第4卷第2期
1993年4月

软 件 学 报
JOURNAL OF SOFTWARE

Vol.4,No.2
Apr.,1993

基于轨迹的归纳程序综合

王志坚 章骏 徐家福

(南京大学计算机软件研究所,南京 210008)

INDUCTIVE PROGRAM SYNTHESIS FROM TRACES

Wang Zhijian, Zhang Jun and Xu Jiafu

(*Institute of Computer Software, Nanjing University, Nanjing* 210008)

Abstract This paper presents the inductive method for synthesizing programs from traces in NDIPS system. The matching identification algorithm is applied to the traces, generated by problem mechanism or provided by users, to form the recursive relations, and to synthesize the target programs. Its theory and implementation are discussed, and the synthesizing process of hanoi program is presented for the illustration.

摘要 本文讨论了NDIPS系统中基于轨迹的归纳程序综合方法.该方法对问题求解机制产生的或手工给出的程序执行轨迹,使用匹配识认算法找出其递归关系,进而生成目标程序.文章从理论和实现上对这种方法进行了探讨,用hanoi程序的归纳综合为例展示了方法的使用.

§1. 引 言

基于枚举识认的归纳程序综合算法要求给定的描述语言相对目标程序是完备的[1].例如,当描述语言中不含append描述成分时,仅通过reverse的输入/输出实例不能综合出正确的reverse程序:

```
reverse([ ],[ ]).
reverse([A|X],Y):-reverse(X,Z),append(Z,[A],Y).
```

没有方法能判定一任意给定的描述语言是否能正确描述目标程序.描述成分过多,将使程序假设空间急剧变大;描述成分太少,则限制了可综合的目标程序的范围.传统的做法是使描述语言仅包含原始递归集上的基本函数,任何其它程序都可以通过这些基本函数去构作,以保证描述语言的完备性.但是,仅用基本函数构作规模较大的程序会产生组合爆炸.解决途径

本文1991年3月定稿.本工作是国家863计划资助项目.作者**王志坚**,讲师,1989年博士毕业于南京大学,主要研究领域为归纳推理、机器学习、面向对象程序设计.**章骏**,讲师,1990年硕士毕业于南京大学,主要研究领域为软件自动化.**徐家福**,教授,所长,主要研究领域为程序设计语言、新型程序设计、软件自动化.

有2个:层次地构作目标程序和通过执行轨迹构作目标程序.

本文讨论NDIPS系统中基于轨迹的归纳程序综合方法.方法的基本思想是:对通过问题求解机制产生的或手工给出的程序执行轨迹,采用结构匹配识认算法找出其递归关系,进而生成目标程序.

§2. 设计思想

为了能归纳综合出目标程序,系统必须能以某种方式获得所需的信息.输入/输出实例是传递信息的一种方式,但当输入与输出之间的差异较大时,直接识认其内在关系极为困难.程序执行轨迹是沟通输入与输出的一种有效手段.与输入/输出实例相比,轨迹能提供更多的信息.例如

实例: sort([3,1,4,2])=[1,2,3,4]

轨迹: sort([3,1,4,2])→[ ]
[1,4,2]→[3]
[4,2]→[1,3]
[2]→[1,3,4]
[ ]→[1,2,3,4]

实例只给出了排序的信息,而轨迹则进一步说明它是插入排序.从这种意义上讲,实例是轨迹的特殊形式.特别地,不同的程序执行轨迹,提供了同一问题的不同算法信息.

在技术上,已有的基于输入/输出实例的归纳程序综合算法都是按某种内部定义的模式,从给定实例形成其半迹(semitrace,具有多种计算次序的轨迹),通过结构匹配找出其递归关系.各种算法的区别在于产生和分析半迹所采用的方法不同[2].基于轨迹的归纳算法的设计思想是,将程序综合作为一问题求解任务[3],二者有着自然的对应关系:

程序综合	问题求解
输 入 域	初 态 集
输 出 域	终 态 集
背景知识	操 作 集

其中,背景知识包括从给定输入计算所需输出的各种转换规则.

