数交给系统. 在验证此元函数的正确性之后,推理系统可应用该元函数. 例如,运用加法消去律,

(*equal* (*equal* (*plus x y*) (*plus x z*)) (*equal* (*fix y*) (*fix z*)))

可把等式 (*equal* (*plus i* (*plus x y*)) (*plus i* (*plus j k*)))

重写为: (*equal* (*plus x y*) (*plus j k*)).

但对于下式 $P$:　(*equal* (*plus* (*plus a i*) (*plus b k*)) (*plus j* (*plus k* (*plus i x*))))

却因为公共因子 $i$,$k$ 不是最外层那个 *plus* 表达式的第一参量,无法应用加法消去律.为证明此公式,可引入消去函数 *cancel*.[2]证明消去函数的正确性,实为证明:

(implies (*form*? *x*) (and (*equal* (*meaning x a*) (*meaning* (*fn x*) *a*)) (*form*? (*fn x*))))

可将上式分成6条引理:

*L*1. (implies (*form*? *x*) (*form-lst*? (*fringe x*)))

*L*2. (implies (*form-lst*? *x*) (*form-lst*? (*bagdiff x y*)))

*L*3. (implies (*form-lst*? *x*) (*form*? (*plus-tree x*)))

*L*4. (implies (*subbag y x*)
(*equal* (*meaning* (*plus-tree* (*bagdiff x y*)))
(*difference* (*meaning* (*plus-tree x*)) (*meaning* (*plus-tree y*)))))

*L*5. (implies (*subbag x y*)
(*lesseq*? (*meaning* (*plus-tree x*)) (*meaning* (*plus-tree y*))))

*L*6. (implies (*plus-tree*? *x*)
(*equal* (*meaning* (*plus-tree* (*fringe x*))) (*meaning x*)))

先证明 *L*1,*L*2、*L*3的证明与 *L*1类似.对 *L*1作用归纳法,生成的归纳法模式为:

*P*1. (implies (not (*list*? *x*)) (implies (*form*? *x*) (*form-lst*? (*fringe x*))))

*P*2. (implies (and (*list*? *x*) (not (*list*? (*car x*))))
(implies (*form*? *x*) (*form-lst*? (*fringe x*))))

*P*3. (implies (and (*list*? *x*) (*list*? (*car x*)) (*equal* (*caar x*) "*quote*"))
(implies (*form*? *x*) (*form-lst*? (*fringe x*))))

*P*4. (implies (and (*list*? *x*) (*list*? (*car x*)) (not (*equal* (*caar x*) "*quote*"))
(not (implies (*form*? (*cdr x*)) (*form-lst*? (*fringe* (*cdr x*))))))
(implies (*form*? *x*) (*form-lst*? (*fringe x*))))

再对每种情况分别作重写、简化等策略,即可得证.

同理可证 *L*4、*L*5、*L*6.证明完消去函数的正确性后,便可将其对应代码加到简化程序模块中,供证明用.将 *cancel* 作用于上面的 $P$ 式后,

(and (*equality*? *P*) (*plus-tree*? (*cadr P*)) (*plus-tree*? (*cadddr P*)))为真,

(*fringe* (*cadr P*))=$\{a,i,b,k\}$,(*fringe* (*caddr P*))=$\{j,i,k,x\}$,则(*cancel P*)的结果为:

(*equal* (*plus a b*) (*plus j x*)).证毕.

## 4　结束语

作者用编译 LISP[5]语言在微机上实现了归纳法推理系统.[6]一般而言,推理系统性能的2个主要测试条件为:①系统表示特定领域中典型事实和定理的能力;②系统在合理时间内完成推理的能力.本系统可以表示许多有趣的事实和定理,并已证明出一批有价值的程序属性及定理,而这些证明很难用手工完成.由于系统可以利用以前已经证明出的引理,所以

系统能在较短时间内完成困难问题的推理过程.

## 参考文献


1 Manna Z, Waldinger R. The deductive foundations of computer programming. Addison—Wesley Pubilshing Company, Inc., 1993.

2 刘叙华. 基于归结方法的自动推理. 北京:科学出版社,1994.

3 Boyer R S, Moore J S. 计算逻辑. 李卫华译. 计算机工程与应用,1982,(4),(5);1983,(7).

4 Boyer R S, Moore J S. 计算机科学中的正确性问题. 李卫华译. 计算机工程与应用,1984,(10),(11).

5 李卫华,陈兆乾,潘金贵. 人工智能程序设计. 北京:科学出版社,1989.

6 李卫华,张黔,刘娟等. 归纳法推理系统. 计算机学报,1996,(3).


# AUTOMATIC GENERATION OF INDUCTION SCHEMA


Li Weihua Zhang Qian Zhang Liang Liu Juan

(*Department of Computer Science Wuhan University Wuhan* 430072)



**Abstract** This paper discusses the kernel program of an induction inference system—— the automatic generator of induction schema. Starting from the definition of a recursive function, the system generates the induction template automatically. Starting from the theorem to be proved, it can also automatically produce candidate induction schema and formula by using the template. The system has been implemented on microcomputers.



**Key words** Induction template, induction schema, induction inference.