如果将Lisp的car,cdr,cons等基本函数作为问题求解的操作集,实例的输入和输出分别作为问题求解的初态和终态,则问题的特解等价于对应实例的半迹,而问题的通解就是所要综合的目标程序.因此,已有的Lisp程序综合系统都可以用问题求解模型刻划.特别是,通过背景知识可以定义不同的半迹形式,较之已有模型更具灵活性.

§3. 基本概念

问题求解的基本技术是搜索. 为了便于从解路径的分析中发现解的一般规律，必须排除那些伪信息——这类信息通常隐藏在冗余的操作中. NDIPS 系统采用宽度优先搜索，该搜索策略是完备的，即如果问题有解，则能找到这个解，并且是最优解（解路径最短者）. 显然，这样的解不含冗余操作.

定义 3.1：从程序开始执行到终止过程中的操作序列称为程序的执行轨迹：

$$[P_1(f_1(\overline{X}_0,\overline{X}_1),\ldots,P_n(f_n(\overline{X}_1,\ldots,\overline{X}_{n-1}),\overline{X}_n)]$$

其中 $\overline{X}_0$ 是输入变量表，$\overline{X}_1,\ldots,\overline{X}_{n-1}$ 是中间变量表，$\overline{X}_n$ 是输出变量表，P_i 是操作名，f_i 是变量表选取函数.

上述的轨迹定义等价于下列的 Prolog 程序：

$$P(\overline{X}_0,\overline{X}_n):-P_1(f_1(\overline{X}_0),\overline{X}_1),$$
$$\ldots,$$
$$P_n(f_n(\overline{X}_0,\ldots,\overline{X}_{n-1}),\overline{X}_n).$$

定理 3.1：任意一个 Prolog 程序都可以用上述的轨迹形式加以表示.

该定理建立了 Prolog 程序与其执行轨迹的对应关系.

定理 3.2：规模为 k 的实例所对应的轨迹为：

$$T_k=[P_1^k(f_1^k(\overline{X}_0^k),\overline{X}_1^k),$$
$$\ldots,$$
$$P_{n^k}^k(f_{n^k}^k(\overline{X}_0^k,\cdots,\overline{X}_{n^k-1}^k),\overline{X}_{n^k}^k)].$$

特别地，当 $\overline{X}_i^k(1\leqslant i\leqslant n^k)$ 中的元素都输入变量表 $\overline{X}_0^k$ 中的元素时有 $f_i^k(\overline{X}_0^k,\ldots,\overline{X}_{i-1}^k)=\overline{X}_i^k$，故上述轨迹可简记为：

$$T_k=[P_1^k(\overline{X}_1^k),\ldots,P_{n^k}^k(\overline{X}_{n^k}^k)]$$

定理 3.3：实例规模为 k 的实例所对应的程序为：

$$P^k(\overline{X}_0^k,\overline{X}_{n^k}^k):-P_1^k(\overline{X}_1^k),\ldots,P_{n^k}^k(\overline{X}_{n^k}^k).$$

其中，程序 P^k 的输入是 $\overline{X}_0^k$，输出是 $\overline{X}_{n^k}^k$.

§4. 匹配识认

对轨迹 T_k 的主要匹配过程如下：

1. 分别用 $T_{k-1},T_{k-2},\ldots$ 等和 T_k 去匹配；
2. 对匹配结果

$$[P_1^k(\overline{X}_1^k),\ldots,P_i^k(\overline{X}_i^k),T_m,P_j^k(\overline{X}_j^k),\ldots,P_{n^k}^k(\overline{X}_{n^k}^k)]$$

中的 $[P_1^k(\overline{X}_1^k),\ldots,P_i^k(\overline{X}_i^k)]$ 和 $[P_j^k(\overline{X}_j^k),\ldots,P_{n^k}^k(\overline{X}_{n^k}^k)]$ 再用 $T_{m-1},T_{m-2},\ldots$ 再去匹配；

3. 重复上述步骤，直至所有匹配都不成功.

对轨迹 T_k 的主要识认过程如下：

1. 利用 T_m 对 T_k 的划分（$m<k$）建立递归关系 R_k；
2. 利用背景知识确定余下的谓词符号 $P_i,P_i\in T_m$.

定理 4.1:轨迹 $T_1,\ldots,T_{k-1}$和关系 R_k 能覆盖规模不超过 k 的实例.

定理 4.2:如果存在正整数 m,使得对所有的 n(>m)都有 $R_n=R_m$,则上述算法在有限时间内正确识认了目标程序(轨迹).

§5. 实例分析

考察汉诺塔程序的综合.

用[A,B,C]表示汉诺塔的三根柱子;用数码表示盘,数码小者为小盘;空表表示相应的柱上无盘.

对依次给出的实例

e1: ([1],[ ],[ ])→([ ],[ ],[1])

e2: ([1,2],[ ],[ ])→([ ],[ ],[1,2])

e3: ([1,2,3],[ ],[ ])→([ ],[ ],[1,2,3])

...

利用背景知识中关于 move 操作的定义,NDIPS 系统中的问题求解机制可自动产生相应的特解

S_1=[move(A,C)]

S_2=[move(A,B),move(A,C),move(B,C)]

S_3=[move(A,C),move(A,B),move(C,B),
move(A,C),move(B,A),move(B,C),move(A,C)]

...

其中,S_i 表示规模为 i (盘的个数)的解序列,即程序执行轨迹.

在具体实现时是用下列三元组来描述相应轨迹的,即

T_i=[i, [A,B,C], S_i]

第一项表示实例的规模,第二项是输入变量表,第三项是输出的解序列.

利用前面给出的匹配算法,可得:

T_1=[1,[A,B,C],[move(A,C)]]

T_2=[2,[A,B,C],[T_1(1,[A,C,B]),move(A,C),T_1(1,[B,A,C])]]

T_3=[3,[A,B,C],[T_2(2,[A,C,B]),move(A,C),T_2(2,[B,A,C])]]

...

利用前面给出的识认算法,可得

R_1(1,[A,B,C]) = [move(A,C)]

R_n(n,[A,B,C]) = [R_{n-1}(n−1,[A,C,B]),move(A,C),R_{n-1}(n−1,[B,A,C])]

最后,系统可生成如下的 Prolog 程序:

```
hanoi(1,[A,B,C]):-move(A,C),
hanoi(n,[A,B,C]):-N1 is N-1,
                  hanoi(N1,[B,A,C]),
                  move(A,C),
                  N2 is N-1,
```

hanoi(N_2,[B,A,C]).

除上述 hanoi 程序外,已综合出的程序还有:表的反转,矩阵转置,回文等等.

§6. 方法评述

NDIPS 系统得到轨迹的途径有两种:通过问题求解机制自动产生轨迹或用户以手工方式给出轨迹.

通过问题求解机制产生轨迹的主要问题在于,搜索空间随实例规模而急剧变大.为了避免搜索的盲目性,必须利用启发式知识指导搜索过程.在 NDIPS 系统中,这类启发式知识是通过"边干边学(learning while doing)"获得的[4].这种学习方式是失败驱动的:在搜索过程中,当发现死点或循环时,记录产生失败的原因,形成启发式规则,使在其后不会重蹈覆辙,从而提高搜索效率.当实例是按其规模的大小给出时,对相当一类问题而言,问题求解机制都能有效地自动产生其轨迹.

对于规模相同的实例,其轨迹则可能是不同的.例如,实例 sort([1,2])→[1,2]和实例 sort([2,1])→[1,2],它们的规模都是 2(按表的长度),但其对应的轨迹显然不同.对于这类问题必须或者限定实例的给出方式,或者手工给出该规模实例的一条公共轨迹.

对问题求解机制而言,产生同一实例的不同轨迹必须通过改变操作集的定义,而手工方式则有较大的灵活性,能充分体现用户的主观意图,但要求用户对目标程序的执行过程有充分了解,并且由于手工方式给出轨迹的随意性,匹配识认过程有时较为困难.采用人机交互方式能在一定程度上克服这一问题.因为归纳必须是在一定范围内进行,所以,领域背景知识能够指导归纳的偏向(inductive bias).以手工给出的轨迹作为一种启发式的归纳偏向,利用问题求解机制自动地或提示用户修改轨迹偏差,使匹配识认过程得以顺利进行.

已有的基于输入/输出实例和基于执行轨迹的归纳程序综合方法,与 NDIPS 系统中所采用的方法,都是依据归纳推理的枚举识认技术.因此,从能力上讲是等价的——适用于处处终止的递归可枚举程序类的综合.但由于具体实现技术有所不同,因而各种方法所能综合出的程序的复杂度有所差异.

与 Summers[5]的方法相比,NDIPS 系统能通过反映程序执行行为的轨迹得到问题的同族算法(或程序).与 Biermann[6]的方法相比,NDIPS 系统通过问题求解机制自动地产生程序执行轨迹.为了清楚地比较各种方法的特点和能力,我们选择其中较有代表性工作比较如下.

综合方法	时间复杂度	循环嵌套次数	所需实例个数	递归重数
Summers	O(n)	1	≥3	1
Regular LISP	O(n)	不限	足够多以产生分枝	1
Biermann	O(n)	不限	1	1
Kodratoff	$O(n^k)$	不限	≥k(n+2)	不限
NDIPS	O(n)	不限	若干个	不限

其中 n 是输入变量的个数，k 是条件分枝数.

结语：程序设计是一种创造性的思维活动，程序设计自动化是人们不懈追求的目标，归纳途径则是其中的一种富有生命力的研究途径. 从当前的研究现状来看，应该把研究重点放在具体领域的自动程序设计，从软件工程与人工智能相结合的角度，探讨问题症结，找到解决方法. 我们将进一步展开这方面的工作.

参考文献

[1]D. Angluin, C. H. Smith, Inductive Inference: Theory and Methods, Comp. Survey 15, 1983.

[2]D. R. Smith, The Synthesis of LISP Programs from Examples: a Survey, Automatic Program Construction Techniques, 1984.

[3]徐家福，戴敏，王志坚，归纳程序综合系统 NDIPS 的设计，《软件学报》，Vol. 1, No. 1, 1990.

[4]S. Ohlsson, On the Automated Learning of Problem Solving Rules, Cybernetics and System Research, 1982.

[5]P. D. Summers, A Methodology for LISP Program Construction from Examples, J. ACM Vol. 24, 1977.

[6]A. W. Biermann, et al., Speeding up the Synthesis of Program from Traces, IEEE Trans. on Computer, Vol. 24(2), 1975.

《第二届亚洲测试会议 ATS'93》征文通知

IEEE 计算机学会测试技术委员会(TTTC)1992 年 9 月在美国开会决定，《第二届亚洲测试会议》于 1993 年 11 月 17—18 日在中国·北京召开。这次会议由 IEEE 计算机学会测试技术委员会主办，中国计算机学会、中国科学院计算所和国家自然科学基金委员会协办。美国 IBM 公司 Ben M. Y. Hsiao 博士任大会主席，中国科学院计算所闵应骅研究员任程度主席。会议论文正在征集。征文范围主要包括：

故障模型、设计和测试集成、故障模拟、系统诊断、计算机辅助测试产生、用于测试的 AI 和专家系统、AC/时滞测试、模拟元件测试、可测试性分析、混合信号元件测试、设计验证、测试综合、自动测试设备、软件测试、可测试性设计、软件可测试性设计、内部自测试、容错系统中的在线测试、边界扫描、可靠性。

欢迎踊跃投稿。论文请用英文撰写，一式 5 份。截稿日期：1993 年 3 月 29 日，最晚不迟于 1993 年 4 月 15 日。联系地址：100080 北京市海淀区中关村中国科学院计算所 CAD 室 李忠诚。电话：(01)2565533—536。